Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Tesi di Laurea

Progettazione e implementazione di un'app Android basata sul crowdsourcing per la segnalazione di "unsafety events" in contesti urbani

Design and implementation of a crowdsourcing-based Android app for reporting unsafety events in urban contexts

Relatore

Prof. Domenico Ursino

Candidato

Michele Florio

Anno Accademico 2020-2021

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Introduzione

Negli ultimi 15 anni, il tema della sicurezza urbana ha assunto un ruolo sempre più rilevante comparendo esplicitamente nei programmi politici di governo delle città, ritagliandosi spazio tra le notizie di cronaca locale, fino a divenire oggetto di studio all'interno di corsi universitari.

Le autorità avevano già provato nel 2017 a promuovere la sicurezza urbana con la "Conversione in legge, con modificazioni, del decreto-legge 20 febbraio 2017, n. 14, recante disposizioni urgenti in materia di sicurezza delle città" (in G.U. 21/04/2017, n. 93). L'obiettivo del provvedimento consisteva nell'aumentare il coinvolgimento degli enti territoriali nel combattere il degrado urbano, grazie a un approccio integrato con le forze di polizia.

Il decreto consente il coinvolgimento di gestori di edilizia residenziale, amministratori di condomini, consorzi o comitati di professionisti e residenti, nelle operazioni di sorveglianza e monitoraggio del vicinato.

Il provvedimento è in linea con il quadro europeo in materia di sorveglianza del vicinato che, nel 2014, ha visto l'istituzione della European Neighbourhood Watch Association, la quale promuove la cooperazione dei cittadini nel contrasto al degrado urbano. Tra il 2018 e il 2019, per la prima volta, in Italia vengono costituite associazioni nazionali di sorveglianza del vicinato che si prefiggono l'obiettivo di promuovere le realtà di sorveglianza costituite spontaneamente dai cittadini.

Tali associazioni che promuovono la sicurezza urbana sono diverse migliaia ed agiscono su un territorio più o meno vasto, partendo dalla supervisione della singola palazzina fino all'intero comune. Per colpa dell'elevata frammentazione, il risultato ottenuto è l'inefficienza organizzativa; sullo stesso territorio operano spesso più associazioni che, tra loro, non comunicano e non condividono le preziose informazioni raccolte; nel peggiore degli scenari, non sono nemmeno a conoscenza della loro reciproca esistenza.

Allo stato attuale non esiste un elenco aggiornato delle realtà territoriali che supportano la sicurezza urbana e, di conseguenza, non esiste un'unica infrastruttura in grado di coordinare o archiviare il lavoro portato avanti quotidianamente da tanti cittadini volontari. Tra le difficoltà organizzative delle realtà locali, la prima è costituita dalla modalità con cui queste possono comunicare tra loro al fine di scambiarsi informazioni preziose che spesso restano confinate al "dominio locale".

Nella maggior parte dei casi, ogni gruppo condivide le informazioni soltanto tra i propri membri iscritti attraverso semplici gruppi affidati ad applicazioni per dispositivi mobili e social network. Questa soluzione, oltre ad impedire la condivisione del proprio lavoro con la comunità esterna al singolo gruppo, non rappresenta la strategia ottimale per la memorizzazione dei dati raccolti.

L'obiettivo del progetto SafeLife è la realizzazione di una piattaforma web (attualmente sviluppata dal prof. Faramondi Luca del Campus Bio-Medico di Roma) e di un'app Android ed essa collegata (argomento della tesi, sviluppata per conto di Res On Network), in grado di strutturare le informazioni raccolte dai singoli cittadini con l'intento di realizzare una mappa delle criticità del territorio al fine di ottimizzare gli sforzi delle autorità. L'output prodotto permetterebbe una pianificazione ottimizzata delle attività di tutte le forze in gioco impiegate nella sicurezza partecipata (prefetture, polizia nazionale, protezione civile, regioni, province, comuni e cittadini) e porterebbe, così, allo sviluppo di un metodo per la validazione delle misure di contrasto al degrado urbano. Infine, partendo dai dati raccolti, sarà possibile analizzare l'evoluzione di fenomeni di interesse (rapine, furti, aggressioni, etc.) al fine di predire lo sviluppo delle attività criminali.

Ogni utente avrà la possibilità di inserire una segnalazione nella piattaforma utilizzando l'app mobile di SafeLife; ogni segnalazione sarà geolocalizzata per permetterne il posizionamento sulla mappa degli eventi della zona. Il processo di acquisizione di informazioni dagli utenti è completato da una fase di verifica delle segnalazioni inviate alla piattaforma. Ogni segnalazione viene sottoposta sia ad un processo di validazione, mediante riscontro automatico su canali di informazione ufficiali, sia ad un processo continuo di riscontro da parte di utenti geograficamente vicini all'area sede dell'evento segnalato.

L'utente di SafeLife non sarà l'unica fonte di informazioni per il patrimonio informativo del sistema proposto. Infatti, la seconda risorsa che contribuirà all'incremento di questo patrimonio informativo è rappresentata dai gruppi istituiti su social network, che affrontano i temi della sicurezza urbana, della sicurezza del vicinato e del degrado urbano.

L'ultima fonte che si intende includere nel set di flussi che popoleranno il patrimonio informativo di SafeLife è rappresentata dalle fonti aperte, come giornali online, dai quali, attraverso algoritmi di machine learning (crawler web), sarà possibile strutturare le informazioni pubblicate, riguardanti gli eventi pericolosi, al fine di renderle comparabili e presentabili con i dati acquisiti dalle segnalazioni degli utenti e dai canali social.

Nel contenuto del progetto SafeLife, la presente tesi si concentra sulla progettazione e implementazione della companion app per Android.

L'obiettivo dell'app SafeLife è, principalmente, quello di permettere ai vari utenti di segnalare, in maniera rapida e semplice, un evento pericoloso di cui sono stati testimoni durante lo svolgimento delle normali attività quotidiane. Ciò permette anche ad altri utenti interessati alla medesima zona dell'accaduto di essere avvisati in merito all'evento, semplicemente guardando la mappa della zona. Inoltre, gli utenti, durante l'inserimento di un nuovo evento, possono specificare delle informazioni opzionali, oltre a quelle obbligatorie, necessarie a catalogare e collocare geograficamente l'evento stesso. L'utente può, infatti, caricare una foto del luogo dell'evento, oppure una breve descrizione dello stesso, in modo da fornire ulteriori dettagli sul-

l'accaduto per permettere una migliore distinzione tra i vari eventi presenti nella stessa zona. Tali aggiunte sono importanti soprattutto durante l'inserimento di un nuovo evento, che potrebbe risultare simile ad un altro già presente nel database; in questo caso, le informazioni extra fornite dagli utenti consentono una più facile distinzione tra gli eventi con informazioni simili presenti nel database.

Le informazioni che gli utenti inseriscono, inoltre, possono essere visualizzate da tutti (sia tramite l'app che mediante la piattaforma web) al fine di garantire la sicurezza urbana, in modo tale che un utente possa effettuare delle ricerche più approfondite degli eventi che lo circondano, grazie anche all'utilizzo di filtri che rendono la mappa dinamica rispetto alle impostazioni scelte.

Un altro aspetto fondamentale dell'app è la possibilità di scegliere la visualizzazione degli eventi sotto forma di heatmap, nella quale gli eventi vengono rappresentati come delle macchie colorate di diversa intensità e diverso colore, in base al peso del singolo evento e di quelli adiacenti. Il valore del peso associato ad un evento viene calcolato a partire dalla categoria dello stesso e dalla "vecchiaia" della segnalazione.

Un'altra caratteristica prevista per l'app è quella di utilizzare il database per realizzare una funzionalità di navigazione, che guiderà l'utente verso una destinazione attraverso una serie di cammini, come ad esempio, il cammino più sicuro che parte dalla posizione dell'utente per arrivare alla destinazione scelta.

In ogni caso, l'app di SafeLife ha come scopo principale quello di permettere agli utenti del sistema complessivo di effettuare delle segnalazioni rapide e semplici, incentivandoli ad effettuare segnalazioni, evitando, così, passaggi complessi che si avrebbero utilizzando la piattaforma web da smartphone.

Per cui, se la controparte web di SafeLife permette un maggior numero di visualizzazioni, di statistiche ed analisi degli eventi nel database, l'app rappresenta il principale strumento tramite cui un utente interagirà con la piattaforma. L'utente, perciò, sarà invogliato nell'utilizzo dell'app quando consulterà SafeLife per le strade della propria città, sia in caso di visualizzazione degli eventi pericolosi, sia quando deciderà di effettuare una segnalazione in tempo reale dell'accaduto.

Lo sviluppo dell'app è stato effettuato secondo l'approccio tipico dell'ingegneria del software. Per prima cosa, è stata reperita la documentazione da analizzare per poter elaborare i requisiti implementativi. Sulla base di questi, è stata portata avanti la progettazione, in cui, nel caso in esame, si è dovuta prestare particolare attenzione alla soluzione software che ha permesso l'integrazione tra le due parti del progetto. Dopodiché si è passati all'implementazione delle funzionalità previste per l'app, utilizzando l'IDE Android Studio per realizzare il codice in Java. L'ultimo passo è stato quello di completare l'integrazione dell'app con i moduli della piattaforma web.

La presente tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 verrà introdotto il crowdsourcing, verranno illustrate le varie tipologie di crowdsourcing e si spiegherà in che forma esso sarà integrato nel progetto.
- Nel Capitolo 2 si parlerà del progetto SafeLife nel suo insieme.
- Nel Capitolo 3 si descriverà il processo di raccolta e analisi dei requisiti.
- Nel Capitolo 4 verrà illustrato l'approccio alla progettazione delle funzionalità previste.

- Nel Capitolo 5 sarà analizzata la loro implementazione.
- Nel Capitolo 6 verrà analizzata l'integrazione dell'app con la piattaforma web.
- Nel Capitolo 7 verranno espresse delle considerazioni riguardo al lavoro svolto.
- Nel Capitolo 8 saranno tratte le conclusioni, mostrando dei possibili sviluppi futuri.

1

Crowdsourcing

In questo primo capitolo verrà descritta la pratica del crowdsourcing per quanto concerne lo sviluppo di un progetto, inoltre verranno approfonditi gli aspetti di tale metodologia impiegati per la realizzazione del progetto in esame.

1.1 Introduzione e tipologie

Con il termine "crowdsourcing" si indica in genere lo sviluppo collettivo di un progetto, base volontaria o su invito, da parte di una moltitudine di persone esterne all'azienda ideatrice.

Questo modello operativo, in genere reso possibile grazie a Internet, non riguarda necessariamente la scrittura di codice in linguaggi di programmazione: infatti tale metodo può essere applicato anche alla raccolta di dati ed informazioni o alla loro analisi da parte di un insieme di utenti al fine di realizzare un archivio alimentato dall'utenza stessa che utilizza tale applicazione oppure di delegare il compito del controllo delle voci di tale archivio all'utenza.

In ambito economico, il crowdsourcing, può essere definito come "esternalizzazione di una parte delle proprie attività", cioè un modello di business nel quale un'azienda o un'istituzione affida la progettazione, la realizzazione o lo sviluppo di un progetto, oggetto o idea a un insieme indefinito di persone non organizzate precedentemente.

Questo processo viene favorito dagli strumenti che mette a disposizione il web. Solitamente il meccanismo delle open call viene reso disponibile attraverso dei portali presenti sulla rete internet.

Il crowdsourcing inizialmente è nato dal lavoro di volontari ed appassionati che dedicavano il loro tempo libero a creare contenuti e risolvere problemi. La community open source è stata la prima a trarne beneficio.

Esistono diversi esempi di crowdsourcing su base volontaria che hanno avuto successo e che continuano tutt'ora a ricevere un costante apporto dalla loro community. Un importante esponente in questo senso è l'enciclopedia Wikipedia, che viene considerata da molti un esempio di crowdsourcing volontario i cui utenti sono sia fruitori, sorgenti e validatori delle informazioni presenti sulla piattaforma stessa.

Altri esempi comuni sono i portali che accettano recensioni libere, sia nel campo dei prodotti (come Amazon o IMDb, cioè "Internet Movie Database"), che dei servizi (come ad esempio TripAdvisor per il territorio europeo o Yelp per quello americano).

Oggi il crowdsourcing è per le aziende un nuovo modello di open enterprise, per i freelance diventa la possibilità di offrire i propri servizi su un mercato globale e per i ricercatori universitari un modo per raccogliere dati importanti e fare nuove scoperte o rivelazioni.

Il crowdsourcing può essere visto essenzialmente come un modello di produzione e risoluzione dei problemi. Nell'accezione classica del termine, viene richiesta la risoluzione di un determinato problema a un gruppo non definito di persone. Gli utenti, ovvero la "crowd" (folla), solitamente si riuniscono in comunità online, le quali forniscono una serie di soluzioni, che vengono poi vagliate dal gruppo stesso alla ricerca di quelle più adatte.

Queste soluzioni appartengono all'istituzione o all'individuo che ha inizialmente presentato il problema e gli utenti che hanno contribuito a trovarle in alcuni casi vengono ricompensati in denaro o con premi o riconoscimenti, in altri con la semplice soddisfazione intellettuale.

Grazie al crowdsourcing, le soluzioni possono provenire da utenti non professionisti o volontari, che lavorano al problema nel loro tempo libero, o da esperti e piccole imprese, che erano sconosciute all'istituzione committente.

Il termine viene spesso usato anche per riferirsi a situazioni in cui le aziende non si rivolgono alla massa intera di utenti, ma solo ad alcuni individui.

La differenza principale fra il crowdsourcing e il tradizionale outsourcing consiste nel fatto che il progetto o il problema viene esternalizzato ad un gruppo indefinito di persone, invece che ad una specifica entità.

Il crowdsourcing si distingue anche dalle produzioni open source perché in quest'ultimo caso, si tratta di attività di cooperazione iniziate e portate avanti volontariamente da taluni individui. Nel crowdsourcing, invece, il progetto viene iniziato da un committente e in seguito sviluppato da un altro individuo o gruppo.

Altre differenze fra le attività open source e quelle di crowdsourcing riguardano le diverse motivazioni che spingono gli individui a partecipare.

Passiamo, ora, ad elencare e descrivere le principali tipologie di crowdsourcing che sempre più spesso si stanno diffondendo come pratica di sviluppo collettivo all'interno delle aziende in tutto il mondo per la realizzazione di progetti altrimenti di difficile realizzazione:

- *Crowdsourcing via Web*: grazie al recente aumento delle potenzialità delle applicazioni Internet, sono migliorate anche le tecniche di crowdsourcing, e ora il termine si riferisce quasi esclusivamente ad attività svolte via Web. Il potenziale per sviluppare progetti tramite crowdsourcing usando la rete Internet esisteva già in precedenza, ma solo recentemente lo si è potuto cominciare a sfruttare appieno. Un esempio importante di crowdsourcing sul Web è rappresentato dal social bookmarking. Grazie ai sistemi di social bookmarking gli utenti possono assegnare dei tag a delle fonti condivise con altri utenti in modo da organizzare le informazioni. Il rappresentante più noto appartenente a questa categoria è Reddit, un forum in cui i propri utenti registrati possono pubblicare contenuti di natura testuale, ipertestuale o multimediale, sono proprio gli stessi utenti autori

dei loro post ad assegnare un tag al loro contenuto in maniera da organizzare le informazioni che man mano vengono inserite sulla piattaforma.

- *Collaboratition*: "Collaboratition" è un neologismo per descrivere un tipo di crowdsourcing usato per progetti che richiedono un sforzo collaborativo per essere portati avanti con successo. In tali progetti, il lavoro può essere suddiviso in team che lavoreranno su uno stesso aspetto del progetto, oppure su parti distinte in modo da poter combinare gli sforzi atti al rapido avanzamento del progetto. Un altro modo di attuare questa tipologia di crowdsourcing consiste nello spronare i vari team di sviluppo tramite la competizione; far partecipare le varie squadre ad una gara di tipo "collabora-competitivo", spesso, permette di ottenere dei risultati migliori rispetto alla collaborazione classica.
- *Crowdfunding*: un'altra forma di collaborazione è rappresentata dal crowdfunding, termine che deriva dal crowdsourcing stesso. In questo caso la collaborazione consiste nel raccogliere fondi, generalmente sul web, per sostenere le iniziative di determinate persone o organizzazioni. Si possono sviluppare progetti di crowdfunding per vari scopi, dagli aiuti umanitari alle vittime di disastri, al finanziamento di artisti da parte dei fan, alle campagne politiche. Il fenomeno del crowdfounding è in continua crescita dagli ultimi 10 anni a questa parte proprio grazie alla proliferazione e dall'affermarsi di applicazioni web e di servizi per dispositivi mobili, con delle vere e proprie piattaforme in cui i singoli imprenditori o le singole imprese possono presentare al pubblico le loro idee per la realizzazione di prodotti o servizi sollecitando il feedback degli utenti su di esse. Le piattaforme di crowdfounding più note ed influenti del momento sono Kikstarter e Indiegogo; entrambe offrono un modello di finanziamento collettivo chiamato reward-based. Tale modello è il più diffuso per numero di piattaforme e prevede, per l'investitore, una ricompensa commisurata con il contributo. Solitamente la piattaforma dà due o più scelte di contributo ordinate per entità, con ognuna associata la sua ricompensa. A seconda che l'obiettivo di finanziamento sia stato raggiunto o meno, le piattaforme di questo tipo seguono due dei seguenti schemi: Keep-it-all (tieni tutto) oppure All-or-nothing (tutto o niente).

1.2 Crowdsourcing delle informazioni

Come spiegato nella Sezione 1.1, quando si è introdotto il concetto di crowdsourcing nonchè le varie tipologie che fanno parte di tale metodo di sviluppo, è stata elencata tra queste, anche, la condivisione di informazioni, come parte fondamentale tramite cui una community di utenti è in grado di alimentare l'insieme dei vari dati messi a disposizione di tutti. In questo modo ogni utente ha potenzialmente la capacità di apportare un proprio contributo nella costruzione del pool di informazioni a cui gli utenti hanno accesso. In questo modo si garantisce, innanzitutto, un enorme sgravio di lavoro per il team di sviluppo dell'applicazione in questione, demandando, in parte o in toto, la ricerca, collezione e in alcuni casi, la verifica dei dati permettendo, al tempo stesso, alla comunità di rimanere attiva, spingendo i vari membri a dare un contributo così da poter ingrandire sempre di più l'archivio di dati del progetto.

Vediamo ora delle applicazioni di tale modalità di crowdsourcing al progetto SafeLife, distinguendo le informazioni collezionate in due diverse categorie in base alla provenienza delle informazioni raccolte.

1.2.1 Informazione utenti

Cominciamo col descrivere le informazioni che gli utenti del progetto SafeLife forniscono all'applicazione. Come già anticipato gli utenti possono inserire le informazioni relative agli eventi pericolosi in cui essi stessi si imbattono durante la loro giornata, ovunque nelle varie città italiane.

Quindi ogni utente dall'app oppure dall'applicazione web, può inserire le proprie segnalazioni, previa registrazione su uno dei due portali tramite i diversi metodi di autenticazione messi a disposizione, per poter, così, continuare ad alimentare il database dell'applicazione con delle informazioni aggiornate che, man mano, vanno a costruire una fitta rete di nodi (eventi) che popolerà la mappa di tutte le città man mano che il numero di utenti crescerà.

Un utente che voglia effettuare un inserimento di un evento, dopo aver selezionato la schermata che permetterà di aggiungere nuovi eventi (dall'app o dalla pagina web) inserirà una serie di informazioni relative all'evento stesso, alcune delle quali sono obbligatorie mentre altre opzionali.

Le informazioni obbligatorie per l'inserimento di un nuovo evento sono:

- *Categoria*: rappresenta la classe di categoria a cui l'evento appartiene; l'utente deve necessariamente scegliere una tra le categorie presenti nel menù a tendina.
- *Data*: data di avvenimento dell'evento, è possibile inserire la data selezionandola dal calendario che si apre premendo il pulsante accanto al campo in questione; la data non può essere scelta oltre quella odierna o precedente ad un anno di tempo.
- *Ora*: orario in cui è avvenuto il fatto; anche qui tramite pulsante si può scegliere comodamente la data dall'orologio.
- *Luogo*: luogo in cui è accaduto l'evento; esso può essere inserito manualmente sotto forma di indirizzo oppure è possibile usare il pulsante della geolocalizzazione, che permette di mettere come luogo dell'evento quello del dispositivo.

Le informazioni facoltative sono:

- *Descrizione*: è un campo vuoto in cui l'utente può inserire una breve descrizione dell'evento.
- *Immagine*: è un'immagine che può essere inserita dall'utente a supplemento dell'evento.

Gli utenti non hanno solo il compito di fornire informazioni per incrementare i numeri di casi registrati dal database dell'app, ma hanno anche il compito cruciale della validazione dei dati inseriti. Infatti, ogni evento inserito da un utente presenta un campo chiamato "verificato" che serve ad indicare il livello di fidatezza dell'evento in questione.

Ogni evento utente parte con un livello di "verificato", pari a 0 (il livello massimo è pari a 3), per poi essere incrementato man mano che un altro utente cerca di

inserire un nuovo, evento che risulta essere simile ad un altro evento già presente nel database. L'incremento avviene nel momento in cui l'utente da la conferma che i due eventi sono, in effetti, lo stesso; in questo caso il nuovo evento non viene inserito per evitare di duplicare le informazioni; però il vecchio evento presente nel database viene aggiornato incrementando di 1 il livello di "verificato".

Per di più, un evento può avere il proprio livello di "verificato" incrementato non solo dagli eventi segnalati dagli utenti, ma anche dagli eventi ricavati dagli articoli. Infatti, ogni volta che il codice che si occupa dell'inserimento degli articoli nel database effettua una nuova operazione di inserimento, si procede con una verifica degli eventi simili già presenti nel database e nel caso ci siano corrispondenze, il nuovo evento ricavato dall'articolo viene inserito normalmente, ma l'evento già presente viene aggiornato con un livello di "verificato" che viene piazzato direttamente al livello massimo 3, in quanto gli eventi ottenuti dagli articoli sono inerentemente informazioni sicure.

Per concludere con gli eventi inseriti dagli utenti e la fase di inserimento di uno di questi eventi, nonchè della verifica stessa che l'utente deve svolgere in questo caso, è necessario illustrare il controllo che l'app deve eseguire ad ogni nuovo inserimento per recuperare le informazioni degli eventi simili già inseriti in precedenza nel database.

Tale controllo è necessario proprio per permettere all'utente di verificare se il nuovo evento coincida con uno vecchio; proprio per poter prendere una decisione, l'utente deve avere a disposizione il maggior numero di informazioni disponibili riguardo al vecchio evento.

Si rende, quindi, necessario definire un set di parametri che ci permettano di stabilire in maniera univoca quali sono gli eventi simili da mostrare all'utente. Dopo delle sessioni di brainstorming abbiamo convenuto che i parametri da prendere in considerazione sono i seguenti:

- *Distanza*: è il primo tra i vari parametri da analizzare, cioè la distanza tra i due eventi. Siccome è necessario trovare un compromesso tra numero di eventi simili che riusciamo ad individuare ed il numero di eventi che dovranno essere analizzati alla ricerca di similarità, si è deciso di impostare la ricerca ad 1 Km di distanza dall'evento che l'utente sta ceracndo di inserire. Gli eventi all'interno del raggio verranno analizzati per scoprire somiglianze e, perciò, proseguiranno con il resto dell'analisi; i restanti eventi, invece, non verranno presi in considerazione.
- *Categoria*: a partire dagli eventi che soddisfano le condizioni precedenti, verrà, poi, analizzata la categoria a cui essi appartengono. Passeranno ai prossimi controlli solo gli eventi che hanno la stessa categoria del nuovo evento.
- *Data*: a partire dagli eventi che soddisfano le condizioni precedenti, verrà poi analizzato il giorno dell'avvenimento dell'evento riportato come informazione; gli eventi che proseguiranno al livello successivo sono quelli con la stessa data del nuovo evento.
- *Ora*: a partire dagli eventi che soddisfano le condizioni precedenti, verrà poi analizzato l'ultimo parametro, cioè l'ora dell'evento. Gli eventi che verranno mostrati all'utente per la verifica sono solo quelli che hanno un orario dell'evento che si differenzia di 30 minuti (in più o in meno) rispetto al nuovo evento.

Tutti gli eventi che hanno superato questi controlli saranno, poi, mostrati all'utente con tutte le informazioni a loro collegate, l'utente avrà la possibilità di decidere se uno tra tali eventi simili è lo stesso evento che sta cercando di inserire in quel momento.

1.2.2 Informazioni estrapolate dal web

In questa sezione ci concentreremo nello spiegare la provenienza dei dati ottenuti dal web che andranno a costituire gli eventi ottenuti dagli articoli all'interno della nostra app; inoltre, spiegheremo più nello specifico il funzionamento del crawler che si occupa della raccolta di tali informazioni.

Come anticipato in precedenza la raccolta di queste informazioni è stata curata dall'Univerità Campus Bio-Medico di Roma.

Cominciamo ora a descrivere cos'è un crawler; successivamente, illustreremo alcuni passaggi fondamentali della sua creazione.

Un crawler è un software che analizza i contenuti di una rete (o database) in modo metodico e automatizzato. Esso è solitamente un bot (programma o script che automatizza la ricerca) che acquisisce una copia dei documenti presenti all'interno di una o più pagine web. Una volta che un sito è stato visitato ne rileva il contenuto che può analizzare, nonchè tutti i collegamenti interni ed esterni e li memorizza in un database.

L'obiettivo dei bot è quello di conoscere di cosa tratta ogni pagina in modo che le informazioni possano essere recuperate quando è necessario Figura 1.1.

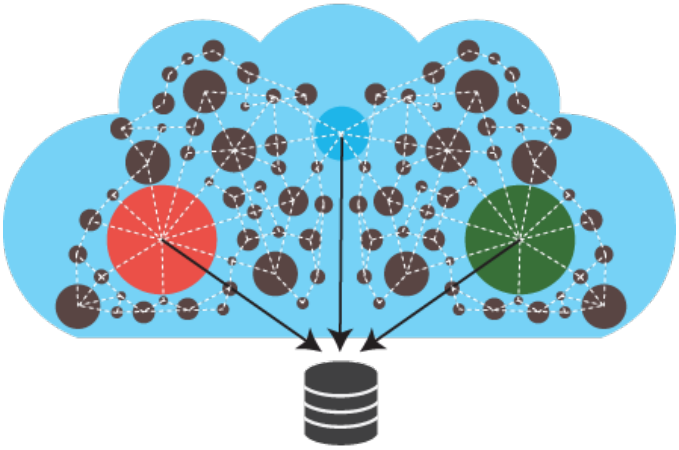

Figura 1.1. Rappresentazione del lavoro di un crawler web

In particolare, si definisce web crawler un software pensato per accedere a un sito web e ottenere dati tramite una scansione automatica. I crawler sono quasi sempre gestiti dai motori di ricerca, che applicano un algoritmo ai dati raccolti fornendo, poi, collegamenti pertinenti in risposta alle query di ricerca dell'utente, che, a loro volta, generano gli elenchi di pagine su Google o Bing (o un altro motore di ricerca).

La maggior parte dei crawler web non esegue la scansione di tutte le risorse disponibili online, ma decide quali pagine scansionare in base al numero di altre pagine collegate, alla quantità di visitatori e ad altri fattori che indicano la probabilità che la pagina contenga informazioni importanti e pertinenti.

Un crawler che esegue la ricerca web ha alla base l'utilizzo degli URL che vengono definiti seed; da essi parte la ricerca. I seed, composti da un URL o da una lista fornita dall'operatore o dall'iniziale ricerca web, da quel momento identificano tutti i contenuti ipertestuali presenti nel documento e inseriscono in una lista o database i collegamenti riscontrati per poterli analizzare in un secondo momento.

Gli URL costitutivi della lista sono chiamati crawl frontier verranno visitati ricorsivamente dal crawler così da poter registrare eventuali modifiche o aggiornamenti. Naturalmente anche gli URL identificati dalla ricerca di frontiera saranno inseriti all'interno della lista in modo da poter essere visitati in un secondo momento. In questo modo si andrà a costruire una "ragnatela" di pagine web, legate le une alle altre attraverso collegamenti ipertestuali.

Il crawler, nel momento in cui agisce in modalità di "archiviazione", come nel nostro caso, copia e conserva i contenuti di ciascuna pagina visitata. Il processo può essere concluso manualmente o dopo che un determinato numero di collegamenti è stato eseguito.

In Figura 1.2 viene riportato uno schema di algoritmo di crawling.

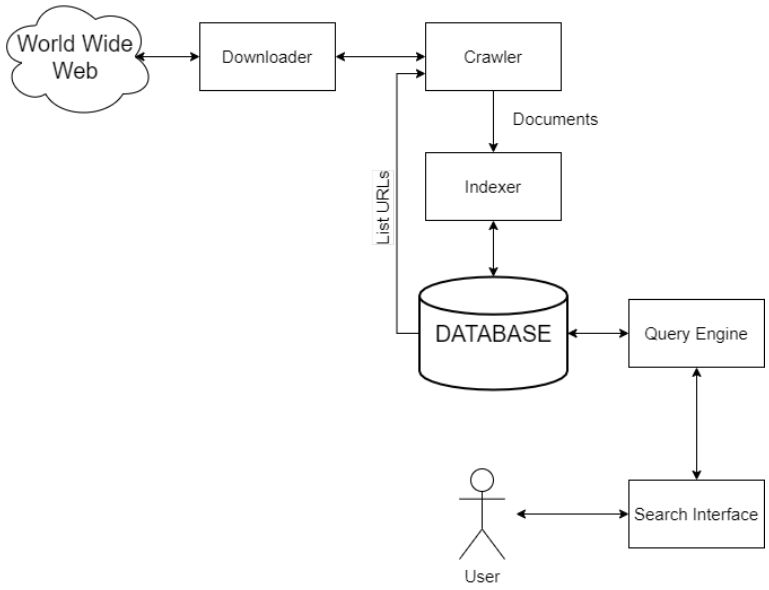

Figura 1.2. Schema di algoritmo di crawling

Vogliamo ora, analizzare un esempio di algoritmo di crawling. Per poterlo spiegare, però, è necessario introdurre alcune definizioni preliminari:

- *frontier*: è un insieme da cui vengono prelevati (e quindi rimossi) i link da seguire ed in cui vengono inseriti i nuovi link trovati;

- *seed*: sono gli indirizzi (URL) di partenza del processo di crawling;
- *link*: identifica tutti i documenti già analizzati;

L'algoritmo presenta, quindi, la seguente struttura:

1. Recupera un link dalla frontiera.
   a) Il link viene eliminato da F ed inserito in L.
2. Scarica il codice HTML della pagina al fine di:
   a) analizzare la pagina secondo gli scopi del crawler.
   b) estrarre tutti i link della pagina.
3. Aggiunge alla frontiera ogni link estratto che non è presente né in L né in F.
4. Ritorna al punto 1 se la frontiera contiene ancora link, altrimenti termina.

Si noti che al punto 3 è possibile aggiungere ulteriori condizioni per l'inserimento dei link nella frontiera, ad esempio per limitare il crawling all'interno di uno specifico dominio. In questo caso più propriamente di parla di web scraping.

Dopo aver visto che cos'è un crawler e quale è il suo funzionamento di base, passiamo ora ad esplorare nel dettaglio come questo specifico web crawler è stato progettato per la raccolta delle informazioni riguardanti articoli di cronaca.

L'estrazione delle informazioni si suddivide in 3 fasi distinte, per ciascuna di queste fasi verrà utilizzato un diverso crawler web che ha compiti diversi e specifici rispetto ai crawler delle altre fasi.

La prima è una fase preparatoria in cui si predispongono le basi per gli step successivi. Si predispone quindi in questo momento il database, fornendo lo schema preliminare e le relazioni che lo costituiranno. Inoltre, viene eseguito il primo crawler web, che raccoglie le notizie presenti sulle diverse testate del quotidiano Today, in particolare nella sezione cronaca. Un esempio di articolo del quotidiano Today lo si può vedere nella Figura 1.3.

Le tabelle sono quindi:

1. *ArticoloDaProcessare*: in questa tabella saranno raccolti gli URL degli articoli, che saranno successivamente analizzati; tali articoli saranno identificati da un id specifico.
2. *Articolo*: questa tabella sarà la più corposa in quanto al suo interno verranno raccolte tutte le informazioni estrapolate da un successivo crawler sugli articoli presenti nella tabella sopra citata.
3. *Categorie*: questa tabella raccoglierà le parole chiave che identificano i contenuti principali degli articoli, quindi che costituiscono dei sottogruppi tematici a cui ricondurre le informazioni ivi contenute.
4. *Art2cat*: una tabella che mette in comunicazione l'identificativo dell'articolo con quello della corrispettiva categoria, di modo da associarli, seguendo le regole dei database relazionali che vogliono evitare ridondanza di informazioni.
5. *Reazioni*: questa tabella entra nell'ambito della raccolta delle informazioni provenienti dai social; una, quindi, mantiene al proprio interno le informazioni riguardo alle reazioni sui social.
6. *TweetScore*: questa tabella è strettamente legata con quella precedente. Infatti, in essa, vi sono raccolte le informazioni ottenute da un'analisi più approfondita riguardo i commenti social alle notizie analizzate, concentrandosi principalmente sul social media Twitter.

Si svolge in questa prima fase anche l'implementazione della prima tabella *ArticoloDaProcessare*. Per fare ciò si realizza un crawler che cattura gli URL connessi alla pagina della sezione del quotidiano d'interesse. Si svolge la ricerca all'interno della sezione cronaca poiché le notizie che si ricercano sono di carattere prevalentemente locale; di conseguenza si orienta la ricerca verso una analisi particolareggiata e che ci si auspica fornisca maggiori risultati.

La profondità del crawler arriva fino il secondo grado; questo perché raccogliendo notizie solo da questa sezione si ottiene una buona copertura informativa senza appesantire eccessivamente la ricerca.

La tabella *ArticoloDaProcessare* si può considerare una tabella di "appoggio", in quanto gli URL saranno contenuti all'interno della stessa solo finché non verrà elaborato l'articolo. Una volta che esso verrà analizzato ed inserito nella tabella *Articolo*, sarà automaticamente cancellato da questa tabella.

Il crawling web sarà eseguito in modo automatizzato attraverso l'esecuzione di un servizio. Ciò permette di eseguire più volte nel corso della giornata la ricerca di eventuali notizie senza un controllo esterno dell'operatore, permettendo di assicurarci una copertura esaustiva della situazione corrente.

Nella seconda fase del lavoro si implementa un secondo crawling che effettua un parse del codice HTML connesso all'URL presente nella tabella *ArticoloDaProcessare*.

In questa fase si popolano, quindi le tre tabelle: *Articolo*, *Categorie* e *Art2Cat*, create in precedenza.

Si inizia con la tabella articolo che costituisce il cuore della raccolta dei dati e della sua elaborazione.

Si esegue, quindi, il crawling partendo dagli articoli raccolti in precedenza nella tabella *ArticoloDaProcessare*. Tale esecuzione si lancia, ovviamente, sull'URL individuato.

Una volta che si è eseguito il parse della pagina, si procede con l'eliminazione dell'URL dalla tabella precedente. Per essere più precisi, si procede con l'eliminazione dell'articolo solo qualora siano stati inseriti correttamente tutti i campi cercati. Inoltre, in questa fase, si inserisce un filtro che seleziona esclusivamente gli articoli attinenti alle tematiche da noi cercate.

Analizziamo la struttura della tabella; questa è costituita dai seguenti campi:

- *IdArticolo*: rappresenta la chiave principale della tabella. È l'identificativo numerico dell'Articolo all'interno del database e coincide con quello della tabella *ArticoloDaProcessare*.
- *URLarticoloRT*: indica l'URL dell'articolo in modo tale da consentire di risalire all'articolo iniziale in qualsiasi momento.
- *DataInserimentoCrawler*: fornisce l'indicazione temporale di quando il record è stato inserito all'interno della tabella *Articolo*.
- *DataArticoloRT*: questo dato viene estrapolato direttamente dal codice HTML e corrisponde alla data di pubblicazione dell'articolo; è auspicabile che coincida con la data in cui è avvenuto l'evento.
- *DataProcessamento*: il dato viene inserito in automatico dall'applicativo. Si inserisce questa data nel momento in cui tutti i restanti campi della tabella sono stati riempiti, prima di eliminare lo stesso articolo dalla tabella *ArticoloDaProcessare*. In questo modo si tiene traccia temporale dell'elaborazione dell'informazione.

- *LuogoRT*: indica il luogo in cui è avvenuto l'evento; anch'esso viene acquisito direttamente dalla pagina web, ove presente.
- *URLimgRT*: riporta l'URL identificativo dell'immagine presente all'interno dell'Articolo.
- *Lat*: indica la latitudine; si ricava partendo dall'indicazione del luogo presente nell'articolo e eseguendo una richiesta HTTP all'APIRest di Google.
- *Lon*: indica la longitudine; si ricava partendo dall'indicazione del luogo presente nell'articolo e eseguendo una richiesta HTTP all'APIRest di Google.
- *Titolo*: riporta il titolo dell'articolo.
- *IntroArticoloRT*: riporta l'introduzione, noto come cappelletto dell'articolo, nella quale vengono riassunte ed espresse le informazioni chiave in esse contenute.
- *Testo*: riporta l'intero contenuto testuale dell'articolo.

Figura 1.3. Esempio di articolo presente su Roma Today

Come detto in precedenza nella tabella *ArticoloDaProcessare* sono già presenti gli ID e gli URL degli articoli presenti nella pagina di cronaca di Roma Today e che verranno riportati nella tabella in esame.

Cominciando da questi URL si richiede la creazione di un file document che conterrà, al proprio interno, le informazioni presenti nella pagina web, sotto forma di linguaggio HTML.

Resta ora da descrivere la terza ed ultima fase, quella che si concentra sugli aspetti social; in questo caso si esegue un'analisi delle reazioni che gli utenti hanno alle notizie sui social media.

Siccome però le informazioni riguardanti i social non sono state ancora prese in considerazione in questa fase del progetto SafeLife (le implementazioni social nell'app sono un punto di sviluppo futuro), ci limiteremo ad una spiegazione rapida di questa terza fase.

La prima scelta fatta in questa fase è stata quella di selezionare il "social" su cui effettuare la ricerca. Per la capillare presenza sul territorio nazionale si è deciso di analizzare le notizie riportate dal gruppo Today. Quindi si è proceduto ad individuare quale fosse il social maggiormente utilizzato, all'interno del circuito Today, per il tipo di informazioni d'interesse. L'idea alla base era di cercare di comprendere quale fosse l'opinione della popolazione rispetto alle notizie prese in esame e di impostare una preliminare analisi del sentiment; in particolare, si sono presi in esame Facebook e Twitter.

Le informazioni sono state raccolte attraverso request alle API messe a disposizione dai social network. Poi su queste informazioni è stata effettuata una sentiment analysis che si tradurrà nell'assegnare uno score numerico al tweet (o post) collezionato.

L'analisi si struttura creando tre liste di radici di parole, una per quelle cosiddette positive che, quindi, andrebbero ad esprimere un sentimento favorevole o positivo nei confronti della notizia, e una per le cosiddette radici negative che al contrario, vanno ad esprimere un sentimento avverso e negativo nei confronti della notizia. Infine, vi è una terza lista con delle radici di parole da escludere e che, quindi, non si devono prendere in considerazione perché non forniscono un sentimento netto e definito nei confronti di ciò che si sta commentando e anzi, potrebbero creare delle ambiguità di giudizio.

Si procede, poi, confrontando le parole presenti nel testo del tweet (o post) con quelle all'interno delle liste citate in precedenza. A seconda che sia presente una radice positiva o negativa si assegna, rispettivamente, un punteggio positivo o negativo. In questo modo si avrà un punteggio quanto più alto quanto più la valutazione del tweet (o post) è positiva al contrario più il tweet (o post) ha un'accezione negativa tanto più esso avrà un punteggio minore di zero.

2

Il progetto SafeLife Advisor

In questo capitolo si fornirà una panoramica sul progetto SafeLife nel suo insieme; successivamente, si discuterà dell'idea che ha portato al concepimento del progetto sia come piattaforma web che come app Android; proprio su quest'ultima si concentrerà il resto della tesi.

2.1 Introduzione al progetto

Il progetto SafeLife Advisor è il frutto della collaborazione tra L'univerità Politecnica delle Marche, l'Università Campus Bio-Medico di Roma e la Res On Network, tramite la Fondazione Margherita Hack.

In questa sezione esporremo il problema che il progetto si prefigge di risolvere e quale soluzione è stata ideata.

2.1.1 Introduzione al problema

Negli ultimi 15 anni, il tema della sicurezza urbana ha assunto un ruolo sempre più rilevante comparendo esplicitamente nei programmi politici di governo delle città, ritagliandosi spazio tra le notizie di cronaca locale, fino a divenire oggetto di studio all'interno di corsi universitari.

L'interesse delle autorità nella promozione della sicurezza urbana emerge nella prima metà del 2017 con la "Conversione in legge, con modificazioni, del decreto-legge 20 febbraio 2017, n. 14, recante disposizioni urgenti in materia di sicurezza delle città" (in G.U. 21/04/2017, n. 93). L'obiettivo principale del provvedimento consiste nel potenziare il coinvolgimento degli enti territoriali nel contrastare il degrado urbano, attraverso un approccio integrato con le forze di polizia.

Con l'intento di incentivare una politica di promozione della sicurezza territoriale locale, il decreto apre al coinvolgimento di gestori di edilizia residenziale, amministratori di condomini, consorzi o comitati di professionisti e residenti, nel processo di sorveglianza e monitoraggio del vicinato.

Il provvedimento è perfettamente in linea con il quadro europeo in materia di controllo del vicinato che, nel 2014, ha visto l'istituzione della European Neighbourhood Watch Association, la quale promuove la cooperazione dei cittadini nel

contrasto al degrado urbano. Tra il 2018 e il 2019, anche in Italia vengono costituite associazioni nazionali di controllo di vicinato che si prefiggono l'obiettivo di dare visibilità alle realtà di controllo promosse spontaneamente dai cittadini. L'organizzazione nazionale passa attraverso la condivisione delle "best practice" collaudate negli anni e la promozione di corsi di formazione su argomenti inerenti alla prevenzione passiva, e l'adozione di strumenti informatici per l'analisi statistica e geo-referenziata dei reati commessi nel proprio territorio. Proprio la condivisione delle informazioni è il vero anello debole della struttura che spontaneamente si è costituita nel tempo.

In Italia, le associazioni che promuovono la sicurezza urbana sono nell'ordine delle migliaia ed agiscono su un territorio più o meno vasto, partendo dal monitoraggio della singola palazzina fino all'intero comune. Il risultato di tale frammentazione è l'inefficienza organizzativa; sullo stesso territorio operano spesso più associazioni che, tra loro, non comunicano e non condividono le preziose informazioni raccolte e nel peggiore degli scenari, non sono a conoscenza della reciproca esistenza.

Attualmente non esiste un elenco aggiornato delle realtà territoriali che supportano la sicurezza urbana e, di conseguenza, non esiste un'infrastruttura in grado di coordinare il lavoro portato avanti quotidianamente da tanti cittadini volontari.

In Italia, un primo passo verso l'organizzazione centralizzata delle attività è stato compiuto dall'ACDV (Associazione Controllo del Vicinato) che conta centinaia di gruppi locali con la partecipazione attiva di oltre 67.000 persone. Tra le difficoltà organizzative delle realtà locali, la prima è rappresentata dalla modalità con cui queste possono dialogare tra loro al fine di scambiarsi informazioni preziose che spesso restano di "dominio locale".

Nella maggior parte dei casi, ogni gruppo condivide le informazioni solo al suo interno attraverso semplici gruppi affidati ad applicazioni per dispositivi mobili e social network. Questa soluzione, oltre ad impedire la condivisione del proprio lavoro con il resto della comunità, non rappresenta la strategia ottimale per la memorizzazione dei dati raccolti.

L'obiettivo del progetto SafeLife è la realizzazione di una piattaforma in grado di strutturare le informazioni raccolte dai singoli cittadini con l'intento di realizzare una mappa delle criticità del territorio al fine di ottimizzare gli sforzi delle autorità. L'output prodotto permetterebbe una pianificazione ottimizzata delle attività di tutte le forze in gioco nella sicurezza partecipata (prefetture, polizia nazionale, protezione civile, regioni, province, comuni e cittadini) e porterebbe allo sviluppo di un metodo per la validazione delle misure di contrasto al degrado urbano. Infine, partendo dai dati raccolti, sarà possibile analizzare l'evoluzione di fenomeni di interesse (rapine, furti, aggressioni, etc.) al fine di predire lo sviluppo delle attività criminali.

2.1.2 Soluzione proposta

SafeLife Advisor è una piattaforma web che nasce con l'obiettivo di fornire un supporto logistico nella gestione del patrimonio informativo di qualsiasi realtà all'interno del circuito della sicurezza partecipata. L'idea alla base di SafeLife Advisor è quella di realizzare un collettore di informazioni, raccolte principalmente mediante segnalazioni volontarie degli utenti, in merito ad eventi legati alla sicurezza urbana.

L'utente che accetta di partecipare al progetto sarà, dunque, la prima fonte di informazioni che popoleranno il patrimonio informativo di SafeLife Advisor in merito al degrado urbano.

Ogni utente potrà inserire una segnalazione nella piattaforma utilizzando l'app mobile di SafeLife Advisor; ogni segnalazione sarà geolocalizzata ed assegnata ad una delle sezioni previste dal sistema. Il processo di acquisizione di informazioni dagli utenti è completato da una fase di verifica delle segnalazioni inviate alla piattaforma. Ogni segnalazione viene sottoposta sia ad un processo di validazione, mediante riscontro automatico su canali di informazione ufficiali sia ad un processo continuo di riscontro da parte di utenti geograficamente vicini all'area sede dell'evento segnalato.

L'utente di SafeLife Advisor non è l'unica fonte di informazioni per il patrimonio informativo del sistema proposto. Infatti, nella roadmap di realizzazione del progetto è prevista l'integrazione di un'altra principale fonte di acquisizione di informazioni.

La seconda risorsa che contribuirà all'incremento del patrimonio informativo del sistema proposto è rappresentata dai gruppi istituiti su social network, che affrontano i temi della sicurezza urbana, della sicurezza del vicinato e del degrado urbano. A partire da tale fonte un processo automatico analizzerà i post pubblici degli utenti al fine di trarre informazioni che possano essere integrate con le segnalazioni volontarie, o semplicemente possano rappresentare una conferma di una segnalazione già ricevuta in precedenza.

L'utilizzo dei canali social per la raccolta di informazioni permette un'analisi molto accurata dell'evento rispetto alla singola segnalazione effettuata dall'utente.

Per i dati estratti dai social network è, infatti, possibile stimare con un buon grado di approssimazione non solo la tipologia di evento in esame ma anche il grado di importanza del fatto riportato. Sarà, infatti, possibile stimare il grado di rilevanza dell'evento sui social network valutando la pervasività dell'informazione condivisa e le reazioni degli altri utenti che esprimono la propria opinione in merito all'evento esaminato.

I commenti degli utenti, le reazioni e le condivisioni del medesimo evento sui social network sono tutti dati aggiuntivi che permetteranno a SaveLife Advisor di raccogliere informazioni aggiuntive in grado di completare la segnalazione dell'evento acquisito.

Secondo la roadmap ipotizzata, l'ultima fonte che si intende includere nel set di flussi che popoleranno il patrimonio informativo di SafeLife Advisor è rappresentata dalle fonti aperte, come giornali online, dai quali, attraverso algoritmi di machine learning, sarà possibile strutturare le informazioni pubblicate di interesse al fine di renderle comparabili e presentabili con i dati acquisiti dalle segnalazioni degli utenti e dai canali social.

A regime, l'obiettivo è quello di raggiungere un alto grado di integrazione tra i dati raccolti da ciascuna delle tre fonti (utenti, canali social e fonti aperte) al fine di attuare un processo di validazione delle informazioni acquisite che utilizza ciascuna fonte come chiave per la validazione delle altre.

Il prodotto che SafeLife Advisor potrà proporre, a partire dalle informazioni acquisite, consisterà in una dashboard in grado di visualizzare il livello di criticità e degrado di aree urbane permettendo all'utente di personalizzare l'analisi fornita attraverso l'applicazione di filtri.

Tale dashboard sarà consultabile attraverso sito web e applicazione mobile. Le informazioni collezionate dalla piattaforma saranno presentate all'utente mediante tre differenti tipologie di analisi dei dati. Queste sono:

- *Analisi georeferenziata*: In questo tipo di analisi i dati saranno rappresentati attraverso su una heatmap dell'area geografica di interesse dell'utente, al fine di visualizzare, con una scala di colori predefinita, il livello di degrado o criticità di una determinata area rispetto ad una o più categorie di eventi selezionate dall'utente. Quest'ultimo, oltre a poter specificare la tipologia di evento e l'area geografica a cui è interessato, potrà filtrare i risultati anche selezionando un orizzonte temporale specifico.
- *Trend degli eventi*: Questo tipo di visualizzazione permetterà all'utente di valutare come evolve nel tempo il numero di eventi legati al degrado urbano, mostrando la frequenza con cui si verificano, in un intervallo di tempo scelto, gli eventi afferenti a una o più categorie di interesse.
- *Pervasività degli eventi*: L'ultima modalità propone all'utente di visualizzare quali sono gli eventi che hanno suscitato maggiore interesse. Tale analisi terrà conto del numero di segnalazioni ricevute per lo stesso evento e della risonanza scaturita su canali social e sulle fonti aperte. Anche in questo caso l'utente potrà filtrare il ranking degli eventi di interesse selezionando aree geografiche e intervalli temporali preferiti.

La dashboard di SafeLife Advisor, oltre a permettere l'elaborazione numerica dei dati raccolti, offrirà un'altra funzionalità, ovvero la pianificazione di un percorso da un punto di partenza a un punto di arrivo di interesse.

Il percorso in questione potrà essere un percorso a piedi, con l'auto o attraverso l'utilizzo di mezzi pubblici; esso rappresenterà il miglior compromesso tra lunghezza del percorso ed il livello di criticità delle aree attraversate.

Anche in questo caso l'utente potrà personalizzare il processo di definizione del percorso specificando l'orario in cui intende intraprendere il viaggio e la tipologia di eventi da tenere in considerazione per il calcolo della criticità delle aree che attraverserà.

L'ultima funzionalità prevista dall'applicazione mobile consente a ciascun utente di inviare ad altri utenti nella zona una notifica in caso di pericolo, al fine di richiedere aiuto e far intervenire nell'area altri utenti in suo soccorso.

L'infrastruttura del sistema viene riportata nella Figura 2.1.

Come si evince dalla figura è possibile individuare quattro livelli per il sistema.

Il primo livello è quello che si occupa del reperimento dei dati. Da quanto descritto in precedenza, a regime, ci saranno tre modalità di reperimento dei dati, ovvero:

- *Reperimento tramite app*: In questo caso è necessario un collettore delle informazioni, che si occupa di ricevere le informazioni fornite dagli utenti tramite app e di passarle al validatore delle informazioni; quest'ultimo si occuperà di verificare se esse sono o meno affidabili. In caso affermativo procederà alla loro memorizzazione nel Data Lake.
- *Reperimento tramite social network*: In questo caso è necessario un crawler che si occupa di reperire le informazioni di interesse dai social network e di passarle

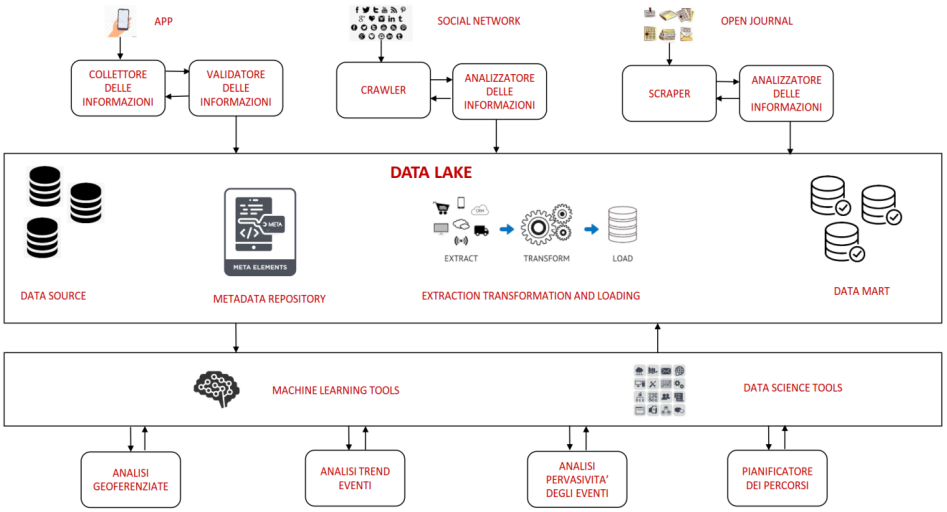

Figura 2.1. Infrastruttura del progetto SafeLife

ad un analizzatore delle informazioni. Quest'ultimo ha il compito di analizzare le informazioni ricevute e di verificare se sono attendibili. In caso affermativo procederà alla loro memorizzazione del Data Lake.

- *Reperimento tramite open journal*: In questo caso è necessario uno scraper che si occupa di esaminare gli open journal per estrarre informazioni di interesse e di passarle ad un analizzatore delle informazioni, il cui comportamento è analogo a quello del corrispettivo modulo visto per le social network.

Il secondo livello è quello che si occupa della memorizzazione dei dati. A tal fine, si è scelta, come architettura di questo livello, quella dei Data Lake in quanto essa garantisce una grande flessibilità e, allo stesso tempo, è la più adatta per scenari come il nostro, in cui i dati variano molto rapidamente. Tale architettura prevede, innanzitutto, la presenza di una serie di Data Source che memorizzano tutte le informazioni provenienti dal livello precedente.

Come previsto dalle architetture di Data Lake, accanto alle sorgenti dati, viene costruito un Metadata Repository, che ha il compito di facilitare l'estrazione dei dati di interesse. Ogniqualvolta è necessaria l'elaborazione di una parte dei dati del Data Lake, i dati grezzi di interesse devono essere ripuliti, trasformati e opportunamente integrati.

Tutto questo viene effettuato dal modulo di Extraction, Transformation e Loading che si occupa di effettuare tali attività e di costruire gli opportuni Data Mart per le elaborazioni successive. Il terzo livello è quello che si occupa di estrarre le informazioni di interesse dai dati grezzi. A tal fine esso prevede la presenza di svariati tool di Machine Learning e di Data Science al proprio interno (tool di Data Mining, di Sentiment Analysis, di Social Network Analysis, di Log Mining e di Cognitive Computing).

Le informazioni così estratte verranno passate ai tool che compongono il quarto livello, che saranno, poi, quelli che forniranno i servizi finali agli utenti. Tali tool,

in linea con quanto descritto in precedenza, saranno un sistema per le analisi georeferenziate, un sistema per l'analisi dei trend degli eventi, un sistema per l'analisi della pervasività degli eventi e, infine, un sistema per la pianificazione dei percorsi.

2.2 L'applicazione web

In questa sezione descriviamo le caratteristiche principali della piattaforma web a supporto del progetto SafeLife Advisor. Essa è una delle componenti che si dovrà interfacciare con l'app oggetto della presente tesi. La piattaforma web prevede i seguenti tool:

- *Sistema di acquisizione di notizie*: attualmente due script Java gestiscono l'acquisizione di news da quotidiani online e possono essere schedulati ad orari predefiniti. Vengono archiviate in un database le statistiche sugli eventi che ci interessano (eventi di sicurezza, in primis), comprensivi di descrizione dell'evento e localizzazione dello stesso su mappa georeferenziata. Nella Figura 2.2 è possibile osservare una panoramica di un'analisi georeferenziata così come si presenta sulla piattaforma web.

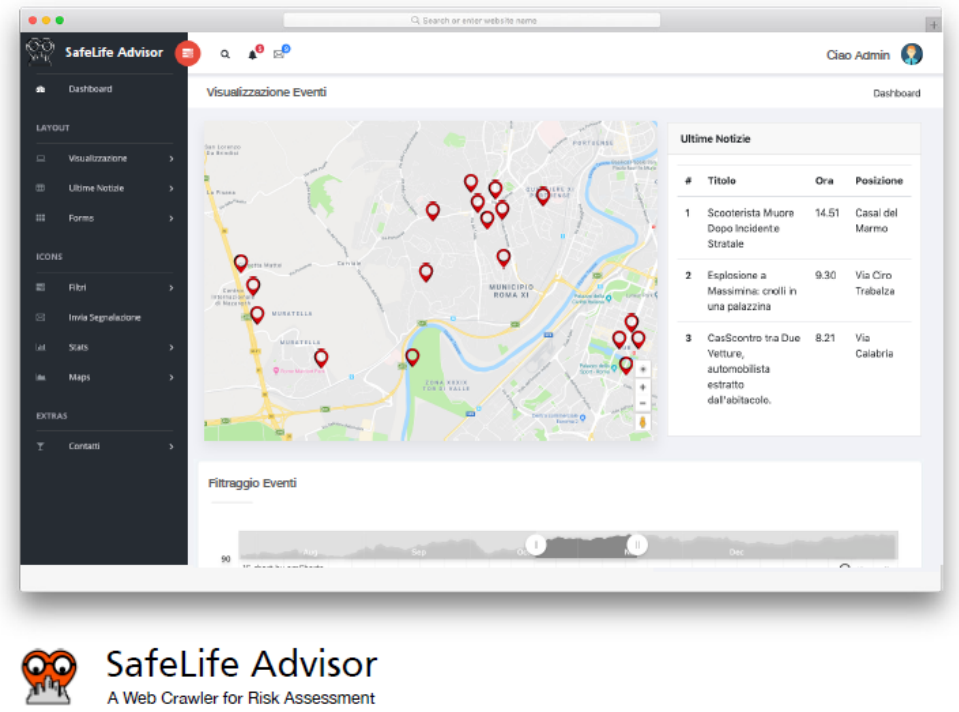

Figura 2.2. Panoramica di un'analisi georeferenziata sulla piattaforma web di SafeLife

- *Applicazione di algoritmi di sentiment analysis*: sono raccolti commenti dagli utenti riguardo alle notizie e agli eventi dai social, per integrare le informazioni e creare una rete strutturata intorno all'evento utile per il monitoraggio del potenziale mediatico.
- *Visualizzazione delle informazioni*: l'interfaccia grafica permette anche di avere una dashboard che possa fondere insieme eventi e commenti degli utenti, oltre alla localizzazione delle notizie, la valutazione della gravità degli eventi e, infine, l'individuazione degli eventi più vicini all'utente che visualizza il sito. Nella Figura 2.3 è possibile osservare la dashboard dell'amministratore della piattaforma web; in particolare è possibile riconoscere le diverse fonti di dati che compongono gli elementi del database del progetto.

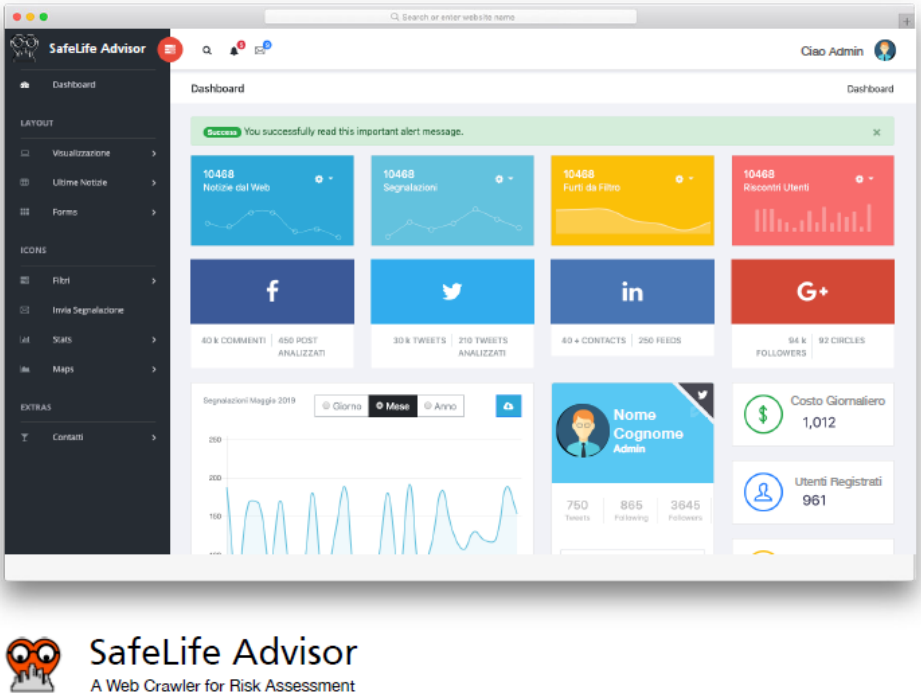

Figura 2.3. Dashboard Amministratore della piattaforma web di SafeLife

- *Contestualizzazione della posizione dell'utente*: attraverso l'app sarà possibile entrare in modalità "inside", tramite essa l'utente potrà visualizzare gli eventi accaduti nella zona dove si trova man mano che percorre il proprio itinerario. In tal modo egli può anche decidere di evitare le zone statisticamente più degradate, dove la probabilità di incidenti è molto più elevata rispetto ad altre aree più sicure. Nella Figura 2.4 invece possiamo visualizzare le informazioni degli eventi presenti nelle vicinanze dell'utente.
- *Classificazione delle aree metropolitane in base alla sicurezza*: la graduatoria

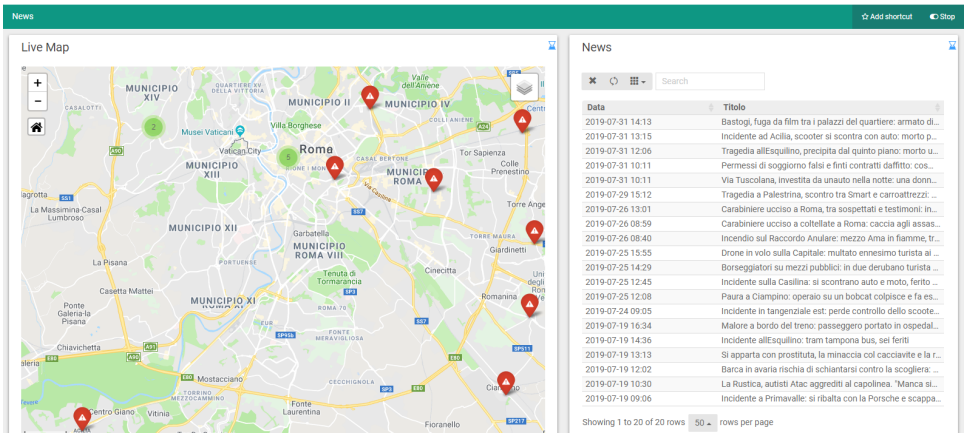

Figura 2.4. Panoramica delle informazioni relative agli eventi nei pressi dell'utente

parte dal database delle statistiche degli eventi, e potrà migliorare man mano che nella zona saranno adottate contromisure di sicurezza (videosorveglianza, maggiori interventi delle forze dell'ordine, arresto dei colpevoli, maggiore partecipazione attiva dei cittadini alla messa in sicurezza del territorio, etc.) o degradare se nell'area aumenteranno gli eventi negativi (aree di spaccio, presidi di malavitosi, etc.).

- *Accesso in Modalità "Active"*: attraverso l'app gli utenti potranno, come accade per un normale gruppo WhatsApp temporaneo, partecipare attivamente alla condivisione di informazioni e azioni nell'area in cui sono posizionati (attivando il GPS). Si creano, in questo modo, delle comunità attive di persone che aiutano le forze dell'ordine, o i normali cittadini, in caso di necessità. Si tratta di attività ben conosciute dalle forze dell'ordine, in particolare dal CASD (Centro alti Studi della Difesa), dove, da anni, sono organizzati corsi di formazione e seminari per rafforzare la "collaborazione civile-militare", che sta diventando una fonte sempre più importante per il contrasto alla criminalità e alla gestione di eventi critici e disastrosi. Esempi di eventi che possono essere gestiti in questo modo sono i seguenti:
  - *Smarrimento di una persona o un bambino*: attivando la modalità active, chi è interessato ad aiutare nella gestione dell'evento potrà dare il proprio contributo a tutti gli altri che sono all'interno del gruppo, compresi i genitori del disperso e le forze dell'ordine.
  - *Furto o violenza*: attivando la modalità active sarà più semplice far circolare informazioni sull'evento, raccogliere testimonianze nonchè, favorire l'intervento delle forze dell'ordine. In questo modo anche la sensazione di "rete di collaborazione ai fini della sicurezza" può creare un senso fiducia e di unione concreta contro la malavita e per una migliore qualità della vita.
  - *Eventi di crisis*: in caso di terremoti, alluvioni, atti terroristici, o altri disastri, la rete di collaborazione è fondamentale per raccogliere informazioni importanti e per agevolare la collaborazione e la cooperazione di tutti gli stakeholder interessati (forze dell'ordine, protezione civile, sindaci, prefetti,

assessori, cittadini, enti pubblici e privati, etc.).

- *Creazione automatica gruppi di quartiere*: il sistema creerà automaticamente un gruppo di prossimità per l'identificazione del personale disponibile ad entrare nella rete SAFE e, in qualche modo, intervenire.
- *Modalità "Help Me"*: sarà possibile interconnettere il sistema con le forze dell'ordine o con gli enti privati autorizzati per permettere gli interventi attraverso i pulsanti SOS e geolocalizzazione.
- *Inserimento informazioni fornite dagli utenti*: tutti potranno inserire nel sistema informazioni e segnalazioni, con foto, mappe e criticità, aiutando, in questo modo, a creare una mappa completa della situazione nelle aree anche meno abitate o sconosciute. Nella Figura 2.5 è possibile vedere com'è strutturato il form per l'inserimento di una nuova segnalazione.

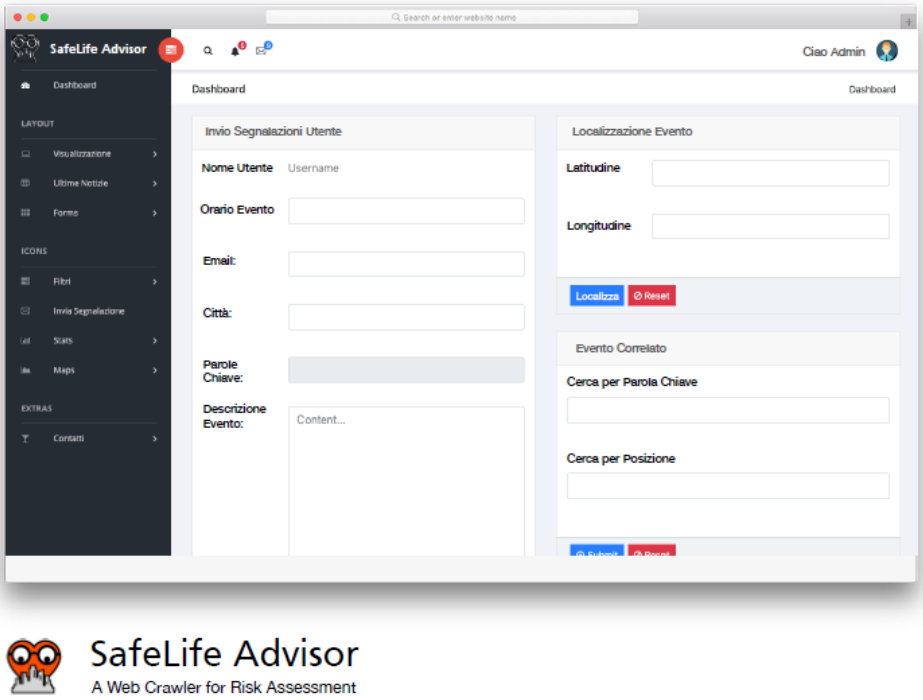

Figura 2.5. Form per l'inserimento di un nuovo evento

- *Evoluzione casistiche sul territorio*: il sistema permetterà di effettuare analisi predittive sul territorio proponendo possibili futuri target di azioni criminali, consentendo di agevolare le azioni di contrasto da parte delle forze dell'ordine e dei normali cittadini.

2.2.1 L'app di SafeLife

Veniamo, infine, a spiegare l'ultimo aspetto del progetto, ovvero la companion app di SafeLife per Android.

L'obiettivo dell'app SafeLife è, principalmente, quello di permettere ai vari utenti di segnalare, in maniera rapida e semplice, un evento pericoloso di cui sono stati testimoni durante lo svolgimento delle normali attività quotidiane. Ciò permette anche ad altri utenti interessati alla medesima zona dell'accaduto di essere avvisati in merito all'evento, semplicemente guardando la mappa della zona. Inoltre, gli utenti, durante l'inserimento di un nuovo evento, possono specificare delle informazioni opzionali, oltre a quelle obbligatorie, necessarie a catalogare e collocare geograficamente l'evento stesso. L'utente può, infatti, caricare una foto del luogo dell'evento, oppure una breve descrizione dello stesso, in modo da fornire ulteriori dettagli sull'accaduto per permettere una migliore distinzione tra i vari eventi presenti nella stessa zona. Tali aggiunte sono importanti soprattutto durante l'inserimento di un nuovo evento, che potrebbe risultare simile ad un altro già presente nel database; in questo caso, le informazioni extra fornite dagli utenti consentono una più facile distinzione tra gli eventi, in modo da aiutare l'utente, in fase di inserimento, a decidere se l'evento segnalato sia lo stesso di quello già presente nel database.

Le informazioni che gli utenti inseriscono, inoltre, possono essere visualizzate da tutti al fine di garantire la sicurezza urbana, in modo tale che un utente possa effettuare delle ricerche più approfondite degli eventi che lo circondano, grazie anche all'utilizzo di filtri che rendono la mappa dinamica rispetto alle impostazioni scelte.

Un altro aspetto fondamentale dell'app è la possibilità di scegliere la visualizzazione degli eventi sotto forma di heatmap, nella quale gli eventi vengono rappresentati come delle macchie colorate di diversa intensità e diverso colore, in base al peso del singolo evento e di quelli adiacenti. Diventa, quindi, indispensabile che di ogni singolo evento si abbia a disposizione la categoria a cui esso appartiene, per determinare il suo peso. Allo stesso modo, anche la data dell'evento è fondamentale; infatti, in base al tempo trascorso dalla segnalazione, l'intensità del colore dell'evento cambierà, diventando man mano sempre meno forte fino quasi a sparire quando oramai l'evento è diventato datato, a confronto delle altre segnalazioni presenti nella zona di interesse.

Un'altra caratteristica prevista per l'app è quella di utilizzare il database, che costituisce il cuore del progetto, per realizzare una funzionalità di navigazione, che guiderà l'utente verso una destinazione attraverso una serie di cammini. Un'opzione è, ad esempio, il cammino più sicuro; ma tale opzione non è l'unica possibile, infatti l'utente, scegliendo un punto di partenza e uno di destinazione insieme ad un mezzo di trasporto (a piedi, macchina, etc.), potrà visulizzare uno o più percorsi suggeriti, come il percorso più breve, il più sicuro, etc, spetterà poi a lui decidere quali di questi seguire.

Per di più, una volta pianificato il percorso, si potrebbero visualizzare delle statistiche riguardanti gli eventi che l'utente intercetterà durante esso, così da avvisarlo e metterlo in guardia da eventi pericolosi che nel frattempo dovessero comparire.

In ogni caso, l'app di SafeLife ha come scopo principale quello di permettere agli utenti del sistema complessivo di effettuare delle segnalazioni rapide e semplici, incentivandoli ad effettuare segnalazioni, evitando, così, passaggi complessi che si avrebbero utilizzando la piattaforma web da smartphone.

Perciò un'app studiata per smartphone garantisce una visualizzazione delle informazioni molto più fluida su smartphone, rispetto ad un'app web sullo stesso dispositivo, la cui navigazione attraverso browser sarebbe molto più problematica.

Quindi l'app si rende necessaria proprio in qui casi in cui l'utente assiste ad un evento pericoloso e decide di effettuare una segnalazione immediata; in questo caso, lo smartphone sarà il principale strumento di segnalazione a disposizione dell'utente.

Per cui, se la controparte web di SafeLife permette un maggior numero di visualizzazioni, di statistiche ed analisi degli eventi nel database, l'app rappresenta il principale strumento tramite cui un utente interagirà con la piattaforma. L'utente, perciò, sarà invogliato nell'utilizzo dell'app quando consulterà SafeLife per le strade della propria città, sia in caso di visualizzazione degli eventi pericolosi, sia quando deciderà di effettuare una segnalazione in tempo reale dell'accaduto. Per cui si progetterà l'app per avere un'interfaccia quanto più snella e semplice possibile, mantenendo, però, le funzioni basilari della piattaforma sempre a disposizione dell'utente.

Questa separazione di funzionalità è già stata mostrata nella Sezione 2.1.2; in particolare, nella Figura 2.1, risulta chiaro come l'app rappresenti una delle principali fonti di alimentazione del database degli eventi.

3

Analisi dei requisiti

In questo capitolo parleremo del processo di analisi della documentazione finalizzato all'elicitazione dei requisiti funzionali e non funzionali. Verranno inoltre approfondite le funzionalità da essi derivate e verranno illustrati i dettagli implementativi.

3.1 Raccolta dei requisiti

La raccolta dei requisiti necessari all'implementazione delle funzionalità dell'app SafeLife si è basata principalmente su una serie di incontri virtuali tra le persone coinvolte nel progetto e gli stakeholder della Res On Network.

In particolare, parliamo di almeno due incontri a cui hanno partecipato dal Campus Bio-Medico di Roma il prof. Luca Faramondi, la studentessa Martina Nobili, mentre dalla Politecnica delle Marche il prof. Domenico Ursino e il prof. Gianluca Bonifazi, e dalla Fondazione il direttore scientifico Marco Santarelli.

I punti fondamentali del progetto sono stati definiti principalmente dal prof. Faramondi, in quanto è attualmente a capo della supervisione della parte web del progetto ed è anche il principale ideatore del progetto e delle sue funzionalità. Tutta una serie di requisiti extra è stata, poi, aggiunta dagli stakeholder, il cui portavoce era il direttore scientifico Santarelli, il quale si è espresso soprattutto su una serie di feature di tipo supplementare rispetto ai requisiti fondamentali del progetto che sono, invece, a carattere maggiormente tecnico.

I vari requisiti concordati sono stati scelti anche tenendo a mente i vincoli temporali dovuti alla durata limitata del tirocinio, per cui alcune delle funzionalità che sono state ritenute non fondamentali, almeno all'inizio dello sviluppo dell'app, sono state lasciate come punti di sviluppo futuro del progetto.

Un'altra sorgente per la raccolta dei requisiti si è basata su una serie di documenti e presentazioni precedentemente realizzati dal prof. Faramondi con lo scopo di introdurre il progetto ai non addetti ai lavori, esponendo tutte le caratteristiche principali che esso mirano ad implementare, sia tramite app che mediante piattaforma web.

3.1.1 Casi d'uso

Inizialmente è stata effettuata un'analisi ad alto livello della documentazione disponibile, al fine di individuare gli attori coinvolti nei processi e delineare, per sommi capi, le operazioni da essi svolte. Da ciò è stato elaborato il diagramma dei casi d'uso mostrato in Figura 3.1.

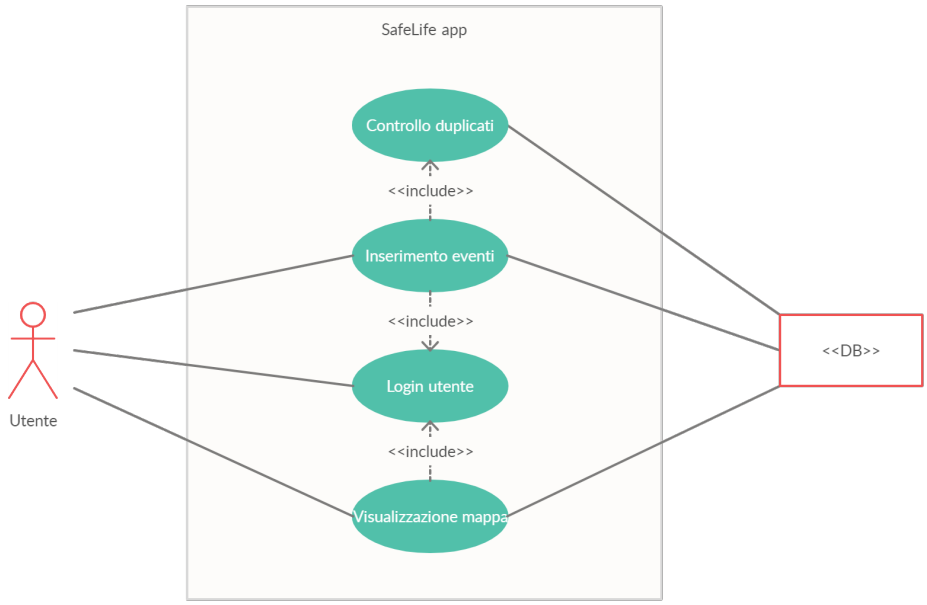

Figura 3.1. Modello dei casi d'uso dello scenario di inserimento e visualizzazione degli eventi nel database

Le uniche azioni eseguite direttamente dall'utente sono il login, la visualizzazione della mappa degli eventi e l'inserimento delle segnalazioni come eventi utente. L'inserimento di nuovi eventi, così come la visualizzazione degli eventi sulla mappa, sono gestite dall'app che salverà le informazioni riguardanti gli eventi su un database.

Il database remoto è fondamentale per rendere permanenti gli eventi inseriti dagli utenti, in modo tale da rendere possibile il riempimento della mappa con tutte le segnalazioni degli utenti nel corso del tempo.

Infine, l'app, sempre tramite il supporto del database, deve gestire il caso in cui siano presenti degli eventi duplicati, che cioè si riferiscono allo stesso evento; per questo è fondamentale implementare un algoritmo automatico in grado di riconoscere e gestire tali evenienze.

3.2 Requisiti funzionali

Dalla successiva analisi, più dettagliata, sono stati ricavati i requisiti funzionali enumerati nella Tabella 3.1. Quasi tutti questi requisiti sono considerati equamente importanti; le uniche eccezioni riguardano le informazioni opzionali degli eventi, che un utente può inserire per permettere una maggiore distinzione da quelli già presenti nel database; essi, per tale ragione, sono considerati meno importanti degli altri.

R1, *R3*, *R4* ed *R5* sono i requisiti di base su cui si sviluppa l'intero progetto; essi riguardano la possibilità dell'utente di inserire i propri eventi nel database, in modo tale da riempire la mappa degli eventi che ogni utente può visualizzare, ma non solo; infatti, il database comprenderà anche gli eventi di articoli di cronaca ottenuti tramite crawler. Tutte queste informazioni sugli eventi dovranno, poi, essere condivise tra l'app Android e la piattaforma web in modo facile, veloce e in tempo reale, in modo tale da rendere SafeLife un'unica piattaforma con database condiviso tra web e app.

R2, *R7* e *R10* sono dei requisiti che non riguardano il funzionamento base dell'app, ma arricchiscono le informazioni della mappa tramite delle visualizzazioni personalizzate dall'utente stesso, oppure, come nel caso della heatmap, permettono un approccio diverso alla visualizzazione degli eventi rispetto a quello classico, tramite icone differenti. Infatti, la heatmap permette il riconoscimento immediato delle zone a maggior densità di eventi pericolosi, sia dovuto all'elevato numero di eventi che alla pericolosità degli stessi.

R6 rendere obbligatorio la registrazione dell'utente dell'app; tale requisito è necessario per soddisfare quelli rimanenti.

R8 e *R9* sono requisiti che riguardano la convalida dei dati inseriti dagli utenti e sono tra loro collegati in fase implementativa; inoltre, come anticipato, la registrazione dell'utente serve proprio a realizzare questi controlli che, altrimenti, risulterebbero facilmente aggirabili.

3.2.1 Funzionalità

Il passo successivo è stato definire le funzionalità da implementare all'interno del programma per far sì che tutti i requisiti funzionali fossero rispettati. Tali funzionalità sono elencate nella Tabella 3.2 e approfondite di seguito.

F1, *F3*, *F4* e *F5* servono a soddisfare i requisiti *R1*, *R3*, *R4*, *R5*. A livello implementativo queste funzionalità rappresentano lo scheletro dell'app SafeLife, permettendo, quindi, all'utente di visualizzare, anche se in maniera basilare, tutti gli eventi (utente o articoli) che sono contenuti nel database.

Quindi, la mappa, già a questo livello, permetterebbe all'utente di cercare la presenza di eventi pericolosi nella sua zona (grazie anche al database remoto che permette di mantenere sia i dati provenienti dal crawler, sia quelli provenienti dagli utenti). Però, la sola visualizzazione degli eventi è solo uno degli aspetti fondamentali dell'app, l'altro è la possibilità di effettuare delle segnalazioni di eventi pericolosi in maniera rapida e semplice; ciò è reso possibile grazie alla schermata apposita che contiene un form guidato in cui l'utente, una volta compilato il form stesso, può completare l'inserimento dell'evento nel database.

CODICE	DESCRIZIONE
R1	La soluzione deve prevedere la possibilità di visualizzare tutti gli eventi pericolosi su una mappa geografica.
R2	La soluzione deve prevedere la possibilità di visualizzare tutti gli eventi pericolosi tramite una modalità heatmap, in modo da rendere facilmente riconoscibili le zone a maggior rischio di eventi pericolosi.
R3	La soluzione deve prevedere la possibilità che gli utenti possano inserire i propri eventi, composti da: 1) *Categoria:* questo campo serve ad identificare il tipo dell'evento, in base alla categoria del crimine a cui esso si riferisce. 2) *Data e ora:* questo campo, come suggerito dal nome, identifica la data e l'ora, indicate dall'utente, in cui l'evento è avvenuto. 3) *Luogo:* questo campo identifica il luogo in cui l'evento è avvenuto. 4) *Descrizione:* questo è un campo opzionale, non fondamentale, quindi, ai fini della distinzione dell'evento dagli altri presenti nel database; di certo è, comunque, un'informazione preziosa per gli utenti durante la visualizzazione delle segnalazioni salvate nel database. 5) *Immagine dell'evento:* un'altra informazione opzionale dell'evento; anche questa, come la precedente, ha un ruolo importante specialmente per gli utenti che visualizzano le informazioni degli eventi in zona, rendendo l'evento in questione facilmente distinguibile dagli altri.
R4	La soluzione deve prevedere la possibilità di recuperare le informazioni sugli eventi ottenuti dagli articoli di cronaca online, estratti dal web tramite crawler.
R5	La soluzione deve permettere di realizzare un database condiviso tra l'applicazione per smartphone e la piattaforma web, in modo da permettere il recupero, in tempo reale, delle informazioni degli eventi articoli e utente.
R6	La soluzione deve avere un sistema di login per la registrazione degli utenti dell'app.
R7	La soluzione deve prevedere la possibilità di applicare dei filtri durante la visualizzazione degli eventi sulla mappa, in modo da permettere all'utente di vedere solo gli eventi di suo interesse.
R8	La soluzione deve essere in grado di risolvere la gestione di eventi duplicati (stessa categoria, stessa data e l'ora che sia al massimo 30 minuti prima o 30 minuti dopo l'ora del nuovo evento) in maniera automatica.
R9	La soluzione deve essere in grado di implementare un sistema che permetta di distinguere le segnalazioni utente legittime da quelle fasulle.
R10	La soluzione deve essere in grado di rendere riconoscibile alla vista la differenza che esiste tra i vari eventi che compongono la mappa.

Tabella 3.1. Elenco dei requisiti funzionali

F2 e *F10* mappano direttamente i requisiti *R2* e *R10*, per cui dalla schermata della mappa degli eventi, sarà possibile premere un pulsante per passare dalla visualizzazione tramite marker colorati, (per distinguerli visivamente tra loro in base ad un parametro stabilito) ad una visualizzazione della mappa tramite heatmap. Nella modalità heatmap gli eventi si trasformano in macchie colorate di diversi colori e tonalità in base alla pericolosità e alla vecchiaia della segnalazione.

F6 è legata al requisito *R6*: ogni utente che avvia l'app deve prima autenticarsi

CODICE	DESCRIZIONE
F1	Utilizzo dell'SDK di Google Maps per la realizzazione della mappa geografica.
F2	Realizzazione di un pulsante che permetta di passare dalla modalità classica di visualizzazione degli eventi, rappresentati come marker, a quella tramite heatmap.
F3	Realizzazione di un form guidato per l'inserimento di un nuovo evento, in grado di controllare che tutte le informazioni obbligatorie siano state inserite.
F4	Rendere visibili sulla mappa geografica la posizione e le informazioni degli articoli estratti dal crawler.
F5	Realizzazione di un database remoto, alimentato sia dalle segnalazioni dell'utente, che dalle informazioni recuperate dal crawler; inoltre il database deve permettere di recuperare facilmente le informazioni inserite sia dall'app che dalla piattaforma web.
F6	Realizzazione di una pagina di login che permetta all'utente di registrarsi ed autenticarsi.
F7	Creazione di un pulsante che permetta all'utente di scegliere che combinazione di filtri applicare alla visualizzazione degli eventi sulla mappa.
F8	Realizzazione di una soluzione automatica basata selle informazioni degli eventi, per riconoscere gli eventi simili tra loro (stessa categoria, stessa data e l'ora che sia al massimo 30 minuti prima o 30 minuti dopo l'ora del nuovo evento) in modo da evitare che dei duplicati vengano salvati nel database.
F9	Realizzazione di un metodo per verificare la validità delle informazioni inserite dagli utenti.
F10	Realizzazione delle icone per i marker degli eventi, in modo da renderli visibilmente diversi tra loro in base ad un parametro stabilito.

Tabella 3.2. Elenco delle funzionalità

per poter svolgere qualsiasi azione; ciò è reso possibile tramite una serie di schermate che hanno lo scopo di permettere all'utente di effettuare il login, utilizzando diversi metodi di autenticazione. Il requisito dell'autenticazione degli utenti è, inoltre, legato a quello del requisito *R9* e della corrispettiva funzionalità *F9*. Infatti è di fondamentale importanza che ogni utente sia autenticato per permettere una corretta valutazione delle segnalazioni inserite dagli utenti.

F8 è legata al requisito *R8*; infatti è necessario sfruttare le informazioni a disposizione con ogni evento per trovare una lista di eventi duplicati già presenti nel database durante la fase di inserimento di una segnalazione.

F7 è legata al requisito *R7*; per migliorare l'esperienza dell'utente durante la visualizzazione della mappa, è necessario garantire all'utente un certo grado di libertà nella scelta degli eventi a cui è interessato; per questo è stata prevista la presenza di un pulsante che permette di selezionare una serie di filtri su di essi.

La matrice in Tabella 3.3 mette in relazione i requisiti e le funzionalità; la "X" nella posizione (i, j) indica che la funzionalità F_j è necessaria al soddisfacimento del requisito R_i.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
R1	X		X	X	X					
R2		X								
R3	X		X	X	X					
R4	X		X	X	X					
R5	X		X	X	X					
R6						X			X	
R7							X			
R8								X	X	
R9								X	X	
R10										X

Tabella 3.3. Matrice di corrispondenza requisiti-funzionalità

3.2.2 Dettagli implementativi

Le funzionalità appena analizzate sono state affinate aggiungendovi ulteriori dettagli implementativi, volti a semplificare la fruizione da parte dell'utente e a migliorare l'integrazione con l'interfaccia esistente. Di seguito viene presentato un elenco di tali dettagli.

- *L'utente deve avere la possibilità di scegliere una città come punto di partenza da cui visualizzare la mappa degli eventi.* Per semplificare la navigazione dell'utente all'interno della mappa degli eventi che ricopre l'intera Italia, è necessario stabilire un punto di partenza nella visualizzazione della stessa, per questo viene chiesto all'utente di scegliere una città di principale. Ogni volta che l'utente passa alla schermata della mappa, la camera di Google Maps viene posizionata sulla città principale scelta, mostrando, quindi, tutti gli eventi nelle immediate vicinanze.
- *Rendere agile gli spostamenti di camera nella mappa degli eventi.* Per migliorare la velocità di spostamento della camera nella mappa degli eventi, si è deciso di implementare un campo di testo che permette di spostare immediatamente la camera nella città indicata.
- *Rendere i valori dei filtri applicati visibili in qualunque momento e facilmente cancellabili.* Per rendere migliore l'esperienza dell'utente, durante la scelta dei filtri da applicare, è necessario implementare un sistema che mantenga visibili le informazioni dei filtri scelti e diventa necessaria anche l'aggiunta di un pulsante apposito per la cancellazione degli stessi.
- *Aggiungere dei pulsanti per la geolocalizzazione del dispositivo.* Anche questo aspetto mira a semplificare l'utilizzo da parte dell'utente; infatti, sono stati ideati dei pulsanti che, tramite l'utilizzo del GPS (dopo aver ottenuto il consenso dell'utente), permettono la compilazione automatica dei campi che necessitano di una città o indirizzo per essere completati, come, ad esempio, la città princi-

pale o la città su cui si vuole spostare la camera; il campo verrà riempito con l'indirizzo o la città in cui il dispositivo si trova attualmente.
- *Gestire la presenza di più eventi nello stesso punto.* È necessario implementare una soluzione che permetta di gestire la sovrapposizione di più eventi sullo stesso punto della mappa.

3.3 Requisiti non funzionali

Per la definizione dei requisiti non funzionali si è cercato di sfruttare il più possibile delle soluzioni commerciali già ampiamente affermate per la realizzazione delle funzionalità di backend al fine di velocizzare lo sviluppo dell'app. In base a ciò sono stati individuati i seguenti requisiti non funzionali:

- *La soluzione deve utilizzare dei servizi commerciali per la gestione del backend.* Tale requisito interessa esclusivamente la componente gestionale dell'app, che prevede l'utilizzo di una soluzione esistente per la realizzazione e il mantenimento di un database remoto, insieme alla gestione dei profili di autenticazione degli utenti.
- *La soluzione deve utilizzare dei servizi commerciali per l'implementazione della mappa geografica, così come per il geocoding delle coordinate geografiche dei luoghi inseriti dagli utenti.* Tale requisito consiste nell'utilizzo delle API di alcuni servizi commerciali per poter implementare l'interfaccia della mappa geografica della nostra app, nonchè, per permettere di passare alle coordinate geografiche di un luogo a partire da un indirizzo, e viceversa.

4

Progettazione

In questo capitolo verrà trattata la fase di progettazione delle funzionalità individuate durante l'analisi dei requisiti. Verranno dapprima discussi i diagrammi UML realizzati per la progettazione dell'app; successivamente, ci soffermeremo sul mockup realizzato, il quale è stato utilizzato come linea guida durante lo sviluppo, in particolare, durante la realizzazione dell'interfaccia utente.

4.1 Progettazione dei diagrammi UML

Incominciamo questo capitolo dando una breve e semplice descrizione sul UML e i suoi diagrammi che andremo ad osservare nelle successive sezioni. UML, Unified Modeling Language, è un linguaggio di modellazione semi formale e grafico (basato su diagrammi) per specificare, visualizzare, costruire e documentare gli artefatti di un sistema software. Si tratta di un linguaggio di modellazione usato per capire e descrivere le caratteristiche di un nuovo sistema o un sistema esistente. Gli stessi modelli UML sono, quindi, artefatti usati per sviluppare il sistema e comunicare con il cliente (ma anche con progettisti/sviluppatori/etc.).

Un modello UML è costituito da una collezione organizzata di diagrammi correlati, costruiti componendo elementi grafici con significato formalmente definito, elementi testuali formali, ed elementi di testo libero.

Vediamo, ora, una breve descrizione dei diagrammi che sono stati realizzati in questa fase di progettazione dell'app.

Il primo diagramma realizzato è il Diagramma delle Classi; esso descrive il tipo delle classi che compongono il sistema e le relazioni statiche esistenti tra esse. Una classe descrive un insieme di oggetti che condividono gli stessi attributi, operazioni, metodi, relazioni e semantica; in questo diagramma, una classe viene rappresentata da un rettangolo suddiviso in 3 compartimenti: nome, attributi e metodi. Invece, le frecce che collegano le varie classi sono chiamate associazioni; esse rappresentano una connessione tra due o più classi, ognuna delle quali ha la responsabilità di notificare una certa informazione all'altra. Le associazioni sono bidirezionali e la loro molteplicità indica quanti oggetti di una classe possono far riferimento ad ogni oggetto dell'altra.

Passiamo, adesso, a dare una breve spiegazione di cos'è un Diagramma delle Attività. Esso descrive il flusso di elaborazione interno di una istanza, una classe, un caso d'uso o una operazione. È un grafo composto da attività (i nodi) e transizioni (gli archi) e presenta un punto di partenza ed uno di arrivo per ogni flusso di elaborazione.

L'ultimo diagramma utilizzato è il Diagramma dei Componenti. Esso ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi componenti principali e delle relazioni fra di essi. Per componente si intende un'unità software dotata di una precisa identità, nonché responsabilità e interfacce ben definite.

4.1.1 Diagramma delle Classi

In questa sezione approfondiremo meglio il Diagramma delle Classi del codice, ottenuto grazie all'utilizzo dello strumento dedicato alla creazione di diagrammi UML, presente nell'IDE (Android Studio) utilizzato per lo sviluppo dell'app.

Prima di tutto, nella Figura 4.1, è possibile osservare il diagramma completo, comprensivo di tutte le classi presenti nel progetto, con le varie frecce che definiscono il tipo della relazione in base al colore e al tratteggio. Passiamo, adesso, ad illustrare sinteticamente il significato delle varie frecce presenti nel diagramma:

- Le frecce bianche e tratteggiate stanno ad indicare una relazione di dipendenza, ciò indica che la definizione di una delle due fa riferimento alla definizione dell'altra.
- Le frecce bianche senza tratteggio, con un rombo bianco da un lato e la punta della freccia dall'altro, rappresentano un'aggregazione, ovvero un tipo di associazione "parte-di". Un'aggregazione si verifica quando esiste una classe che è una collezione o contenitore di altre classi. Il rombo rappresenta la classe contenitore, mentre la punta della freccia denota la classe contenuta.
- Le frecce verdi e tratteggiate rappresentano, invece, una relazione di implementazione di un'interfaccia all'interno di una classe. L'estremità con la punta denota l'interfaccia, mentre all'altra estremità troviamo la classe che la implementa.
- Per ultimo abbiamo le linee rosse; queste stanno ad indicare la presenza di classi interne. L'estremità con il pallino rosso indica la classe esterna, mentre all'altro lato troviamo la classe interna.

Passiamo, adesso, a spiegare brevemente anche il significato dei simboli contenuti nelle classi:

- I pallini gialli con all'interno la lettera "f", presenti nelle classi, stanno ad indicare gli attributi della classe.
- I pallini rossi con all'interno la lettera "m", presenti nelle classi, stanno ad indicare i metodi della classe.
- I lucchetti arancioni a fianco agli attributi e dei metodi rappresentano la visibilità dei suddetti, in questo caso impostata su privata.
- I lucchetti verdi, come i precedenti, rappresentano la visibilità dei metodi e attributi, in questo caso impostata su pubblica.

Figura 4.1. Diagramma delle Classi integrale

Ora che abbiamo terminato di descrivere i simboli presenti nel diagramma, passiamo ad analizzarne in maniera più specifica alcune viste ricavate da esso. Incominciamo con la Figura 4.2; da questa vista è possibile osservare più nel dettaglio tre classi che si occupano, nello specifico, di realizzare una lista di elementi che hanno in comune lo stesso layout, in modo tale da realizzare un elenco composto dalle informazioni di eventi pericolosi.

Nella vista presente nella Figura 4.3 è possibile osservare la classe **`CustomAdapterF`** che si occupa di realizzare una lista di elementi che si può scorrere, come

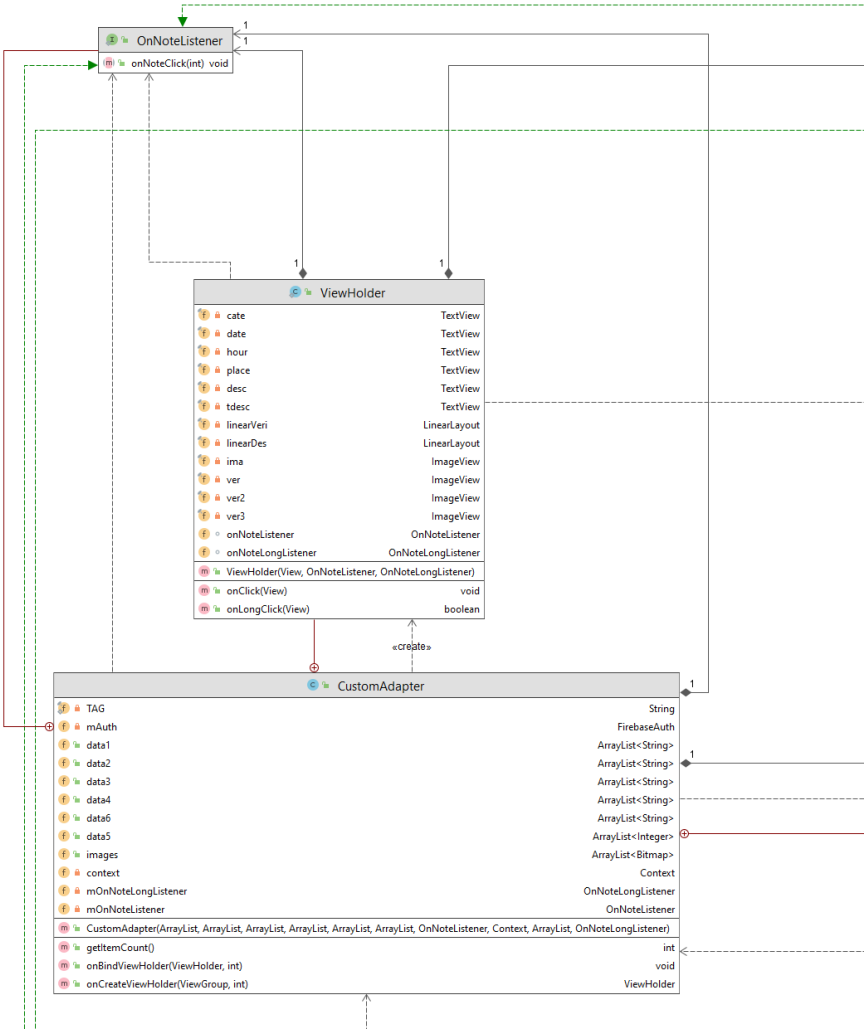

Figura 4.2. Prima vista del Diagramma delle Classi

nella vista precedente; essa permette di realizzare l'elenco delle FAQ dell'app. Altra classe importante è **`DuplicateActivity`**, che, come dice il nome, serve alla realizzazione dell'Activity che gestisce il caso in cui l'evento che l'utente cerca di inserire sia simile ad uno o più eventi già presenti nel database. In particolare, questa Activity mostra all'utente le informazioni relative agli eventi simili rilevati e, gli chiede se uno di questi è uguale alla sua segnalazione.

Nella vista presente nella Figura 4.4 è possibile osservare la classe **`Event`** che ha lo scopo di realizzare una serie di oggetti evento. Questi, poi, vengono utilizzati allo scopo di memorizzare, e successivamente recuperare, tutte le informazioni relative ad un evento pericoloso (sia inserito da un utente che ricavato da un articolo).

Nella vista presente nella Figura 4.5 è possibile osservare le classi **`InsEveU-`**

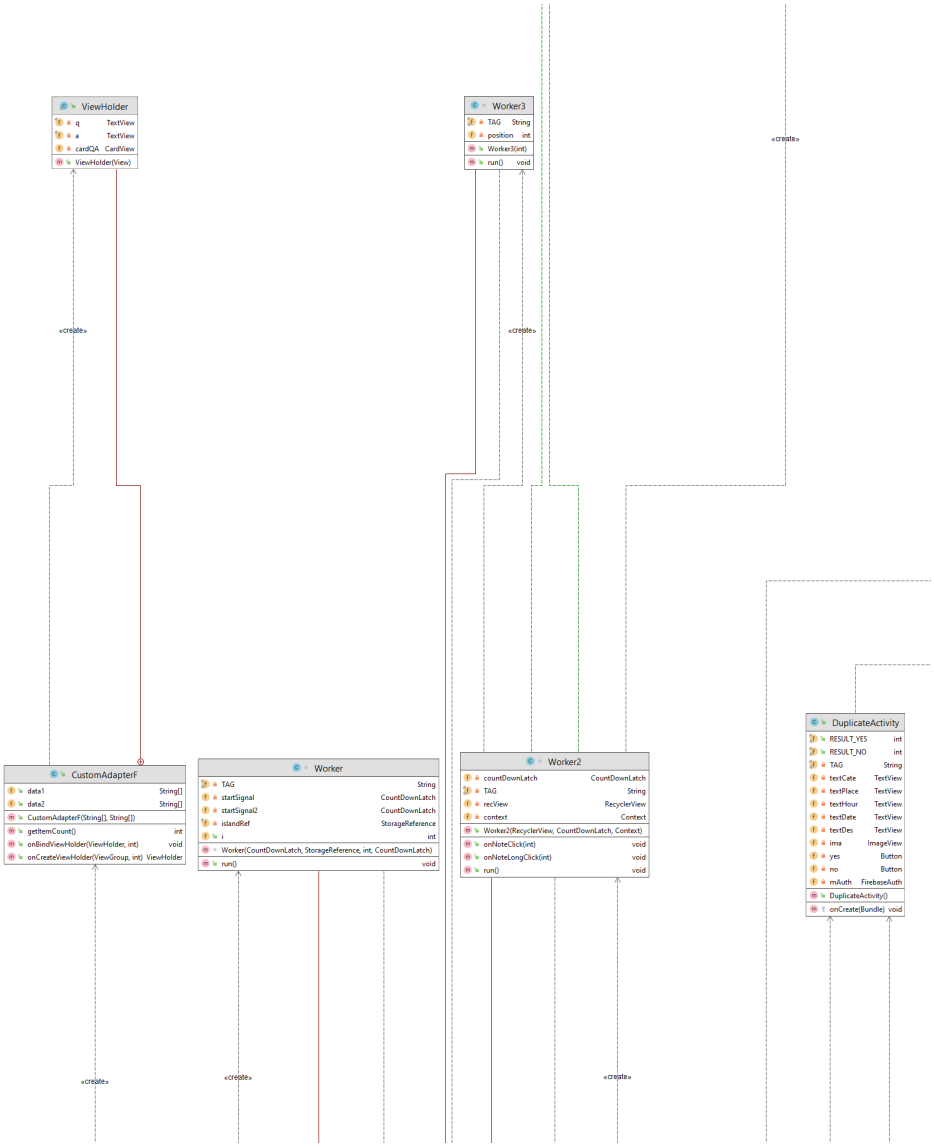

Figura 4.3. Seconda vista del Diagramma delle Classi

`List` e **`InsEveUList`**, che hanno entrambe lo scopo di memorizzare una serie di HashMap. Esse contengono le informazioni riguardanti gli eventi, con la differenza che la prima classe riguarda quelli ottenuti dalle segnalazioni degli utenti mentre la seconda riguarda gli eventi ottenuti dagli articoli. Inoltre, entrambe le classi hanno dei metodi per il recupero delle informazioni, così come un metodo dedicato alla pulizia degli attributi.

Nella vista presente nella Figura 4.6 è possibile osservare le classi **`LisEve-MultiActivity`** e **`User`**. La prima si occupa di creare l'Activity che permette di

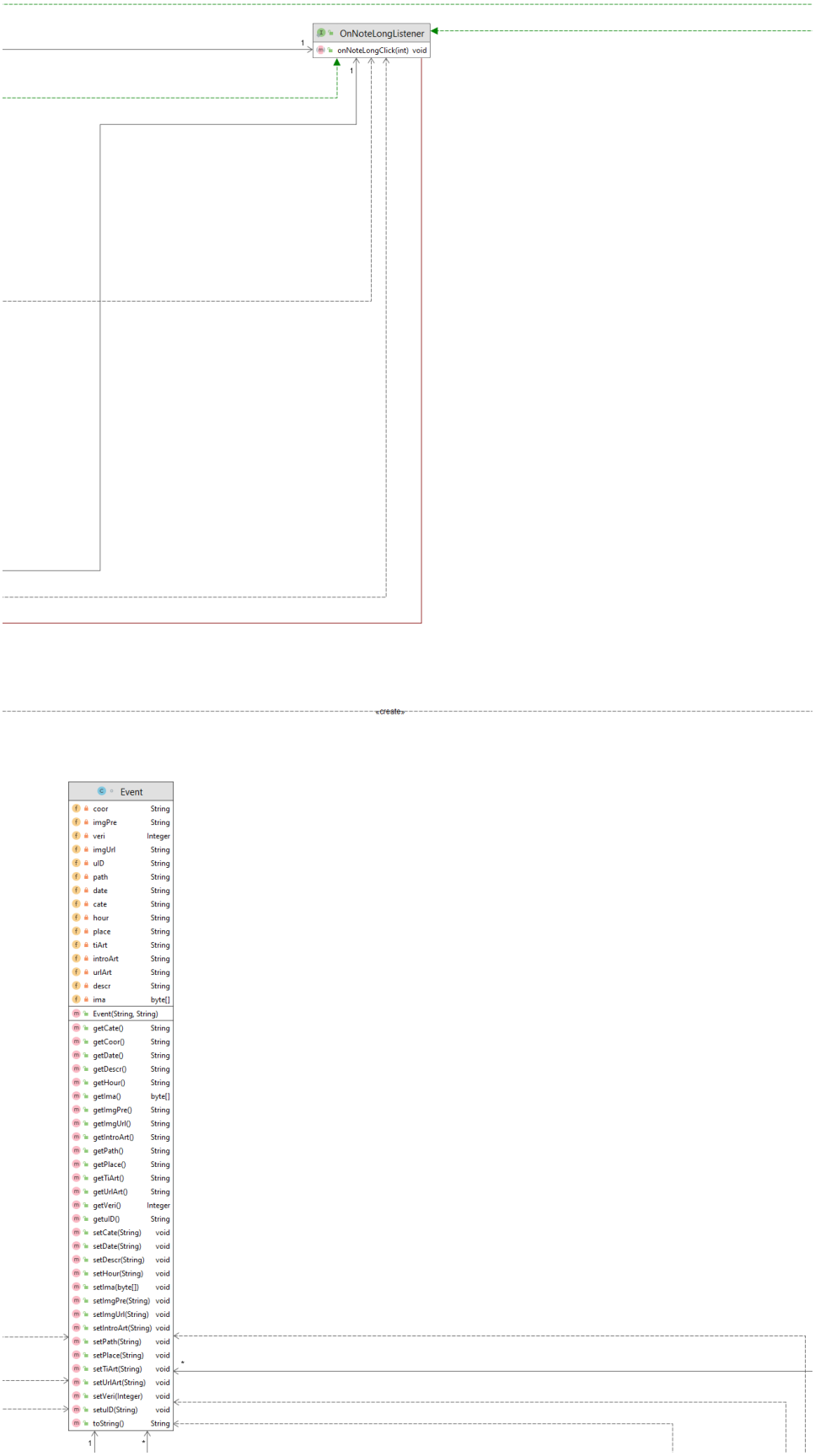

Figura 4.4. Terza vista del Diagramma delle Classi

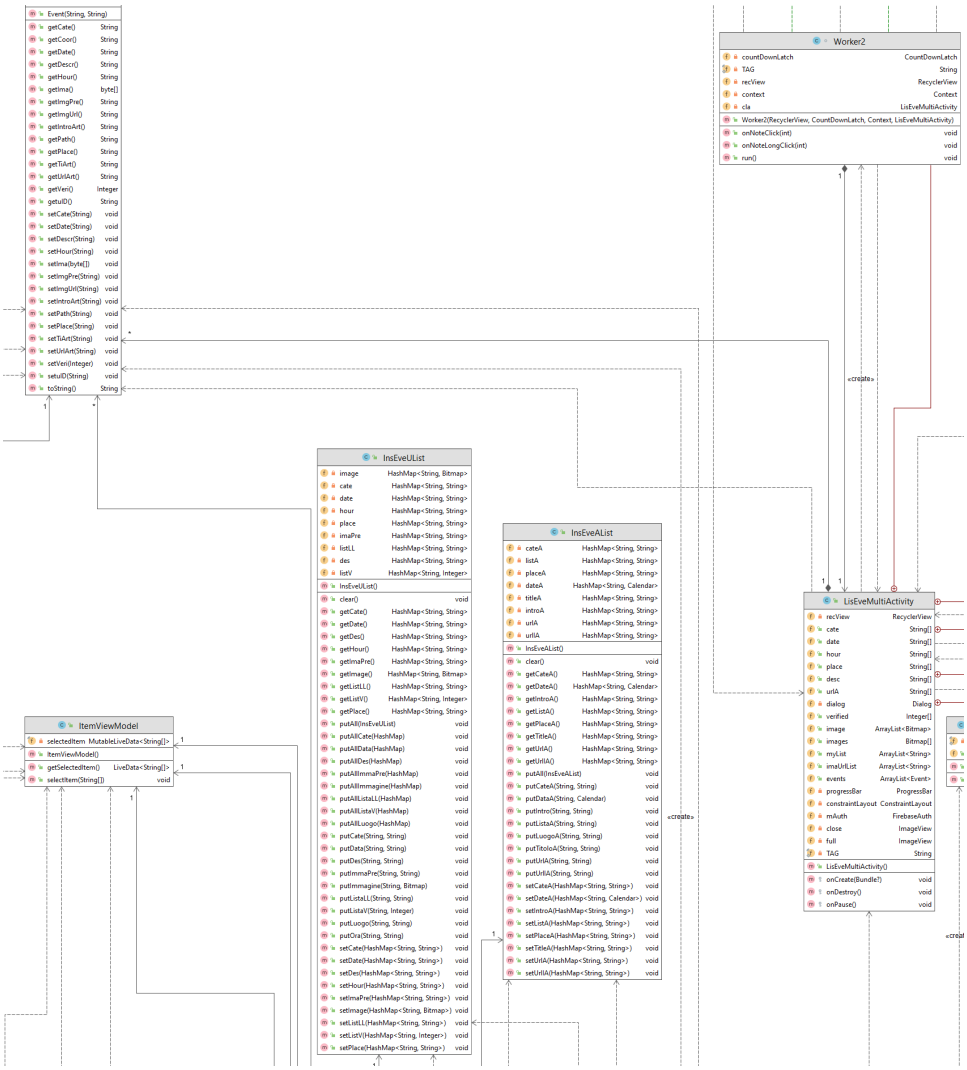

Figura 4.5. Quarta vista del Diagramma delle Classi

visualizzare una lista di elementi contenenti le informazioni degli eventi che sulla mappa appaiono in uno stesso punto. La seconda, invece serve per memorizzare tutte le informazioni che costituiscono il marker dell'evento all'interno della schermata della mappa, quando la visualizzazione selezionata è quella tramite marker.

Nella vista presente nella Figura 4.7 è possibile osservare le classi **`LisEveActivity`**, **`SelCiActivity`**, **`FaqActivity`**, **`OptionsFragment`** e **`EnterActivity`**. Le prime tre si occupano della creazione delle rispettive Activity; in particolare, la prima realizza e gestisce la schermata contenente la lista di eventi che l'utente ha segnalato tramite l'app. La seconda, invece, realizza e gestisce la schermata di selezione della città principale dell'utente, cioè la città che verrà visualizzata ogni volta che la schermata della mappa verrà selezionata. La terza realizza e gestisce la

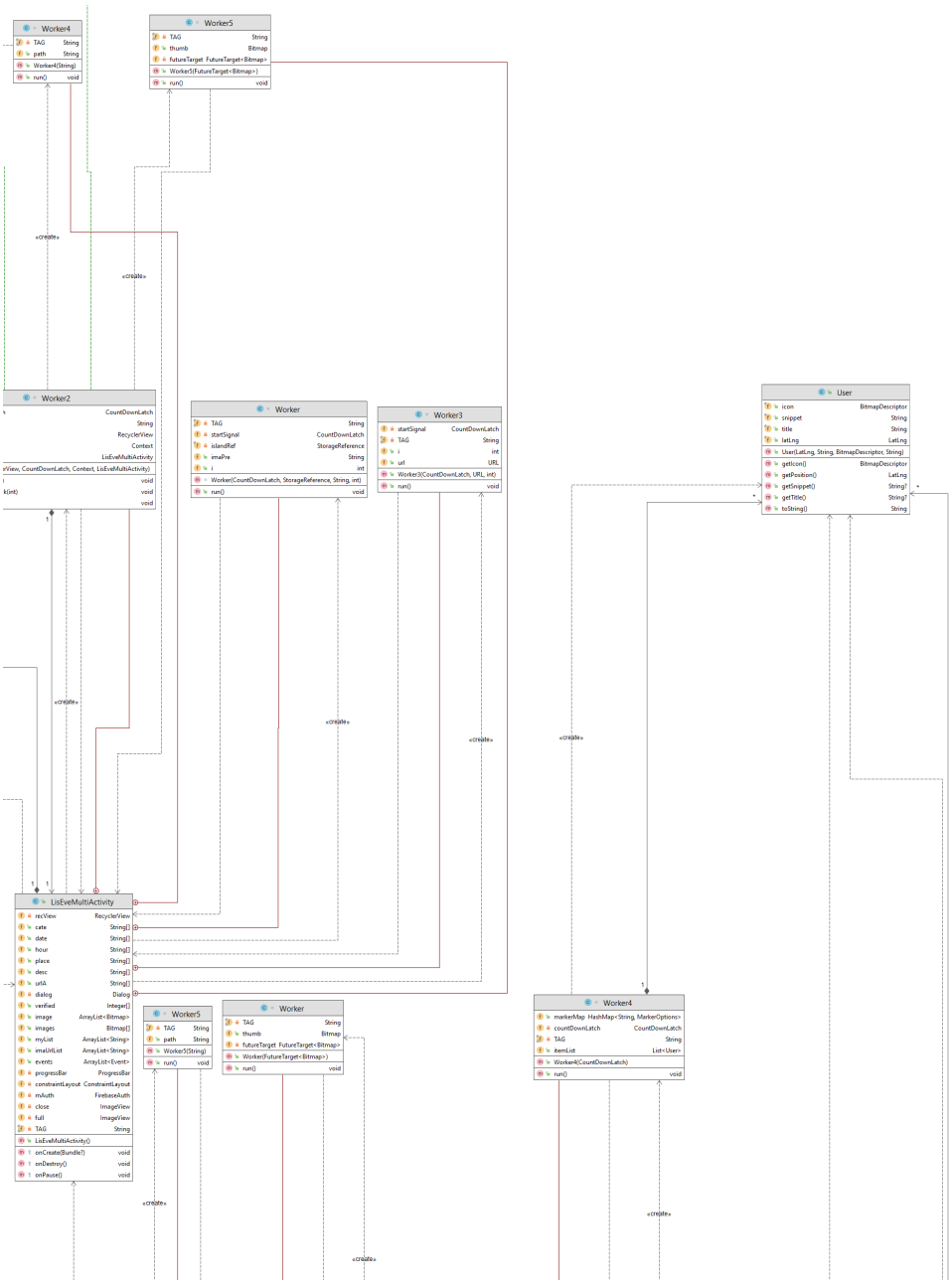

Figura 4.6. Quinta vista del Diagramma delle Classi

schermata delle FAQ dell'app, composta da una serie di domande e rispettive risposte, che hanno lo scopo di chiarire all'utente il funzionamento dell'app. La quarta, **`OptionFragment`**, come dice il nome, si occupa di realizzare e gestire il Fragment che permette all'utente di selezionare le opzioni a sua disposizione. Le opzioni previste sono la possibilità di effettuare il logout, di cancellare il proprio account, di cambiare la città principale, di visualizzare la lista delle proprie segnalazioni oppure di visualizzare le FAQ dell'app. L'ultima classe, invece, permette all'utente di autenticarsi tramite l'utilizzo della e-mail e della password.

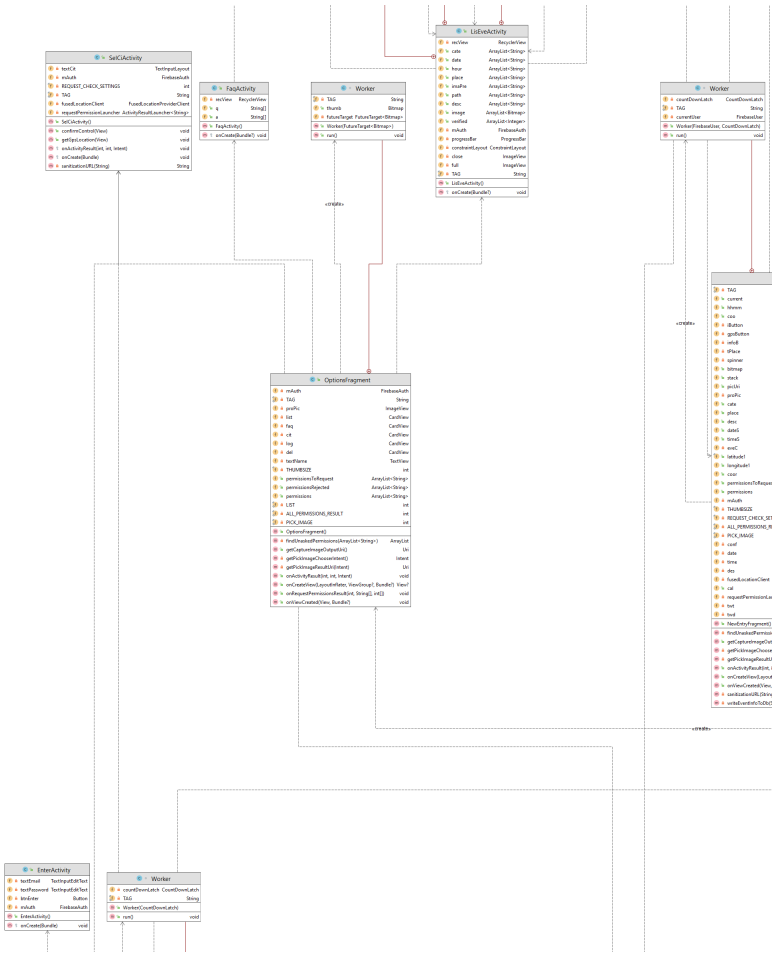

Figura 4.7. Sesta vista del Diagramma delle Classi

Nella vista presente nella Figura 4.8 è possibile osservare le classi **`BottomNavActivity`**, **`NewEntryFragment`**, **`FilDialogFragment`**, **`HomeFragment`** e **`NothingSelectedSpinnerAdapter`**.

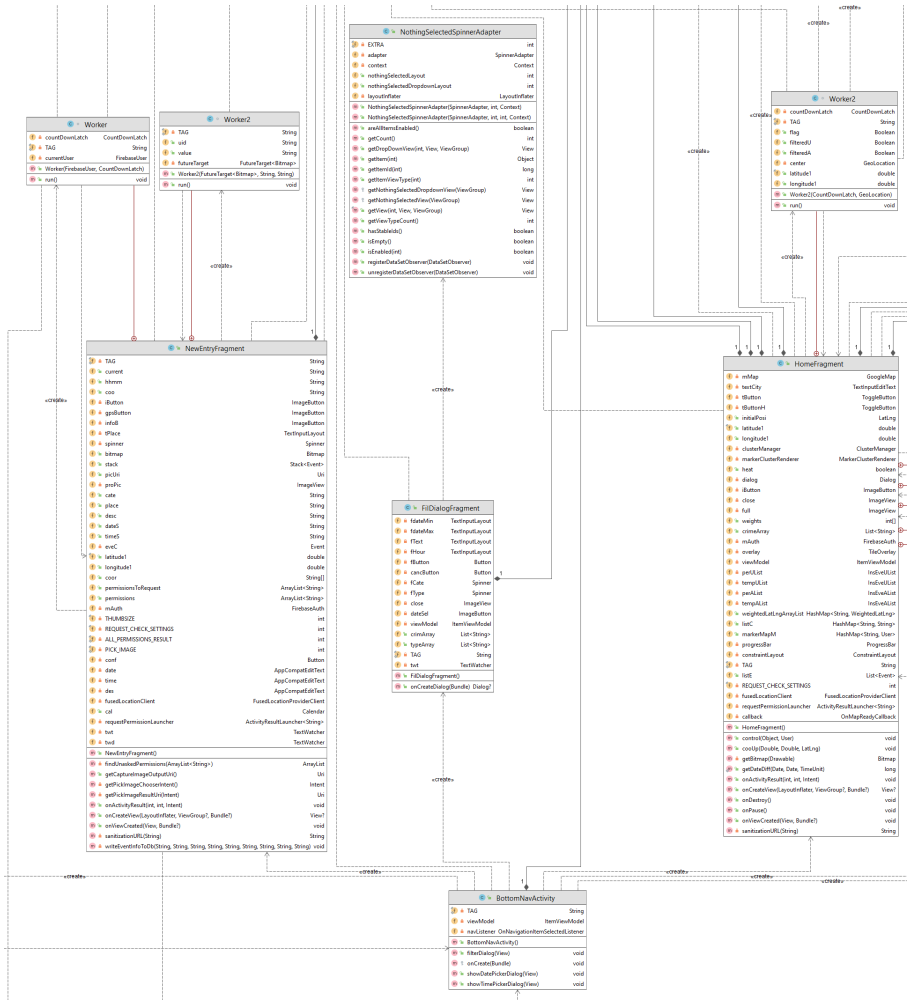

Figura 4.8. Settima vista del Diagramma delle Classi

La prima si occupa della creazione dell'Activity che realizza e gestisce la Bottom Navigation View, cioè la barra di navigazione tra i tre Fragment principali dell'app, ovvero quello della mappa, quello delle nuove segnalazioni e quello delle opzioni. Le successive tre classi, invece, creano i rispettivi Fragment; nello specifico, il primo realizza e gestisce la schermata in cui l'utente può effettuare una nuova segnalazione dopo aver compilato i campi obbligatori. Il secondo, invece, realizza e gestisce il Dialog utilizzato per la selezione dei filtri da applicare agli eventi mostrati sulla mappa. Il terzo, infine, realizza e gestisce la schermata della mappa degli eventi; da essa l'utente, oltre a visualizzare le varie segnalazioni nella zona, può anche decidere di selezionare una diversa modalità di visualizzazione, oppure applicare dei filtri sugli eventi presenti. L'ultima classe, **`NothingSelectedSpinnerAdapter`**, si occupa di realizzare l'elemento "nessuna opzione selezionata" dello Spinner (cioè è l'elemento dello Spinner presente di default quando nessuna opzione è stata ancora

selezionata), presente nel Dialog della selezione dei filtri.

Nella vista presente nella Figura 4.9 è possibile osservare le classi **`MainActivity`**, **`ChooseEnterActivity`**, **`LoginActivity`** e **`ResetActivity`**. Tutte queste classi si occupano della creazione delle loro rispettive Activity. La prima, ad esempio, realizza la schermata di splash screen dell'app durante il suo avvio. La seconda, invece, realizza e gestisce la schermata della selezione del metodo di Login, da questa schermata l'utente può decidere come autenticarsi prima di accedere alle funzionalità dell'app. La terza permette la registrazione di un nuovo account utente tramite l'utilizzo della e-mail e della password. L'ultima consente di poter reimpostare la password associata all'account utente legato alla e-mail inserita.

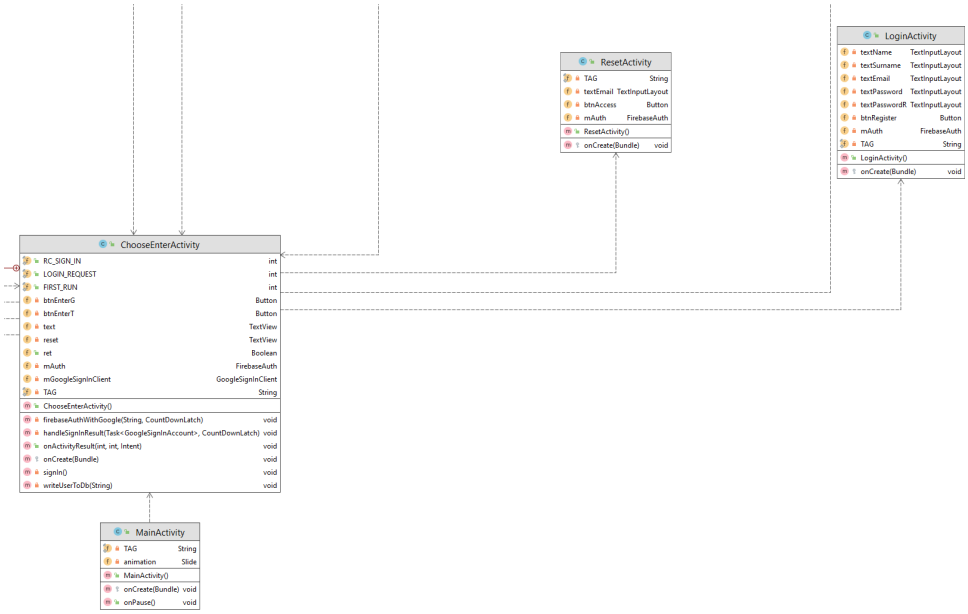

Figura 4.9. Ottava vista del Diagramma delle Classi

Nella vista presente nella Figura 4.10 è possibile osservare le classi **`DatePickerFragment`**, **`TimePickerFragment`** e **`MyInfoWindowAdapter`**. Le prime due classi si occupano della creazione dei loro rispettivi Fragment; la prima classe, per esempio, realizza un Dialog Fragment con cui l'utente può selezionare una data dal calendario che compare sopra la schermata attualmente visualizzata. La seconda classe, analogamente alla precedente, realizza un Dialog Fragment con cui l'utente può selezionare un orario dall'orologio che compare sopra la schermata attualmente visualizzata. La terza classe realizza le Info Windows che appaiono sopra i marker selezionati dalla mappa degli eventi e contengono al loro interno le informazioni relative alla segnalazione in esame.

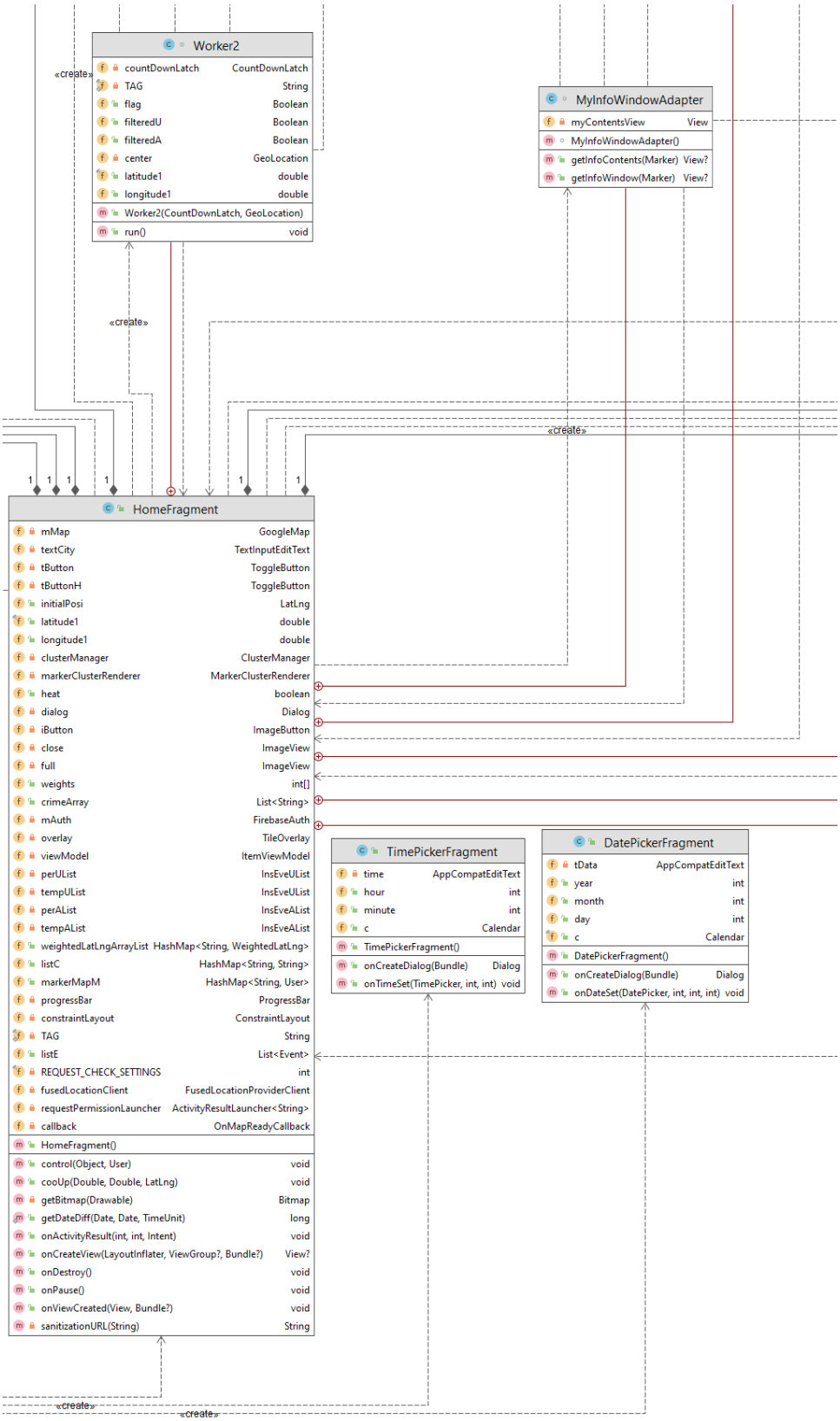

Figura 4.10. Nona vista del Diagramma delle Classi

Nella vista presente nella Figura 4.11 è possibile osservare la classe **`MarkerClusterRenderer`**. Essa ha lo scopo di implementare il renderer dei cluster di marker sulla mappa, nella pratica, quando sono presenti sulla mappa almeno quattro marker ravvicinati tra loro, questi verranno inglobati all'interno di un cluster. I cluster hanno lo scopo di ridurre il calcolo computazionale richiesto per il rendering di un numero elevato di marker contemporanei sulla mappa.

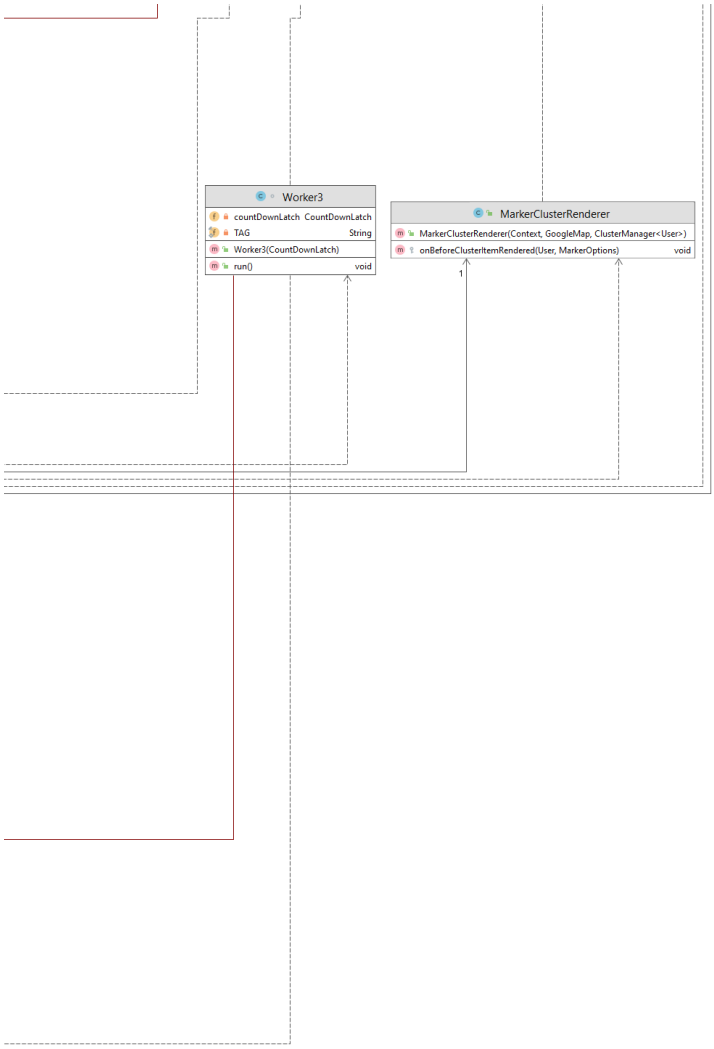

Figura 4.11. Decima vista del Diagramma delle Classi

Per concludere questa sezione riguardante il Diagramma delle Classi, è necessaria una piccola precisazione sulle classi "Worker" presenti in diverse viste analizzate. Esse sono classi interne ad un'altra classe e non possono essere istanziate senza

essere collegate ad una classe principale. Tutte queste classi interne sono dei thread che hanno lo scopo di realizzare una serie di operazioni asincrone, come quelle web, e implementare una forma di sincronizzazione tra di esse.

4.1.2 Diagramma delle Attività

La definizione del Diagramma delle Attività è già avvenuta nella Sezione 4.1, per cui passiamo subito ad esporre i vari diagrammi ottenuti dall'analisi dei casi d'uso. La prima attività realizzata è quella relativa al caso d'uso "Login utente", come possiamo osservare dalla Figura 4.12. Questa attività comprende sia il caso in cui l'utente possiede già un account a cui accedere tramite e-mail e password, sia il caso in cu egli non ha ancora un account con tali credenziali oppure voglia crearne un altro completamente nuovo. Come si evince dall'immagine, vengono eseguiti dei controlli sia in fase di autenticazione sia durante la fase di registrazione di un nuovo account; questi hanno lo scopo di evitare che un utente riesca ad accedere all'app senza prima autenticarsi, oppure, nel caso della registrazione, impediscono all'utente di realizzare un nuovo account senza che le informazioni inserite siano corrette o complete.

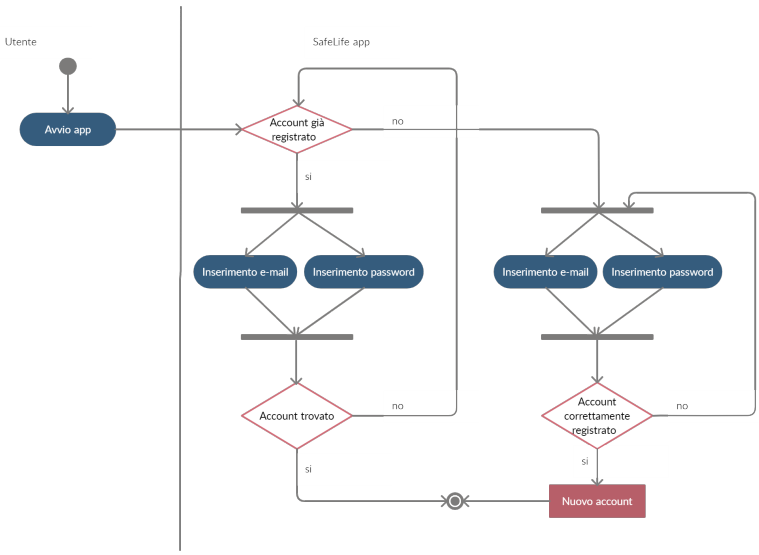

Figura 4.12. Diagramma delle Attività del caso d'uso "Login utente"

La seconda attività realizzata è quella relativa al caso d'uso "Visualizzazione mappa", come possiamo osservare dalla Figura 4.13. Questa attività riguarda la visualizzazione degli eventi sulla mappa tramite marker. Come si può vedere, alla richiesta dell'utente di mostrare gli eventi nella zona, l'app interroga il database remoto per il recupero delle informazioni. Tra queste spiccano, in particolare, quelle relative alla posizione geografica dell'evento che risultano essenziali per il posizionamento della segnalazione sulla mappa.

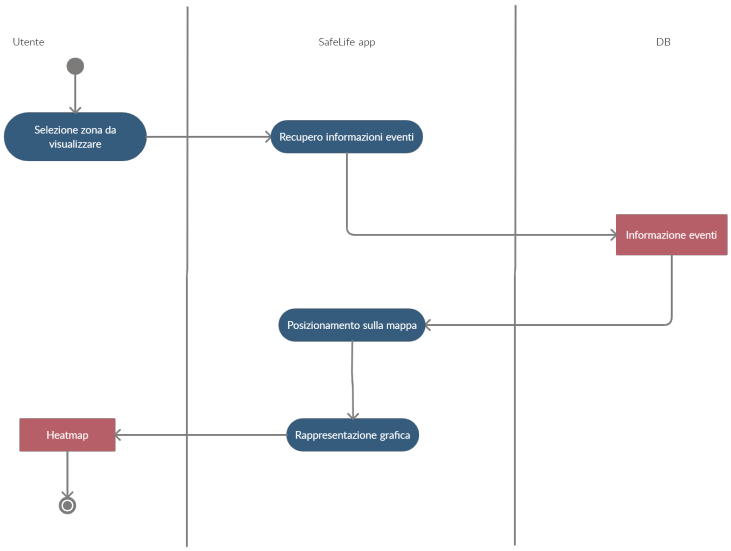

Figura 4.13. Diagramma delle Attività del caso d'uso "Visualizzazione mappa"

La terza attività realizzata è quella relativa al caso d'uso "Inserimento eventi", che possiamo osservare nella Figura 4.14.

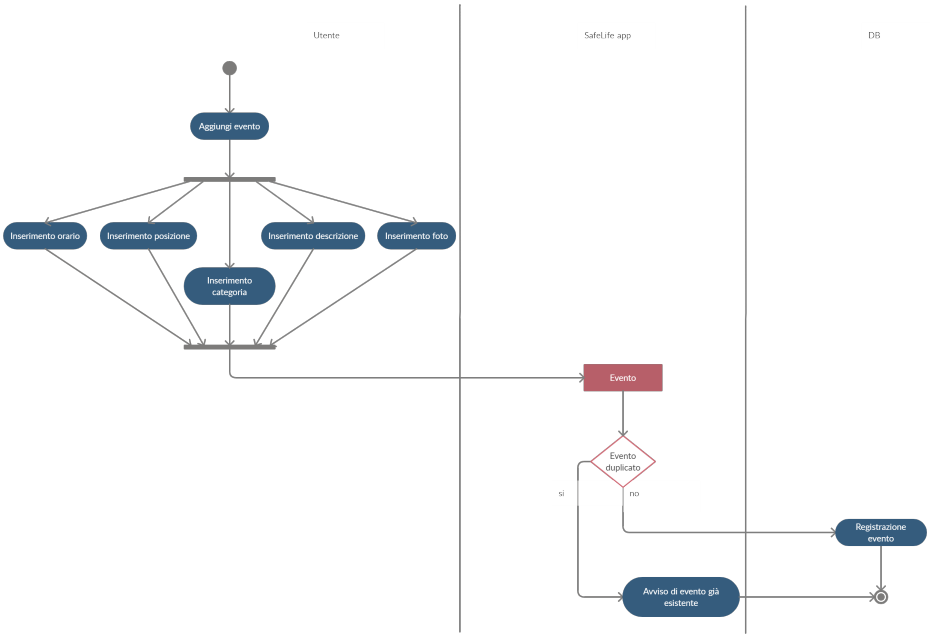

Figura 4.14. Diagramma delle Attività del caso d'uso "Inserimento eventi"

Questa attività riguarda il caso in cui l'utente decida di inserire una nuova segnalazione; dal diagramma possiamo notare che solo dopo l'inserimento delle informazioni l'evento viene creato. Però fino a questo punto il nuovo evento ancora non è stato salvato; infatti, prima di ciò, l'app compie una ricerca nel database per trovare la presenza di eventi duplicati. Se il database contiene tali eventi, questi verranno mostrati all'utente e gli chiederà se il nuovo evento corrisponde ad uno di questi; solo in caso di risposta negativa esso viene inserito nel database.

La quarta attività realizzata è quella relativa al caso d'uso "Controllo duplicati", come possiamo osservare dalla Figura 4.15.

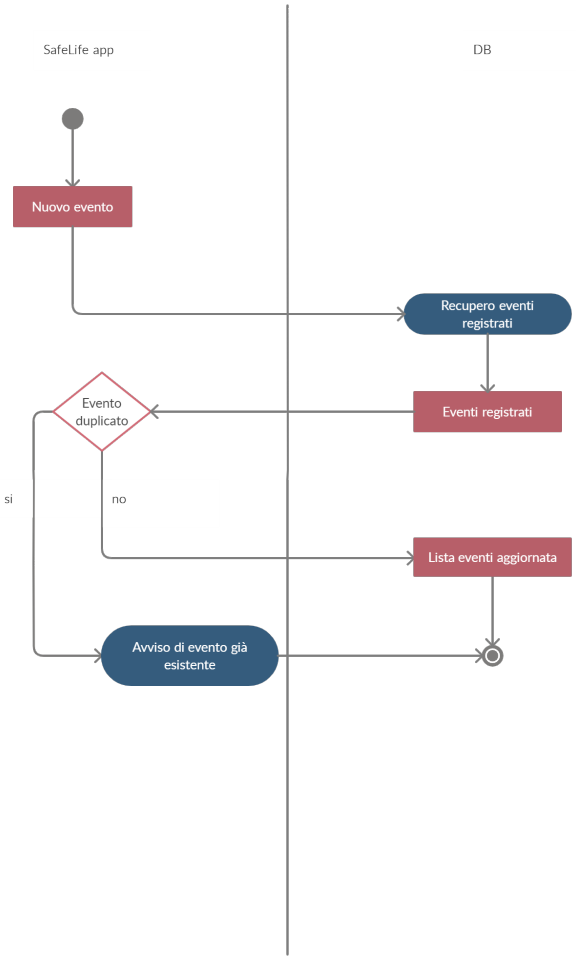

Figura 4.15. Diagramma delle Attività del caso d'uso "Controllo duplicati"

Questa attività riguarda il caso particolare della presenza di un evento duplicato che risulta simile ad altri eventi già presenti. Questo caso è già stato discus-

so nel diagramma precedente; però qui è maggiormente dettagliato e, soprattutto, mostra che l'attività di "Controllo duplicati" non è eseguita necessariamente dall'utente, ma viene eseguita ogniqualvolta avviene la registrazione di un nuovo evento. Questo implica che anche l'inserimento di un evento ottenuto da un articolo deve, necessariamente, passare attraverso questo controllo.

4.1.3 Diagramma dei Componenti

La definizione del Diagramma dei Componenti è già avvenuta nella Sezione 4.1, per cui passiamo subito ad esporre il diagramma ottenuto. In Figura 4.16 possiamo vedere che il diagramma è diviso tramite l'utilizzo delle Swim lane; concentriamoci sul livello più profondo, quello dell'Utilità. Qui appare chiaro come è stata concepita l'integrazione tra la piattaforma web e l'app. Come si vede, le due comunicano per mezzo del Backend del progetto, il quale è composto in parte da un database remoto per la memorizzazione delle informazioni riguardanti gli eventi. Il Backend funge, quindi, da punto di collegamento, ed essendo il database dinamico, ogni modifica compiuta da una delle due parti si riflette anche sull'altra in tempo reale.

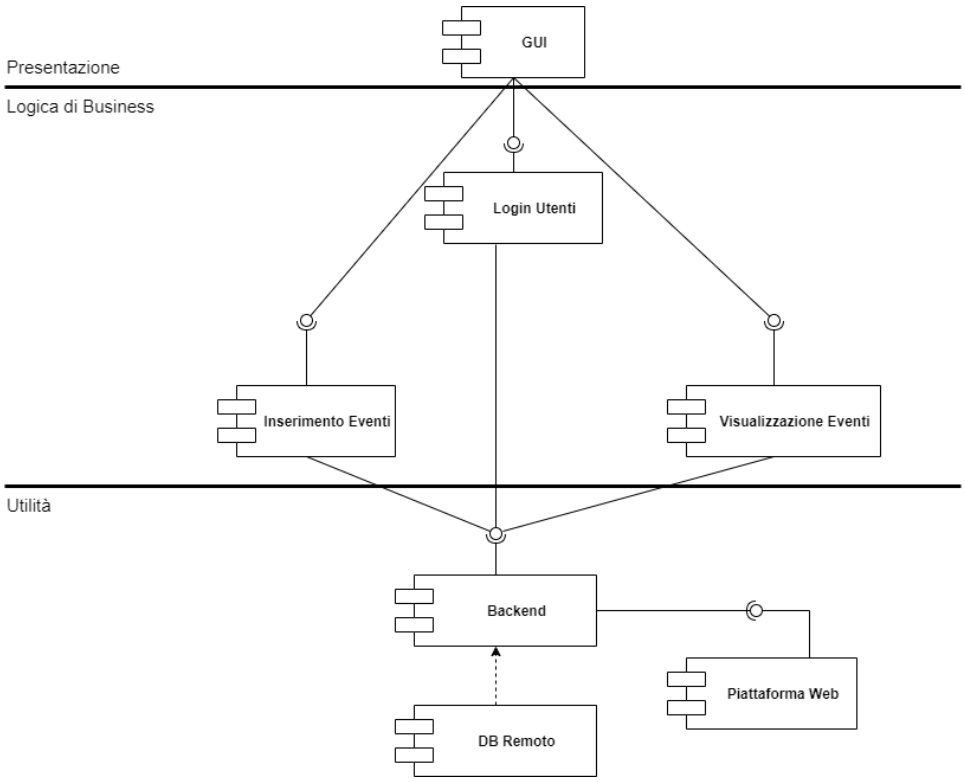

Figura 4.16. Diagramma delle Attività del caso d'uso "Controllo duplicati"

4.2 Diagramma dei Mockup

Prima di cominciare col diagramma è necessario spiegare cosa si intende per Mockup. In informatica per mockup si intende una realizzazione grafica di un prototipo della UI (User Interface) e delle funzionalità di un programma, con lo scopo di rendere comprensibile anche ai non addetti ai lavori (come gli stakeholder) come verranno realizzate le specifiche che l'applicazione deve soddisfare.

Il Diagramma dei Mockup non fa parte dei diagrammi UML in quanto è più simile ad un Diagramma di Flusso che mostra i passaggi che l'utente può compiere per muoversi da un Mockup ad un altro. Il diagramma in questione è riportato in Figura 4.17 e da esso possiamo osservare tutti i passaggi permessi all'utente in fase di progettazione dell'app.

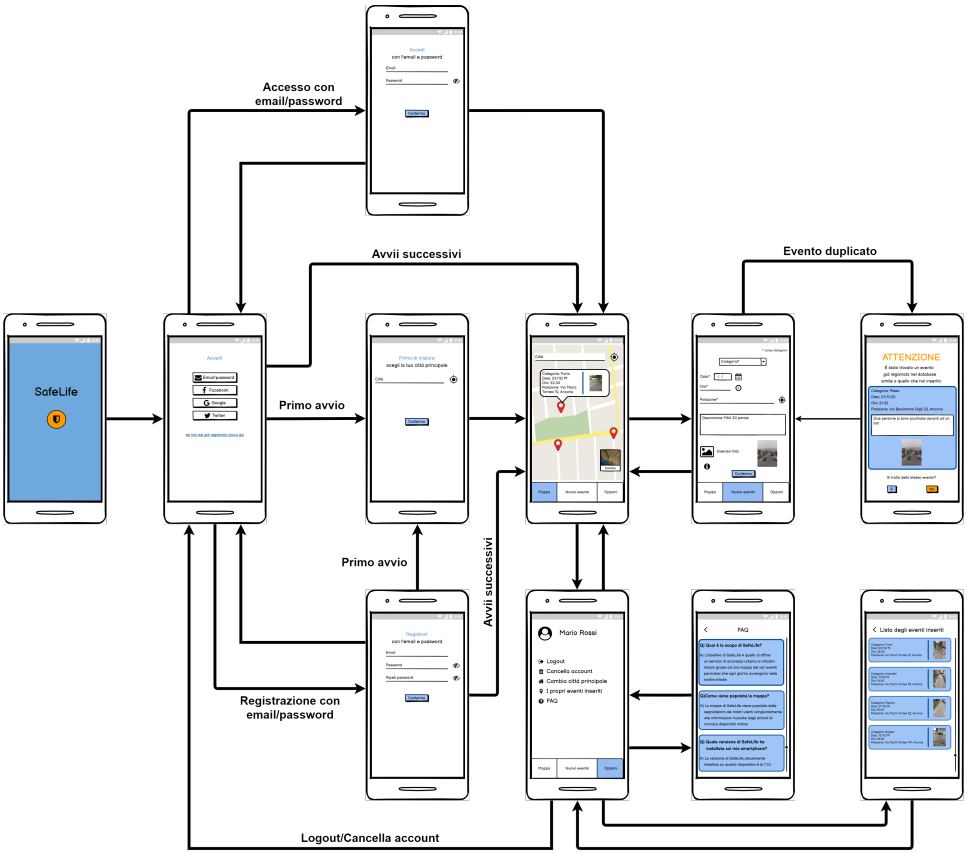

Figura 4.17. Diagramma dei Mockup costituito dai singoli mockup delle schermate dell'app e dall'insieme delle operazioni permesse per muoversi da una schermata all'altra

Partendo da sinistra, possiamo notare le prime schermate che l'utente incontra appena avviata l'app; superata la splash screen ci si imbatte nella schermata di selezione della modalità di autenticazione; da qui l'utente può scegliere se effettuare

il Login con un account già esistente e con il metodo che più preferisce (ovvero tramite il collegamento di un account esterno all'app, oppure, mediante e-mail e password). In caso l'utente sia sprovvisto di un account, può crearne uno inserendo un'e-mail e una password che gli verranno successivamente richieste in fase di Login.

Nel caso questa sia la prima volta che l'utente avvia l'app sul dispositivo corrente, gli verrà richiesto di inserire il nome di una città di partenza, chiamata "città principale", che ha lo scopo di impostare la camera della mappa (cioè una vista virtuale di una porzione della mappa che viene visualizzata dallo schermo del dispositivo, può essere modificata dall'utente tramite i controlli touch screen) sulla città in questione ogni volta che tale schermata viene avviata.

La prossima schermata che l'utente visualizzerà sarà quella della mappa. Questa è una delle tre schermate principali; le altre sono la schermata opzioni; è possibile passare da una all'altra tramite la pressione del pulsante corrispondente nella Bottom Navigation View presente in fondo a queste tre schermate. Come anticipato dai nomi, tali schermate permettono, rispettivamente, di visualizzare la mappa degli eventi presenti nella zona, di inserire un nuovo evento che possiede le informazioni specificate dall'utente e, per ultima, di scegliere una serie di opzioni dall'elenco fornito. Le restanti schermate mostrano alcune delle opzioni che è possibile selezionare dalla schermata "Opzioni", come la schermata delle FAQ e quella della lista degli eventi inseriti.

L'unica tra queste che non fa parte delle opzioni è la schermata degli eventi duplicati. Quest'ultima ha lo scopo di esporre all'utente le informazioni degli eventi simili a quello che egli cerca di inserire come nuova segnalazione; pertanto essa non può essere raggiunta dall'utente in maniera diretta, ma è l'app stessa che, in base alla presenza o meno di eventi simili, mostra questa schermata oppure no.

5

Implementazione

Ora che le funzionalità dell'app sono state descritte, comprese e progettate, verranno esposte le principali implementazioni svolte. Dopo aver discusso il codice delle classi, sarà illustrato il normale funzionamento dell'app tramite un manuale atto a mostrare le varie schermate con cui l'utente può interagire.

5.1 Implementazione delle classi

In accordo con quanto definito durante la fase di progettazione, sono state implementate le classi descritte nel Diagramma delle Classi.

5.1.1 La classe MainActivity

La prima classe che affronteremo è la **`MainActivity`**. Il codice della classe è mostrato per intero nel Listato 5.1. Possiamo notare che la classe **`MainActivity`** eredita il comportamento della classe **`AppCompatActivity`**, rendendola un'Activity a tutti gli effetti (riga 1).

Un'Activity è una classe che implementa una schermata dell'app, per cui, di solito, un'app è costituita da tante Activity quante sono le schermate; in particolare un'Activity può implementare i metodi **`onCreate`** e **`onPause`** come vediamo dal codice della nostra classe (riga 7 e 32). Come si può immaginare, i metodi **`onCreate`** e **`onPause`** vengono automaticamente eseguiti, rispettivamente, quando l'Activity viene creata e quando viene messa in pausa.

Sempre dal codice possiamo vedere che questa classe realizza la prima schermata che l'utente potrà vedere ad ogni avvio dell'app, cioè lo splash screen. In pratica, il compito di questa schermata è quello di rendere la transizione di apertura dell'app maggiormente piacevole per l'utente. Infatti, dal codice si può osservare che lo splash screen dura un secondo, dopodichè viene fatta partire una nuova Activity, chiamata **`ChooseEnterActivity`**, che costituirà la prima vera schermata con cui l'utente può interagire; una volta lanciata la nuova Activity, la **`MainActivity`** viene terminata (righe 17-27).

```
1   public class MainActivity extends AppCompatActivity {
2
3       private static final String TAG ="MainActivity" ;
4       private Slide animation;
5
6       @Override
7       protected void onCreate(Bundle savedInstanceState) {
8           super.onCreate(savedInstanceState);
9           getWindow().requestFeature(Window.FEATURE_ACTIVITY_TRANSITIONS);
10          animation=new Slide();
11          animation.setSlideEdge(Gravity.LEFT);
12          getWindow().setExitTransition(animation);
13          setContentView(R.layout.activity_main);
14
15          new Handler().postDelayed(new Runnable() {
16              @Override
17              public void run() {
18                  try {
19                      Intent intent = new Intent(MainActivity.this, ChooseEnterActivity.class);
20                      startActivity(intent);
21                      finish();
22                      overridePendingTransition(R.anim.fadein, R.anim.fadeout);
23                  }catch (Exception e){
24                      e.printStackTrace();
25                  }
26              }
27          }, 1000);
28
29      }
30
31      @Override
32      public void onPause() {
33          super.onPause();
34      }
35  }
36
37  }
```

Listato 5.1. Codice della classe **`MainActivity`**

5.1.2 La classe User

La classe successiva che analizzeremo è **`User`**. Questa classe implementa i metodi dell'interfaccia **`ClusterItem`**, con lo scopo di realizzare i metodi necessari a creare i marker degli eventi (icone presenti sulla mappa per identificare questi ultimi) degli eventi da posizionare sulla mappa e che dovranno essere inseriti all'interno dei Cluster (agglomerati di marker), in modo tale da ridurre il carico computazionale sulla CPU del dispositivo. Come possiamo vedere dal Listato 5.2, la classe implementa i metodi necessari alla registrazione e al recupero delle informazioni che costituiscono un marker (righe 7-42).

```
1   public class User implements ClusterItem {
2       public final BitmapDescriptor icon;
3       public final String snippet;
4       public final String title;
5       public final LatLng latLng;
6
7       public User( LatLng latLng, String snippet,BitmapDescriptor icon, String title) {
8           this.icon = icon;
9           this.title=title;
10          this.latLng = latLng;
11          this.snippet = snippet;
12      }
13
14      @Override
15      public LatLng getPosition() {
16          return latLng;
17      }
18
19      @Nullable
20      @org.jetbrains.annotations.Nullable
21      @Override
22      public String getTitle() {
23          return title;
24      }
25
26      @Override
```

```
27      public String getSnippet() {
28          return snippet;
29      }
30
31      public BitmapDescriptor getIcon() {
32          return icon;
33      }
34
35      @NonNull
36      @Override
37      public String toString() {
38          return "User{" +
39                  "username='" + snippet + '\'' +
40                  ", latLng=" + latLng +
41                  '}';
42      }
43  }
```

Listato 5.2. Codice della classe **`User`**

5.1.3 La classe LoginActivity

La prossima classe che analizzeremo è **`LoginActivity`**, mostrata dal Listato 5.3. Anche questa classe è un'Activity; in particolare essa si occupa di realizzare e gestire la schermata di registrazione di un nuovo account, utilizzando l'e-mail e la password come credenziali. La schermata realizzata da questa Activity presenta una serie di campi di testo in cui l'utente può e deve inserire le informazioni richieste per potersi registrare.

Come si legge dal codice, tali informazioni sono: il nome e cognome dell'utente, un'e-mail valida per la registrazione ed una password (di almeno sei caratteri), che dovrà essere ripetuta per evitare errori da parte dell'utente (riga 3). Una volta inserite tutte le informazioni, egli può premere il pulsante di conferma e, a questo punto, verrà eseguito un controllo sulle informazioni inserite. Viene analizzata l'e-mail per verificare che sia correttamente formattata e che non ci sia un altro account con la stessa e-mail; successivamente la password viene analizzata per controllare che sia almeno di sei caratteri e che corrisponda alla password ripetuta (righe 21-34).

Nel caso in cui tutti i controlli descritti vadano a buon fine, il nuovo account viene creato e viene restituita all'Activity chiamante il risultato "RESULT_OK", che indica che il nuovo account è stato correttamente creato, dopodichè l'Activity viene terminata (righe 36-53).

Nel caso in cui i dati inseriti non dovessero superare questi controlli, l'utente verrà avvertito con un messaggio in sovraimpressione sulla schermata che indica quale sia il problema riscontrato (righe 54-85).

```
1   public class LoginActivity extends AppCompatActivity {
2
3       private TextInputLayout textName, textSurname, textEmail, textPassword,textPasswordR;
4       private Button btnRegister;
5       private FirebaseAuth mAuth;
6       private static final String TAG ="LoginActivity" ;
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          overridePendingTransition(R.anim.slide_in, R.anim.fadeout);
12          setContentView(R.layout.activity_login);
13          mAuth = FirebaseAuth.getInstance();
14          textName = findViewById(R.id.textNom);
15          textSurname = findViewById(R.id.textCo);
16          textEmail = findViewById(R.id.textEm);
17          textPassword = findViewById(R.id.textPa);
18          textPasswordR = findViewById(R.id.textPaR);
19          btnRegister = findViewById(R.id.btn_registra2);
20
```

```
21          btnRegister.setOnClickListener(new View.OnClickListener() {
22
23              @Override
24              public void onClick(View v) {
25                  try {
26
27                      InputMethodManager inputMethodManager = (InputMethodManager)getSystemService(
                               INPUT_METHOD_SERVICE);
28                      inputMethodManager.hideSoftInputFromWindow(v.getApplicationWindowToken(),0);
29                      final String name = textName.getEditText().getText().toString();
30                      final String surname = textSurname.getEditText().getText().toString();
31                      final String email = textEmail.getEditText().getText().toString();
32                      final String password = textPassword.getEditText().getText().toString();
33                      final String passwordR = textPasswordR.getEditText().getText().toString();
34
35                      if(password.equals(passwordR)) {
36
37                          mAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(new
                                   OnCompleteListener<AuthResult>() {
38                              @Override
39                              public void onComplete(@NonNull Task<AuthResult> task) {
40                                  if (task.isSuccessful()) {
41                                      final FirebaseUser user = mAuth.getCurrentUser();
42                                      UserProfileChangeRequest profileChangeRequest = new UserProfileChangeRequest.
                                               Builder()
43                                              .setDisplayName(name + " " + surname)
44                                              .build();
45
46                                      user.updateProfile(profileChangeRequest).addOnCompleteListener(new
                                               OnCompleteListener<Void>() {
47                                          @Override
48                                          public void onComplete(@NonNull Task<Void> task) {
49
50                                              Intent intent = new Intent();
51                                              setResult(RESULT_OK, intent);
52                                              finish();
53                                          }
54                                      });
55                                  } else {
56                                      try {
57
58                                          throw task.getException();
59                                      } catch(FirebaseAuthUserCollisionException e) {
60
61                                          Toast.makeText(LoginActivity.this, getString(R.string.emailR), Toast.
                                                   LENGTH_SHORT).show();
62                                      }  catch (FirebaseAuthWeakPasswordException e){
63                                          Toast.makeText(LoginActivity.this, getString(R.string.passC), Toast.
                                                   LENGTH_SHORT).show();
64
65                                      } catch(FirebaseAuthInvalidCredentialsException e) {
66
67                                          Toast.makeText(LoginActivity.this, getString(R.string.emaE), Toast.
                                                   LENGTH_SHORT).show();
68                                      } catch(FirebaseTooManyRequestsException e) {
69
70                                          Toast.makeText(LoginActivity.this, getString(R.string.errorblo), Toast.
                                                   LENGTH_SHORT).show();
71                                      }
72                                    catch(Exception e) {
73                                          task.getException().printStackTrace();
74                                        Toast.makeText(LoginActivity.this, getString(R.string.errorsignup), Toast.
                                                 LENGTH_SHORT).show();
75                                      }
76                                  }
77                              }
78                          });
79                      }else
80                          Toast.makeText(LoginActivity.this, getString(R.string.errorpassu), Toast.LENGTH_SHORT).show()
                                   ;
81
82                  } catch(Exception e) {
83                      Toast.makeText(LoginActivity.this, getString(R.string.inforequired), Toast.LENGTH_SHORT).show();
84                  }
85              }
86          });
87
88      }
89  }
```

Listato 5.3. Codice della classe **`LoginActivity`**

5.1.4 La classe Event

La prossima classe che analizzeremo è **`Event`**, mostrata nel Listato 5.4. Questa classe ha lo scopo di istanziare oggetti rappresentanti un evento pericoloso (segnalato

da un utente o ricavato da un articolo); per questo motivo, i suoi attributi servono a registrare le informazioni che costituiscono un evento, mentre i metodi servono a memorizzare e recuperare tali informazioni (righe 20-55).

```
1   public class Event implements Serializable {
2
3       private String coor;
4       private String imgPre;
5       private Integer veri;
6       private String imgUrl;
7       private String uID;
8       private String path;
9       private String date;
10      private String cate;
11      private String hour;
12      private String place;
13      private String tiArt;
14      private String introArt;
15      private String urlArt;
16
17      private String descr;
18      private byte[] ima;
19
20      public Event(String coordinate, String path) {
21          this.coor = coordinate;
22          this.path = path;
23
24      }
25
26       public void setTiArt(String tiArt) {
27           this.tiArt = tiArt;
28       }
29
30       public String getTiArt() {
31           return tiArt;
32       }
33
34       public void setIntroArt(String introArt) {
35           this.introArt = introArt;
36       }
37
38       public String getIntroArt() {
39           return introArt;
40       }
41
42       public void setUrlArt(String urlArt) {
43           this.urlArt = urlArt;
44       }
45
46       public String getUrlArt() {
47           return urlArt;
48       }
49  ...
50  ...
51  ...
52
53      public String toString() {
54          return coor + ";" + path+";"+ cate +";"+ date +";"+ hour +";"+ place +";"+ descr;
55      }
56  }
```

Listato 5.4. Codice della classe **`Event`**

5.1.5 La classe InsEveAList

La classe **`InsEveAList`** viene mostrata dal Listato 5.5. Essa ha lo scopo di istanziare oggetti contenenti una serie di HashMap delle informazioni riguardanti una lista di eventi pericolosi ricavati dagli articoli; per questo motivo, i suoi attributi servono a registrare le informazioni che costituiscono una lista di eventi, mentre i metodi implementati memorizzano e recuperano tali informazioni (righe 11-43).

Un metodo particolare della classe è **`clear`**, che ha lo scopo di pulire gli attributi da ogni informazione precedentemente memorizzata (righe 44-63).

```
1   public class InsEveAList implements Serializable {
2       private HashMap<String,String> cateA = new HashMap<>(100);
```

```
3       private HashMap<String,String> listA = new HashMap<>(100);
4       private HashMap<String,String> placeA = new HashMap<>(100);
5       private HashMap<String, Calendar> dateA = new HashMap<>(100);
6       private HashMap<String,String> titleA = new HashMap<>(100);
7       private HashMap<String,String> introA = new HashMap<>(100);
8       private HashMap<String,String> urlA = new HashMap<>(100);
9       private HashMap<String,String> urlIA = new HashMap<>(100);
10
11      public HashMap<String, Calendar> getDateA() {
12          return dateA;
13      }
14
15      public HashMap<String, String> getCateA() {
16          return cateA;
17      }
18  ...
19  ...
20  ...
21      public void setCateA(HashMap<String, String> cateA) {
22          this.cateA = cateA;
23      }
24
25      public void setDateA(HashMap<String, Calendar> dateA) {
26          this.dateA = dateA;
27      }
28  ...
29  ...
30  ...
31      public void putCateA(String key,String val){
32          this.cateA.put(key,val);
33      }
34      public void putDataA(String key,Calendar val){
35          this.dateA.put(key,val);
36      }
37      public void putTitoloA(String key,String val){
38          this.titleA.put(key,val);
39      }
40
41  ...
42  ...
43  ...
44      public void putAll(InsEveAList val){
45          this.cateA.putAll(val.getCateA());
46          this.dateA.putAll(val.getDateA());
47          this.urlIA.putAll(val.getUrlIA());
48          this.urlA.putAll(val.getUrlA());
49          this.introA.putAll(val.getIntroA());
50          this.listA.putAll(val.getListA());
51          this.placeA.putAll(val.getPlaceA());
52          this.titleA.putAll(val.getTitleA());
53      }
54      public void clear(){
55          this.cateA.clear();
56          this.dateA.clear();
57          this.introA.clear();
58          this.listA.clear();
59          this.placeA.clear();
60          this.urlA.clear();
61          this.urlIA.clear();
62          this.titleA.clear();
63      }
64  }
```

Listato 5.5. Codice della classe **`InsEveAList`**

5.1.6 La classe InsEveUList

La classe **`InsEveUList`** viene mostrata dal Listato 5.6. Essa ha lo scopo di istanziare oggetti contenenti una serie di HashMap delle informazioni riguardanti una lista di eventi pericolosi ricavati dalle segnalazioni degli utenti; per questo motivo, i suoi attributi servono a registrare le informazioni che costituiscono una lista di eventi, mentre i metodi implementati servono a memorizzare e recuperare tali informazioni (righe 7-51).

Un metodo particolare della classe è **`clear`**, che ha lo scopo di pulire gli attributi da ogni informazione precedentemente memorizzata (righe 52-57).

```
1   public class InsEveUList implements Serializable {
```

```
2       private HashMap<String, Bitmap> image = new HashMap<>(100);
3   ...
4   ...
5   ...
6
7       public HashMap<String, Bitmap> getImage() {
8           return image;
9       }
10
11      public HashMap<String, String> getCate() {
12          return cate;
13      }
14  ...
15  ...
16  ...
17
18      public void setCate(HashMap<String, String> cate) {
19          this.cate = cate;
20      }
21
22      public void setDate(HashMap<String, String> date) {
23          this.date = date;
24      }
25  ...
26  ...
27  ...
28      public void putCate(String key,String val){
29          this.cate.put(key,val);
30      }
31      public void putData(String key,String val){
32          this.date.put(key,val);
33      }
34  ...
35  ...
36  ...
37      public void putAllCate(HashMap val){
38          this.cate.putAll(val);
39      }
40      public void putAllData(HashMap val){
41          this.date.putAll(val);
42      }
43  ...
44  ...
45  ...
46      public void putAll(InsEveUList val){
47          this.cate.putAll(val.getCate());
48  ...
49  ...
50  ...
51      }
52      public void clear(){
53          this.cate.clear();
54  ...
55  ...
56  ...
57      }
58  }
```

Listato 5.6. Codice della classe **`InsEveUList`**

5.1.7 La classe ItemViewModel

La classe **`ItemViewModel`** viene mostrata nel Listato 5.7. Questa classe ha lo scopo di permettere il passaggio di un Array di String tra un'Activity e un'altra, o tra Fragment, consentendo quindi lo scambio di informazioni tra classi diverse. I suoi metodi servono quindi per la scrittura e lettura dell'Array da parte di un'altra classe (righe 3-8).

```
1   public class ItemViewModel extends ViewModel {
2       private final MutableLiveData<String[]> selectedItem = new MutableLiveData<>();
3       public void selectItem(String[] item) {
4           selectedItem.setValue(item);
5       }
6       public LiveData<String[]> getSelectedItem() {
7           return selectedItem;
8       }
9   }
```

Listato 5.7. Codice della classe **`ItemViewModel`**

5.1.8 La classe DatePickerFragment

La classe **DatePickerFragment** viene mostrata nel Listato 5.8. Essa ha lo scopo di realizzare un Dialog Fragment che permetta all'utente di selezionare una data dal calendario che appare sopra la schermata attualmente visualizzata.

Come si può notare dal codice, il risultato, cioè la data selezionata dall'utente, viene scritto all'interno del campo di testo dell'Activity **NewEventFragment** dedicata all'inserimento della data (riga 24).

```
1   public class DatePickerFragment extends DialogFragment
2         implements DatePickerDialog.OnDateSetListener{
3       private AppCompatEditText tData;
4       public int year,month,day;
5       public final Calendar c= Calendar.getInstance();
6
7       @Override
8       public Dialog onCreateDialog(Bundle savedInstanceState) {
9
10          year = c.get(YEAR);
11          month = c.get(MONTH);
12          day = c.get(DAY_OF_MONTH);
13          DatePickerDialog dp= new DatePickerDialog(getActivity(), this, year, month, day);
14          dp.getDatePicker().setMaxDate(c.getTimeInMillis());
15          c.set(YEAR,(year-1));
16          dp.getDatePicker().setMinDate(c.getTimeInMillis());
17
18          return dp;
19      }
20
21      public void onDateSet(DatePicker view, int year, int month, int day) {
22
23          tData=getActivity().findViewById(R.id.editTextDate);
24          tData.setText(String.format("\%02d",day) +"/"+ String.format("\%02d",(month+1) ) +"/"+String.format("\%04d"
                ,year));
25      }
26  }
```

Listato 5.8. Codice della classe **DatePickerFragment**

5.1.9 La classe TimePickerFragment

La classe **TimePickerFragment** viene mostrata nel Listato 5.9. Essa ha lo scopo di realizzare un Dialog Fragment che permetta all'utente di selezionare un orario dall'orologio che appare sopra la schermata attualmente visualizzata.

Come si può notare dal codice, il risultato, cioè l'ora selezionata dall'utente, viene scritta all'interno del campo di testo dell'Activity **NewEventFragment**, dedicata all'inserimento dell'orario (riga 19).

```
1   public class TimePickerFragment extends DialogFragment
2         implements TimePickerDialog.OnTimeSetListener {
3       private AppCompatEditText time;
4       public int hour,minute;
5       public Calendar c;
6
7       @Override
8       public Dialog onCreateDialog(Bundle savedInstanceState) {
9           c = Calendar.getInstance();
10          hour = c.get(Calendar.HOUR_OF_DAY);
11          minute = c.get(Calendar.MINUTE);
12          return new TimePickerDialog(getActivity(), this, hour, minute,true);
13      }
14
15      public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
16          time=getActivity().findViewById(R.id.editTextTime);
17          time.setText(String.format("\%02d",hourOfDay) +":"+ String.format("\%02d",minute ) );
18      }
19  }
```

Listato 5.9. Codice della classe **TimePickerFragment**

5.1.10 La classe ResetActivity

La classe **ResetActivity** viene mostrata nel Listato 5.10. Essa ha lo scopo di creare un'Activity che permette di realizzare e gestire una schermata che consente di reimpostare la password dell'account collegato alla e-mail inserita dall'utente.

Una volta che l'utente ha inserito l'e-mail, e dopo aver premuto il tasto di conferma, viene inviata una mail contenente un link per reimpostare la password all'indirizzo specificato (righe 22-35).

```
1   public class ResetActivity extends AppCompatActivity {
2       private static final String TAG ="ResetActivity" ;
3       private TextInputLayout textEmail;
4       private Button btnAccess;
5       private FirebaseAuth mAuth;
6
7       @Override
8       protected void onCreate(Bundle savedInstanceState) {
9           super.onCreate(savedInstanceState);
10          setContentView(R.layout.activity_reset);
11          mAuth = FirebaseAuth.getInstance();
12          overridePendingTransition(R.anim.slide_in, R.anim.fadeout);
13          textEmail = findViewById(R.id.textInputLayoutReset);
14
15          btnAccess = findViewById(R.id.buttonReset);
16
17          btnAccess.setOnClickListener(new View.OnClickListener() {
18
19              @Override
20              public void onClick(View v) {
21                  try {
22                      String email = textEmail.getEditText().getText().toString();
23                      mAuth.sendPasswordResetEmail(email)
24                              .addOnCompleteListener(new OnCompleteListener<Void>() {
25                                  @Override
26                                  public void onComplete(@NonNull Task<Void> task) {
27                                      if (task.isSuccessful()) {
28                                          Toast.makeText(ResetActivity.this, getString(R.string.maili), Toast.
                                                    LENGTH_SHORT).show();
29                                      }
30                                   else {
31                                      task.getException().printStackTrace();
32                                      Toast.makeText(ResetActivity.this, getString(R.string.mailn), Toast.LENGTH_SHORT).
                                                show();
33                                  }
34                                  }
35                              });
36
37                  } catch (Exception e) {
38                      Toast.makeText(ResetActivity.this, getString(R.string.inforequired), Toast.LENGTH_SHORT).show();
39                  }
40              }
41          });
42
43      }
44  }
```

Listato 5.10. Codice della classe **ResetActivity**

5.1.11 La classe MarkerClusterRenderer

La classe **MarkerClusterRenderer** viene mostrata nel Listato 5.11. Questa classe ha lo scopo di realizzare un renderer dei Cluster ereditando da **DefaultClusterRenderer** il suo comportamento. Come possiamo vedere, i Cluster realizzati contengono elementi della classe **User** da cui estrae le informazioni chiave del marker, come il titolo, il testo e l'icona (righe 12-14).

Con le informazioni ottenute vengono realizzati degli elementi del Cluster costituiti dalle stesse informazioni appena ricevute. In questo modo il Cluster ingloba al proprio interno i marker che sono ravvicinati tra loro, dopodichè il numero di elementi contenuto al suo interno viene incrementato e mostrato sopra di esso.

Quando, invece, si ingrandisce la visuale della mappa su un Cluster, tramite zoom, esso si divide mostrando i singoli marker sulla mappa nella posizione in cui l'evento è accaduto.

```
1   @SuppressLint("InflateParams")
2   public class MarkerClusterRenderer extends DefaultClusterRenderer<User> {
3
4       public MarkerClusterRenderer(@NonNull Context context, GoogleMap map, ClusterManager<User> clusterManager) {
5           super(context, map, clusterManager);
6
7       }
8
9       @Override
10      protected void onBeforeClusterItemRendered(User item, MarkerOptions markerOptions) {
11          try {
12              markerOptions.icon(item.getIcon());
13              markerOptions.snippet(item.getSnippet());
14              markerOptions.title(item.getTitle());
15          }catch (Exception e){
16              e.printStackTrace();
17          }
18
19      }
20  }
```

Listato 5.11. Codice della classe **`MarkerClusterRenderer`**

5.1.12 La classe FaqActivity

La classe **`FaqActivity`** viene mostrata nel Listato 6.2. Questa classe ha lo scopo di realizzare un'Activity che crea e gestisce la schermata delle FAQ dell'app; come si vede dal codice, viene realizzato un Recycler View composto dalla lista di domande e delle rispettive risposte.

Sia le domande che le risposte sono conservate sotto forma di Array di String; ogni volta che viene visualizzata questa schermata delle FAQ, le informazioni corrispondenti vengono recuperate e passate ad un'istanza della classe **`CustomAdapterF`**, che ha lo scopo di realizzare i singoli elementi che costituiscono il Recycler View (righe 14-19).

```
1   public class FaqActivity extends AppCompatActivity {
2       private RecyclerView recView;
3       public String[] q,a;
4
5       @Override
6       protected void onCreate(@Nullable Bundle savedInstanceState) {
7           super.onCreate(savedInstanceState);
8           overridePendingTransition(R.anim.slide_in, R.anim.fadeout);
9           setContentView(R.layout.activity_faq);
10          Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbarF);
11  ...
12  ...
13  ...
14          q= getResources().getStringArray(R.array.Question);
15          a= getResources().getStringArray(R.array.Answer);
16          recView= findViewById(R.id.recyclerF);
17          CustomAdapterF myAdapter= new CustomAdapterF(q,a);
18          recView.setAdapter(myAdapter);
19          recView.setLayoutManager(new LinearLayoutManager(getBaseContext()));
20      }
21  }
```

Listato 5.12. Codice della classe **`FaqActivity`**

5.1.13 La classe EnterActivity

La prossima classe che analizzeremo è **`EnterActivity`**, viene mostrata nel Listato 5.13. Essa ha lo scopo di realizzare un'Activity che crea e gestisce la schermata di

login dell'utente tramite e-mail e password. Questa classe è simile alla classe **LoginActivity**, che permetteva di registrare un nuovo account; infatti, come vediamo dal codice, l'utente deve inserire l'e-mail e la password di un account registrato e poi premere il tasto di conferma (riga 3).

Alla pressione di questo tasto, vengono effettuati dei controlli sulle informazioni inserite per verificare che corrispondano ad un account esistente. In caso di fallimento l'utente viene avvertito tramite un messaggio in sovraimpressione; in caso di successo viene restituito all'Activity chiamante il risultato "RESULT_OK", che indica che le credenziali sono corrette; dopodichè l'Activity viene terminata (righe 14-52).

```
1   public class EnterActivity extends AppCompatActivity {
2
3       private TextInputEditText textEmail, textPassword;
4       private Button btnEnter;
5       private FirebaseAuth mAuth;
6
7       @Override
8       protected void onCreate(Bundle savedInstanceState) {
9           super.onCreate(savedInstanceState);
10          overridePendingTransition(R.anim.slide_in, R.anim.fadeout);
11  ...
12  ...
13  ...
14          btnEnter.setOnClickListener(new View.OnClickListener() {
15
16              @Override
17              public void onClick(View v) {
18                  try {
19                      InputMethodManager inputMethodManager = (InputMethodManager)getSystemService(
                                INPUT_METHOD_SERVICE);
20                      inputMethodManager.hideSoftInputFromWindow(v.getApplicationWindowToken(),0);
21                      String email = textEmail.getText().toString();
22                      String password = textPassword.getText().toString();
23                      mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener(new OnCompleteListener<
                                AuthResult>() {
24
25                          @Override
26                          public void onComplete(@NonNull Task<AuthResult> task) {
27                              if (task.isSuccessful()) {
28                                  Intent intent = new Intent();
29                                  setResult(RESULT_OK, intent);
30                                  finish();
31                              } else {
32                                  task.getException().printStackTrace();
33                                  try {
34                                      throw task.getException();
35                                  } catch(FirebaseAuthInvalidUserException e) {
36                                      Toast.makeText(EnterActivity.this,getString(R.string.erroruser) , Toast.
                                                LENGTH_SHORT).show();
37                                  } catch(FirebaseAuthInvalidCredentialsException e) {
38                                      Toast.makeText(EnterActivity.this, getString(R.string.errorpass), Toast.
                                                LENGTH_SHORT).show();
39                                  } catch(FirebaseTooManyRequestsException e) {
40                                      Toast.makeText(EnterActivity.this, getString(R.string.errorblo), Toast.
                                                LENGTH_SHORT).show();
41                                  }catch(Exception e) {
42                                      task.getException().printStackTrace();
43                                      Toast.makeText(EnterActivity.this, getString(R.string.errorsignup), Toast.
                                                LENGTH_LONG).show();
44                                  }
45                              }
46                             }
47                      });
48                  } catch (Exception e) {
49                      Toast.makeText(EnterActivity.this, getString(R.string.inforequired), Toast.LENGTH_SHORT).show();
50                  }
51              }
52          });
53      }
54  }
```

Listato 5.13. Codice della classe **EnterActivity**

5.1.14 La classe BottomNavActivity

La classe **`BottomNavActivity`** viene mostrata nel Listato 5.14. Essa ha lo scopo di realizzare un'Activity che permette di creare e gestire una Bottom Navigation View con tre pulsanti: Mappa, Nuovo evento e Opzioni. Alla pressione di uno di essi, viene caricato il Fragment corrispondente, così come vediamo dal codice (righe 35-50).

Inoltre, prima di effettuare il cambio di schermata, l'Array di String che contiene i valori dei filtri selezionati dall'utente viene pulito, in modo tale che, al cambio della schermata, i valori scelti non vengano conservati (riga 64).

Gli ultimi tre metodi della classe hanno lo scopo di creare i rispettivi Dialog (**`DatePickerFragment`**, **`TimePickerFragment`** e **`FilDialogFragment`**) alla pressione del tasto dedicato ad ognuno di essi (righe 76-89).

```
1   public class BottomNavActivity extends AppCompatActivity{
2
3
4       private static final String TAG ="BottomNavActivity" ;
5       private ItemViewModel viewModel;
6
7       @SuppressLint("WrongViewCast")
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          try {
11              super.onCreate(savedInstanceState);
12              if (android.os.Build.VERSION.SDK_INT >= 28) {
13                  getWindow().setFlags(
14                          WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED,
15                          WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED);
16              }
17
18              getWindow().requestFeature(Window.FEATURE_ACTIVITY_TRANSITIONS);
19              this.setContentView(R.layout.activity_bottomnav);
20              BottomNavigationView bottomNav = findViewById(R.id.bottom_navigation);
21              bottomNav.setOnNavigationItemSelectedListener(navListener);
22              getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container, new HomeFragment()).
                        commit();
23
24
25          } catch (Exception e) {
26              throw e;
27          }
28
29      }
30      private BottomNavigationView.OnNavigationItemSelectedListener navListener =
31              new BottomNavigationView.OnNavigationItemSelectedListener() {
32
33
34                  @Override
35                  public boolean onNavigationItemSelected(@NonNull MenuItem item) {
36                      Fragment selectedFragment=null;
37                      View view = getCurrentFocus();
38                      switch (item.getItemId()) {
39                          case R.id.nav_blankFragment:
40                              selectedFragment = new HomeFragment();
41                              break;
42                          case R.id.nav_blankFragment2:
43                              selectedFragment = new NewEntryFragment();
44
45                              break;
46                          case R.id.nav_blankFragment3:
47                              selectedFragment = new OptionsFragment();
48                              break;
49
50                      }
51                      if (view != null) {
52                          InputMethodManager imm = (InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);
53                          imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
54                      }
55                      Fragment finalSelectedFragment = selectedFragment;
56                      new Handler().postDelayed(new Runnable() {
57                          @Override
58                          public void run() {
59                              try {
60                                  viewModel = new ViewModelProvider(BottomNavActivity.this).get(ItemViewModel.class);
61                                  String[] filters=viewModel.getSelectedItem().getValue();
62
63                                  if (filters==null  !filters[0].equals("1") ) {
64                                      viewModel.selectItem(new String[]{"", "", "", "", "", "0", ""});
```

```
65                                  getSupportFragmentManager().beginTransaction().setCustomAnimations(R.anim.slide_in
                                           , R.anim.fade_out).replace(R.id.fragment_container, finalSelectedFragment)
                                           .commit();
66                                  }
67                              }catch (Exception e){
68                                  e.printStackTrace();
69                              }
70                          }
71                      }, 50);
72
73                      return true;
74                  }
75              };
76      public void showDatePickerDialog(View v) {
77          DialogFragment newFragment = new DatePickerFragment();
78          newFragment.show(getSupportFragmentManager(), "datePicker");
79      }
80
81      public void showTimePickerDialog(View v) {
82          DialogFragment newFragment = new TimePickerFragment();
83          newFragment.show(getSupportFragmentManager(), "timePicker");
84      }
85
86      public void filterDialog(View v){
87          DialogFragment newFragment = new FilDialogFragment();
88          newFragment.show(getSupportFragmentManager(), "filterPicker");
89      }
90
91  }
```

Listato 5.14. Codice della classe **`BottomNavActivity`**

5.1.15 La classe NothingSelectedSpinnerAdapter

La classe **`MarkerClusterRenderer`** viene mostrata nel Listato 5.15. Essa ha lo scopo di realizzare l'elemento di default di uno Spinner quando nessun elemento è ancora stato selezionato. Tale elemento non viene considerato come facente parte della lista di elementi che lo Spinner permette di scegliere; infatti, non compare come selezionabile dall'elenco e può essere impostato solo tramite codice.

```
1   public class NothingSelectedSpinnerAdapter implements SpinnerAdapter, ListAdapter {
2
3       private final static int EXTRA = 1;
4       private SpinnerAdapter adapter;
5       private Context context;
6       public int nothingSelectedLayout;
7       public int nothingSelectedDropdownLayout;
8       private LayoutInflater layoutInflater;
9
10      public NothingSelectedSpinnerAdapter(
11              SpinnerAdapter spinnerAdapter,
12              int nothingSelectedLayout, Context context) {
13
14          this(spinnerAdapter, nothingSelectedLayout, -1, context);
15      }
16
17      public NothingSelectedSpinnerAdapter(SpinnerAdapter spinnerAdapter,
18                                      int nothingSelectedLayout, int nothingSelectedDropdownLayout, Context context
                                                ) {
19          this.adapter = spinnerAdapter;
20          this.context = context;
21          this.nothingSelectedLayout = nothingSelectedLayout;
22          this.nothingSelectedDropdownLayout = nothingSelectedDropdownLayout;
23          layoutInflater = LayoutInflater.from(context);
24      }
25
26      @Override
27      public final View getView(int position, View convertView, ViewGroup parent) {
28          if (position == 0) {
29              return getNothingSelectedView(parent);
30          }
31          return adapter.getView(position - EXTRA, null, parent);
32      }
33
34      protected View getNothingSelectedView(ViewGroup parent) {
35          return layoutInflater.inflate(nothingSelectedLayout, parent, false);
36      }
37
38      @Override
39      public View getDropDownView(int position, View convertView, ViewGroup parent) {
40
```

```
41          if (position == 0) {
42              return nothingSelectedDropdownLayout == -1 ?
43                      new View(context) :
44                      getNothingSelectedDropdownView(parent);
45          }
46
47
48          return adapter.getDropDownView(position - EXTRA, null, parent);
49      }
50
51      protected View getNothingSelectedDropdownView(ViewGroup parent) {
52          return layoutInflater.inflate(nothingSelectedDropdownLayout, parent, false);
53      }
54
55  ...
56  ...
57  ...
58
59  }
```

Listato 5.15. Codice della classe **`NothingSelectedSpinnerAdapter`**

5.1.16 La classe CustomAdapter

La classe **`CustomAdapter`** viene mostrata nel Listato 5.16. Essa, e le sue classi interne, hanno lo scopo di realizzare i singoli elementi di una Recycler View di eventi pericolosi. Se guardiamo il metodo **`onBindViewHolder`**, possiamo notare la presenza dei controlli sulle informazioni che costituiscono l'evento. Tali controlli hanno lo scopo di distinguere l'origine degli eventi (ricavata da una segnalazione utente o dagli articoli di cronaca) allo scopo di modificare il layout dell'elemento della Recicler View di conseguenza. Infatti, le informazioni disponibili per gli eventi ottenuti dagli articoli sono diverse da quelle ottenute dalle segnalazioni utente; ad esempio, gli articoli possiedono l'introduzione e il titolo, a differenza delle segnalazioni utente. Un altro controllo effettuato dal metodo in esame è quello del livello di "verificato", nel caso in cui l'evento sia stato ricavato da una segnalazione utente, in modo da modificarne il layout in base a tale livello (righe 66-113).

Infine, le ultime classi interne, **`OnNoteListener`** e **`OnNoteLongListener`**, sono delle interfacce necessarie alla realizzazione di listener per un singolo tocco breve, il primo, e un tocco prolungato, il secondo, in modo tale da permettere l'esecuzione di codice specifico per queste azioni (righe 125-130).

```
1   public class CustomAdapter extends RecyclerView.Adapter<CustomAdapter.ViewHolder> {
2       private static final String TAG ="CustomAdapter" ;
3       private FirebaseAuth mAuth;
4
5       public ArrayList<String>data1,data2,data3,data4,data6;
6       public ArrayList<Integer> data5;
7       public ArrayList<Bitmap> images;
8       private Context context;
9       private OnNoteLongListener mOnNoteLongListener;
10      private OnNoteListener mOnNoteListener;
11
12      public static class ViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener, View.
            OnLongClickListener {
13          private final TextView cate, date, hour, place,desc,tdesc;
14          private final LinearLayout linearVeri,linearDes;
15          private final ImageView ima,ver,ver2,ver3;
16          OnNoteListener onNoteListener;
17          OnNoteLongListener onNoteLongListener;
18          public ViewHolder(View itemView, OnNoteListener onNoteListener,OnNoteLongListener onNoteLongListener) {
19              super(itemView);
20  ...
21  ...
22  ...
23              ima = itemView.findViewById(R.id.immaE);
24              this.onNoteListener=onNoteListener;
25              this.onNoteLongListener=onNoteLongListener;
26              itemView.setOnClickListener(this);
27              itemView.setOnLongClickListener(this);
```

```
28          }
29
30
31          @Override
32          public void onClick(View v) {
33              onNoteListener.onNoteClick(getAdapterPosition());
34          }
35
36          @Override
37          public boolean onLongClick(View v) {
38              onNoteLongListener.onNoteLongClick(getAdapterPosition());
39              return true;
40          }
41      }
42
43
44      public CustomAdapter(ArrayList cate,ArrayList date,ArrayList hour,ArrayList luogo,ArrayList img,ArrayList ve,
            OnNoteListener onNoteListener, Context cont, ArrayList desc, OnNoteLongListener onNoteLongListener) {
45          data1 = cate;
46          data2=date;
47          data3=hour;
48          data4=luogo;
49          data5=ve;
50          images=img;
51          data6=desc;
52          context=cont;
53          this.mOnNoteListener=onNoteListener;
54          this.mOnNoteLongListener=onNoteLongListener;
55      }
56
57      @Override
58      public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
59          View view = LayoutInflater.from(parent.getContext())
60                  .inflate(R.layout.text_row_item,parent, false);
61
62          return new ViewHolder(view,mOnNoteListener,mOnNoteLongListener);
63      }
64
65      @Override
66      public void onBindViewHolder(ViewHolder viewHolder, final int position) {
67
68          viewHolder.cate.setText(data1.get(position));
69  ...
70  ...
71  ...
72          if (data5.get(position)==null){
73              viewHolder.linearVeri.setVisibility(View.GONE);
74              viewHolder.desc.setText(context.getString(R.string.intrA));
75          }else {
76               viewHolder.linearVeri.setVisibility(View.VISIBLE);
77               viewHolder.desc.setText(context.getString(R.string.des));
78               if (data5.get(position)>=3) {
79                                   viewHolder.ver.setImageDrawable(context.getDrawable(R.drawable.
                                           ic_baseline_check_circle_24));
80                                   viewHolder.ver.setColorFilter(context.getResources().getColor(R.color.colorPrimary));
81                                   viewHolder.ver2.setVisibility(View.VISIBLE);
82                                   viewHolder.ver3.setVisibility(View.VISIBLE);
83                               }
84                               else {
85                                   if (data5.get(position)==2) {
86                                       viewHolder.ver.setImageDrawable(context.getDrawable(R.drawable.
                                                   ic_baseline_check_circle_24));
87                                       viewHolder.ver.setColorFilter(context.getResources().getColor(R.color.colorPrimary
                                                   ));
88                                       viewHolder.ver2.setVisibility(View.VISIBLE);
89                                       viewHolder.ver3.setVisibility(View.GONE);
90                                   }
91                                   else {
92                                       if (data5.get(position)==1) {
93                                           viewHolder.ver.setImageDrawable(context.getDrawable(R.drawable.
                                                   ic_baseline_check_circle_24));
94                                           viewHolder.ver.setColorFilter(context.getResources().getColor(R.color.
                                                   colorPrimary));
95                                           viewHolder.ver2.setVisibility(View.GONE);
96                                           viewHolder.ver3.setVisibility(View.GONE);
97                                       } else {
98                                           viewHolder.ver.setImageDrawable(context.getDrawable(R.drawable.
                                                   ic_baseline_cancel_24));
99                                           viewHolder.ver.setColorFilter(context.getResources().getColor(R.color.
                                                   colorAccent));
100                                          viewHolder.ver2.setVisibility(View.GONE);
101                                          viewHolder.ver3.setVisibility(View.GONE);
102                                      }
103                                  }
104                              }
105         }
106         if(images.size()!=0)
107             if(images.get(position)!=null) {
108
109                 viewHolder.ima.setImageBitmap(images.get(position));
110             }else{
111                 viewHolder.ima.setImageResource(R.drawable.ic_baseline_image_not_supported_24);
112             }
113     }
```

```
114
115     @Override
116     public int getItemCount() {
117         int ris=0;
118         try {
119             ris= data1.size();
120         }catch (Exception e){
121
122         }
123         return ris;
124     }
125     public interface OnNoteListener{
126         void onNoteClick(int position);
127     }
128     public interface OnNoteLongListener{
129         void onNoteLongClick(int position);
130     }
131
132 }
```

Listato 5.16. Codice della classe **`CustomAdapter`**

5.1.17 La classe CustomAdapterF

La classe **`CustomAdapterF`** viene mostrata nel Listato 6.1. Essa, e le sue classi interne, hanno lo scopo di realizzare i singoli elementi di una Recycler View delle domande e risposte della FAQ dell'app. Guardando il codice possiamo notare la somiglianza con la classe **`CustomAdapter`**; infatti, entrambe le classi realizzano delle Recycler View, solo che la **`CustomAdapterF`** è più semplice rispetto alla **`CustomAdapter`**; per tale motivo non è necessario aggiungere altro (righe 24-36).

```
1   public class CustomAdapterF extends RecyclerView.Adapter<CustomAdapterF.ViewHolder> {
2       public String[] data1,data2;
3
4       public static class ViewHolder extends RecyclerView.ViewHolder {
5           private final TextView q,a;
6   ...
7   ...
8   ...
9       }
10
11      public CustomAdapterF(String[] question, String[] answer) {
12          data1 = question;
13          data2=answer;
14      }
15
16      @Override
17      public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
18          View view = LayoutInflater.from(parent.getContext())
19                  .inflate(R.layout.text_row_faq,parent, false);
20          return new ViewHolder(view);
21      }
22
23      @Override
24      public void onBindViewHolder(ViewHolder viewHolder, final int position) {
25          viewHolder.q.setText(data1[position]);
26          viewHolder.a.setText(data2[position]);
27          viewHolder.cardQA.setOnClickListener(new View.OnClickListener() {
28
29              @Override
30              public void onClick(View v) {
31                  if (viewHolder.a.getVisibility()==View.GONE)
32                      viewHolder.a.setVisibility(View.VISIBLE);
33                  else
34                      viewHolder.a.setVisibility(View.GONE);
35              }
36          });
37      }
38
39      @Override
40      public int getItemCount() {
41          return data1.length;
42      }
43  }
```

Listato 5.17. Codice della classe **`CustomAdapterF`**

5.1.18 La classe DuplicateActivity

La classe **DuplicateActivity** viene mostrata nel Listato 5.18. Questa classe ha lo scopo di realizzare un'Activity che crea e gestisce la schermata che mostra all'utente le informazioni riguardanti l'evento simile a quello che l'utente stava cercando di inserire. Perciò, questa schermata compare soltanto a seguito del tentativo dell'utente di inserire una nuova segnalazione se, contemporaneamente, l'app, a seguito dei dovuti controlli, trova nel database uno o più eventi simili alla nuova segnalazione. In questa situazione l'utente vedrà comparire automaticamente questa schermata contenente le varie informazioni sull'evento; in fondo alla schermata è presente una domanda rivolta all'utente in cui gli viene chiesto se questo è lo stesso evento che cercava di inserire. Dopo la domanda, in fondo alla schermata, sono presenti due pulsanti, "Si" e "No". Come si vede dal codice, le informazioni riguardanti l'evento sono state fornite all'Activity dalla classe chiamante; l'unica informazione che deve essere ottenuta interrogando Firebase è l'eventuale immagine dell'evento (riga 12).

Sempre dal codice si vede che, nel caso l'evento sia stato ricavato da un articolo, l'immagine non è ottenuta tramite interrogazione del servizio di Storage di Firebase, come avviene nel caso delle segnalazioni di utenti, ma viene interrogato il servizio di Firestore che contiene l'URL dell'immagine contenuta nell'articolo (righe 25-68).

Quando l'utente preme uno dei due pulsanti in fondo alla schermata, viene terminata la classe **DuplicateActivity**, ma solo dopo aver restituito un codice di risposta alla classe chiamante in base al pulsante premuto. Per di più, nel caso venga premuto il pulsante "Si", si effettua un controllo sull'identità dell'utente per verificare se coincide con colui che aveva segnalato in precedenza l'evento descritto dalla schermata in esame; tale risultato è, poi, restituito alla classe chiamante (righe 75-108).

```
1   public class DuplicateActivity extends AppCompatActivity {
2       public static final int RESULT_YES =118 ;
3       public static final int RESULT_NO =117 ;
4   ...
5   ...
6   ...
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          overridePendingTransition(R.anim.slide_in, R.anim.fadeout);
12          Event evento = (Event) getIntent().getSerializableExtra("evento");
13  ...
14  ...
15  ...
16          yes = findViewById(R.id.yes);
17          no = findViewById(R.id.no);
18          mAuth = FirebaseAuth.getInstance();
19          FirebaseStorage storage = FirebaseStorage.getInstance();
20          StorageReference storageRef = storage.getReference();
21          StorageReference islandRef = storageRef.child("/utenti/" + evento.getPath());
22          ima = findViewById(R.id.immagineD);
23          final long ONE_MEGABYTE = 1024 * 1024;
24
25              if (!evento.getPath().contains("/")) {
26              FirebaseFirestore db = FirebaseFirestore.getInstance();
27              DocumentReference docRef = db.collection("articoli").document(evento.getPath());
28              docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
29
30                  @Override
31                  public void onComplete(@NonNull Task<DocumentSnapshot> task) {
32                      if (task.isSuccessful()) {
33                          DocumentSnapshot document = task.getResult();
34                          if (document.exists()) {
35                              RequestOptions options = new RequestOptions()
36                                      .fitCenter()
37                                      .placeholder(R.drawable.ic_baseline_add_photo_alternate_50)
38                                      .error(R.drawable.ic_baseline_add_photo_alternate_50);
39                              Glide.with(getBaseContext()).load(document.getString("URLImg")).apply(options).into(ima);
40
41                          } else {
```

```
42                          Log.d(TAG, "No such document");
43                      }
44                  } else {
45                      Log.d(TAG, "get failed with ", task.getException());
46                  }
47              }
48          });
49
50      } else {
51              if (evento.getImgPre().equals("si")){
52              islandRef.getBytes(ONE_MEGABYTE * 10).addOnCompleteListener(new OnCompleteListener<byte[]>() {
53
54                  @Override
55                  public void onComplete(@NonNull Task<byte[]> task) {
56                      if (task.isSuccessful()) {
57                          byte[] bytes = task.getResult();
58                          Bitmap bitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
59                          ima.setImageBitmap(bitmap);
60                      } else {
61
62                          Log.d(TAG, "get failed with ", task.getException());
63
64                      }
65                  }
66              });
67      }
68      }
69
70      textCate.setText(evento.getCate().replace("[","").replace("]",""));
71  ...
72  ...
73  ...
74
75      no.setOnClickListener(new View.OnClickListener() {
76          @Override
77          public void onClick(View v) {
78              try {
79
80                  Intent intent = new Intent();
81
82                  setResult(117, intent);
83                  finish();
84              } catch (Exception e) {
85              }
86          }
87      });
88
89      yes.setOnClickListener(new View.OnClickListener() {
90          @Override
91          public void onClick(View v) {
92              try {
93                  boolean diffUser=false;
94                  if (evento.getPath().contains("/")) {
95
96                      if (!evento.getuID().equals(mAuth.getCurrentUser().getUid())) {
97                          diffUser = true;
98                      }
99                  }
100                 Intent intent = new Intent();
101                 intent.putExtra("diffUser", diffUser);
102
103                 setResult(118, intent);
104                 finish();
105             } catch (Exception e) {
106             }
107         }
108     });
109     }
110 }
```

Listato 5.18. Codice della classe **`DuplicateActivity`**

5.1.19 La classe ChooseEnterActivity

La classe **`ChooseEnterActivity`** viene mostrata nel Listato 5.19. Essa ha lo scopo di realizzare un'Activity che crea e gestisce la schermata della selezione della modalità di autenticazione dell'utente, prima di poter accedere alle funzionalità dell'app. Come vediamo dal codice, alla creazione della schermata, vengono realizzati dei pulsanti e delle righe di testo, che, alla loro pressione, lanciano delle nuove Activity con lo scopo di ottenere un risultato al termine della loro esecuzione (riga 16

e 23). Poi, in base al risultato ottenuto, verranno effettuate delle operazioni, come, ad esempio, l'accesso alla schermata della mappa dopo la corretta autenticazione dell'utente.

Sempre dal codice, possiamo notare che vengano messi a disposizione degli utenti due metodi per autenticarsi usando un account esistente (righe 17-31). Il primo metodo è basato su un account Google; con esso, alla pressione del tasto corrispondente, l'utente selezionerà un account di Google che aveva già creato in precedenza, e non appena i controlli daranno esito positivo l'utente verrà direttamente indirizzato verso la prossima Activity dell'app.

Un altro modo che l'utente ha di effettuare l'accesso è premendo il pulsante di Login tramite e-mail e password; in tal caso, viene lanciata l'Activity **`EnterActivity`**, che permette all'utente di autenticarsi con le credenziali menzionate.

Un'altra opzione è quella di realizzare un nuovo account usando una e-mail e password; in tal caso verrà lanciata l'Activity **`LoginActivity`** e la schermata corrente resterà in attesa di un risultato dalla nuova classe.

L'ultima opzione a disposizione dell'utente è il reset della password; com'è possibile immaginare, questa opzione consente all'utente di reimpostare la password di un account precedentemente creato; a tal scopo, viene lanciata l'Activity **`ResetActivity`**.

Al termine di ogni Activity lanciata da questa classe, verrà analizzato il risultato restituito allo scopo di decidere quale sarà la prossima schermata a cui l'utente dovrà accedere. A questo scopo vediamo nel codice la presenza di un metodo **`onActivityResult`** che si attiva in automatico alla ricezione di una risposta da parte delle Activity lanciate (righe 43-82). Come possiamo vedere, il primo controllo viene effettuato con lo scopo di verificare che il risultato ottenuto sia un messaggio che confermi la riuscita dell'obiettivo dell'Activity lanciata, quindi nel nostro caso, che l'utente sia riuscito effettivamente ad autenticarsi correttamente tramite uno qualsiasi dei metodi visti in precedenza.

Il prossimo passo consiste, invece, nel verificare se questo sia oppure no il primo avvio dell'app su questo dispositivo. In caso di risposta affermativa, la prossima Activity che verrà lanciata sarà **`SelCiActivity`** che gli permetterà di inserire una "città principale" dalla quale partire ogni volta che si accede alla schermata della mappa. Inoltre, vengono aggiornate le Shared Preference che consentono la memorizzazione di informazioni riguardanti l'app in maniera permanente anche alla sua chiusura, in modo da memorizzare che il primo avvio dell'app sia già avvenuto. In caso di risposta negativa, l'utente verrà direttamente reindirizzato verso il Fragment **`HomeFragment`** che realizza e gestisce la schermata della mappa.

Adesso passiamo ad esporre brevemente l'operato del metodo **`writeUserToDb`**; esso permette la creazione su Firestore di un documento identificato tramite UID (User ID, cioè un codice identificativo univoco assegnato ad ogni utente registrato tramite il servizio Firebase Authentication), che contiene al proprio interno un contatore che verrà utilizzato per realizzare dei codici identificativi univoci per ogni evento che un utente inserisce (righe 119-145).

Infine, passiamo a descrivere la classe interna **`Worker`**, essa è innanzitutto un thread, cioè un sottoprocesso che esegue il proprio codice concorrentemente agli altri (righe 146-174). Lo scopo di questo thread è quello di realizzare una sincronizzazione

tra le operazioni asincrone contenute nella classe principale, come l'accesso tramite account Google, utilizzando un oggetto **`CountDownLatch`**. Esso permette di creare dei contatori che bloccano l'esecuzione di un thread fino a quando il contatore non raggiungerà il valore stabilito in fase di istanziazione. Pertanto, nel nostro caso, la parte di codice contenuta nel thread (che si occupa di scegliere la prossima schermata da avviare in base ai controlli visti in precedenza con gli altri metodi di autenticazione) viene bloccata fino a quando l'autenticazione tramite Google non darà esito positivo.

```
1   public class ChooseEnterActivity extends AppCompatActivity {
2   ...
3   ...
4   ...
5
6       @Override
7       protected void onCreate(Bundle savedInstanceState) {
8           super.onCreate(savedInstanceState);
9           setContentView(R.layout.activity_enter_o);
10          GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT\_SIGN\_IN)
11                  .requestIdToken(BuildConfig.GOO\_TOKEN+".apps.googleusercontent.com")
12                  .requestEmail()
13                  .build();
14          mGoogleSignInClient = GoogleSignIn.getClient(this, gso);
15          mAuth = FirebaseAuth.getInstance();
16          btnEnterG = findViewById(R.id.btn_registra1);
17          btnEnterG.setOnClickListener(new View.OnClickListener() {
18              @Override
19              public void onClick(View v) {
20                  signIn();
21              }
22          });
23           btnEnterT = findViewById(R.id.btn_registra2);
24           btnEnterT.setOnClickListener(new View.OnClickListener() {
25              @Override
26              public void onClick(View v) {
27
28                    Intent intent= new Intent(ChooseEnterActivity.this,EnterActivity.class);
29                    startActivityForResult(intent, LOGIN_REQUEST);
30              }
31          });
32  ...
33  ...
34  ...
35      }
36
37      private void signIn() {
38          Intent signInIntent = mGoogleSignInClient.getSignInIntent();
39          startActivityForResult(signInIntent, RC_SIGN_IN);
40      }
41
42      @Override
43      public void onActivityResult(int requestCode, int resultCode, Intent data) {
44          super.onActivityResult(requestCode, resultCode, data);
45          if (requestCode == LOGIN_REQUEST) {
46              if(resultCode == RESULT_OK) {
47
48                  mAuth = FirebaseAuth.getInstance();
49                  writeUserToDb(mAuth.getCurrentUser().getUid());
50                  SharedPreferences preferences = getSharedPreferences("login1", MODE_PRIVATE);
51                  if(preferences.getBoolean("firstrun1", true)) {
52
53                      Intent intent1 = new Intent(ChooseEnterActivity.this, SelCiActivity.class);
54                      startActivityForResult(intent1, FIRST_RUN);
55                  }
56
57                  if(!preferences.getBoolean("firstrun1", true)) {
58
59                      startActivity(new Intent(ChooseEnterActivity.this, BottomNavActivity.class));
60                      finish();
61                  }
62              }
63          }
64
65          if (requestCode == RC_SIGN_IN) {
66              CountDownLatch counter= new CountDownLatch(1);
67              Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
68              handleSignInResult(task,counter);
69              new Thread(new ChooseEnterActivity.Worker(counter)).start();
70          }
71          if(requestCode == FIRST_RUN) {
72              if (resultCode == RESULT_OK) {
73
74                  SharedPreferences preferences = getSharedPreferences("login1", MODE_PRIVATE);
75                  SharedPreferences.Editor editor = preferences.edit();
76                  editor.putBoolean("firstrun1", false);
```

```
77                editor.apply();
78                startActivity(new Intent(ChooseEnterActivity.this, BottomNavActivity.class));
79                finish();
80            }
81        }
82    }
83
84    private void handleSignInResult(Task<GoogleSignInAccount> completedTask, CountDownLatch counter) {
85
86        try {
87            GoogleSignInAccount account = completedTask.getResult(ApiException.class);
88            firebaseAuthWithGoogle(account.getIdToken(),counter);
89        } catch (ApiException e) {
90
91            Log.e(TAG, "signInResult:failed code=" + e.getStatusCode());
92            Toast.makeText(ChooseEnterActivity.this, getString(R.string.errorsignup), Toast.LENGTH_SHORT).show();
93            counter.countDown();
94        }
95    }
96    private void firebaseAuthWithGoogle(String idToken, CountDownLatch counter) {
97
98        AuthCredential credential = GoogleAuthProvider.getCredential(idToken, null);
99        mAuth.signInWithCredential(credential)
100               .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
101                   @Override
102                   public void onComplete(@NonNull Task<AuthResult> task) {
103                       if (task.isSuccessful()) {
104                           Log.d(TAG, "signInWithCredential:success");
105                           ret =true;
106                           mAuth = FirebaseAuth.getInstance();
107                           writeUserToDb(mAuth.getCurrentUser().getUid());
108                           counter.countDown();
109
110                       } else {
111
112                           Toast.makeText(ChooseEnterActivity.this, getString(R.string.errorsignup), Toast.
                                  LENGTH_SHORT).show();
113                           counter.countDown();
114                       }
115                   }
116               });
117   }
118
119   private void writeUserToDb(String uid) {
120       Map<String, Object> count = new HashMap<>();
121       count.put("count", 0);
122       FirebaseFirestore db = FirebaseFirestore.getInstance();
123       DocumentReference dr= db.collection("utenti").document(uid);
124
125       dr.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
126           @Override
127           public void onComplete(@NonNull Task<DocumentSnapshot> task) {
128               if (task.isSuccessful()) {
129                   if (task.getResult().get("count")==null) {
130                       dr.set(count).addOnCompleteListener(new OnCompleteListener<Void>() {
131                           @Override
132                           public void onComplete(@NonNull Task<Void> task) {
133                           }
134                       })
135                               .addOnFailureListener(new OnFailureListener() {
136                                   @Override
137                                   public void onFailure(@NonNull Exception e) {
138                                       Log.w(TAG, "Error writing document", e);
139                                   }
140                               });
141                   }
142               }
143           }
144       });
145   }
146       class Worker implements Runnable {
147           private CountDownLatch countDownLatch;
148           private static final String TAG ="ChooseEnterActivity" ;
149
150           public Worker( CountDownLatch countDownLatch) {
151               this.countDownLatch=countDownLatch;
152           }
153
154           @Override
155           public void run() {
156               try {
157                   countDownLatch.await();
158                   SharedPreferences preferences = getSharedPreferences("login1", MODE_PRIVATE);
159                   if (ret) {
160                       if (preferences.getBoolean("firstrun1", true)) {
161
162                           Intent intent1 = new Intent(ChooseEnterActivity.this, SelCiActivity.class);
163                           startActivityForResult(intent1, FIRST_RUN);
164                       }
165                       if (!preferences.getBoolean("firstrun1", true)) {
166                           startActivity(new Intent(ChooseEnterActivity.this, BottomNavActivity.class));
167                           finish();
168                       }
169                   }
```

```
170             } catch (Exception e) {
171                 e.printStackTrace();
172             }
173         }
174     }
175 }
```

Listato 5.19. Codice della classe **`ChooseEnterActivity`**

5.1.20 La classe FilDialogFragment

La classe **`FilDialogFragment`** viene mostrata nel Listato 5.20. Essa ha lo scopo di realizzare un Dialog per la selezione dei filtri da applicare sugli eventi visualizzati dalla schermata della mappa. Il metodo **`onTextChanged`** permette di limitare l'input dell'utente al solo inserimento di un orario valido all'interno del campo apposito del Dialog; esso fornisce, anche, una guida visuale sul formato dell'ora da inserire (righe 12-34). La guida fornita è del tipo "HH", dove la "H" indica un numero da dover inserire, per cui è sempre previsto l'inserimento di due cifre.

Il metodo **`onCreateDialog`**, invece, si occupa di creare l'interfaccia del Dialog e, in particolare, gli Spinner (cioè un menu dropdown di elementi tra cui scegliere) che permettono di scegliere la categoria dell'evento e il tipo di evento che si vuole visualizzare (righe 54-69). Vediamo, anche, che gli Spinner vengono inizializzati con i valori presenti nelle liste di Array salvate tra le risorse dell'app, a cui viene aggiunta anche un'istanza della classe **`NothingSelectedSpinnerAdapter`**, per la creazione dell'elemento zero, cioè quello presente quando nessuna opzione è ancora stata selezionata.

Un'altra cosa realizzata in fase di inizializzazione è la scrittura dei valori selezionati per i filtri che erano già stati specificati dall'utente in precedenza; per fare ciò si utilizza un'istanza della classe **`ItemViewModel`** per tenere traccia dei filtri applicati e dei valori scelti. In fondo al Dialog realizzato si trovano due pulsanti, "Applica" e "Cancella", che hanno, rispettivamente, il compito di leggere i valori selezionati per i filtri e di salvarli utilizzando un'istanza di **`ItemViewModel`**, così che possano essere utilizzati dalla classe **`HomeFragment`**, di rimuovere tutti i filtri precedentemente applicati e pulire i campi del Dialog riportandoli allo stato di partenza (righe 104-121 e 132-136).

```
1   public class FilDialogFragment extends DialogFragment {
2   ...
3   ...
4   ...
5
6       private TextWatcher twt = new TextWatcher() {
7           public String current = "";
8           public String hhmm = "HH";
9           public Calendar cal = Calendar.getInstance();
10
11          @Override
12          public void onTextChanged(CharSequence s, int start, int before, int count) {
13
14              if (!s.toString().equals(current)) {
15                     String clean = s.toString().replaceAll("[^\\d.]\\.", "");
16                     int cl = clean.length();
17                     int sel = cl;
18                     if (clean.length() < 2) {
19                         clean = clean + hhmm.substring(clean.length());
20                     } else {
21                         int hour = Integer.parseInt(clean);
22                         hour = hour > 23 ? 23 : hour;
23                         cal.set(Calendar.HOUR, hour);
24                         clean = String.format("\%02d", hour);
25
```

```
                    }
                    clean = String.format("\%s", clean);
                    sel = sel < 0 ? 0 : sel;
                    current = clean;
                    fHour.getEditText().setText(current);
                    fHour.getEditText().setSelection(sel < current.length() ? sel : current.length());

                }
            }

            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {}

            @Override
            public void afterTextChanged(Editable s) {}
        };

        @Nullable
        @Override
        public Dialog onCreateDialog(Bundle savedInstanceState) {
            Dialog dialog = new Dialog(getContext());
            crimArray = Arrays.asList(getResources().getStringArray(R.array.categorie\_array));
            typeArray= Arrays.asList(getResources().getStringArray(R.array.tipi\_array));
            viewModel = new ViewModelProvider(getActivity()).get(ItemViewModel.class);
            String[] filtri=viewModel.getSelectedItem().getValue();
    ...
    ...
    ...
            ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(getActivity(),
                    R.array.categorie\_array, android.R.layout.simple\_spinner\_item);
            ArrayAdapter<CharSequence> adapterT = ArrayAdapter.createFromResource(getActivity(),
                    R.array.tipi\_array, android.R.layout.simple\_spinner\_item);
            adapter.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);
            adapterT.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);
            fType.setAdapter(
                    new NothingSelectedSpinnerAdapter(
                            adapterT,
                            R.layout.contact\_spinner\_row\_nothing\_selected2,
                            getActivity()));
            fCate.setAdapter(
                    new NothingSelectedSpinnerAdapter(
                            adapter,
                            R.layout.contact\_spinner\_row\_nothing\_selected,
                            getActivity()));
    ...
    ...
    ...
            if (filtri!=null) {

                    fType.setSelection(typeArray.indexOf(filtri[6])+1);
    ...
    ...
    ...



            }
            dateSel.setOnClickListener(new View.OnClickListener() {

                @Override
                public void onClick(View v) {
                    MaterialDatePicker.Builder<Pair<Long, Long>> builder= MaterialDatePicker.Builder.dateRangePicker();
                    MaterialDatePicker<Pair<Long,Long>> picker = builder.build();
                    picker.show(getFragmentManager(), picker.toString());
                    picker.addOnPositiveButtonClickListener(new MaterialPickerOnPositiveButtonClickListener<Pair<Long,
                            Long>>() {

                        @Override
                        public void onPositiveButtonClick(Pair<Long, Long> selection) {
    ...
    ...
    ...
                        }
                    });
                }
            });
            fButton.setOnClickListener(new View.OnClickListener() {

                @Override
                public void onClick(View v) {
                    InputMethodManager inputMethodManager = (InputMethodManager)getActivity().getSystemService(INPUT\
                            _METHOD\_SERVICE);
                    inputMethodManager.hideSoftInputFromWindow(v.getApplicationWindowToken(),0);
                    String[] filters;
                    String type="";
                    String cate="";
                    if (fCate.getSelectedItem()!=null)
                        cate=fCate.getSelectedItem().toString();
                    if (fType.getSelectedItem()!=null)
                        type=fType.getSelectedItem().toString();
                    if (fHour.getEditText().getText().toString().contains("H")){
                        filters = new String[]{cate, fText.getEditText().getText().toString(), fdateMin.getEditText().
                                getText().toString(), fdateMax.getEditText().getText().toString(),"", "1",type};

                    }else {
```

```
117                 filters = new String[]{cate, fText.getEditText().getText().toString(), fdateMin.getEditText().
                            getText().toString(), fdateMax.getEditText().getText().toString(), fHour.getEditText().
                            getText().toString(), "1",type};
118             }
119             viewModel = new ViewModelProvider(getActivity()).get(ItemViewModel.class);
120             viewModel.selectItem(filters);
121             dialog.cancel();
122
123         }
124     });
125 ...
126 ...
127 ...
128     cancButton.setOnClickListener(new View.OnClickListener() {
129
130         @Override
131         public void onClick(View v) {
132             viewModel.selectItem(new String[]{"", "", "", "", "", "1",""});
133             fType.setSelection(0);
134 ...
135 ...
136 ...
137         }
138     });
139     dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
140     return dialog;
141   }
142 }
```

Listato 5.20. Codice della classe **`FilDialogFragment`**

5.1.21 La classe LisEveActivity

La classe **`LisEveActivity`** viene mostrata nel Listato 5.21. Essa ha lo scopo di realizzare un'Activity che crea e gestisce la schermata della lista degli eventi inseriti dall'utente. Come vediamo, il codice presente all'interno del metodo **`onCreate`** serve a recuperare le informazioni degli eventi inseriti dall'utente presenti su Firestore, e al tempo stesso, fa partire l'esecuzione del thread **`Worker`**, necessario al recupero delle immagini di thumbnail dell'evento dallo Storage di Firebase (righe 6-50 e 53-91).

Il secondo thread, cioè **`Worker2`**, si occupa di creare un'istanza di **`CustomAdapter`** con le informazioni sugli eventi, per poter creare la lista degli eventi da mostrare all'utente. Questa lista, però, per poter essere creata, deve prima possedere tutte le informazioni necessarie; per questo motivo il secondo thread, prima di poter essere eseguito, deve aspettare che il primo abbia terminato di recuperare i dati necessari (questo sempre grazie all'utilizzo del contatore **`CountDownLatch`**) (righe 93-152).

Sempre dentro il secondo thread è presente un listener che si attiva alla pressione breve di un evento della lista, ed ha lo scopo di far partire il terzo thread **`worker3`**, con l'obiettivo di recuperare e mostrare l'immagine dell'evento nella dimensione più elevata possibile per le specifiche del dispositivo (righe 153-202).

```
1   public class LisEveActivity extends AppCompatActivity {
2   ...
3   ...
4   ...
5     @Override
6      protected void onCreate(@Nullable Bundle savedInstanceState) {
7   ...
8   ...
9   ...
10        FirebaseUser currentUser = mAuth.getCurrentUser();
11        db.collection("utenti").document("dati").collection("eventi").whereEqualTo("uID",currentUser.getUid()).get
                ().addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
12            @Override
13            public void onComplete(@NonNull Task<QuerySnapshot> task) {
14                if (task.isSuccessful()) {
```

```
                        cate= new ArrayList<>();
...
...
...
                        CountDownLatch startSignal2 = new CountDownLatch(task.getResult().size());
                        for (QueryDocumentSnapshot document : task.getResult()) {
                            cate.add(document.getString("crimine"));
...
...
...
                            startSignal2.countDown();
                        }
                        CountDownLatch startSignal = new CountDownLatch(task.getResult().size());
                        for (int i=0;i<task.getResult().size();i++) {
                            try {
                                FirebaseStorage storage = FirebaseStorage.getInstance();
                                StorageReference storageRef = storage.getReference();
                                if (path.get(i)!=null) {
                                    StorageReference islandRef = storageRef.child("utenti/" + path.get(i) + "thumb
                                            ");
                                    recView = findViewById(R.id.recyclerView);
                                    new Thread(new Worker(startSignal, islandRef, i, startSignal2)).start();
                                }else {
                                    startSignal.countDown();
                                }
                                } catch (Exception e) {
                                    e.printStackTrace();

                                }
                        }
                        new Thread(new Worker2(recView, startSignal,getBaseContext())).start();
                    } else {
                        Log.d(TAG, "Error getting documents: ", task.getException());
                    }
                }
            });
    }


    class Worker implements Runnable {
...
...
...
        Worker(CountDownLatch startSignal, StorageReference islandRef,int i, CountDownLatch startSignal2) {
...
...
...
        }
        public void run() {
            try {
                startSignal2.await();
                if (imaPre.get(i).equals("si")) {
                islandRef.getBytes(1024 * 1024*10).addOnCompleteListener(new OnCompleteListener<byte[]>() {

                    @Override
                    public void onComplete(@NonNull Task<byte[]> task) {
                        if (task.isSuccessful()) {
                            byte[] bytes = task.getResult();
                            Bitmap imm=BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
                            image.set(i,imm);
                            startSignal.countDown();
                        } else {
                            image.set(i,null);
                            startSignal.countDown();
                        }

                    }
                });

            } else {
                    image.set(i,null);
                    startSignal.countDown();
                }
            } catch (Exception ex) {

            }
        }
     }

    class Worker2 implements Runnable , CustomAdapter.OnNoteListener, CustomAdapter.OnNoteLongListener {
...
...
...
    public Worker2(RecyclerView recView, CountDownLatch countDownLatch, Context context){
            this.recView = recView;
            this.context = context;
            this.countDownLatch = countDownLatch;
        }

        @Override
        public void run() {
            try {
                countDownLatch.await();

```

```
108                 CustomAdapter myAdapter= new CustomAdapter(cate, date, hour, place, image, verified,this,
                            getBaseContext(),desc,this);
109                 if(recView!=null) {
110                     runOnUiThread(new Runnable() {
111
112                         @Override
113                         public void run() {
114                             try {
115                                 recView.setAdapter(myAdapter);
116                                 recView.setLayoutManager(new LinearLayoutManager(context));
117                             }catch (Exception e){
118                                 e.printStackTrace();
119                             }
120                         }
121                     });
122                 }
123             } catch (InterruptedException e) {
124                 e.printStackTrace();
125             }
126         }
127         @Override
128         public void onNoteClick(int position) {
129            if (path.get(position)!=null) {
130                Dialog dialog = new Dialog(LisEveActivity.this);
131    ...
132    ...
133    ...
134                 new Thread(new Worker3(position)).start();
135                 close.setOnClickListener(new View.OnClickListener() {
136                    @Override
137                    public void onClick(View v) {
138                       dialog.dismiss();
139                    }
140                 });
141                 dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
142                 dialog.show();
143            } else {
144                Toast.makeText(LisEveActivity.this, getString(R.string.immaA) ,Toast.LENGTH\_SHORT).show();
145            }
146         }
147
148         @Override
149         public void onNoteLongClick(int position) {
150
151         }
152     }
153     class Worker3 implements Runnable {
154         private static final String TAG ="LiseveActivity" ;
155         private int position;
156
157     public Worker3(int position) {
158        this.position=position;
159
160     }
161
162     @Override
163     public void run() {
164         try {
165    ...
166    ...
167    ...
168             islandRef.getBytes(ONE\_MEGABYTE*10).addOnCompleteListener(new OnCompleteListener<byte[]>() {
169
170                 @Override
171                 public void onComplete(@NonNull Task<byte[]> task) {
172                     if (task.isSuccessful()) {
173
174                         byte[] bytes = task.getResult();
175                         Bitmap bitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
176                         int scaled\_height = constraintLayout.getWidth() * bitmap.getHeight() / bitmap.getWidth();
177                         FrameLayout.LayoutParams layoutParamsI = new FrameLayout.LayoutParams(constraintLayout.
                                getWidth(), scaled\_height);
178                         full.setLayoutParams(layoutParamsI);
179                         RoundedBitmapDrawable dr = RoundedBitmapDrawableFactory.create(getResources(), bitmap);
180                         dr.setCornerRadius(20);
181                         full.setImageDrawable(dr);
182                         full.setVisibility(View.VISIBLE);
183                         close.setVisibility(View.VISIBLE);
184                         progressBar.setVisibility(View.GONE);
185
186                     } else {
187
188                         FrameLayout.LayoutParams layoutParamsI = new FrameLayout.LayoutParams(constraintLayout.
                                getWidth(), constraintLayout.getWidth());
189                         full.setLayoutParams(layoutParamsI);
190                         full.setImageDrawable(getResources().getDrawable(R.drawable.ic\_baseline\_image\_not\
                                _supported\_24,null));
191                         full.setVisibility(View.VISIBLE);
192                         close.setVisibility(View.VISIBLE);
193                         progressBar.setVisibility(View.GONE);
194
195                     }
196                 }
197             });
```

```
198
199         } catch (Exception e) {
200             e.printStackTrace();
201         }
202     }
203 }
204 }
```

Listato 5.21. Codice della classe **LisEveActivity**

5.1.22 La classe LisEveMultiActivity

La classe **LisEveMultiActivity** viene mostrata nel Listato 5.22. Essa ha lo scopo di realizzare un'Activity che crea e gestisce la schermata della lista degli eventi presenti in uno stesso punto della mappa. Come vediamo dal codice, questa classe non differisce troppo dalla precedente, cioè **LisEveActivity**, com'era prevedibile, dal momento che entrambe sono state pensate per scopi simili.

La prima differenza che vediamo con la classe precedente è la presenza di un controllo all'interno del metodo **onCreate**, necessario a distinguere gli eventi ottenuti dalle segnalazioni degli utenti e quelli ottenuti dagli articoli (righe 6-53). Questa distinzione si tramuta in un codice leggermente differente per quanto riguarda la lettura delle informazioni salvate su Firestore, in quanto i documenti degli eventi salvati su Firebase hanno elementi diversi in base all'origine degli stessi.

Il primo e quarto thread della classe compiono le stesse operazioni del primo e terzo thread presenti in **LisEveActivity**; per questo motivo non le tratteremo di nuovo (righe 55-96 e 188-219).

Il secondo thread, cioè **Worker2**, è simile al secondo thread della classe **LisEveActivity** perchè crea lista degli eventi da mostrare all'utente, ma si differenzia per la presenza di due listener (righe 101-186). Il primo, che si attiva alla pressione breve di un evento della lista, ha lo scopo di far partire il quarto e quinto thread, che hanno l'obiettivo di recuperare e mostrare l'immagine dell'evento nella dimensione più elevata possibile per le specifiche del dispositivo (righe 224-283). La sola differenza tra questi due thread è che il primo viene usato per il recupero delle immagini degli eventi segnalati dagli utenti, mentre il secondo per il recupero delle immagini degli eventi ottenuti dagli articoli. Il secondo listener, invece, si attiva alla pressione prolungata di un evento; al verificarsi di questa condizione viene eseguito il codice che permette di aprire la pagina web riguardante l'articolo da cui deriva l'evento sul browser predefinito dall'utente.

Il terzo thread, invece, contiene un codice che recupera le immagini degli eventi ottenuti dagli articoli per poterle usare come immagini di thumbnail.

```
1   public class LisEveMultiActivity extends AppCompatActivity {
2   ...
3   ...
4   ...
5    @Override
6       protected void onCreate(@Nullable Bundle savedInstanceState) {
7   ...
8   ...
9   ...
10          myList = (ArrayList<String>) getIntent().getSerializableExtra("lista");
11          events = (ArrayList<Event>) getIntent().getSerializableExtra("eventi");
12          imaUrlList = (ArrayList<String>) getIntent().getSerializableExtra("immaUrl");
13          setContentView(R.layout.activity\_lisevemulti);
14  ...
15  ...
16  ...
```

```
17
18
19          CountDownLatch startSignal = new CountDownLatch(myList.size());
20          recView = findViewById(R.id.recyclerView1);
21          cate = new String[myList.size()];
22    ...
23    ...
24    ...
25          int count=0;
26          for (Event evento: events){
27
28              if(evento.getPath().contains("-")) {
29                  cate[count]=evento.getCate();
30    ...
31    ...
32    ...
33                  StorageReference islandRef = storageRef.child("utenti/" + evento.getPath().replace("-","/") + "thumb
                          ");
34                  new Thread(new LisEveMultiActivity.Worker(startSignal, islandRef,evento.getImgPre(),count)).start();
35
36              }else {
37                  String[] split=evento.getDate().split(" ");
38                  cate[count]=evento.getCate();
39    ...
40    ...
41    ...
42                  URL url = null;
43                  try {
44                      url = new URL(evento.getImgUrl());
45                  } catch (MalformedURLException e) {
46                      e.printStackTrace();
47                  }
48                  new Thread(new Worker3(startSignal,url , count)).start();
49              }
50              count++;
51          }
52          new Thread(new Worker2(recView, startSignal,getBaseContext(),this)).start();
53      }
54
55      class Worker implements Runnable {
56    ...
57    ...
58    ...
59
60          Worker(CountDownLatch startSignal, StorageReference islandRef,String imaPre,int i) {
61    ...
62    ...
63    ...
64          }
65          public void run() {
66              try {
67                  if (imaPre.equals("si")) {
68                      Log.d(TAG, islandRef.toString());
69                      islandRef.getBytes(1024 * 1024).addOnCompleteListener(new OnCompleteListener<byte[]>() {
70
71                          @Override
72                          public void onComplete(@NonNull Task<byte[]> task) {
73                              if (task.isSuccessful()) {
74                                  byte[] bytes = task.getResult();
75                                  images[i]=BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
76                                  image.add(BitmapFactory.decodeByteArray(bytes, 0, bytes.length)) ;
77                                  startSignal.countDown();
78
79
80                              } else {
81                                  images[i]=null;
82                                  image.add(null);
83                                  startSignal.countDown();
84                              }
85
86                          }
87                      });
88
89                  } else {
90                      images[i]=null;
91                      image.add(null);
92                      startSignal.countDown();
93                  }
94              } catch (Exception ex) {}
95
96          }
97
98
99      }
100
101       class Worker2 implements Runnable, CustomAdapter.OnNoteListener, CustomAdapter.OnNoteLongListener {
102   ...
103   ...
104   ...
105         public Worker2(RecyclerView recView, CountDownLatch countDownLatch, Context context, LisEveMultiActivity
                  cla ) {
106   ...
107   ...
108   ...
```

```
109
110         }
111
112         @Override
113         public void run() {
114             try {
115                 countDownLatch.await();
116                 ArrayList<String> ca = new ArrayList<>();
117 ...
118 ...
119 ...
120                 ArrayList<Integer> ve = new ArrayList<>();
121                 Collections.addAll(ve, verified);
122                 ArrayList<Bitmap> im = new ArrayList<>();
123                 Collections.addAll(im, images);
124                 CustomAdapter myAdapter= new CustomAdapter(ca,da,ora,luo,im,ve,this,getBaseContext(),des,this);
125                 if(recView!=null) {
126                     runOnUiThread(new Runnable() {
127
128                         @Override
129                         public void run() {
130                             try {
131                                 recView.setAdapter(myAdapter);
132                                 recView.setLayoutManager(new LinearLayoutManager(context));
133                             }catch (Exception e){
134                                 e.printStackTrace();
135                             }
136                         }
137                     });
138                 }
139             } catch (Exception e) {
140                 e.printStackTrace();
141             }
142         }
143
144         @Override
145         public void onNoteClick(int position) {
146             if (images[position]!=null) {
147             dialog.getWindow().getAttributes().windowAnimations = R.style.PauseDialogAnimation;
148             dialog.setContentView(R.layout.image_popup);
149             full = dialog.findViewById(R.id.full_image);
150             constraintLayout=dialog.findViewById(R.id.constraintLayoutP);
151             progressBar=dialog.findViewById(R.id.progressBar);
152                 close = dialog.findViewById(R.id.close_image);
153                 close.setOnClickListener(new View.OnClickListener() {
154                     @Override
155                     public void onClick(View v) {
156                         dialog.dismiss();
157                     }
158                 });
159                 if (imaUrlList.get(position)==null)
160                     new Thread(new Worker4(cla.myList.get(position))).start();
161                 else {
162
163
164                     FutureTarget<Bitmap> futureTarget =
165                             Glide.with(LisEveMultiActivity.this)
166                                     .asBitmap()
167                                     .load(imaUrlList.get(position))
168                                     .submit();
169                     new Thread(new Worker5(futureTarget)).start();
170                 }
171                 dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
172                 dialog.show();
173
174             } else {
175                 Toast.makeText(LisEveMultiActivity.this, getString(R.string.immaA) ,Toast.LENGTH\_SHORT).show();
176             }
177         }
178
179         @Override
180         public void onNoteLongClick(int position) {
181             if (cla.urlA[position]!=null) {
182                 Intent browserIntent = new Intent(Intent.ACTION\_VIEW, Uri.parse(cla.urlA[position]));
183                 startActivity(browserIntent);
184             }
185         }
186     }
187
188     class Worker3 implements Runnable {
189         private CountDownLatch startSignal;
190         private static final String TAG ="LiseveMultipliActivity" ;
191         public int i;
192         public URL url;
193
194         public Worker3(CountDownLatch startSignal, URL url, int i ) {
195             this.url = url;
196             this.i= i;
197             this.startSignal = startSignal;
198
199
200         }
201
202         @Override
```

```
203         public void run() {
204             try {
205                 Bitmap image = BitmapFactory.decodeStream(url.openConnection().getInputStream());
206                 images[i]=image;
207                 LisEveMultiActivity.this.image.add(image);
208                 startSignal.countDown();
209
210
211             } catch (Exception e) {
212                 e.printStackTrace();
213                 images[i]=null;
214                 LisEveMultiActivity.this.image.add(null);
215                 startSignal.countDown();
216             }
217         }
218
219     }
220 ...
221 ...
222 ...
223
224     class Worker5 implements Runnable {
225
226         private static final String TAG ="LiseveMultipliActivity" ;
227         public Bitmap thumb;
228         private FutureTarget<Bitmap> futureTarget;
229
230         public Worker5(FutureTarget<Bitmap> futureTarget ) {
231             this.futureTarget=futureTarget;
232
233         }
234
235         @Override
236         public void run() {
237             try {
238                 thumb = futureTarget.get();
239                 try {
240                     Thread.sleep(100);
241                 } catch (InterruptedException e) {
242                     e.printStackTrace();
243                 }
244                 int scaled_height = constraintLayout.getWidth() * thumb.getHeight() / thumb.getWidth();
245                 FrameLayout.LayoutParams layoutParamsI = new FrameLayout.LayoutParams(constraintLayout.getWidth(),
                            scaled_height);
246
247                 LisEveMultiActivity.this.runOnUiThread(new Runnable() {
248                     @Override
249                     public void run() {
250                         try {
251                             full.setLayoutParams(layoutParamsI);
252                             RoundedBitmapDrawable dr = RoundedBitmapDrawableFactory.create(getResources(), thumb);
253                             dr.setCornerRadius(20);
254                             full.setImageDrawable(dr);
255                             full.setVisibility(View.VISIBLE);
256                             close.setVisibility(View.VISIBLE);
257                             progressBar.setVisibility(View.GONE);
258                         }catch (Exception e){
259                             e.printStackTrace();
260                         }
261                     }
262                 });
263             } catch (ExecutionException  InterruptedException e) {
264
265                 LisEveMultiActivity.this.runOnUiThread(new Runnable() {
266                     @Override
267                     public void run() {
268                         try {
269                             FrameLayout.LayoutParams layoutParamsI = new FrameLayout.LayoutParams(constraintLayout.
                                        getWidth(), constraintLayout.getWidth());
270                             full.setLayoutParams(layoutParamsI);
271                             full.setImageDrawable(getResources().getDrawable(R.drawable.ic\_baseline\_image\_not\
                                        _supported\_24, null));
272                             full.setVisibility(View.VISIBLE);
273                             close.setVisibility(View.VISIBLE);
274                             progressBar.setVisibility(View.GONE);
275                         }catch (Exception e){
276                             e.printStackTrace();
277                         }
278                     }
279                 });
280                 e.printStackTrace();
281             }
282         }
283     }
284 }
```

Listato 5.22. Codice della classe **`LisEveMultiActivity`**

5.1.23 La classe OptionsFragment

La classe **`OptionsFragment`** viene mostrata nel Listato 5.23. Essa ha lo scopo di realizzare un Fragment che crea e gestisce la schermata delle opzioni dell'app. Guardando il codice si possono distinguere due parti fondamentali. La prima riguarda il metodo **`onViewCreated`** che definisce la schermata e i suoi elementi, compresi i pulsanti associati alle opzioni disponibili per l'utente; alla pressione di ciascuno di questi pulsanti viene eseguito il codice che si occupa di lanciare le rispettive Activity associate alle varie opzioni (righe 6-89).

La seconda parte, invece, è costituita da tutti i restanti metodi, che hanno lo scopo di permettere all'utente di selezionare un'immagine di profilo dalla galleria delle immagini del dispositivo e di mostrarla nella cornice dedicata, posizionata a fianco del nome dell'utente (righe 142-237). Alla creazione della schermata si può notare una chiamata al servizio di Storage di Firebase per verificare la presenza di una immagine di profilo già impostata per l'account in uso; nel caso non fosse presente, un placeholder viene caricato al suo posto nel frame dedicato. Da sottolineare è il fatto che, nel momento in cui l'utente preme sulla cornice dell'immagine di profilo, per poter scegliere l'immagine da impostare tra quelle salvate sul dispositivo, gli verrà chiesto, se è la prima volta, di consentire all'app di leggere e scrivere sulla memoria interna; in caso di rifiuto non è possibile proseguire con l'operazione.

Un'ultima menzione va all'unico thread presente nella classe; esso si occupa di memorizzare l'immagine scelta come una thumbnail sul servizio Storage di Firebase solo e soltanto quando l'immagine è stata correttamente recuperata dalla galleria, in modo tale che possa essere recuperata in futuro (righe 90-139).

```
1   public class OptionsFragment extends Fragment {
2   ...
3   ...
4   ...
5       @Override
6       public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
7           proPic = getView().findViewById(R.id.propic);
8           textName = getView().findViewById(R.id.text_nome);
9           log= getView().findViewById(R.id.card5);
10          log.setOnClickListener(new View.OnClickListener() {
11
12              @Override
13              public void onClick(View v) {
14                  FirebaseAuth.getInstance().signOut();
15                  Intent intent = new Intent(getActivity(), ChooseEnterActivity.class);
16                  startActivity(intent);
17              }
18          });
19  ...
20  ...
21  ...
22          textName.setText(currentUser.getDisplayName());
23          FirebaseStorage storage = FirebaseStorage.getInstance();
24          StorageReference storageRef = storage.getReference();
25          StorageReference islandRef = storageRef.child("/utenti/"+currentUser.getUid()+ "/pic");
26
27          islandRef.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
28
29                 @Override
30                 public void onSuccess(Uri uri) {
31                     try {
32                         getActivity().runOnUiThread(new Runnable() {
33
34                             @Override
35                             public void run() {
36                                 try {
37                                     Glide.with(getActivity()).load(uri).apply(RequestOptions.circleCropTransform()).
                                              error(getActivity().getDrawable(R.drawable.placeholder\_profile)).into(
                                              proPic);
38
39                                 }catch (Exception e){
40                                     e.printStackTrace();
41                                 }
42                             }
43                         });
```

```
                }catch (NullPointerException e){


                }
            }
        }).addOnFailureListener(new OnFailureListener() {

            @Override
            public void onFailure(@NonNull Exception exception) {
                try {
                    Glide.with(getActivity()).load(getActivity().getDrawable(R.drawable.placeholder\_profile)).
                            apply(RequestOptions.circleCropTransform()).into(proPic);

                }catch (Exception e){
                    e.printStackTrace();

                }
            }
        });

    proPic.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            permissions.add(Manifest.permission.WRITE\_EXTERNAL\_STORAGE);
            permissions.add(Manifest.permission.READ\_EXTERNAL\_STORAGE);
            permissionsToRequest = findUnaskedPermissions(permissions);
            if (permissionsToRequest.size() > 0) {
                requestPermissions(permissionsToRequest.toArray(new String[permissionsToRequest.size()]), ALL\
                        _PERMISSIONS\_RESULT);
            } else {
                startActivityForResult(getPickImageChooserIntent(), PICK\_IMAGE);
            }
        }
    });
}

private ArrayList findUnaskedPermissions(ArrayList<String> wanted) {
    ArrayList<String> result = new ArrayList<>();

    for (String perm : wanted) {
        if (!(ActivityCompat.checkSelfPermission(getContext(),perm) == PackageManager.PERMISSION\_GRANTED)) {
            result.add(perm);
        }
    }

    return result;
}
class Worker implements Runnable {

    private static final String TAG ="LiseveActivity" ;
    public Bitmap thumb;
    private FutureTarget<Bitmap> futureTarget;
    public Worker(FutureTarget<Bitmap> futureTarget ) {
        this.futureTarget=futureTarget;
    }

    @Override
    public void run() {
        try {
            try {
                thumb = futureTarget.get();
            } catch (ExecutionException  InterruptedException e) {
                e.printStackTrace();
            }
            FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
            FirebaseStorage storage = FirebaseStorage.getInstance();
            StorageReference storageRef = storage.getReference();
            ByteArrayOutputStream baos1 = new ByteArrayOutputStream();
            thumb.compress(Bitmap.CompressFormat.JPEG, 50, baos1);
            byte[] data1 = baos1.toByteArray();
            StorageReference eventImagesRef1 = storageRef.child("utenti/" + user.getUid() + "/pic");
            UploadTask uploadTask1 = eventImagesRef1.putBytes(data1);
            uploadTask1.addOnFailureListener(new OnFailureListener() {

                @Override
                public void onFailure(@NonNull Exception exception) {
                    Toast.makeText(getActivity(), getString(R.string.pron), Toast.LENGTH\_SHORT).show();
                }
            }).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {

                @Override
                public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                    uploadTask1.addOnCompleteListener(new OnCompleteListener<UploadTask.TaskSnapshot>() {

                        @Override
                        public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {

                            Toast.makeText(getActivity(), getString(R.string.proa), Toast.LENGTH\_SHORT).show();
                        }
                    });
                }
            });
        }catch (Exception e){
```

```
136                 e.printStackTrace();
137             }
138         }
139     }
140 
141     @Override
142     public void onActivityResult(int requestCode, int resultCode, Intent intent) {
143         super.onActivityResult(requestCode, resultCode, intent);
144         if (requestCode == PICK\_IMAGE) {
145             if (resultCode == RESULT\_OK) {
146                 if (getPickImageResultUri(intent) != null) {
147                     Uri picUri = getPickImageResultUri(intent);
148                     FutureTarget<Bitmap> futureTarget =
149                             Glide.with(getContext())
150                                     .asBitmap()
151                                     .load(picUri)
152                                     .submit(THUMBSIZE, THUMBSIZE);
153 
154                     Glide.with(getContext()).load(picUri).apply(RequestOptions.circleCropTransform()).into(proPic);
155                     new Thread(new OptionsFragment.Worker(futureTarget)).start();
156                 } else {
157                 }
158             }
159         }
160     }
161 
162     private Uri getPickImageResultUri(Intent data) {
163         boolean isCamera = true;
164         if (data != null) {
165             String action = data.getAction();
166             isCamera = action != null \&\& action.equals(MediaStore.ACTION\_IMAGE\_CAPTURE);
167         }
168 
169         return isCamera ? getCaptureImageOutputUri() : data.getData();
170 
171     }
172 
173     private Intent getPickImageChooserIntent() {
174 
175         Uri outputFileUri = getCaptureImageOutputUri();
176         List<Intent> allIntents = new ArrayList<>();
177         PackageManager packageManager = getActivity().getPackageManager();
178         Intent captureIntent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);
179         List<ResolveInfo> listCam = packageManager.queryIntentActivities(captureIntent, 0);
180         for (ResolveInfo res : listCam) {
181             Intent intent = new Intent(captureIntent);
182             intent.setComponent(new ComponentName(res.activityInfo.packageName, res.activityInfo.name));
183             intent.setPackage(res.activityInfo.packageName);
184             if (outputFileUri != null) {
185                 intent.putExtra(MediaStore.EXTRA\_OUTPUT, outputFileUri);
186             }
187         }
188         String action;
189         action = Intent.ACTION\_PICK;
190         Intent galleryIntent = new Intent(action, MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI);
191         List<ResolveInfo> listGallery = packageManager.queryIntentActivities(galleryIntent, 0);
192         for(ResolveInfo res : listGallery) {
193             Intent intent = new Intent(galleryIntent);
194             intent.setComponent(new ComponentName(res.activityInfo.packageName, res.activityInfo.name));
195             intent.setPackage(res.activityInfo.packageName);
196             allIntents.add(intent);
197         }
198 
199         Intent mainIntent = allIntents.get(allIntents.size()-1);
200         for(Intent intent : allIntents) {
201             if(intent.getComponent().getClassName().equals("com.android.documentsui.DocumentsActivity")) {
202                 mainIntent = intent;
203                 break;
204             }
205         }
206         allIntents.remove(mainIntent);
207         Intent chooserIntent = Intent.createChooser(mainIntent, getString(R.string.sorg));
208         chooserIntent.putExtra(Intent.EXTRA\_INITIAL\_INTENTS, allIntents.toArray(new Parcelable[allIntents.size()
                ]));
209         return chooserIntent;
210     }
211 
212     public Uri getCaptureImageOutputUri() {
213         Uri outputFileUri = null;
214         File getImage = getActivity().getExternalCacheDir();
215         if(getImage != null) {
216             outputFileUri = Uri.fromFile(new File(getImage.getPath(), "propic.png"));
217         }
218         return outputFileUri;
219     }
220 
221     public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
222         if(requestCode == ALL\_PERMISSIONS\_RESULT) {
223             for(String perm: permissionsToRequest) {
224                 if(!(ActivityCompat.checkSelfPermission(getContext(),perm)==PackageManager.PERMISSION\_GRANTED)) {
225                     permissionsRejected.add(perm);
226                 }
227             }
228             if(permissionsRejected.size() > 0) {
```

```
229                 if(shouldShowRequestPermissionRationale(permissionsRejected.get(0))) {
230                     Toast.makeText(getActivity(),getString(R.string.appr), Toast.LENGTH\_SHORT).show();
231                 }
232             }
233             else {
234                 startActivityForResult(getPickImageChooserIntent(), PICK\_IMAGE);
235             }
236         }
237     }
238
239 }
```

Listato 5.23. Codice della classe **`OptionsFragment`**

5.1.24 La classe SelCiActivity

La classe **`SelCiActivity`** viene mostrata nel Listato 5.24. Essa ha lo scopo di realizzare un'Activity che crea e gestisce la schermata della selezione della "città principale" (cioè la città che verrà visualizzata ogni volta che si passa alla schermata della mappa). Come vediamo dal codice, la schermata è composta da pochi elementi: un campo di testo in cui l'utente può inserire il nome della città scelta, un pulsante di conferma e un pulsante per poter utilizzare il GPS del dispositivo, in modo da individuare automaticamente la città in cui esso si trova (il risultato verrà inserito nel campo apposito).

I metodi **`getGpsLocation`** e **`confirmControl`** sono simili tra loro; infatti, il primo viene eseguito alla pressione del tasto del GPS con lo scopo di identificare il nome della città in cui il dispositivo si trova in quel momento (righe 35-121 e 124-205). L'utilizzo del GPS, però, restituisce le coordinate geografiche del dispositivo per cui, per trovare il nome della città, è necessario utilizzare un servizio di geocoding tramite le API fornite da Mapquest Developer (tutte le operazioni di geocoding o reverse geocoding da qui in avanti faranno uso di queste API) che permettono di ottenere l'indirizzo completo delle coordinate fornite. A partire dall'indirizzo, viene recuperato soltanto il nome della città che verrà, poi, inserito nel campo ad esso dedicato presente nella schermata. Vale la pena specificare che esistono dei controlli riguardanti la posizione della città restituita dal GPS; in particolare, viene controllato che la città sia all'interno dei confini dello stato italiano; in caso contrario un messaggio di errore verrà restituito all'utente. Inoltre, alla prima pressione del tasto di GPS, verrà richiesto all'utente di fornire il permesso di utilizzare il GPS durante il funzionamento dell'app; per di più, nel caso in cui il GPS sia disattivato sul dispositivo, comparirà un Dialog che chiederà il permesso di attivarlo in automatico; chiaramente se uno dei due permessi viene a mancare, verrà restituito all'utente un messaggio d'errore.

Il secondo metodo, **`confirmControl`**, si occupa, invece, di realizzare un'operazione di reverse geocoding alla pressione del tasto di conferma. Il reverse geocoding, come intuibile dal nome, consiste nell'operazione inversa al geocoding, ossia ottenere le coordinate geografiche di un punto a partire dal suo indirizzo. Come nel caso precedente, anche il reverse geocoding viene realizzato tramite l'utilizzo di API fornite da Mapquest Developer. Anche in questo caso vengono effettuati dei controlli sul campo della città inserita; ad esempio, si verificherà che il campo non sia vuoto oppure che il nome della città non sia troppo corto per essere identificata o, ancora, che sia all'interno dell'Italia. Il risultato ottenuto da questo metodo viene, poi, salvato tramite le Shared Preference, che abbiamo già introdotto in precedenza, allo

scopo di rendere permanente questa informazione, in modo che sia accessibile ad ogni avvio dell'app.

Vediamo, infine, l'ultimo metodo della classe, cioè **sanitizationURL**; esso si occupa della sanificazione del contenuto del campo della città in modo tale da impedire ad un utente malintenzionato di inserire qualsiasi carattere non permesso, prevenendo così una qualsiasi forma di attacco di iniezione di codice (righe 222-228).

```
1   public class SelCiActivity extends AppCompatActivity {
2
3   ...
4   ...
5   ...
6       private ActivityResultLauncher<String> requestPermissionLauncher =
7               registerForActivityResult(new ActivityResultContracts.RequestPermission(), isGranted -> {
8                   if (isGranted) {
9                       Toast.makeText(SelCiActivity.this, getString(R.string.perc), Toast.LENGTH_SHORT).show();
10                  } else {
11                      Toast.makeText(SelCiActivity.this, getString(R.string.pern), Toast.LENGTH_SHORT).show();
12                  }
13              });
14
15      @Override
16      protected void onCreate(Bundle savedInstanceState) {
17          super.onCreate(savedInstanceState);
18          overridePendingTransition(R.anim.slide_in, R.anim.fadeout);
19          setContentView(R.layout.activity_selci);
20          mAuth = FirebaseAuth.getInstance();
21          textCit = findViewById(R.id.textInputLayoutSelCi);
22          textCit.setOnKeyListener(new View.OnKeyListener() {
23              public boolean onKey(View v, int keyCode, KeyEvent event) {
24
25                  if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
26                          (keyCode == KeyEvent.KEYCODE_ENTER)) {
27
28                      return true;
29                  }
30                  return false;
31              }
32          });
33      }
34
35      public void getGpsLocation(View v) {
36          RequestQueue queue = Volley.newRequestQueue(this);
37
38                  if (ActivityCompat.checkSelfPermission(SelCiActivity.this, Manifest.permission.ACCESS_FINE_LOCATION)
                            != PackageManager.PERMISSION_GRANTED) {
39                      requestPermissionLauncher.launch(
40                              Manifest.permission.ACCESS_FINE_LOCATION);
41                  }else {
42                      final LocationRequest locationRequest = LocationRequest.create();
43                      locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
44                      LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder()
45                              .addLocationRequest(locationRequest);
46                      SettingsClient client = LocationServices.getSettingsClient(SelCiActivity.this);
47                      Task<LocationSettingsResponse> task = client.checkLocationSettings(builder.build());
48
49
50                      task.addOnSuccessListener(SelCiActivity.this, new OnSuccessListener<LocationSettingsResponse>()
                                {
51
52                          @SuppressLint("MissingPermission")
53                          @Override
54                          public void onSuccess(LocationSettingsResponse locationSettingsResponse) {
55                              fusedLocationClient = LocationServices.getFusedLocationProviderClient(SelCiActivity.this)
                                        ;
56                              fusedLocationClient.getCurrentLocation(LocationRequest.PRIORITY_HIGH_ACCURACY, null).
                                        addOnSuccessListener(SelCiActivity.this, new OnSuccessListener<Location>() {
57
58                                  @Override
59                                  public void onSuccess(Location location) {
60                                      if (location != null) {
61
62                                          String urlStr = "https://www.mapquestapi.com/geocoding/v1/reverse?key=" +
                                                    BuildConfig.GEO_API_KEY + "&location=" + location.getLatitude() + "\%2C
                                                    " + location.getLongitude() + "&outFormat=json&thumbMaps=false&
                                                    maxResults=1";
63                                          StringRequest stringRequest = new StringRequest(Request.Method.GET, urlStr,
64                                                  new Response.Listener<String>() {
65                                                      @Override
66                                                      public void onResponse(String response) {
67                                                          String jsonString = response;
68                                                          try {
69                                                              JSONObject obj = new JSONObject(jsonString);
70                                                              JSONArray arr = obj.getJSONArray("results");
71                                                              String city = arr.getJSONObject(0).getJSONArray("locations")
                                                                        .getJSONObject(0).getString("adminArea5");
72                                                              String country = arr.getJSONObject(0).getJSONArray("
```

```
                                                locations").getJSONObject(0).getString("adminArea1");
73                                          if (!country.isEmpty() && !country.equals("null")) {
74                                              if (country.equals("IT")) {
75                                                  textCit.getEditText().setText(city);
76                                                  Toast.makeText(SelCiActivity.this, getString(R.string
                                                            .post), Toast.LENGTH_SHORT).show();
77                                              } else
78                                                  Toast.makeText(SelCiActivity.this, getString(R.string
                                                            .posf), Toast.LENGTH_SHORT).show();
79                                          } else
80                                              Toast.makeText(SelCiActivity.this, getString(R.string.
                                                        posf), Toast.LENGTH_SHORT).show();
81
82
83                                          } catch (JSONException e) {
84
85                                              e.printStackTrace();
86                                              Toast.makeText(SelCiActivity.this, getString(R.string.prop),
                                                        Toast.LENGTH_SHORT).show();
87                                          }
88
89                                      }
90                                      }, new Response.ErrorListener() {
91                                  @Override
92                                  public void onErrorResponse(VolleyError error) {
93                                      Toast.makeText(SelCiActivity.this, getString(R.string.conn), Toast.
                                                LENGTH_SHORT).show();
94                                  }
95                              });
96                              queue.add(stringRequest);
97                          } else
98                              Toast.makeText(SelCiActivity.this, getString(R.string.posn), Toast.
                                    LENGTH_SHORT).show();
99                      }
100                 });
101             }
102         });
103         task.addOnFailureListener(SelCiActivity.this, new OnFailureListener() {
104             @Override
105             public void onFailure(@NonNull Exception e) {
106                 e.printStackTrace();
107                 if (e instanceof ResolvableApiException) {
108
109                     try {
110
111                         ResolvableApiException resolvable = (ResolvableApiException) e;
112                         resolvable.startResolutionForResult(SelCiActivity.this,
113                                 REQUEST_CHECK_SETTINGS);
114                     } catch (IntentSender.SendIntentException sendEx) {
115
116                     }
117                 }
118             }
119         });
120     }
121 }
122
123 @SuppressLint({"CutPasteId", "WrongViewCast"})
124 public void confirmControl(View v) {
125     try {
126         InputMethodManager inputMethodManager = (InputMethodManager)getSystemService(INPUT_METHOD_SERVICE);
127         inputMethodManager.hideSoftInputFromWindow(v.getApplicationWindowToken(),0);
128         textCit = findViewById(R.id.textInputLayoutSelCi);
129         String place = String.valueOf(Objects.requireNonNull(textCit.getEditText()).getText());
130
131         if (place.length()>=3) {
132
133                 RequestQueue queue = Volley.newRequestQueue(this);
134                 String urlStr = "https://www.mapquestapi.com/geocoding/v1/address?key="+BuildConfig.GEO_API_KEY+
                            "&inFormat=kvp&outFormat=json&location=" + sanitizationURL(textCit.getEditText().getText
                            ().toString()) + "&outFormat=json&thumbMaps=false&maxResults=5";
135                 StringRequest stringRequest = new StringRequest(Request.Method.GET, urlStr.replace(",", "\%2C").
                            replace(" ", "+"),
136                         new Response.Listener<String>() {
137                             @Override
138                             public void onResponse(String response) {
139                                 double maxA=0;
140                                 ArrayList<Integer> indexMaxA= new ArrayList<>();
141                                 String jsonString = response;
142                                 DecimalFormat dec = new DecimalFormat("#0.000000");
143                                 try {
144                                     JSONObject obj = new JSONObject(jsonString);
145                                     JSONArray arr = obj.getJSONArray("results").getJSONObject(0).getJSONArray("
                                                locations");
146                                     ArrayList<Double> acc = new ArrayList<>();
147                                     ArrayList<String> data = new ArrayList<>();
148
149                                     for (int i = 0; i < arr.length(); i++) {
150                                         String label = arr.getJSONObject(i).getString("street");
151                                         String ci = arr.getJSONObject(i).getString("adminArea5");
152                                         String pa = arr.getJSONObject(i).getString("adminArea1");
153                                         if (((!label.isEmpty() && !label.equals("null"))  (!ci.isEmpty() && !ci.
                                                    equals("null"))) && pa.equals("IT")) {
154                                             double distance = StringUtils.getJaroWinklerDistance(label+", "+ci,
```

```
                                        textCit.getEditText().getText());
155                                 if(maxA<=distance)
156                                     maxA=distance;
157                                 acc.add(distance);
158                                 data.add(dec.format(Double.valueOf(arr.getJSONObject(i).getJSONObject("
                                        latLng").getString("lat"))) + ";" + dec.format(Double.valueOf(
                                        arr.getJSONObject(i).getJSONObject("latLng").getString("lng")))
                                        );
159                             }
160                         }
161
162                         for (int i=0;i<acc.size();i++){
163                             if(maxA==acc.get(i))
164                                 indexMaxA.add(i);
165                         }
166                         if (indexMaxA.size() == 1) {
167                             SharedPreferences preferences = getSharedPreferences("citta", MODE_PRIVATE)
                                    ;
168                             String[] coord = data.get(0).split(";");
169                             SharedPreferences.Editor editor = preferences.edit();
170
171                             editor.putString("latitudine",coord[0]);
172                             editor.putString("longitudine",coord[1]);
173                             editor.commit();
174
175                             Intent intent = new Intent();
176                             setResult(RESULT_OK, intent);
177                             finish();
178                         }
179                         if (indexMaxA.size()> 1)
180                             Toast.makeText(SelCiActivity.this, getString(R.string.cinc), Toast.
                                    LENGTH_SHORT).show();
181                         if (indexMaxA.isEmpty())
182                             Toast.makeText(SelCiActivity.this, getString(R.string.cint), Toast.
                                    LENGTH_SHORT).show();
183                     } catch (JSONException e) {
184                         Log.i(TAG, response );
185                         Toast.makeText(SelCiActivity.this, getString(R.string.cint), Toast.
                                LENGTH_SHORT).show();
186                     }
187
188                 }
189             }, new Response.ErrorListener() {
190             @Override
191             public void onErrorResponse(VolleyError error) {
192                 Log.d(TAG, "onErrorResponse: " + error.getMessage());
193                 Toast.makeText(SelCiActivity.this, getString(R.string.connr), Toast.LENGTH_SHORT).show();
194             }
195         });
196         queue.add(stringRequest);
197
198     }
199     else
200         Toast.makeText(SelCiActivity.this, getString(R.string.citc), Toast.LENGTH_SHORT).show();
201
202 }catch (NullPointerException  NumberFormatException e){
203     Toast.makeText(SelCiActivity.this, getString(R.string.comco), Toast.LENGTH\_SHORT).show();
204 }
205 }
206
207 protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
208     super.onActivityResult(requestCode, resultCode, intent);
209     if(requestCode==REQUEST\_CHECK\_SETTINGS){
210
211         if(resultCode==RESULT\_OK){
212
213             Toast.makeText(this, getString(R.string.gps), Toast.LENGTH\_SHORT).show();
214             Log.d("result ok",intent.toString());
215
216         }else if(resultCode==RESULT\_CANCELED){
217             Log.d("result cancelled",intent.toString());
218         }
219     }
220 }
221
222 private String sanitizationURL(String urlU){
223
224     urlU= urlU.replaceAll("[^[0-9][a-zA-Z],\\-]"," ");
225     urlU=urlU.replace(",", "\%2C").replace(" ", "+");
226
227     return urlU;
228 }
229
230 }
```

Listato 5.24. Codice della classe **`SelCiActivity`**

5.1.25 La classe NewEntryFragment

La classe **NewEntryFragment** ha lo scopo di realizzare un Fragment che crea e gestisce la schermata di inserimento di nuove segnalazioni da parte dell'utente. Come mostrato nel Listato 5.25, possiamo notare che il codice contenuto nel metodo **onViewCreated** serve a creare gli elementi della schermata in esame; in particolare, possiamo notare il codice necessario a gestire i pulsanti presenti nella schermata.

Tra questi pulsanti notiamo quello dedicato al GPS, che è simile a quello presente nella classe **SelCiActivity**, ed ha lo stesso scopo. A differenza del precedente, però, questo permette di ottenere l'indirizzo completo della posizione del dispositivo. L'altro importante pulsante presente nella schermata è quello che permette di confermare l'inserimento del nuovo evento.

Alla pressione di questo pulsante di conferma, vengono effettuati dei controlli sulle informazioni inserite per verificare la presenza di tutte le informazioni obbligatorie; successivamente, superati i controlli, si ricercano le coordinate dell'indirizzo inserito, utilizzando le API di reverse geocoding già viste in precedenza (righe 8-68).

Il passo successivo consiste nella ricerca di eventuali eventi simili a quello che l'utente sta cercando di inserire; a tal scopo, vengono effettuate due ricerche su Firestore, una per cercare tra le segnalazioni degli utenti e l'altra per cercare tra quelle ricavate dagli articoli (righe 69-198). Le ricerche vengono effettuate per trovare eventi simili che hanno la stessa categoria, la stessa data ed un orario che si discosti di al massimo 30 minuti (prima o dopo) rispetto all'orario dell'evento che si sta cercando di inserire; in più, si verifica che i due eventi si trovino all'interno di un raggio di 1 Km dal luogo della segnalazione. Una volta ottenuti i risultati di queste interrogazioni, si creerà una coda contenente tutte le informazioni riguardanti tali eventi; a tal fine ciò si utilizzano delle istanze della classe **Event** di cui abbiamo già discusso in questo capitolo. Se la coda contiene almeno un elemento, viene estratto quello in cima ad essa per poi lanciare l'Activity **DuplicateActivity** con in più le informazioni sull'evento estratto.

```
1    @Override
2       public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
3           date = getView().findViewById(R.id.editTextDate);
4           date.addTextChangedListener(twd);
5    ...
6    ...
7    ...
8           conf.setOnClickListener(new View.OnClickListener() {
9
10              @SuppressLint("MissingPermission")
11              @Override
12              public void onClick(View v) {
13
14                  try {
15                      InputMethodManager inputMethodManager = (InputMethodManager)getActivity().getSystemService(
                                INPUT_METHOD_SERVICE);
16                      inputMethodManager.hideSoftInputFromWindow(v.getApplicationWindowToken(),0);
17                      conf = getView().findViewById(R.id.button4);
18   ...
19   ...
20   ...
21                      if (NewEntryFragment.this.cate.equals("")  place.equals("")  dateS.equals("")  timeS.equals(""))
                                {
22                          viewModel.selectItem(new String[]{"0"});
23                          conf.setClickable(true);
24                          Toast.makeText(getActivity(), getString(R.string.comco), Toast.LENGTH_SHORT).show();
25                      } else {
26                              RequestQueue queue = Volley.newRequestQueue(getActivity());
27                              String urlStr = "https://www.mapquestapi.com/geocoding/v1/address?key="+BuildConfig.
                                        GEO_API_KEY+"&inFormat=kvp&outFormat=json&location=" + sanitizationURL(tPlace.
                                        getEditText().getText().toString()) + "&outFormat=json&thumbMaps=false&maxResults
                                        =5";
28                              StringRequest stringRequest = new StringRequest(Request.Method.GET, urlStr, //url.
                                        toString(),
29                                      new Response.Listener<String>() {
```

```
30                                  @Override
31                                  public void onResponse(String response) {
32                                      double maxA = 0;
33                                      ArrayList<Integer> indexMaxA = new ArrayList<>();
34                                      String jsonString = response;
35                                      DecimalFormat dec = new DecimalFormat("#0.000000");
36                                      try {
37                                          JSONObject obj = new JSONObject(jsonString);
38                                          JSONArray arr = obj.getJSONArray("results").getJSONObject(0).
                                                getJSONArray("locations");
39                                          ArrayList<Double> acc = new ArrayList<>();
40                                          ArrayList<String> data = new ArrayList<>();
41  ...                                     
42  ...                                     
43  ...                                     
44                                          for (int i = 0; i < acc.size(); i++) {
45                                              if (maxA == acc.get(i))
46                                                  indexMaxA.add(i);
47                                          }
48                                          if (indexMaxA.size() == 1) {
49                                              
50                                              mAuth = FirebaseAuth.getInstance();
51                                              FirebaseUser currentUser = mAuth.getCurrentUser();
52                                              coor = data.get(indexMaxA.get(0)).split(";");
53                                              FirebaseFirestore db = FirebaseFirestore.getInstance();
54                                              CountDownLatch countDownLatch;
55                                              try {
56                                                  countDownLatch = new CountDownLatch(2);
57                                                  GeoLocation center = new GeoLocation(Double.parseDouble(coor[0].
                                                        replace(",", ".")), Double.parseDouble(coor[1].replace("
                                                        ,", ".")));
58                                                  final List<Task<QuerySnapshot>> tasks = new ArrayList<>();
59                                                      try {
60                                                          Query q = db.collection("utenti").document("dati").
                                                                collection("eventi")
61                                                              .orderBy("lat")
62                                                              .whereGreaterThanOrEqualTo("lat", (center.latitude -
                                                                    (latitude1)))
63                                                              .whereLessThanOrEqualTo("lat", (center.latitude + (
                                                                    latitude1)));
64                                                      tasks.add(q.get());
65                                                  
66                                                  } catch (Exception e) {
67                                                      e.printStackTrace();
68                                                  }
69                                                  Tasks.whenAllComplete(tasks)
70                                                          .addOnCompleteListener(new OnCompleteListener<List<Task
                                                                <?>>>() {
71                                                              
72                                                              @RequiresApi(api = Build.VERSION_CODES.N)
73                                                              @Override
74                                                              public void onComplete(@NonNull Task<List<Task<?>>> t
                                                                    ) {
75                                                                  
76                                                                  for (Task<QuerySnapshot> task : tasks) {
77                                                                      QuerySnapshot snap = task.getResult();
78                                                                      for (DocumentSnapshot doc : snap.getDocuments
                                                                            ()) {
79                                                                          
80                                                                          DecimalFormat dec = new DecimalFormat("
                                                                                #0.000000");
81                                                                          double lat = doc.getDouble("lat");
82                                                                          double lng = doc.getDouble("lng");
83                                                                          Calendar cale = Calendar.getInstance();
84                                                                          SimpleDateFormat sdf = new
                                                                                SimpleDateFormat("dd/MM/yyyy HH:mm"
                                                                                , Locale.ITALY);
85                                                                          
86                                                                          try {
87                                                                              cale.setTime(sdf.parse(doc.getString("
                                                                                    dataora")));
88                                                                              Timestamp time = new Timestamp(cale.
                                                                                    getTime());
89                                                                          
90                                                                              if (lat >= (Double.parseDouble(coor[0].
                                                                                    replace(",", ".")) - latitude1)
                                                                                     && lng >= (Double.parseDouble(
                                                                                    coor[1].replace(",", ".")) -
                                                                                    longitude1 ) && lat <= (Double.
                                                                                    parseDouble(coor[0].replace(","
                                                                                    , ".")) + latitude1) && lng <=
                                                                                    (Double.parseDouble(coor[1].
                                                                                    replace(",", ".")) + longitude1
                                                                                    )) {
91                                                                                  Calendar calendar = Calendar.
                                                                                        getInstance();
92                                                                                  calendar.setTime(time.toDate());
93                                                                                  String categ = doc.getString("
                                                                                        crimine");
94  ...                                                                             
95  ...                                                                             
96  ...                                                                             
97                                                                                  Calendar cal = Calendar.getInstance
                                                                                        ();
```

```
98                                                  Calendar calendarP = Calendar.
                                                           getInstance();
99                                                  Calendar calendarM = Calendar.
                                                           getInstance();
100                                                 try {
101                                                     calendarP.setTime(time.toDate()
                                                            );
102                                                     calendarM.setTime(time.toDate()
                                                            );
103                                                     cal.setTime(sdf.parse(dateS + "
                                                             " + timeS));
104                                                     calendarP.add(Calendar.MINUTE,
                                                            30);
105                                                     calendarM.add(Calendar.MINUTE,
                                                            -30);
106                                                   if (categ.equals(
                                                            NewEntryFragment.this.
                                                            cate) && (calendarM.
                                                            before(cal)  calendarM.
                                                            equals(cal) ) && (
                                                            calendarP.after(cal)
                                                            calendarP.equals(cal))) {
107                                                     String latLng = dec.format(
                                                             lat).replace(",", ".
                                                             ") + ";" + dec.
                                                             format(lng).replace(
                                                             ",", ".");
108                                                     Event eve;
109                                                     eve = new Event(latLng, doc.
                                                             getId().replace("-",
                                                              "/"));
110                                                     eve.setImgPre(doc.getString(
                                                             "immagine"));
111 ...
112 ...
113 ...
114                                                     stack.push(eve);
115
116                                                 }
117                                                 } catch (Exception e) {
118                                                     e.printStackTrace();
119                                                 }
120                                             }
121                                         } catch (ParseException e) {
122                                             e.printStackTrace();
123                                         }
124                                     }
125                                 }
126                                 countDownLatch.countDown();
127                             }
128                         });
129
130                         longitude1 = (1 / (111.320 * Math.cos(Math.toRadians(Double.
                                parseDouble(coor[1].replace(",", "."))))));
131                         final List<Task<QuerySnapshot>> tasksA = new ArrayList<>();
132                         Query q = db.collection("articoli")
133                                 .orderBy("Lat")
134                                 .whereGreaterThanOrEqualTo("Lat", (center.latitude - (
                                        latitude1)))
135                                 .whereLessThanOrEqualTo("Lat", (center.latitude + (
                                        latitude1))); tasksA.add(q.get());
136                         Tasks.whenAllComplete(tasksA)
137                                 .addOnCompleteListener(new OnCompleteListener<List<Task
                                        <?>>>() {
138
139                                     @RequiresApi(api = Build.VERSION_CODES.N)
140                                     @Override
141                                     public void onComplete(@NonNull Task<List<Task<?>>> t
                                            ) {
142                                         List<DocumentSnapshot> matchingDocs = new
                                                ArrayList<>();
143                                         for (Task<QuerySnapshot> task : tasksA) {
144                                             QuerySnapshot snap = task.getResult();
145                                             for (DocumentSnapshot doc : snap.getDocuments
                                                    ()) {
146                                                 DecimalFormat dec = new DecimalFormat("
                                                        #0.000000");
147                                                 double lat = doc.getDouble("Lat");
148                                                 double lng = doc.getDouble("Lon");
149                                                 try {
150
151                                                     Date time = new SimpleDateFormat("yyyy-
                                                            MM-dd HH:mm:ss").parse(doc.
                                                            getString("DataArticolo"));
152                                                     Calendar calendar = Calendar.
                                                            getInstance();
153                                                     if (lat >= (Double.parseDouble(coor[0].
                                                            replace(",", ".")) - latitude1)
                                                             && lng >= (Double.parseDouble(
                                                            coor[1].replace(",", ".")) -
                                                            longitude1) && lat <= (Double.
                                                            parseDouble(coor[0].replace(","
                                                            , ".")) + latitude1) && lng <=
                                                            (Double.parseDouble(coor[1].
```

```
                                                                    replace(",", ".")) + longitude1
                                                                    )) {
154                                                                 calendar.setTime(time);
155                                                                 String categ = doc.getString("
                                                                            Categoria").toLowerCase();
156                                                                 String pl = doc.getString("Luogo");
157                                                                 String da = calendar.get(Calendar.
                                                                            DAY_OF_MONTH) + "/" + (
                                                                            calendar.get(Calendar.MONTH
                                                                            ) + 1) + "/" + calendar.get
                                                                            (Calendar.YEAR);
158                                                                 String ho = calendar.get(Calendar.
                                                                            HOUR_OF_DAY) + ":" +
                                                                            calendar.get(Calendar.
                                                                            MINUTE);
159                                                                 String des = doc.getString("
                                                                            TitoloArticolo");
160                                                                 Calendar cal = Calendar.getInstance
                                                                            ();
161                                                                 Calendar calendarP = Calendar.
                                                                            getInstance();
162                                                                 Calendar calendarM = Calendar.
                                                                            getInstance();
163                                                                 SimpleDateFormat sdf = new
                                                                            SimpleDateFormat("dd/MM/
                                                                            yyyy HH:mm", Locale.ITALY);
164                                                                 calendarP.setTime(time);
165                                                                 calendarM.setTime(time);
166                                                                 try {
167                                                                    cal.setTime(sdf.parse(dateS + "
                                                                               " + timeS));
168                                                                    calendarP.add(Calendar.MINUTE,
                                                                               30);
169                                                                    calendarM.add(Calendar.MINUTE,
                                                                               -30);
170
171                                                                    if (categ.equals(
                                                                               NewEntryFragment.this.
                                                                               cate) && (calendarM.
                                                                               before(cal)  calendarM.
                                                                               equals(cal)) && (
                                                                               calendarP.after(cal)
                                                                               calendarP.equals(cal)))
                                                                               {
172                                                                       String latLng = dec.format(
                                                                                  lat).replace(",", ".
                                                                                  ") + ";" + dec.
                                                                                  format(lng).replace(
                                                                                  ",", ".");
173                                                                       Event eve;
174                                                                       eve = new Event(latLng, doc.
                                                                                  getId());
175                                                                       eve.setCate(categ.toString()
                                                                                  );
176    ...
177    ...
178    ...
179
180                                                                       stack.push(eve);
181                                                                    }
182                                                                 } catch (ParseException e) {
183                                                                    e.printStackTrace();
184                                                                 }
185                                                              }
186                                                           } catch (ParseException e) {
187                                                              e.printStackTrace();
188                                                           }
189
190                                                        }
191                                                     }
192                                                     countDownLatch.countDown();
193                                                  }
194                                               });
195                                            new Thread(new NewEntryFragment.Worker(currentUser,
                                                      countDownLatch)).start();
196                                         } catch (Exception e) {
197                                            e.printStackTrace();
198                                         }
199                                      }
200    ...
201    ...
202    ...
203                                   } catch (JSONException e) {
204                                      viewModel.selectItem(new String[]{"0"});
205                                      conf.setClickable(true);
206                                      Log.i(TAG, e.toString());
207                                      Toast.makeText(getActivity(), getString(R.string.luont), Toast.
                                                LENGTH_SHORT).show();
208                                   }
209
210                                }
211                             }, new Response.ErrorListener() {
212
213                             @Override
                                public void onErrorResponse(VolleyError error) {
```

```
214                                 viewModel.selectItem(new String[]{"0"});
215                                 conf.setClickable(true);
216                                 Toast.makeText(getActivity(), getString(R.string.connr), Toast.LENGTH_SHORT).show
                                          ();
217                             }
218                         });
219                         queue.add(stringRequest);
220                     }
221                 } catch (NullPointerException  NumberFormatException e) {
222                     conf.setClickable(true);
223                     Toast.makeText(getActivity(), getString(R.string.comco), Toast.LENGTH\_SHORT).show();
224                 }
225             }
226         });
227     ...
228     ...
229     ...
230     }
```

Listato 5.25. Estratto della prima parte della classe **`NewEntryFragment`**

Come mostrato nel Listato 5.26, possiamo notare che il codice contenuto nel metodo **`onActivityResult`** serve a gestire l'inserimento dell'immagine selezionata dall'utente dalla galleria del dispositivo, nel frame dedicato all'immagine da allegare all'evento che si sta cercando di inserire (righe 5-38).

Il restante codice del metodo serve a gestire le diverse risposte date dall'utente nel caso in cui vengano individuati degli eventi simili a quello che egli stava cercando di inserire; ad esempio, in caso di risposta affermativa, viene incrementato il valore di "verificato" dell'evento che l'utente conferma essere uguale alla sua segnalazione, ma solo se l'utente artefice di questa segnalazione è diverso dall'utente attuale (righe 40-100). Il nuovo evento non viene aggiunto al database remoto per evitare duplicati nel caso di risposta affermativa, e non verranno mostrati all'utente gli eventuali altri eventi simili trovati. In caso di risposta negativa, invece, verrà mostrato all'utente il prossimo evento simile della coda; se la coda è vuota, allora il nuovo evento verrà inserito all'interno del database come una segnalazione diversa da tutte le altre. Per inserire il nuovo evento viene richiamato il metodo **`writeEventInfoToDb`** che descriveremo successivamente.

```
1    public void onActivityResult(int requestCode, int resultCode, Intent intent) {
2           super.onActivityResult(requestCode, resultCode, intent);
3           mAuth = FirebaseAuth.getInstance();
4           FirebaseUser currentUser= mAuth.getCurrentUser();
5           if(requestCode==REQUEST_CHECK_SETTINGS){
6
7               if(resultCode==RESULT_OK){
8
9                   Toast.makeText(getActivity(), getString(R.string.gps), Toast.LENGTH_SHORT).show();
10                  Log.d("result ok",intent.toString());
11
12              }else if(resultCode==RESULT_CANCELED){
13
14                  Log.d("result cancelled",intent.toString());
15              }
16          }
17          if(requestCode==PICK_IMAGE) {
18              float fileSizeInMB=0;
19              if(resultCode == RESULT_OK) {
20                  if(getPickImageResultUri(intent) != null) {
21                      picUri = getPickImageResultUri(intent);
22                      File file = new File(picUri.getLastPathSegment());
23                      float fileSizeInBytes = file.length();
24                      Log.d(TAG, "file size in B "+fileSizeInBytes);
25                      float fileSizeInKB = fileSizeInBytes / 1024;
26                      fileSizeInMB = fileSizeInKB / 1024;
27                  }
28              }
29              proPic = getView().findViewById(R.id.imageViewIns);
30              if (fileSizeInMB<=10) {
31                  Glide.with(this)
32                          .load(picUri)
33                          .into(proPic);
34              }
35              else
36                  Toast.makeText(getActivity(), getString(R.string.immaT), Toast.LENGTH_SHORT).show();
```

```
37
38          }
39
40          if(requestCode==LOGIN_REQUEST) {
41              if(resultCode==118){
42                 if ( intent.getBooleanExtra("diffUser",false)) {
43                     if (eveC.getPath().contains("/")) {
44                         String[] co = eveC.getPath().split("/");
45                         FirebaseFirestore db = FirebaseFirestore.getInstance();
46                         DocumentReference washingtonRef = db.collection("utenti").document("dati/eventi/" + co[0] + "
                                   -" + co[1]);
47                         washingtonRef
48                                .update("verificato", FieldValue.increment(1))
49                                .addOnSuccessListener(new OnSuccessListener<Void>() {
50                                   @Override
51                                   public void onSuccess(Void aVoid) {
52                                       ItemViewModel viewModel = new ViewModelProvider(requireActivity()).get(
                                                ItemViewModel.class);
53                                       viewModel.selectItem(new String[]{"0"});
54                                       Log.d(TAG, "DocumentSnapshot successfully updated! " +co[0]+"-"+co[1]);
55                                   }
56                               })
57                               .addOnFailureListener(new OnFailureListener() {
58                                   @Override
59                                   public void onFailure(@NonNull Exception e) {
60                                       conf.setClickable(true);
61                                       ItemViewModel viewModel = new ViewModelProvider(requireActivity()).get(
                                                ItemViewModel.class);
62                                       viewModel.selectItem(new String[]{"0"});
63                                       Log.w(TAG, "Error updating document", e);
64                                   }
65                               });
66                     }else {
67                         conf.setClickable(true);
68                         ItemViewModel viewModel = new ViewModelProvider(requireActivity()).get(ItemViewModel.class);
69                         viewModel.selectItem(new String[]{"0"});
70                     }
71                 }else {
72                     conf.setClickable(true);
73                     ItemViewModel viewModel = new ViewModelProvider(requireActivity()).get(ItemViewModel.class);
74                     viewModel.selectItem(new String[]{"0"});
75                 }
76
77
78
79              }
80
81              if(resultCode==117){
82
83                  if (!stack.empty()){
84                      Event eveS = stack.pop();
85                      eveC=eveS;
86                      Intent intent1 = new Intent(getContext(), DuplicateActivity.class);
87                      intent1.putExtra("evento", eveS);
88                      startActivityForResult(intent1, LOGIN_REQUEST);
89                  }else {
90                      conf.setClickable(true);
91                      if (proPic.getDrawable()==null) {
92                          writeEventInfoToDb(cate, dateS, timeS, tPlace.getEditText().getText().toString(), coor[0].
                                    replace(",", "."), coor[1].replace(",", "."), desc, "no", currentUser.getUid());
93                      }else {
94                          writeEventInfoToDb(cate, dateS, timeS, tPlace.getEditText().getText().toString(), coor[0].
                                    replace(",", "."), coor[1].replace(",", "."), desc, "si", currentUser.getUid());
95                      }
96                  }
97
98              }
99
100         }
101     }
```

Listato 5.26. Estratto della seconda parte della classe **`NewEntryFragment`**

Il prossimo metodo che analizzeremo è **`writeEventInfoToDb`**. Come mostrato nel Listato 5.27, possiamo notare che il codice contenuto nel metodo serve a realizzare l'inserimento delle informazioni relative all'evento che si vuole aggiungere (righe 1-81). Il metodo gestisce l'inserimento delle informazioni testuali che compongono la segnalazione tramite una chiamata al servizio Firestore di Firebase; l'inserimento dell'immagine è, invece, affidata al secondo thread, il quale verrà lanciato solo al completamento delle operazioni di scrittura su Firestore.

L'inserimento delle immagini avviene, perciò, grazie al secondo thread della classe, cioè **`Worker2`**, il quale si occupa della creazione e dell'inserimento dell'immagine

di thumbnail ottenuta a partire dall'immagine allegata all'articolo (righe 119-221). Oltre alla thumbnail anche l'immagine originale (opportunamente compressa tramite conversione in un JPEG) viene inserita sullo Storage di Firebase, insieme a tutte le altre immagini degli eventi inseriti dall'utente.

Il primo thread, invece, si occupa di richiamare il metodo **`writeEventInfoToDb`** ogni volta che c'è la necessità di inserire un nuovo evento nel database, oppure lancia una nuova istanza di **`DuplicateActivity`** contenente le informazioni sul prossimo elemento estratto dalla coda di eventi simili, nel caso in cui quest'ultima non sia ancora terminata (righe 83-117). Il primo thread viene richiamato dopo la ricerca di eventi simili nel database, come si vede nel Listato 5.25.

```
1     private void writeEventInfoToDb(String cate, String date,String hour,String place,String lati,String lon,String
          desc,String ima, String uid) {
2         Map<String, Object> user = new HashMap<>();
3   ...
4   ...
5   ...
6         try {
7             cal.setTime(sdf.parse(date+" "+hour));
8             user.put("dataora", date+" "+hour);
9   ...
10  ...
11  ...
12        } catch (ParseException e) {
13            e.printStackTrace();
14        }
15
16        FirebaseFirestore db = FirebaseFirestore.getInstance();
17        DocumentReference dr= db.collection("utenti").document("dati");
18        DocumentReference drc= db.collection("utenti").document(uid);
19        drc.update("count", FieldValue.increment(1));
20        drc.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
21
22            @Override
23            public void onComplete(@NonNull Task<DocumentSnapshot> task) {
24                try {
25                    if (task.isSuccessful()) {
26                        DocumentSnapshot document = task.getResult();
27                        if (document.exists()) {
28                            String value = document.get("count").toString();
29                            if (picUri != null) {
30                                FutureTarget<Bitmap> futureTarget =
31                                        Glide.with(getContext())
32                                                .asBitmap()
33                                                .load(picUri)
34                                                .submit();
35                                new Thread(new Worker2(futureTarget, uid, value)).start();
36                            }else {
37                                ItemViewModel viewModel = new ViewModelProvider(getActivity()).get(ItemViewModel.
                                        class);
38                                viewModel.selectItem(new String[]{"0"});
39                            }
40                            dr.collection("eventi").document(uid + "-" + value).set(user).addOnSuccessListener(new
                                    OnSuccessListener<Void>() {
41
42                                @Override
43                                public void onSuccess(Void aVoid) {
44                                    spinner.setSelection(0);
45                                    tPlace.getEditText().setText("");
46                                    time.setText("");
47                                    NewEntryFragment.this.date.setText("");
48                                    des.setText("");
49                                    bitmap = null;
50                                    if (picUri==null)
51                                        conf.setClickable(true);
52                                    Toast.makeText(getContext(), getString(R.string.insc), Toast.LENGTH_SHORT).show();
53                                    Log.d(TAG, "DocumentSnapshot successfully written!");
54                                }
55                            })
56                                    .addOnFailureListener(new OnFailureListener() {
57                                        @Override
58                                        public void onFailure(@NonNull Exception e) {
59                                            if (picUri==null) {
60                                                ItemViewModel viewModel = new ViewModelProvider(requireActivity()).get(
                                                        ItemViewModel.class);
61                                                viewModel.selectItem(new String[]{"0"});
62                                                conf.setClickable(true);
63                                            }
64                                            Log.w(TAG, "Error writing document", e);
65                                        }
66                                    });
67
68                        } else {
```

```
69
70                          Log.d(TAG, "No such document");
71                      }
72                  } else {
73
74                      Log.d(TAG, "get failed with ", task.getException());
75                  }
76              }catch (Exception e){
77                  e.printStackTrace();
78              }
79          }
80      });
81  }
82
83  class Worker implements Runnable {
84  ...
85  ...
86  ...
87
88      public Worker(FirebaseUser currentUser , CountDownLatch countDownLatch) {
89          this.currentUser=currentUser;
90          this.countDownLatch=countDownLatch;
91
92      }
93
94      @Override
95      public void run() {
96          try {
97              countDownLatch.await();
98              if(stack.empty()){
99                  if (proPic.getDrawable()==null) {
100                     writeEventInfoToDb(cate, dateS, timeS, tPlace.getEditText().getText().toString(), coor[0].
                            replace(",", "."), coor[1].replace(",", "."), desc, "no", currentUser.getUid());
101                 }else {
102                     writeEventInfoToDb(cate, dateS, timeS, tPlace.getEditText().getText().toString(), coor[0].
                            replace(",", "."), coor[1].replace(",", "."), desc, "si", currentUser.getUid());
103                 }
104             }else {
105                 Event eveS= stack.pop();
106                 eveC=eveS;
107                 Intent intent = new Intent(getContext(), DuplicateActivity.class);
108                 intent.putExtra("evento", eveS);
109                 startActivityForResult(intent, LOGIN_REQUEST);
110             }
111
112         } catch (Exception e) {
113             e.printStackTrace();
114         }
115     }
116
117 }
118
119 class Worker2 implements Runnable {
120
121     private static final String TAG ="NewEntryFragment" ;
122     public String uid, value;
123     private FutureTarget<Bitmap> futureTarget;
124
125     public Worker2(FutureTarget<Bitmap> futureTarget, String uid, String value) {
126         this.futureTarget=futureTarget;
127         this.uid=uid;
128         this.value=value;
129
130
131     }
132
133     @Override
134     public void run() {
135         try {
136             try {
137                 bitmap = futureTarget.get();
138             } catch (ExecutionException e) {
139                 e.printStackTrace();
140             } catch (InterruptedException e) {
141                 e.printStackTrace();
142             }
143             FirebaseStorage storage = FirebaseStorage.getInstance();
144             StorageReference storageRef = storage.getReference();
145             ByteArrayOutputStream baos = new ByteArrayOutputStream();
146             ByteArrayOutputStream baos1 = new ByteArrayOutputStream();
147             Bitmap thumb = ThumbnailUtils.extractThumbnail(bitmap,
148                     THUMBSIZE, THUMBSIZE);
149             bitmap.compress(Bitmap.CompressFormat.JPEG, 50, baos);
150             thumb.compress(Bitmap.CompressFormat.JPEG, 100, baos1);
151             byte[] data = baos.toByteArray();
152             byte[] data1 = baos1.toByteArray();
153             StorageReference eventImagesRef = storageRef.child("utenti/" + uid + "/" + value);
154             StorageReference eventImagesRef1 = storageRef.child("utenti/" + uid + "/" + value + "thumb");
155             UploadTask uploadTask = eventImagesRef.putBytes(data);
156             UploadTask uploadTask1 = eventImagesRef1.putBytes(data1);
157             uploadTask.addOnFailureListener(new OnFailureListener() {
158
159                 @Override
160                 public void onFailure(@NonNull Exception exception) {
```

```
161                     ItemViewModel viewModel = new ViewModelProvider(requireActivity()).get(ItemViewModel.class);
162                     viewModel.selectItem(new String[]{"0"});
163                     conf.setClickable(true);
164                 }
165             }).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
166
167                 @Override
168                 public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
169                     uploadTask.addOnCompleteListener(new OnCompleteListener<UploadTask.TaskSnapshot>() {
170
171                         @Override
172                         public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {
173 ...
174 ...
175 ...
176                             try {
177                                 ItemViewModel viewModel = new ViewModelProvider(requireActivity()).get(
                                            ItemViewModel.class);
178                                 viewModel.selectItem(new String[]{"0"});
179                             }catch (Exception e){
180                                 e.printStackTrace();
181                             }
182                             picUri = null;
183                         }
184                     });
185                 }
186             });
187             uploadTask1.addOnFailureListener(new OnFailureListener() {
188                 @Override
189                 public void onFailure(@NonNull Exception exception) {
190                 }
191             }).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
192                 @Override
193                 public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
194                     uploadTask1.addOnCompleteListener(new OnCompleteListener<UploadTask.TaskSnapshot>() {
195                         @Override
196                         public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {
197 ...
198 ...
199 ...
200                         }
201                     });
202                 }
203             });
204         }catch (Exception e){
205             ItemViewModel viewModel = new ViewModelProvider(requireActivity()).get(ItemViewModel.class);
206             viewModel.selectItem(new String[]{"0"});
207             conf.setClickable(true);
208             e.printStackTrace();
209         }
210     }
211 }
```

Listato 5.27. Estratto della terza parte della classe **`NewEntryFragment`**

5.1.26 La classe HomeFragment

La prossima classe che analizzeremo è **`HomeFragment`**. Questa ha lo scopo di realizzare un'Activity che crea e gestisce la schermata della mappa degli eventi. In particolare, essa permette di realizzare un'interfaccia costituita dalla mappa geografica dell'Italia e dagli eventi presenti nell'area geografica attualmente visualizzata dall'utente; questi eventi vengono rappresentati sulla mappa tramite marker (in pratica delle icone che rappresentano il luogo esatto in cui l'evento è accaduto) sparsi per la mappa.

La parte di codice che analizzeremo adesso è quella contenuta nel Listato 5.28. Il metodo che analizzeremo è chiamato **`getInfoContents`**, questo è il metodo che realizza e gestisce le Info Window (delle finestre contenenti le informazioni dell'evento collegato al marker corrispondente) che compaiono quando un marker presente sulla mappa viene premuto dall'utente. Sempre guardando il codice, notiamo che l'aspetto e il contenuto delle informazioni presenti in queste Info Window varia in base all'origine dell'evento o alla presenza di più eventi presenti in uno stesso punto geografico. Questo metodo, infatti, viene richiamato automaticamente alla pressione

di un marker ed ha a disposizione le informazioni relative al marker premuto (riga 2).

Il primo controllo che viene effettuato riguarda la ricerca di eventi multipli presenti nello stesso punto (righe 8-16). Esso è necessario per gestire la presenza di più eventi collocati nello stesso identico luogo; per far ciò viene effettuata una ricerca sulla lista contenente tutti gli eventi attualmente posizionati sulla mappa in modo da trovare la presenza di altri eventi con le stesse coordinate del marker appena premuto. In caso si trovino più eventi nello stesso punto, la Info Window contiene un messaggio che informa l'utente della presenza di eventi multipli e ne specifica la quantità; poi, in caso l'utente decida di voler visualizzare questa lista di eventi, premerà la finestra (righe 148-163). A quel punto il codice farà partire l'Activity **`LisEveMultiActivity`** che abbiamo già trattato.

Se, invece, nel punto risulta presente un solo evento, gli elementi della finestra varieranno in base ad esso (righe 26-146). Infatti, se negli eventi ottenuti dalle segnalazioni di utenti è presente il campo "verificato", lo stesso non si può dire di quelli ricavati dagli articoli che, a loro volta, possiedono il titolo e l'introduzione dell'articolo che dovranno essere mostrati all'utente.

Un'altra caratteristica di queste finestre è che, nel caso in cui l'utente le prema, verrà mostrata in sovraimpressione l'immagine dell'evento (se presente) nella massima dimensione consentita. Nel caso in cui, invece, l'evento sia stato ricavato da un articolo, alla pressione prolungata della finestra, verrà aperto sul browser predefinito l'articolo originale dell'evento.

```
1   @Override
2           public View getInfoContents(Marker marker) {
3               DecimalFormat dec = new DecimalFormat("#0.000000");
4               ArrayList<String> dupliP= new ArrayList<String>();
5               ArrayList<String> dupliI= new ArrayList<String>();
6               ArrayList<Event> dupliE= new ArrayList<Event>();
7               ArrayList<Integer> dupliV= new ArrayList<>();
8               for (Event evento: listE) {
9                   if (evento.getCoor().equals(dec.format(marker.getPosition().latitude).replace(",",".")+";"+dec.
                            format(marker.getPosition().longitude).replace(",","."))) {
10                      dupliP.add(evento.getPath());
11                      if(evento.getVeri()!=null)
12                          dupliV.add(evento.getVeri());
13                      dupliE.add(evento);
14                      dupliI.add(evento.getImgUrl());
15                  }
16              }
17
18              clusterManager.setOnClusterItemInfoWindowLongClickListener(new ClusterManager.
                        OnClusterItemInfoWindowLongClickListener() {
19              @Override
20                  public void onClusterItemInfoWindowLongClick(ClusterItem item) {
21
22                  }
23              });
24
25              if (dupliP.size() < 2) {
26              if(marker.getTitle().contains("articolo")){
27                  myContentsView = LayoutInflater.from(getActivity()).inflate(R.layout.custom_info_contents_article,
                            null);
28                  TextView tvTitle = ((TextView)myContentsView.findViewById(R.id.titleA));
29                  tvTitle.setText(marker.getTitle());
30                  TextView tvSnippet = ((TextView)myContentsView.findViewById(R.id.snippetA));
31                  tvSnippet.setText(marker.getSnippet());
32                  String latLng =dec.format(marker.getPosition().latitude).replace(",",".")+";"+dec.format(marker.
                            getPosition().longitude).replace(",",".");
33                  String key= perAList.getListA().get(latLng);
34
35                  clusterManager.setOnClusterItemInfoWindowLongClickListener(new ClusterManager.
                            OnClusterItemInfoWindowLongClickListener() {
36                      @Override
37                      public void onClusterItemInfoWindowLongClick(ClusterItem item) {
38
39                          Intent browserIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(perAList.getUrlA().get(key)))
                                    ;
40                          startActivity(browserIntent);
41                      }
42                  });
```

```
43
44                clusterManager.setOnClusterItemInfoWindowClickListener(new ClusterManager.
                          OnClusterItemInfoWindowClickListener() {
45                    @Override
46                    public void onClusterItemInfoWindowClick(ClusterItem item) {
47                        dialog.setContentView(R.layout.image_popup);
48                        full = dialog.findViewById(R.id.full_image);
49                        constraintLayout =dialog.findViewById(R.id.constraintLayoutP);
50                        progressBar=dialog.findViewById(R.id.progressBar);
51                        FutureTarget<Bitmap> futureTarget =
52                                Glide.with(HomeFragment.this)
53                                        .asBitmap()
54                                        .load(perAList.getUrlIA().get(key))
55                                        .submit();
56                        new Thread(new Worker(futureTarget)).start();
57                        close = dialog.findViewById(R.id.close_image);
58                        close.setOnClickListener(new View.OnClickListener() {
59                            @Override
60                            public void onClick(View v) {
61                                dialog.dismiss();
62                            }
63                        });
64                        dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
65                        dialog.show();
66
67                    }
68                });
69            }
70            else {
71                clusterManager.setOnClusterItemInfoWindowClickListener(new ClusterManager.
                          OnClusterItemInfoWindowClickListener() {
72                    @Override
73                    public void onClusterItemInfoWindowClick(ClusterItem item) {
74                        String key = perUList.getListLL().get(dec.format(marker.getPosition().latitude).replace(",",
                                  ".") + ";" + dec.format(marker.getPosition().longitude).replace(",", "."));
75                        if (perUList.getImaPre().get(key).equals("si")) {
76                            dialog.getWindow().getAttributes().windowAnimations = R.style.PauseDialogAnimation;
77                            dialog.setContentView(R.layout.image_popup);
78                            constraintLayout =dialog.findViewById(R.id.constraintLayoutP);
79                            progressBar=dialog.findViewById(R.id.progressBar);
80                            full = dialog.findViewById(R.id.full_image);
81                            new Thread(new Worker5(perUList.getListLL().get(dec.format(marker.getPosition().latitude)
                                      .replace(",", ".") + ";" + dec.format(marker.getPosition().longitude).replace(","
                                      , ".")).toString())).start();
82                            close = dialog.findViewById(R.id.close_image);
83
84                            close.setOnClickListener(new View.OnClickListener() {
85                                @Override
86                                public void onClick(View v) {
87                                    dialog.dismiss();
88                                }
89                            });
90                            dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.TRANSPARENT));
91                            dialog.show();
92
93                        } else {
94                            Toast.makeText(getActivity(), getString(R.string.immaA) ,Toast.LENGTH_SHORT).show();
95                        }
96                    }
97                });
98
99                    myContentsView = LayoutInflater.from(getActivity()).inflate(R.layout.custom_info_contents, null)
                              ;
100                   TextView tvTitle = ((TextView) myContentsView.findViewById(R.id.title));
101                   tvTitle.setText(marker.getTitle());
102                   TextView tvSnippet = ((TextView) myContentsView.findViewById(R.id.snippet));
103                   tvSnippet.setText(marker.getSnippet());
104                   ImageView eveVe = myContentsView.findViewById(R.id.imageViewVer);
105                   ImageView eveVe2 = myContentsView.findViewById(R.id.imageViewVer2);
106                   ImageView eveVe3 = myContentsView.findViewById(R.id.imageViewVer3);
107
108                       if (!dupliV.isEmpty()) {
109                           if (dupliV.get(0)>=3) {
110                               eveVe.setImageDrawable(getContext().getDrawable(R.drawable.
                                          ic_baseline_check_circle_24));
111                               eveVe.setColorFilter(getContext().getResources().getColor(R.color.colorPrimary));
112                               eveVe2.setVisibility(View.VISIBLE);
113                               eveVe3.setVisibility(View.VISIBLE);
114                           }
115                           else {
116                               if (dupliV.get(0)==2) {
117                                   eveVe.setImageDrawable(getContext().getDrawable(R.drawable.
                                              ic_baseline_check_circle_24));
118                                   eveVe.setColorFilter(getContext().getResources().getColor(R.color.colorPrimary));
119                                   eveVe2.setVisibility(View.VISIBLE);
120                                   eveVe3.setVisibility(View.GONE);
121                               }
122                               else {
123                                   if (dupliV.get(0)==1) {
124                                       eveVe.setImageDrawable(getContext().getDrawable(R.drawable.
                                                  ic_baseline_check_circle_24));
125                                       eveVe.setColorFilter(getContext().getResources().getColor(R.color.colorPrimary
                                                  ));
126                                       eveVe2.setVisibility(View.GONE);
```

```
127                                 eveVe3.setVisibility(View.GONE);
128                             }else {
129                                 eveVe.setImageDrawable(getContext().getDrawable(R.drawable.
                                            ic_baseline_cancel_24));
130                                 eveVe.setColorFilter(getContext().getResources().getColor(R.color.colorAccent)
                                            );
131                                 eveVe2.setVisibility(View.GONE);
132                                 eveVe3.setVisibility(View.GONE);
133                             }
134                         }
135                     }
136                 }
137                 ImageView im = ((ImageView) myContentsView.findViewById(R.id.immagine));
138                 String key = perUList.getListLL().get(dec.format(marker.getPosition().latitude).replace(",", "."
                        ) + ";" + dec.format(marker.getPosition().longitude).replace(",", "."));
139                 if (perUList.getImaPre().get(key).equals("si")) {
140
141                     im.setVisibility(View.VISIBLE);
142                     im.setImageBitmap((Bitmap) perUList.getImage().get(key));
143                 } else {
144                     im.setVisibility(View.GONE);
145                 }
146             }
147         }else {
148             myContentsView = LayoutInflater.from(getActivity()).inflate(R.layout.custom_info_contents_du, null);
149             TextView tvTitle = ((TextView) myContentsView.findViewById(R.id.title1));
150             tvTitle.setText(getString(R.string.cisono) + dupliP.size() + getString(R.string.evein));
151             TextView tvSnippet = ((TextView) myContentsView.findViewById(R.id.snippet1));
152             tvSnippet.setText(getString(R.string.premfi));
153             clusterManager.setOnClusterItemInfoWindowClickListener(new ClusterManager.
                        OnClusterItemInfoWindowClickListener() {
154
155                 @Override
156                 public void onClusterItemInfoWindowClick(ClusterItem item) {
157                     Intent i = new Intent(getActivity(), LisEveMultiActivity.class);
158                     i.putExtra("lista", (Serializable) dupliP);
159                     i.putExtra("immaUrl", (Serializable) dupliI);
160                     i.putExtra("eventi", (Serializable) dupliE);
161                     getActivity().startActivity(i);
162                 }
163             });
164         }
165         return myContentsView;
166     }
167 }
```

Listato 5.28. Estratto della prima parte della classe **`HomeFragment`**

La parte di codice che adesso è quella contenuta nel Listato 5.29. Il metodo che analizzeremo è chiamato **`onMapReady`**; esso realizza e gestisce la mappa geografica ottenuta tramite l'utilizzo delle API di Google Maps. Questo metodo viene richiamato alla creazione della mappa, a seguito della richiesta effettuata utilizzando le API di Google durante la creazione della dell'Activity. Dal codice possiamo vedere come, non appena la mappa viene creata, vengono impostati una serie di parametri, come il minimo livello di zoom che l'utente può effettuare tramite gesture, il bounding box (cioè i limiti della mappa che è possibile visualizzare) attorno l'Italia, per impedire che l'utente possa spostare la visuale al di fuori di essa, ed, infine, la posizione iniziale da visualizzare; essa sarà incentrata sulla "città principale" impostata dall'utente durante il suo primo avvio dell'app (righe 13-83).

Sempre dentro questo metodo ci sono anche le definizioni che riguardano i due Toggle Button posizionati, all'interno della schermata, sopra la mappa degli eventi. Il primo riguarda la modalità di visualizzazione tramite heatmap degli eventi sulla mappa, con la possibilità di tornare alla visualizzazione tramite marker; il secondo realizza il pulsante per poter passare dalla mappa geografica a quella satellitare, e viceversa (righe 21-40 e 88-96). È presente nel metodo anche la definizione del codice del pulsante che permette di utilizzare il GPS per spostare la visuale automaticamente sulla posizione attuale del dispositivo.

Un altro modo per permettere all'utente di spostarsi agilmente da una città all'altra è dato dalla presenza di un campo di testo sulla schermata della mappa; in esso l'utente può inserire manualmente un indirizzo sul quale spostare la visuale. Il

codice che gestisce questa operazione è anch'esso presente all'interno del metodo in esame (righe 106-116).

Possiamo notare, anche, la presenza di un listener all'interno del metodo, che risponde al movimento della visuale della mappa; il codice in esso contenuto permette di effettuare un controllo per verificare se dei filtri precedentemente impostati siano stati cancellati o modificati; in questo caso si effettua una pulizia della mappa da tutti i marker posizionati (righe 45-78).

Sempre dentro il listener possiamo notare, anche, l'esecuzione di un metodo chiamato **`cooUp`**; questo è un metodo speciale perchè permette di gestire la concorrenza tra i thread in esecuzione, rendendo atomiche cioè non interrompibili, le istruzioni al suo interno, infatti, dal codice che possiamo vedere nel Listato 5.30, viene effettuato un controllo che verifica se il punto geografico in cui la visuale è centrata, una volta terminato un movimento, è ad almeno 5 Km di distanza dal punto precedentemente salvato (righe 470- 479). In caso la visuale si sia spostata di una distanza sufficiente dalla precedente posizione, verrà impostata la posizione corrente come posizione di partenza della visuale, per i controlli futuri dovuti al movimento di quest'ultima, inoltre, vengono lanciati i thread **`Worker2`** e **`Worker3`**.

Tutto questo ha lo scopo di realizzare un sistema che permetta di aggiornare i marker della mappa solo a seguito di uno spostamento della visuale. Tuttavia, per evitare un eccessivo sovraccarico delle richieste di lettura del database, viene effettuato il controllo che permette di ridurre il numero di richieste effettuate. Tale controllo consente l'aggiornamento della mappa solo nel caso in cui la visuale si sia spostata di almeno 5 Km dall'ultimo punto salvato per il controllo.

```
1    private OnMapReadyCallback callback = new OnMapReadyCallback() {
2
3          @Override
4          public void onMapReady(GoogleMap googleMap) {
5
6
7              fusedLocationClient = LocationServices.getFusedLocationProviderClient(getActivity());
8              mMap = googleMap;
9              mMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(getActivity(), R.raw.style_json));
10             mMap.getUiSettings().setMapToolbarEnabled(false);
11             clusterManager = new ClusterManager<>(getActivity(), mMap);
12             markerClusterRenderer = new MarkerClusterRenderer(getActivity(), mMap, clusterManager);
13             mMap.setMinZoomPreference(5.3f);
14             LatLngBounds italyBounds = new LatLngBounds(
15                    new LatLng(37.020710, 5.854287),
16                    new LatLng(47.363126, 20.039189)
17             );
18
19             mMap.setLatLngBoundsForCameraTarget(italyBounds);
20             tButtonH = getView().findViewById(R.id.toggleButtonH);
21             tButtonH.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
22                 public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
23                     if (isChecked) {
24                         initialPosi=new LatLng(0,0);
25                         longitude1=0.01;
26                         mMap.clear();
27                         clusterManager.clearItems();
28                         markerMapM.clear();
29                         heat=true;
30                     } else {
31                         initialPosi=new LatLng(0,0);
32                         longitude1=0.01;
33                         if (overlay!=null) {
34                             heat = false;
35                             overlay.remove();
36                         }
37
38                     }
39                 }
40             });
41
42             mMap.setOnCameraChangeListener(new GoogleMap.OnCameraChangeListener() {
43
44                 @Override
45                 public void onCameraChange(CameraPosition cameraPosition) {
46                     LatLng finalPosi=mMap.getCameraPosition().target;
```

```
47                  Double lat =finalPosi.latitude;
48                  Double lng =finalPosi.longitude;
49                  viewModel = new ViewModelProvider(getActivity()).get(ItemViewModel.class);
50                  String[] filters=viewModel.getSelectedItem().getValue();
51                  if (filters!=null) {
52                      if (filters[5].equals("1")){
53                          weightedLatLngArrayList.clear();
54  ...
55  ...
56  ...
57                          initialPosi=new LatLng(0,0);
58                          longitude1=0.01;
59                          clusterManager.clearItems();
60                          markerMapM.clear();
61                          filters[5]="0";
62                          getActivity().runOnUiThread(new Runnable() {
63
64                              @Override
65                              public void run() {
66                                  try {
67                                      mMap.clear();
68                                      viewModel.selectItem(filters);
69                                  }catch (Exception e){
70                                      e.printStackTrace();
71                                  }
72                              }
73                          });
74                      }
75                  }
76                  cooUp(lat,lng,finalPosi);
77              }
78          });
79          SharedPreferences preferences = getActivity().getSharedPreferences("citta", MODE_PRIVATE);
80          Double lati= Double.parseDouble(preferences.getString("latitudine", "42.402194").replace(",","."));
81          Double longi= Double.parseDouble(preferences.getString("longitudine", "12.862167").replace(",","."));
82          LatLng camera1 = new LatLng(lati, longi);
83          mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(camera1, (float) 10));
84          clusterManager.getMarkerCollection().setInfoWindowAdapter((new HomeFragment.MyInfoWindowAdapter()));
85  ...
86  ...
87  ...
88          tButton.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
89              public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
90                  if (isChecked) {
91                      mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
92                  } else {
93                      mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
94                  }
95              }
96          });
97          iButton = getView().findViewById(R.id.imageButton);
98          iButton.setOnClickListener(new View.OnClickListener() {
99  ...
100 ...
101 ...
102         });
103
104         textCity = getView().findViewById(R.id.text_citta);
105         RequestQueue queue = Volley.newRequestQueue(getActivity());
106         textCity.setOnKeyListener(new View.OnKeyListener() {
107             public boolean onKey(View v, int keyCode, KeyEvent event) {
108                 if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
109                         (keyCode == KeyEvent.KEYCODE_ENTER)) {
110 ...
111 ...
112 ...
113                 }
114                 return false;
115             }
116         });
117     }
118 };
```

Listato 5.29. Estratto della seconda parte della classe **`HomeFragment`**

La successiva parte di codice che analizzeremo è quella contenuta nel Listato 5.30. In particolare, ci concentreremo sull'analisi dei thread della classe, incominciando dal thread **`Worker2`**; questo è uno dei thread che viene richiamato dal listener presente nel Listato 5.29 già discusso in precedenza (righe 6-206). Osservando il contenuto del thread, possiamo notare che le principali operazioni svolte sono quelle relative al recupero delle informazioni riguardanti gli eventi presenti nel raggio di 5 Km dal punto che rappresenta il centro della vista della mappa. Difatti sono presenti due diverse interrogazioni del database remoto (Firestore), una per il recupero delle

informazioni testuali degli eventi di segnalazione utente e l'altro per gli eventi ottenuti dagli articoli. Per di più, sui risultati ottenuti da queste interrogazioni, viene effettuato un controllo per poter filtrare tutti gli eventi che non rispettano i valori attualmente impostati per i filtri attivi, i cui valori sono stati impostati dall'utente tramite un pulsante presente sulla schermata della mappa, che una volta premuto, creerà il Dialog **`FilDialogFragment`** da cui selezionare i vari filtri.

L'altro thread che viene eseguito dal listener nel Listato 5.29 è **`Worker3`**; esso permette di recuperare le immagini di thumbnail presenti nel database (Storage di Firabase) in modo da poter essere utilizzate all'interno delle Info Window degli eventi segnalati dagli utenti, come anteprima dell'immagine allegata all'evento (righe 207-234). Un'altra funzione di questo thread è che, alla ricezione della risposta contenente il thumbnail dell'evento, viene lanciato il thread **`Worker4`**.

Questo nuovo thread si occupa della creazione dei marker a partire dalle informazioni sugli eventi ottenuti dai thread precedenti nonchè dell'aggiunta di questi nuovi marker ai cluster presenti sulla mappa (righe 236-417). Infatti, come abbiamo già discusso, i cluster sono agglomerati di marker rappresentati tramite un pallino con all'interno il numero di marker attualmente contenuti; ogni volta che due o più marker presenti sulla mappa risultano molto vicini tra loro per via del livello di zoom scelto dall'utente, questi vengono inglobati dal cluster più vicino. Il contrario è, naturalmente, possibile, a partire da un cluster, se l'utente imposta un livello di zoom abbastanza elevato da permettere che i marker al suo interno risultino abbastanza distanti tra loro; questi si separeranno dal cluster per posizionarsi sul luogo dell'evento da essi identificato. Tornando al codice del quarto thread, possiamo vedere la parte dedicata alla vera e propria creazione del marker, compreso di icona che si differenzia per colore (nel caso di eventi ottenuti dalle segnalazioni degli utenti), in base al livello del campo di "verificato" associato, in modo tale da rendere visivamente distinguibili gli eventi in base al loro livello. Continuando, possiamo notare il codice che si occupa di verificare la presenza di più eventi sullo stesso punto, con lo scopo di impostare l'icona del marker di colore viola per distinguerlo dal resto degli eventi presenti sulla mappa. Per far ciò si impiega il metodo **`control`** (anch'esso, come il metodo **`cooUp`**, è un metodo usato per risolvere il problema della concorrenza dei thread; infatti, le operazioni al suo interno sono considerate atomiche) che si occupa di realizzare un controllo sulle icone dei marker presenti sulla mappa per aggiornarli, nel caso differissero per via di un aggiornamento delle informazioni contenute nel database avvenuto durante la visualizzazione della mappa.

Un'altra importante operazione svolta da questo metodo è l'aggiunta del marker appena creato al cluster dei marker visualizzati sulla mappa, come possiamo vedere dal codice. La parte successiva del codice del thread si occupa di calcolare il livello di pericolosità associato all'evento (che viene utilizzato per stabilire il colore che l'evento assumerà sulla mappa durante la modalità di visualizzazione tramite heatmap) in base alla categoria e al tempo trascorso dalla sua segnalazione; quindi man mano che il tempo trascorre la pericolosità dell'evento diminuisce. Tutte queste operazioni descritte finora vengono realizzate due volte, la prima per gli eventi ottenuti dalle segnalazioni degli utenti e la seconda per gli eventi ottenuti dagli articoli, in quanto, come già discusso, le informazioni degli eventi variano in base alla fonte da cui sono stati generati e necessitano, quindi, di operazioni leggermente differenti tra loro.

I thread restanti, cioè **`Worker`** e **`Worker5`**, si occupano entrambi del recupero delle immagini originali degli eventi e consentono di visualizzare tali immagini alla dimensione massima consentita dal dispositivo quando l'utente preme sulla Info Window dell'evento (righe 419-468). La differenza tra di essi è dovuta al tipo di evento associato, il primo viene utilizzato per il recupero delle immagini di eventi ottenuti dalle segnalazioni degli utenti mentre il secondo per gli eventi ottenuti dagli articoli.

```
1   class Worker2 implements Runnable {
2   ...
3   ...
4   ...
5
6           public Worker2(CountDownLatch countDownLatch, GeoLocation center) {
7
8               this.countDownLatch=countDownLatch;
9               this.center=center;
10              this.longitude1 =(1/(111.320*Math.cos(Math.toRadians( center.latitude ))));
11          }
12
13          @Override
14          public void run() {
15              try {
16                  viewModel = new ViewModelProvider(getActivity()).get(ItemViewModel.class);
17                  String[] filters=viewModel.getSelectedItem().getValue();
18                  if (filters!=null) {
19                      flag = true;
20
21                  }
22                  FirebaseFirestore db = FirebaseFirestore.getInstance();
23                  final List<Task<QuerySnapshot>> tasks = new ArrayList<>();
24                      Query q = db.collection("utenti").document("dati").collection("eventi")
25                                  .orderBy("lat")
26                                  .whereGreaterThanOrEqualTo("lat", (center.latitude - (latitude1*5)))
27                                  .whereLessThanOrEqualTo("lat", (center.latitude + (latitude1*5)));
28
29                      tasks.add(q.get());
30                      Tasks.whenAllComplete(tasks)
31                             .addOnCompleteListener(new OnCompleteListener<List<Task<?>>>() {
32                                 @RequiresApi(api = Build.VERSION_CODES.N)
33                                 @Override
34                                 public void onComplete(@NonNull Task<List<Task<?>>> t) {
35                                     for (Task<QuerySnapshot> task : tasks) {
36                                         QuerySnapshot snap = task.getResult();
37                                         for (DocumentSnapshot doc : snap.getDocuments()) {
38                                             DecimalFormat dec = new DecimalFormat("#0.000000");
39                                             double lat = doc.getDouble("lat");
40                                             double lng = doc.getDouble("lng");
41
42                                             if(lat>=(center.latitude - latitude1*5) && lng>= (center.longitude -
                                                       longitude1 *5) && lat<=(center.latitude + latitude1*5) && lng<=(
                                                       center.longitude + longitude1 *5)) {
43                                                 if (flag) {
44                                                     if (filters[6].equals("eventi utenti")  filters[6].equals("")  (
                                                               filters[6].equals("eventi verificati") && doc.getLong("
                                                               verificato").intValue()>0)) {
45                                                         if (filters[0].equals(doc.getString("crimine"))  filters[0].equals(
                                                                   "")) {
46
47                                                             if ((doc.getString("descrizione").toLowerCase().replaceAll("\\p{
                                                                       Punct}", " ").matches("^" + filters[1].toLowerCase() + "
                                                                       \\s.*.*\\s" + filters[1].toLowerCase() + "\\s.*.*\\s" +
                                                                       filters[1].toLowerCase() + "$"))  filters[1].equals(""))
                                                                        {
48                                                                 try {
49                                                                     Date oraC = new SimpleDateFormat("HH:mm").parse(doc.
                                                                               getString("ora"));
50                                                                     if (!filters[4].equals("")) {
51                                                                         if (oraC.getHours() == Integer.valueOf(filters[4])) {
52                                                                             if (!filters[2].equals("") && !filters[3].equals(
                                                                                       "")) {
53                                                                                 Date dateC = new SimpleDateFormat("dd/MM/yyyy"
                                                                                           ).parse(doc.getString("data"));
54                                                                                 Date dateMin = new SimpleDateFormat("dd/MM/
                                                                                           yyyy").parse(filters[2]);
55                                                                                 Date dateMax = new SimpleDateFormat("dd/MM/
                                                                                           yyyy").parse(filters[3]);
56                                                                                 if ((!dateC.before(dateMin) && !dateC.after(
                                                                                           dateMax))) {
57                                                                                     filteredU = true;
58                                                                                 }
59                                                                             } else {
60                                                                                 filteredU = true;
61                                                                             }
62                                                                         }
63                                                                     } else {
```

```
                                                if (!filters[2].equals("") && !filters[3].equals(""))
                                                        {
                                                    Date dateC = new SimpleDateFormat("dd/MM/yyyy").
                                                            parse(doc.getString("data"));
                                                    Date dateMin = new SimpleDateFormat("dd/MM/yyyy")
                                                            .parse(filters[2]);
                                                    Date dateMax = new SimpleDateFormat("dd/MM/yyyy")
                                                            .parse(filters[3]);
                                                    if ((!dateC.before(dateMin) && !dateC.after(
                                                            dateMax))) {
                                                        filteredU = true;
                                                    }
                                                } else {
                                                    filteredU = true;
                                                }
                                            }
                                            } catch (Exception e) {
                                                e.printStackTrace();
                                            }
                                        }
                                    }
                                }
                            }
                            if (!flag  filteredU) {
                                String latLng = dec.format(lat).replace(",", ".") + ";" + dec.
                                        format(lng).replace(",", ".");

                                if (!listC.containsKey(doc.getId().replace("-","/"))) {
                                    Event even = new Event(latLng, doc.getId());
                                    even.setVeri(doc.getLong("verificato").intValue() );
...
...
...
                                    listE.add(even);
                                }
                                listC.put(doc.getId().replace("-","/"), latLng);
                                InsEveULIst tempUListS = new InsEveULIst();
                                tempUListS.putListaV(latLng, doc.getLong("verificato").intValue());
...
...
...
                                tempUList.putAll(tempUListS);
                                filteredU = false;
                            }
                        }
                    }
                }
                countDownLatch.countDown();
            }
        });

    final List<Task<QuerySnapshot>> tasksA = new ArrayList<>();
    Query qA = db.collection("articoli")
            .orderBy("Lat")
            .whereGreaterThanOrEqualTo("Lat", (center.latitude - (latitude1*5)))
            .whereLessThanOrEqualTo("Lat", (center.latitude + (latitude1*5)));

    tasksA.add(qA.get());
    Tasks.whenAllComplete(tasksA)
            .addOnCompleteListener(new OnCompleteListener<List<Task<?>>>() {
                @RequiresApi(api = Build.VERSION_CODES.N)
                @Override
                public void onComplete(@NonNull Task<List<Task<?>>> t) {
                    List<DocumentSnapshot> matchingDocs = new ArrayList<>();

                    for (Task<QuerySnapshot> task : tasksA) {
                        QuerySnapshot snap = task.getResult();
                        for (DocumentSnapshot doc : snap.getDocuments()) {
                            DecimalFormat dec = new DecimalFormat("#0.000000");
                            double lat = doc.getDouble("Lat");
                            double lng = doc.getDouble("Lon");

                            try {
                            if(lat>=(center.latitude - latitude1*5) && lng>= (center.longitude - longitude1
                                    *5) && lat<=(center.latitude + latitude1*5) && lng<=(center.longitude
                                    + longitude1 *5)) {
                                Calendar c = Calendar.getInstance();
                                Date time = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse(doc.getString
                                        ("DataArticolo"));
                                Date dateC = new SimpleDateFormat("dd/MM/yyyy").parse(time.getDate()+"/"+(
                                        time.getMonth()+1)+"/"+(time.getYear()+1900));
                                if (flag ) {
                                    if (filters[6].equals("eventi articoli")  filters[6].equals("")
                                            filters[6].equals("eventi verificati") ) {
                                        String category = doc.getString("Categoria").toLowerCase();
                                    if (category.equals(filters[0])  filters[0].equals("")) {
                                        if (((doc.getString("TitoloArticolo").toLowerCase().replaceAll("\\p
                                                {Punct}", " ").matches("^" + filters[1].toLowerCase() + "\\s
                                                .*.*\\s" + filters[1].toLowerCase() + "\\s.*.*\\s" + filters
                                                [1].toLowerCase() + "$")  doc.getString("IntroArticolo").
                                                toLowerCase().replaceAll("\\p{Punct}", " ").matches("^" +
                                                filters[1].toLowerCase() + "\\s.*.*\\s" + filters[1].
                                                toLowerCase() + "\\s.*.*\\s" + filters[1].toLowerCase() + "$
                                                ")))  filters[1].equals("")) {
```

```
140                                             try {
141                                                 if (!filters[4].equals("")) {
142                                                     if (time.getHours() == Integer.valueOf(filters[4])) {
143                                                         if (!filters[2].equals("") && !filters[3].equals(""))
                                                                {
144                                                             Date dateMin = new SimpleDateFormat("dd/MM/yyyy")
                                                                    .parse(filters[2]);
145                                                             Date dateMax = new SimpleDateFormat("dd/MM/yyyy")
                                                                    .parse(filters[3]);
146                                                             if ((!dateC.before(dateMin) && !dateC.after(
                                                                    dateMax))) {
147                                                                 filteredA = true;
148                                                             }
149                                                         } else {
150                                                             filteredA = true;
151                                                         }
152                                                     }
153                                                 } else {
154                                                     if (!filters[2].equals("") && !filters[3].equals("")) {
155                                                         Date dateMin = new SimpleDateFormat("dd/MM/yyyy").
                                                                parse(filters[2]);
156                                                         Date dateMax = new SimpleDateFormat("dd/MM/yyyy").
                                                                parse(filters[3]);
157                                                         if ((!dateC.before(dateMin) && !dateC.after(dateMax))
                                                                ) {
158                                                             filteredA = true;
159                                                         }
160                                                     } else {
161                                                         filteredA = true;
162                                                     }
163                                                 }
164                                             } catch (Exception e) {
165                                                 e.printStackTrace();
166                                             }
167                                         }
168                                     }
169                                 }
170                                 }
171 
172                                 if (!flag  filteredA) {
173                                     String latLng = dec.format(lat).replace(",", ".") + ";" + dec.format(
                                            lng).replace(",", ".");
174                                     c.setTime(time);
175                                     if (!listC.containsKey(doc.getId())) {
176                                         Event ev = new Event(latLng, doc.getId());
177                                         ev.setImgUrl(doc.getString("URLImg"));
178 ...
179 ...
180 ...
181                                         listE.add(ev);
182                                     }
183                                     listC.put(doc.getId(), latLng);
184                                     InsEveAList tempAListS= new InsEveAList();
185                                     tempAListS.putCateA(doc.getId(), doc.getString("Categoria").toLowerCase
                                            ());
186 ...
187 ...
188 ...
189                                     tempAList.putAll(tempAListS);
190                                     filteredA =false;
191                                 }
192                             }
193                             } catch (ParseException e) {
194                                 e.printStackTrace();
195                             }
196                         }
197                     }
198                     countDownLatch.countDown();
199                 }
200             });
201 
202             } catch (Exception e) {
203                 e.printStackTrace();
204             }
205         }
206     }
207     class Worker3 implements Runnable {
208 ...
209 ...
210 ...
211 
212         public Worker3(CountDownLatch countDownLatch) {
213             this.countDownLatch=countDownLatch;
214 
215         }
216 
217         @Override
218         public void run() {
219             try {
220                 countDownLatch.await();
221                 CountDownLatch countDownLatch2= new CountDownLatch(perUList.getImaPre().size());
222                 new Thread(new Worker4(countDownLatch2)).start();
223                 for (Object key:perUList.getImaPre().keySet().toArray()){
224                       if (perUList.getImaPre().get(key).equals("si")) {
```

```
...
...
...
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}

class Worker4 implements Runnable {
...
...
...
    public Worker4(CountDownLatch countDownLatch) {

        this.countDownLatch=countDownLatch;

    }

    @Override
    public void run() {
        try {
            countDownLatch.await();
            if (getActivity()!=null) {
                getActivity().runOnUiThread(new Runnable() {

                    @Override
                    public void run() {
                        try {
                            for (Object key : tempUList.getListLL().keySet().toArray()) {
                                String[] l = key.toString().split(";");
                                if (!heat) {
                                    listC.remove(tempUList.getListLL().get(key));
                                    if (listC.containsValue(key)) {
                                        MarkerOptions markerOptions = new MarkerOptions();
                                        LatLng posi = new LatLng(Double.parseDouble(l[0]), Double.parseDouble(l[1])
                                                );
                                        markerOptions.position(posi).icon(BitmapDescriptorFactory.defaultMarker(
                                                BitmapDescriptorFactory.HUE_VIOLET));
                                        User item = new User(markerOptions.getPosition(), markerOptions.getSnippet
                                                (), markerOptions.getIcon(), markerOptions.getTitle());

                                        control(key, item);

                                        listC.put(tempUList.getListLL().get(key), key.toString());
                                        continue;
                                    } else {
                                        listC.put(tempUList.getListLL().get(key), key.toString());
                                    }
                                }
                                String otherK = tempUList.getListLL().get(key.toString());
                                MarkerOptions markerOptions = new MarkerOptions();
                                LatLng posi = new LatLng(Double.parseDouble(l[0]), Double.parseDouble(l[1]));
                                WeightedLatLng wecoo = null;
                                Calendar cal = Calendar.getInstance();
                                Date today = new Date();
                                SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");
                                int old = 0;
                                try {
                                    cal.setTime(sdf.parse(tempUList.getDate().get(otherK) + " " + tempUList.
                                            getHour().get(otherK)));// all done
                                } catch (ParseException e) {
                                    e.printStackTrace();
                                }
                                try {
                                    old = (int) (getDateDiff(cal.getTime(), today, TimeUnit.DAYS) / 30 - weights[
                                            crimeArray.indexOf(tempUList.getCate().get(otherK))]);

                                } catch (ArrayIndexOutOfBoundsException e) {
                                }
                                if (old <= 0) {
                                    old = 1;
                                }
                                try {
                                    wecoo = new WeightedLatLng(posi, ((double) weights[crimeArray.indexOf(
                                            tempUList.getCate().get(otherK))] / (double) old));
                                } catch (ArrayIndexOutOfBoundsException e) {
                                    wecoo = new WeightedLatLng(posi, 0);
                                }
                                if (heat) {
                                    if (weightedLatLngArrayList.containsKey(key.toString())) {
                                        if (wecoo.getIntensity() > weightedLatLngArrayList.get(key).getIntensity())
                                            weightedLatLngArrayList.put(key.toString(), wecoo);
                                    } else
                                        weightedLatLngArrayList.put(key.toString(), wecoo);
                                }

                                    switch (tempUList.getListV().get(key)) {
                                        case 0:
                                            markerOptions.position(posi)
                                                    .title(getString(R.string.evep) + tempUList.getPlace().get(
                                                            otherK))
```

```
312                                     .snippet(getString(R.string.data) + tempUList.getDate().get(
                                                otherK) + " " + tempUList.getHour().get(otherK) +
                                                getString(R.string.cat) + tempUList.getCate().get(otherK
                                                ) + getString(R.string.luogo) + tempUList.getPlace().get
                                                (otherK) + getString(R.string.des) + tempUList.getDes().
                                                get(otherK))
313                                     .icon(BitmapDescriptorFactory.defaultMarker(
                                                BitmapDescriptorFactory.HUE_RED));
314
315                                 break;
316                             case 1:
317                                 markerOptions.position(posi)
318                                     .title(getString(R.string.evep) + tempUList.getPlace().get(
                                                otherK))
319                                     .snippet(getString(R.string.data) + tempUList.getDate().get(
                                                otherK) + " " + tempUList.getHour().get(otherK) +
                                                getString(R.string.cat) + tempUList.getCate().get(otherK
                                                ) + getString(R.string.luogo) + tempUList.getPlace().get
                                                (otherK) + getString(R.string.des) + tempUList.getDes().
                                                get(otherK))
320                                     .icon(BitmapDescriptorFactory.defaultMarker(
                                                BitmapDescriptorFactory.HUE_ORANGE));
321
322                                 break;
323
324                             case 2:
325                                 markerOptions.position(posi)
326                                     .title(getString(R.string.evep) + tempUList.getPlace().get(
                                                otherK))
327                                     .snippet(getString(R.string.data) + tempUList.getDate().get(
                                                otherK) + " " + tempUList.getHour().get(otherK) +
                                                getString(R.string.cat) + tempUList.getCate().get(otherK
                                                ) + getString(R.string.luogo) + tempUList.getPlace().get
                                                (otherK) + getString(R.string.des) + tempUList.getDes().
                                                get(otherK))
328                                     .icon(BitmapDescriptorFactory.defaultMarker(
                                                BitmapDescriptorFactory.HUE_YELLOW));
329                                 break;
330                             default:
331                                 markerOptions.position(posi)
332                                     .title(getString(R.string.evep) + tempUList.getPlace().get(
                                                otherK))
333                                     .snippet(getString(R.string.data) + tempUList.getDate().get(
                                                otherK) + " " + tempUList.getHour().get(otherK) +
                                                getString(R.string.cat) + tempUList.getCate().get(otherK
                                                ) + getString(R.string.luogo) + tempUList.getPlace().get
                                                (otherK) + getString(R.string.des) + tempUList.getDes().
                                                get(otherK))
334                                     .icon(BitmapDescriptorFactory.defaultMarker(
                                                BitmapDescriptorFactory.HUE_GREEN));
335                                 break;
336
337                         }
338                     User item = new User(markerOptions.getPosition(), markerOptions.getSnippet(),
                                markerOptions.getIcon(), markerOptions.getTitle());
339                     control(key, item);
340                 }
341                 perUList.putAll(tempUList);
342                 tempUList.clear();
343                 for (Object key : tempAList.getListA().keySet().toArray()) {
344                     String l[] = key.toString().split(";");
345                     listC.remove(tempAList.getListA().get(key));
346                     if (listC.containsValue(key)) {
347                         MarkerOptions markerOptions = new MarkerOptions();
348                         LatLng posi = new LatLng(Double.parseDouble(l[0]), Double.parseDouble(l[1]));
349                         markerOptions.position(posi).icon(BitmapDescriptorFactory.defaultMarker(
                                    BitmapDescriptorFactory.HUE_VIOLET));
350                         User item = new User(markerOptions.getPosition(), markerOptions.getSnippet(),
                                    markerOptions.getIcon(), markerOptions.getTitle());
351                         control(key, item);
352                         listC.put(tempAList.getListA().get(key), key.toString());
353                         continue;
354                     } else {
355                         listC.put(tempAList.getListA().get(key), key.toString());
356                     }
357                     String otherK = tempAList.getListA().get(key.toString()).toString();
358                     SimpleDateFormat sdf1 = new SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.ITALY);
359                     String dat = sdf1.format(tempAList.getDateA().get(otherK).getTime());
360                     MarkerOptions markerOptions = new MarkerOptions();
361                     LatLng coordi = new LatLng(Double.parseDouble(l[0]), Double.parseDouble(l[1]));
362                     WeightedLatLng wecoo = null;
363                     Date today = new Date();
364                     int old = 0;
365                     try {
366                         old = (int) (getDateDiff(tempAList.getDateA().get(otherK).getTime(), today,
                                    TimeUnit.DAYS) / 30 - weights[crimeArray.indexOf(tempAList.getCateA().
                                    get(otherK))]);
367                     } catch (ArrayIndexOutOfBoundsException e) {
368                     }
369                     if (old <= 0) {
370                         old = 1;
371                     }
372                     try {
373                         wecoo = new WeightedLatLng(coordi, ((double) weights[crimeArray.indexOf(
```

```
                                        tempAList.getCateA().get(otherK))] / (double) old));
374                                 } catch (ArrayIndexOutOfBoundsException e) {
375                                     wecoo = new WeightedLatLng(coordi, 0);
376                                 }
377                                 if (heat) {
378                                     if (weightedLatLngArrayList.containsKey(key.toString())) {
379                                         if (wecoo.getIntensity() > weightedLatLngArrayList.get(key).getIntensity())
380                                             weightedLatLngArrayList.put(key.toString(), wecoo);
381                                     } else
382                                         weightedLatLngArrayList.put(key.toString(), wecoo);
383                                 }
384
385                                     markerOptions.position(coordi)
386                                             .title(getString(R.string.tito) + tempAList.getTitleA().get(otherK))
387                                             .snippet(getString(R.string.data) + dat + getString(R.string.cate) +
                                                     tempAList.getCateA().get(otherK).toString().replace("[", "").
                                                     replace("]", "") + getString(R.string.luogo) + tempAList.
                                                     getPlaceA().get(otherK) + getString(R.string.intro) + tempAList
                                                     .getIntroA().get(otherK))
388                                             .icon(BitmapDescriptorFactory.fromBitmap(getBitmap(AppCompatResources.
                                                     getDrawable(getContext(), R.drawable.news2))));
389                                     User item = new User(markerOptions.getPosition(), markerOptions.getSnippet(),
                                             markerOptions.getIcon(), markerOptions.getTitle());
390                                     control(key, item);
391                                 }
392                                 perAList.putAll(tempAList);
393                                 tempAList.clear();
394                                 markerMap.clear();
395                                 if (!weightedLatLngArrayList.isEmpty()) {
396                                     mMap.clear();
397                                     clusterManager.clearItems();
398                                     markerMapM.clear();
399                                     HeatmapTileProvider heatmap = new HeatmapTileProvider.Builder().weightedData(
                                             weightedLatLngArrayList.values()).build();
400                                     overlay = mMap.addTileOverlay(new TileOverlayOptions().tileProvider(heatmap));
401                                 } else {
402
403                                     if (overlay != null)
404                                         overlay.remove();
405
406                                 }
407                             }catch ( Exception e) {
408                                 e.printStackTrace();
409                             }
410                         }
411                     });
412                 }
413             } catch ( Exception e) {
414                 e.printStackTrace();
415             }
416         }
417     }
418
419     class Worker5 implements Runnable {
420 ...
421 ...
422 ...
423
424         public Worker5(String path) {
425             this.path=path;
426
427         }
428
429         @Override
430         public void run() {
431             try {
432                 FirebaseStorage storage = FirebaseStorage.getInstance();
433                 StorageReference storageRef = storage.getReference();
434 ...
435 ...
436 ...
437             } catch (Exception e) {
438                 e.printStackTrace();
439             }
440         }
441     }
442
443     class Worker implements Runnable {
444 ...
445 ...
446 ...
447
448         public Worker(FutureTarget<Bitmap> futureTarget ) {
449             this.futureTarget=futureTarget;
450
451         }
452
453         @Override
454         public void run() {
455             try {
456                 thumb = futureTarget.get();
457                 try {
458                     Thread.sleep(100);
459                 } catch (InterruptedException e) {
```

```
460                 e.printStackTrace();
461             }
462             int scaled_height = constraintLayout.getWidth() * thumb.getHeight() / thumb.getWidth();
463             FrameLayout.LayoutParams layoutParamsI = new FrameLayout.LayoutParams(constraintLayout.getWidth(),
                        scaled_height);
464 ...
465 ...
466 ...
467         }
468     }
469
470     public synchronized void cooUp(Double lat, Double lng, LatLng finalPosi){
471         if (!(lat >= (initialPosi.latitude - latitude1 * 5) && lng >= (initialPosi.longitude - longitude1 * 5)
                    && lat <= (initialPosi.latitude + latitude1 * 5) && lng <= (initialPosi.longitude + longitude1 *
                     5))) {
472             CountDownLatch countDownLatch1 = new CountDownLatch(2);
473             initialPosi = finalPosi;
474             longitude1 = (1 / (111.320 * Math.cos(Math.toRadians(finalPosi.latitude))));
475             GeoLocation center = new GeoLocation(lat, lng);
476             new Thread(new Worker2(countDownLatch1, center)).start();
477             new Thread(new Worker3(countDownLatch1)).start();
478         }
479     }
480
481     public synchronized void control(Object key, User markerOptions){
482         if (!markerMapM.containsKey(key.toString())) {
483             clusterManager.addItem(markerOptions);
484             markerMapM.put(key.toString(), markerOptions);
485         }
486         if (!(markerMapM.get(key.toString()).getIcon().equals(markerOptions.getIcon()))) {
487             clusterManager.removeItem(markerMapM.get(key.toString()));
488             clusterManager.addItem(markerOptions);
489             markerMapM.put(key.toString(), markerOptions);
490         }
491         if (!heat) {
492             clusterManager.cluster();
493             weightedLatLngArrayList.clear();
494         }
495     }
```

Listato 5.30. Estratto della terza parte della classe **`HomeFragment`**

5.2 Manuale dell'app

Prima di concludere questo capitolo, descriveremo i passaggi fondamentali che un utente può compiere utilizzando l'app, realizzando, così, un manuale di utilizzo in grado di illustrare tutte questi passaggi.

Cominciamo dal principio, cioè dall'icona dell'app, che apparirà nella home del dispositivo una volta installato l'APK (Android Package). Quello presente nella Figura 5.1 è un esempio dell'aspetto dell'icona dell'app una volta installata sul dispositivo di prova (Xiaomi Redmi Note 7, versione Android: 10 QKQ1.190910.002 API 29).

Il primo passo è l'avvio dell'applicazione tramite la pressione dell'icona dell'app; successivamente partirà una breve animazione e la prima schermata dell'app comparirà per un istante, come mostrato nella Figura 5.2. Subito dopo comparirà la prima vera schermata dell'app con cui l'utente può interagire, cioè la schermata presente nella Figura 5.3. Da essa l'utente può selezionare il metodo di Login che più preferisce, oppure, se non è ancora registrato, può farlo tramite e-mail/password premendo il testo "Oppure premi qui per registrarti". Un'altra opzione a disposizione dell'utente consiste nel premere il testo "Reimposta password" per poter accedere alla schermata che consentirà di reimpostare la password di un account precedentemente creato. Lasceremo per ultimo il caso in cui l'utente decida di effettuare il Login per accedere alle funzionalità dell'app; invece, adesso esploreremo le altre opzioni a disposizione dell'utente.

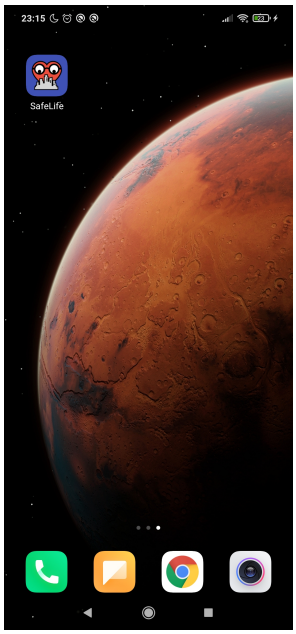

Figura 5.1. Immagine dell'icona dell'app sulla home del dispositivo di prova

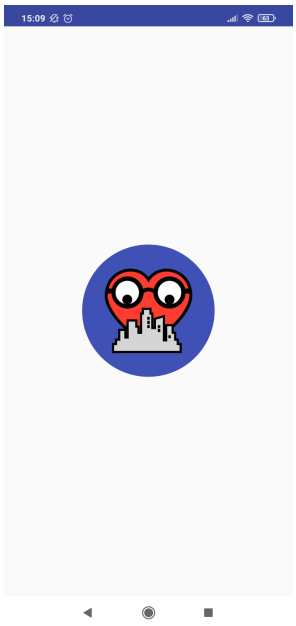

Figura 5.2. Immagine della schermata di avvio dell'app

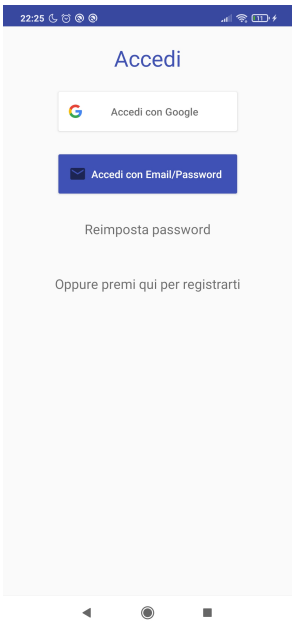

Figura 5.3. Immagine della schermata della scelta del metodo di Login

Il primo caso che analizzeremo è quello in cui l'utente abbia deciso di effettuare il reset della password di un account esistente. Nella Figura 5.4 abbiamo la schermata dedicata al reset; come possiamo osservare è presente solo un campo di testo in cui inserire l'e-mail dell'account che abbiamo intenzione di reimpostare e un pulsante di conferma. Una volta inserita l'informazione richiesta e premuto il tasto di conferma, verrà inviata una e-mail all'indirizzo fornito, contenente un link per reimpostare la password.

La prossima schermata che vedremo è quella della Figura 5.5. Questa si apre se l'utente decide di registrare un nuovo account usando l'e-mail/password come metodo di autenticazione. Essa è composta da una serie di campi di testo ognuno dedicato all'inserimento di una diversa informazione essenziale alla creazione dell'account; in fondo è presente un pulsante per completare la registrazione. Una volta che l'utente preme il tasto per registrarsi, viene effettuato un controllo sui dati inseriti; se sono corretti allora egli può proseguire con la schermata successiva e il nuovo account viene creato; se, invece, i dati non sono corretti, (ad esempio password troppo corta, l'e-mail è già registrata, etc.) viene mostrato sul video un messaggio di errore.

La prossima schermata che vedremo è quella della Figura 5.6. Questa si apre se l'utente decide di effettuare il Login di un account già creato usando l'e-mail/password come metodo di autenticazione. Essa è composta da due campi di testo necessari per l'inserimento delle informazioni essenziali ad effettuare il Login; in fondo alla schermata è presente il pulsante per effettuare l'acceso. Una volta che l'utente preme il tasto per l'accesso, viene effettuato un controllo sui dati inseriti, per verificare che l'account sia già stato creato in precedenza e che i dati inseriti siano corretti; in questo caso, l'utente può proseguire con la schermata successiva,

in caso contrario viene mostrato a video un messaggio di errore.

L'ultimo caso riguarda il Login tramite account Google, come mostrato dalla Figura 5.7. In questo caso la schermata è simile a quella della Figura 5.3; l'unica differenza è la comparsa in sovraimpressione del Dialog in cui viene richiesto all'utente di scegliere con quale account Google autenticarsi.

Una volta che l'utente è riuscito ad autenticarsi (sia tramite Login sia tramite registrazione di un nuovo account), potrà accedere a tutte le funzionalità offerte dall'app. Però, in caso di primo avvio dell'app sullo smartphone, la successiva schermata visualizzata sarà quella di selezione della "città principale"; altrimenti si passerà direttamente alla schermata della mappa.

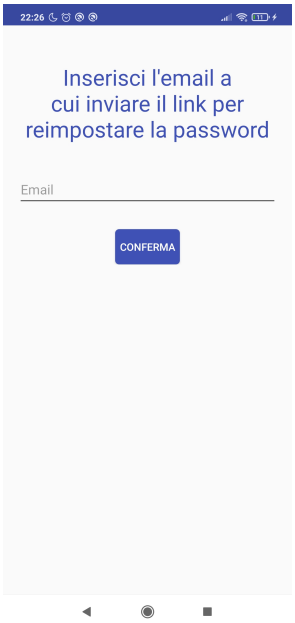

Figura 5.4. Immagine della schermata di reset della password

La prossima schermata che vedremo è quella riportata dalla Figura 5.8. Come possiamo vedere, si tratta della schermata di selezione della "città principale" (la città che verrà visualizzata ogni volta che si passerà alla schermata della mappa). Essa è costituita da un campo di testo in cui inserire il nome della città, un pulsante per attivare la localizzazione tramite GPS del dispositivo, ed un pulsante di conferma. Il nome della città può essere inserita manualmente dall'utente oppure si può premere il tasto GPS (solo nel caso in cui l'utente abbia accettato l'utilizzo del GPS durante l'esecuzione dell'app) in modo da determinare in maniera automatica la città in cui si trova il dispositivo e, così, riempire il campo in maniera automatica. Una volta premuto il tasto di conferma, ed aver individuato le coordinate della città inserita, si passa alla schermata successiva, cioè quella della mappa.

La prossima schermata che vedremo è quella riportata dalla Figura 5.9. Come possiamo vedere, si tratta della schermata della mappa. Questa è la schermata prin-

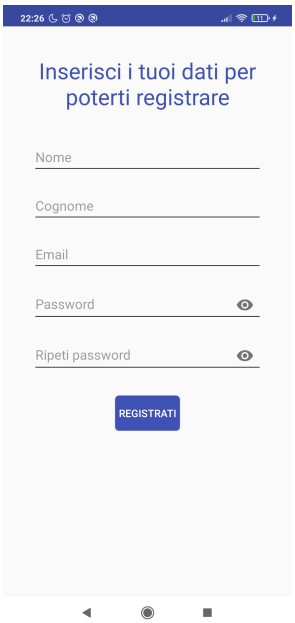

Figura 5.5. Immagine della schermata della registrazione tramite e-mail/password di un nuovo account

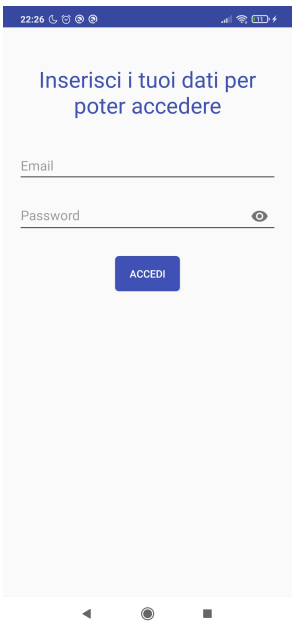

Figura 5.6. Immagine della schermata di Login tramite e-mail/password

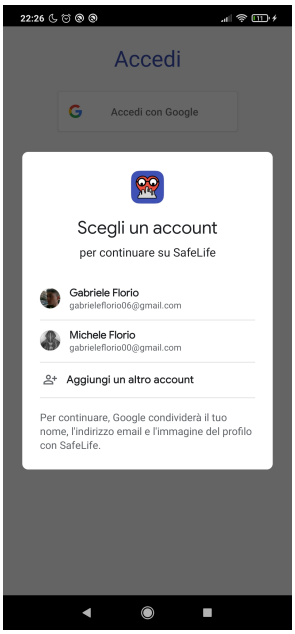

Figura 5.7. Immagine di Login tramite account Google

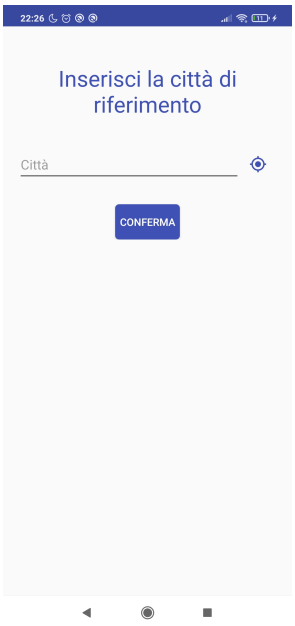

Figura 5.8. Immagine della schermata di selezione della “città principale”

cipale dell'app; da qui è possibile vedere la mappa delle varie città italiane e degli eventi pericolosi che sono stati inseriti dai vari utenti congiuntamente alle segnalazioni ottenute dai giornali di cronaca online. La visuale, inizialmente, è centrata sulla "città principale", ma può essere spostata tramite scroll sul touchscreen del dispositivo o inserendo una città o indirizzo nel campo di testo opportuno o, ancora, usando il GPS per spostare la visuale esattamente sulla posizione del dispositivo.

Gli eventi pericolosi sono visualizzati come marker sulla mappa; premendoli si possono visualizzare le informazioni riguardanti l'evento specifico a cui si riferiscono. Nel caso in cui più marker risultino molto vicini tra loro, questi vengono aggregati al cluster più vicino, con lo scopo di ridurre il carico computazionale dell'app sul dispositivo. I colori dei marker rappresentano i vari livelli di "verificato" degli eventi ottenuti dalle segnalazioni degli utenti; essi partono dal rosso passando poi dall'arancione al giallo al verde, man mano che il livello sale, mentre il colore viola rappresenta dei punti in cui sono presenti più eventi contemporaneamente. Gli eventi ottenuti dagli articoli sono rappresentati tramite l'icona di un giornale.

Sulla mappa sono presenti anche due pulsanti che hanno lo scopo di modificare la vista della mappa stessa, permettendo di passare dalla mappa geografica a quella satellitare, e viceversa, oppure di passare dalla visualizzazione degli eventi tramite marker a quella tramite heatmap, in cui tutti i marker vengono sostituiti da macchie colorate di divere tonalità in base alla gravità dell'evento che rappresentano. È presente, inoltre, un altro pulsante, situato sotto quello del GPS, che permette di aprire il Dialog per selezionare i filtri da applicare alla mappa per consentire la visualizzazione dei soli eventi che rispettano tali vincoli.

Vediamo, adesso, una carrellata di immagini che mostrano alcune delle operazioni consentite dalla schermata (Figure 5.9 - 5.15). Come si vede dalle immagini, è possibile leggere le varie informazioni che costituiscono l'evento, ed è possibile vedere un esempio di visualizzazione della heatmap e del satellite.

Le prossime schermate che vedremo potranno essere accedute dall'utente direttamente dalla schermata della mappa. Nella Figura 5.16 è possibile vedere il Dialog della selezione dei filtri che compare sopra la schermata della mappa; esso è composto da una serie di campi testuali che consentono l'inserimento manuale dei valori. Sono presenti, anche, due Spinner da cui scegliere i filtri da applicare grazie ad una tendina contenente le varie opzioni tra cui scegliere; è presente, infine, anche un pulsante per la scelta dell'intervallo temporale su cui filtrare gli eventi. In fondo al Dialog sono presenti due pulsanti, uno per applicare i filtri selezionati e l'altro per cancellare qualunque filtro precedentemente selezionato.

Nella Figura 5.17 è possibile vedere la schermata che gestisce la lista di eventi presenti su uno stesso punto della mappa. Questa schermata comparirà quando l'utente decide di visualizzare le informazioni sugli eventi presenti in uno stesso punto della mappa, identificati da un marker di colore viola. La lista contiene tutte le informazioni di questi eventi; è anche possibile premere su uno di essi per visualizzare l'immagine allegata all'evento. Nel caso di eventi ottenuti dagli articoli, è anche possibile aprire direttamente l'articolo originale sul browser predefinito, semplicemente tramite pressione prolungata dell'evento.

La prossima schermata che vedremo è quella riportata dalla Figura 5.18. Come possiamo vedere si tratta della schermata che permette di inserire un nuovo evento nel database; questa schermata può essere raggiunta tramite la pressione del pul-

Figura 5.9. Immagine della schermata della mappa con la visualizzazione degli eventi come marker su una mappa geografica

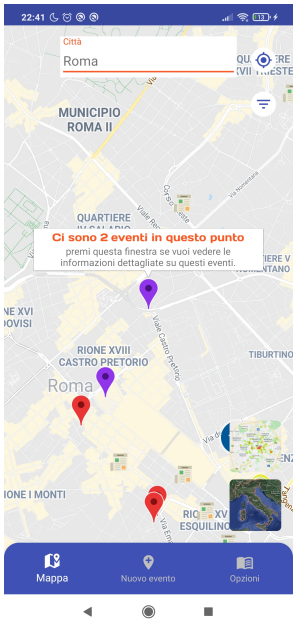

Figura 5.10. Immagine della schermata della mappa in cui è possibile osservare la Info Window di un marker contenente più eventi

Figura 5.11. Immagine della schermata della mappa in cui è possibile osservare la Info Window di un evento ottenuto da una segnalazione utente

Figura 5.12. Immagine della schermata della mappa in cui è possibile osservare la Info Window di un evento ottenuto da un articolo di cronaca

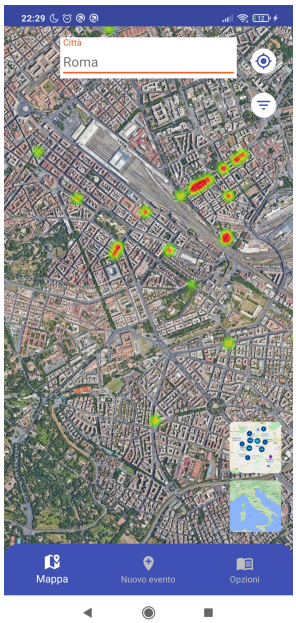

Figura 5.13. Immagine della schermata della mappa con la visualizzazione degli eventi come heatmap su una mappa satellitare

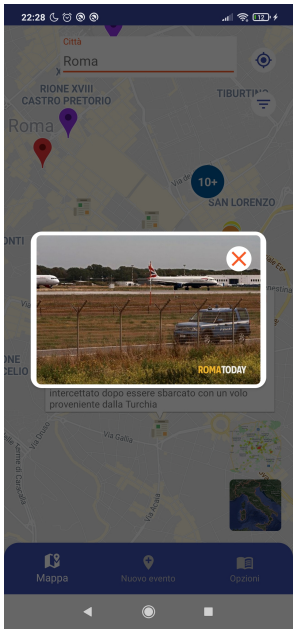

Figura 5.14. Immagine della schermata della mappa in cui viene visualizzata l'immagine di un evento

Figura 5.15. Immagine della schermata della mappa in cui è possibile osservare la Info Window di un evento ottenuto da una segnalazione utente

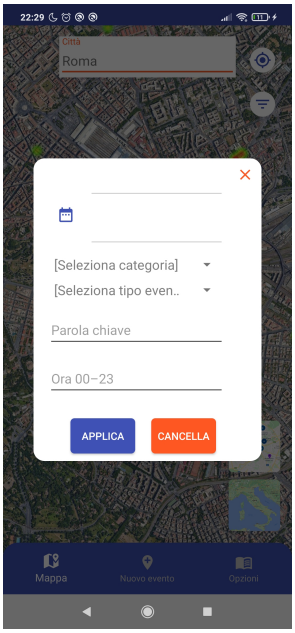

Figura 5.16. Immagine del Dialog che permette di selezionare i filtri da applicare agli eventi visualizzati dalla schermata della mappa

Figura 5.17. Immagine della schermata della lista di eventi multipli presenti in uno stesso punto della mappa

sante corrispondente nella Bottom Navigation View. Per inserire un nuovo evento l'utente deve completare tutti i campi obbligatori e premere il tasto conferma. Alcuni campi, come il luogo, l'ora e la data, possono essere riempiti automaticamente semplicemente premendo il pulsante a fianco ad ognuno di essi. Nelle Figure 5.19 e 5.20 è possibile vedere i Dialog che compaiono premendo i tasti per la selezione della data e dell'ora, mentre, nella Figura 5.21, si vede il Dialog che compare nel caso in cui si sceglie di inserire un'immagine per l'evento. Una volta che l'utente ha premuto il tasto di conferma, e che tutti i campi obbligatori siano completi e validi, viene creato un nuovo evento che verrà aggiunto al database degli eventi. Da quel momento in poi, l'evento sarà visibile a tutti gli utenti; questo, però, avverrà solo nel caso in cui non vengano trovati eventi simili a quello che si sta cercando di inserire. In caso contrario verrà mostrata all'utente una schermata che conterrà i dati dell'evento simile e gli verrà chiesto di decidere se quell'evento è lo stesso della nuova segnalazione oppure no. Nel caso nessuno degli eventi sottoposti all'utente sia un duplicato della nuova segnalazione, allora il nuovo evento viene creato sul database senza problemi; in caso contrario, nessun nuovo evento viene salvato, ma quello duplicato, già sul database, viene incrementato di un livello di "verificato".

La prossima schermata che vedremo è quella riportata dalle Figure 5.22 e 5.23. Come possiamo vedere si tratta della schermata che permette di visualizzare le informazioni degli eventi simili trovati sul database durante l'inserimento di un nuovo evento da parte dell'utente. Siccome il suo funzionamento è già stato descritto quando abbiamo parlato della schermata di inserimento di un nuovo evento, non ci ripeteremo.

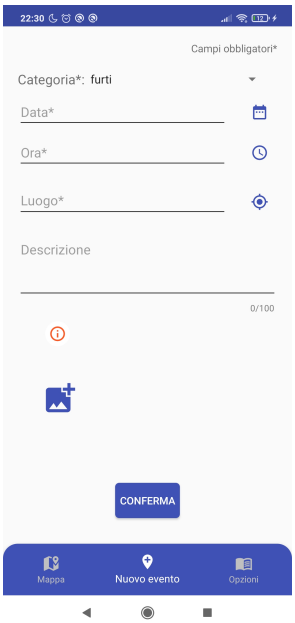

Figura 5.18. Immagine della schermata che consente di inserire un nuovo evento nel database

Figura 5.19. Immagine del Dialog che permette di selezionare una data dal calendario

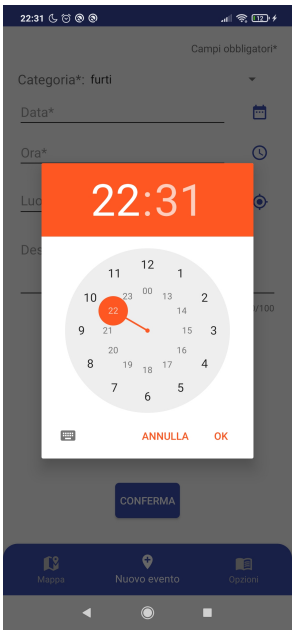

Figura 5.20. Immagine del Dialog che permette di selezionare un orario dall'orologio

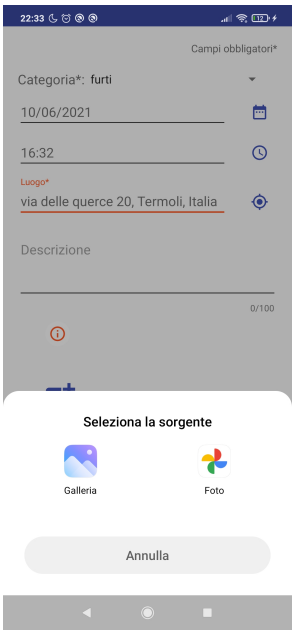

Figura 5.21. Immagine del Dialog che permette di selezionare una fonte da cui scegliere l'immagine da caricare

Figura 5.22. Immagine della schermata che mostra le informazioni sull'evento simile trovato in fase di inserimento di un nuovo evento

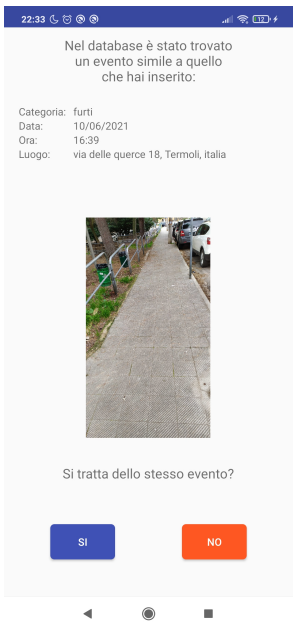

Figura 5.23. Immagine della schermata che mostra le informazioni sull'evento simile trovato in fase di inserimento di un nuovo evento, però in questa immagine vediamo la domanda posta all'utente

La prossima schermata che vedremo è quella riportata dalla Figura 5.24. Come possiamo vedere, si tratta della schermata che permette di scegliere tra le opzioni a disposizione dell'utente. Questa schermata può essere raggiunta tramite la pressione del pulsante corrispondente nella Bottom Navigation View. Essa permette di effettuare delle operazioni di gestione account, come, ad esempio, effettuare il logout, eliminare l'account, cambiare la "città principale", visualizzare l'elenco di tutti gli eventi inseriti dall'utente, ed infine, visualizzare le FAQ dell'app. Vediamo dalle Figure 5.25 e 5.26 le schermate delle opzioni di visualizzazione della lista degli eventi inseriti dall'utente e della lista delle FAQ dell'app, rispettivamente. La prima schermata è molto simile per funzionamento a quella della lista degli eventi presenti sullo stesso punto della mappa che abbiamo già discusso in precedenza; l'unica differenza con essa è che ad essere visualizzati sono solo gli eventi che l'utente ha inserito con l'account attualmente in uso. La seconda schermata, quella delle FAQ, contiene una lista delle domande più richieste dagli utenti e, alla pressione di una di queste domande, compare, al di sotto di esse, la risposta corrispondente. Per quanto riguarda il resto delle opzioni selezionabili dall'utente nell'opportuna schermata, le opzioni di Logout e di eliminazione dell'account riporteranno l'utente alla schermata di selezione del metodo di Login, solo dopo aver effettuato l'operazione descritta. Nel caso in cui l'utente scelga di modificare la "città principale", allora si verrà reindirizzati alla schermata di selezione di questa città. E con quest'ultima operazione abbiamo descritto ogni possibile funzionalità dell'app, per cui si conclude qui questo manuale sul funzionamento dell'app.

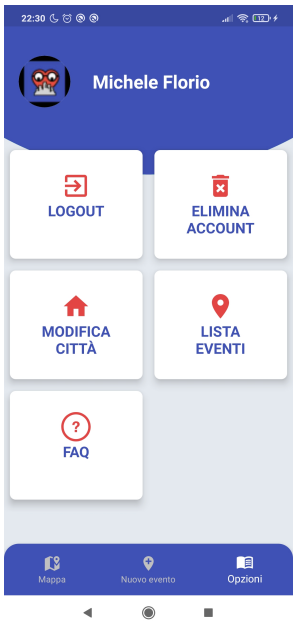

Figura 5.24. Immagine della schermata delle opzioni a disposizione dell'utente

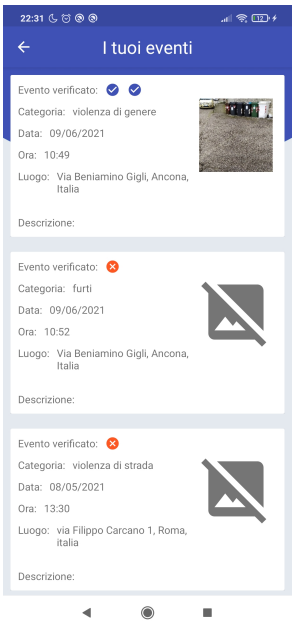

Figura 5.25. Immagine della schermata che mostra la lista degli eventi inseriti dall'utente con questo account

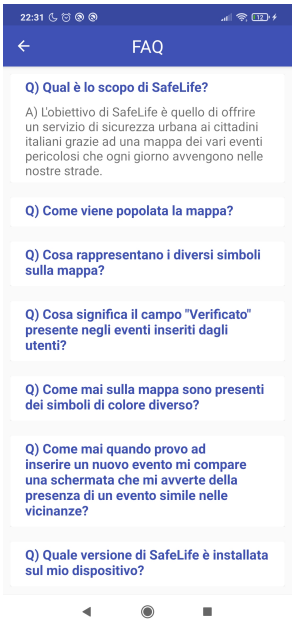

Figura 5.26. Immagine della schermata che mostra le FAQ dell'app

6

Integrazione dell'app con i moduli di SafeLife

In questo capitolo discuteremo dell'ultima fase del lavoro finora presentato, ovvero l'integrazione dell'app con i moduli della piattaforma web di SafeLife

6.1 Il problema dell'integrazione

Per cominciare, illustreremo il problema da risolvere prima di mostrare la soluzione trovata, in conformità con i requisiti funzionali e non. Il problema in questione riguarda la ricerca ed implementazione di un modo per realizzare l'integrazione tra l'applicazione Android di SafeLife ed i moduli della piattaforma web del progetto. In breve, è necessario realizzare una soluzione backend che sia in grado di garantire l'integrazione tra le due piattaforme.

Ricordiamo che la realizzazione di una soluzione di backend (o di un database remoto, come è stato definito durante la descrizione dei requisiti funzionali, nel Capitolo 3) è indispensabile per il soddisfacimento dei requisiti funzionali *R4* e *R5*, definiti nel Capitolo 3 di questa tesi. Entrambi i requisiti, per poter essere soddisfatti, necessitano della creazione di un database remoto che, a sua volta per poter essere gestito, necessita di una struttura di backend adeguata.

Ricordiamo, inoltre, che esiste anche un requisito non funzionale da dover soddisfare riguardante il backend, e cioè che la soluzione deve utilizzare dei servizi commerciali per la gestione del backend. Tale requisito interessa esclusivamente la componente gestionale dell'app, che prevede l'utilizzo di una soluzione esistente per la realizzazione e il mantenimento di un database remoto, insieme alla gestione dei profili di autenticazione degli utenti. Per questo, fin dall'inizio della fase di progettazione, è incominciata la ricerca di un servizio che permettesse la facile gestione di un backend che fosse in grado di gestire sia le informazioni degli utenti che quelle degli eventi pericolosi, e che permettesse l'alimentazione di un database remoto sia tramite le segnalazioni degli utenti, che mediante gli eventi ottenuti dal crawler web.

Un'altra caratteristica fondamentale che deve possedere il servizio è la capacità di effettuare degli aggiornamenti in tempo reale dei valori contenuti nel database. Esso, infine, deve permettere una facile lettura dei dati contenuti anche da parte della piattaforma web del progetto.

6.2 Gli strumenti utilizzati

Varie ricerche sono state compiute per trovare il servizio ideale che avesse le seguenti caratteristiche:

1. facile da usare ed implementare;
2. costi di gestione contenuti;
3. possibilità di memorizzare le informazioni degli eventi su un database remoto;
4. aggiornamento delle informazioni in tempo reale;
5. facilità di lettura delle informazioni dall'esterno;
6. capacità di gestire l'autenticazione degli utenti.

Dopo aver visionato diverse possibili soluzioni, ciascuno con i suoi pro e contro, siamo giunti al servizio che più ci ha convinto per le sue caratteristiche, cioè Firebase di Google.

Firebase è una piattaforma per la creazione di applicazioni sia per dispositivi mobili che web. In particolare, abbiamo deciso di utilizzare alcuni dei servizi offerti dalla piattaforma, in modo da implementare soltanto le caratteristiche che il nostro backend necessita. I servizi che sono stati utilizzati per la gestione del backend sono Cloud Firestore, Cloud Storage e il servizio Authentication di Firebase. I primi due consentono la realizzazione e gestione del database remoto così come è richiesto dai requisiti, mentre l'ultimo servizio permette la creazione e gestione degli account degli utenti. Vediamo, adesso, più nello specifico, i singoli servizi e cosa offrono allo sviluppatore.

Partiamo dal primo, cioè Cloud Firestore. Questo è un servizio che permette di utilizzare il database cloud NoSQL flessibile e scalabile di Firebase per archiviare e sincronizzare i dati per lo sviluppo lato client e server. Cloud Firestore è un database flessibile e scalabile per lo sviluppo di dispositivi mobili, web e server da Firebase e Google Cloud. Esso mantiene i dati sincronizzati tra le app client tramite listener in tempo reale e offre supporto offline per dispositivi mobili e Web in modo da poter creare app reattive che funzionano indipendentemente dalla latenza di rete o dalla connettività Internet. Cloud Firestore offre, anche, una perfetta integrazione con altri prodotti Firebase e Google Cloud. Vediamo, adesso, di esplorare le funzionalità chiave offerte da questo strumento:

- *Operazioni robuste*: il modello di dati Cloud Firestore supporta strutture dati flessibili e gerarchiche. Esso permette di archiviare i dati in documenti e di organizzarli in raccolte. I documenti possono contenere oggetti nidificati complessi, oltre a sottoraccolte.
- *Interrogazione espressiva*: in Cloud Firestore si possono utilizzare le query per recuperare singoli documenti specifici; è possibile, altresì, recuperare tutti i documenti in una raccolta che corrispondono ai nostri parametri di query. Queste ultime possono includere più filtri concatenati e combinare filtri e ordinamenti. Esse sono, anche indicizzate per impostazione predefinita, quindi le loro prestazioni sono proporzionali alla dimensione del set di risultati, e non al set di dati.

- *Aggiornamenti in tempo reale*: Cloud Firestore utilizza la sincronizzazione dei dati per aggiornare i dati su qualsiasi dispositivo connesso. Tuttavia, è anche progettato per eseguire query di recupero semplici e una tantum in modo efficiente.
- *Supporto offline*: Cloud Firestore memorizza nella cache i dati che l'app sta utilizzando attivamente, in modo che essa possa scrivere, leggere, ascoltare ed eseguire query sui dati anche se il dispositivo è offline. Quando il dispositivo torna online, Cloud Firestore sincronizza tutte le modifiche locali al proprio interno.
- *Progettato per scalare*: Cloud Firestore offre il meglio della potente infrastruttura di Google Cloud: replica automatica dei dati in più aree geografiche, solide garanzie di coerenza, operazioni batch atomiche e supporto reale delle transazioni. Esso è stato progettato per gestire i carichi di lavoro necessari per gestire le app più importanti attualmente presenti sul mercato.

Queste sono tutte le caratteristiche offerte dal servizio e le modalità con cui possono essere sfruttate per poter creare e gestire il database contenente le informazioni testuali degli eventi. Per quanto riguarda, invece, le informazioni multimediali allegate alle segnalazioni, nel nostro caso le immagini degli eventi in questione, bisognerà utilizzare un servizio diverso, cioè Cloud Storage, che Firebase mette a disposizione.

Cloud Storage di Firebase è pensato per gli sviluppatori di app che hanno bisogno di archiviare e fornire contenuti generati dagli utenti, come foto o video. Esso è un servizio di archiviazione di oggetti potente, semplice ed economico. Gli SDK Firebase per Cloud Storage aggiungono la sicurezza di Google ai caricamenti e ai download di file per le app Firebase, indipendentemente dalla qualità della rete. È possibile utilizzare i corrispettivi SDK per archiviare immagini, audio, video o altri contenuti generati dagli utenti. Sul server, è possibile utilizzare le API di Google Cloud Storage per accedere agli stessi file. Vediamo, adesso, di esplorare le funzionalità chiave offerte:

- *Flessibilità*: gli SDK Firebase per Cloud Storage eseguono caricamenti e download indipendentemente dalla qualità della rete. I caricamenti e i download sono robusti, il che significa che ricominciano da dove si sono fermati, risparmiando tempo e larghezza di banda agli utenti.
- *Sicurezza forte*: gli SDK Firebase per Cloud Storage si integrano con Firebase Authentication per fornire un'autenticazione semplice e intuitiva per gli sviluppatori. È possibile utilizzare il modello di sicurezza dichiarativa per consentire l'accesso in base a nome del file, dimensione, tipo di contenuto, e altri metadati.
- *Elevata scalabilità*: Cloud Storage è progettato per una scala di exabyte. È possibile passare senza sforzo dal prototipo alla produzione utilizzando la stessa infrastruttura che alimenta Spotify e Google Foto.

Queste sono tutte le caratteristiche offerte dal servizio; come possiamo vedere, esso offre tutto il necessario per poter creare e gestire il database contenente le immagini degli eventi inseriti. Passiamo, adesso, a vedere le caratteristiche dell'ultimo servizio di Firebase utilizzato nello sviluppo dell'app.

Firebase Authentication fornisce servizi di backend, SDK di facile utilizzo e librerie dell'interfaccia utente già pronte per autenticare gli utenti della nostra app.

Supporta l'autenticazione tramite password, numeri di telefono, provider di identità federati popolari, come Google, Facebook e Twitter, e altro ancora. Firebase Authentication si integra perfettamente con altri servizi Firebase e sfrutta standard di settore come OAuth 2.0 e OpenID Connect; quindi, può essere facilmente integrato con il nostro backend personalizzato. Vediamo, adesso, di esplorare i vari metodi di autenticazione offerti:

- *Autenticazione basata su e-mail e password*: autentica gli utenti con i loro indirizzi e-mail e password. L'SDK di autenticazione Firebase fornisce metodi per creare e gestire gli utenti che utilizzano i propri indirizzi e-mail e le proprie password per accedere. L'autenticazione Firebase gestisce anche l'invio di e-mail di reimpostazione della password.
- *Integrazione del provider di identità federato*: autentica gli utenti mediante l'integrazione con provider di identità federati. L'SDK di autenticazione Firebase fornisce metodi che consentono agli utenti di accedere con i propri account Google, Facebook, Twitter e GitHub.
- *Autenticazione del numero di telefono*: autentica gli utenti inviando messaggi SMS ai loro telefoni.
- *Integrazione del sistema di autenticazione personalizzato*: collega il sistema di accesso esistente dell'app all'SDK di autenticazione Firebase consentendo l'accesso a Firebase Realtime Database e ad altri servizi Firebase.
- *Autorizzazione anonima*: utilizza le funzionalità che richiedono l'autenticazione senza richiedere agli utenti di accedere prima creando account anonimi temporanei. Se in seguito l'utente sceglie di registrarsi, è possibile aggiornare l'account anonimo a un account specifico, in modo che l'utente possa continuare da dove si era fermato.

Queste sono tutte le caratteristiche offerte dal servizio; come possiamo vedere, esso offre tutto il necessario per poter creare e gestire gli account di autenticazione degli utenti della nostra app. Per di più, l'integrazione offerta tra i primi due servizi descritti e il servizio di Authentication permetterà la realizzazione delle regole di sicurezza per l'accesso alle informazioni degli eventi, in modo da controllare il più possibile l'accesso al database remoto. Questo conclude il nostro approfondimento riguardante gli strumenti utilizzati per la realizzazione del backend del progetto.

6.3 La soluzione

Dopo aver descritto il problema dell'integrazione, ed aver approfondito i servizi utilizzati per l'implementazione del backend del progetto, costituito dal database remoto e della gestione delle autenticazioni degli utenti, possiamo descrivere la soluzione realizzata. A partire dai servizi di Firebese per la gestione del database, si è ideato un sistema che permettesse all'app di scrivere e leggere le informazioni degli eventi utilizzando le chiamate asincrone ai servizi di Cloud Firestore, per la memorizzazione delle informazioni testuali, e a Cloud Storage, per la memorizzazione delle immagini.

Le chiamate, sia di lettura che di scrittura, da parte dell'app sul database sono già state descritte e mostrate durante il capitolo dedicato all'implementazione del

codice; infatti, molti dei thread che avevamo visto allora, che contenevano nel nome la parola “Worker”, servivano proprio a gestire tali chiamate asincrone, allo scopo di realizzare una sorta di sincronizzazione tra le varie operazioni, grazie all’ausilio di diverse tecniche rivolte a tale scopo. Per questo motivo non le tratteremo di nuovo; piuttosto riteniamo importante spiegare come è stato possibile realizzare l’integrazione dal lato della piattaforma web del progetto.

Per permettere l’integrazione tra le due piattaforme era necessario che entrambe le parti fossero capaci di effettuare liberamente operazioni di lettura e scrittura sul database remoto; per quanto riguarda la comunicazione lato app è già stato detto tutto; dal lato della piattaforma web, era necessario trovare una soluzione alternativa a quella trovata per la parte su Android. Infatti, la soluzione trovata per l’app è utilizzabile solo per applicazioni Android, mentre per la piattaforma web è stato necessario cambiare approccio. Per questo motivo abbiamo cercato un modo per mettere in comunicazione il database con la parte web; le nostre ricerche hanno portato ad impiegare come soluzione le API REST messe a disposizione dei servizi Firestore e Storage.

Le API REST utilizzate permettono il recupero delle informazioni contenute nel database, ed anche l’aggiunta di nuove informazioni, semplicemente inviando una richiesta HTTP ai servizi, specificando il tipo della richiesta stessa, le informazioni coinvolte e l’operazione da svolgere. Tale approccio ha permesso di realizzare un piccolo script in Java che consentisse di effettuare tutti i controlli necessari durante l’inserimento di un nuovo evento ottenuto, tramite il crawler web, realizzato da Nobili Martina e Faramondi Luca del Campus Bio-Medico di Roma, con cui abbiamo collaborato.

Il codice in questione è stato realizzato in collaborazione con Nobili Martina; esso permette l’inserimento degli eventi ottenuti dal crawler in modo tale che, prima che tale operazione sia effettuata, vengano cercate nel database le segnalazioni di utenti dell’app che risultano simili ad esse, (quindi vengono cercati gli eventi degli utenti che abbiano la stessa categoria e la stessa data dell’evento da inserire e che siano all’interno di un raggio di 1 Km da esso) in modo tale da poter incrementare il valore del campo “verificato”, portandolo al livello massimo, cioè 3. Ciò serve ad indicare agli utenti che le segnalazioni che erano già presenti nel database, e che riportavano la stessa notizia dell’evento descritto dall’articolo appena inserito, erano vere, grazie al riscontro avuto con l’articolo di cronaca corrispondente che lo certifica.

Per poter realizzare questo codice è necessario avere a disposizione delle API che permettano non solo di effettuare delle semplici operazioni di lettura e scrittura dei documenti sul database, ma anche delle interrogazioni complesse, per ridurre il numero di documenti restituiti, limitandoci ad ottenere solo quelli davvero necessari, in modo tale da ridurre i tempi di risposta ed i costi associati ad ogni documento.

Prima di passare ad esporre il funzionamento delle API utilizzate, cominciamo con il mostrare la struttura dei documenti e delle raccolte all’interno del database. La prima struttura ad essere analizzata è quella dei documenti contenuti all’interno di Cloud Firestore. Nella Figura 6.1 possiamo vedere la radice delle raccolte di documenti all’interno di Firestore, insieme con gli articoli salvati sotto forma di documenti all’interno della raccolta “articoli”; ciascun documento ha, come nome identificativo, un codice generato automaticamente da Firebase.

Se, invece, ci spostiamo sulla raccolta “utenti” vediamo che, al suo interno, sono

presenti dei documenti denominati con l'UID dei vari utenti che hanno utilizzato l'app, ciascuno dei quali contiene un solo campo usato come contatore, come possiamo vedere dalla Figura 6.2. Tale contatore, inizializzato a 0, viene incrementato dall'app ogni volta che l'utente associato all'UID del documento in questione inserisce una nuova segnalazione nel database.

Oltre a questi documenti, nella raccolta, è presente, anche, un documento chiamato "dati", che contiene al proprio interno una raccolta nidificata, chiamata "eventi", come mostrato dalla Figura 6.3. All'interno di quest'altra raccolta sono contenuti i documenti che rappresentano i singoli eventi degli utenti, ciascuno dei quali è identificato da un nome composto dall'UID dell'utente seguito dal simbolo "-", a sua volta seguito dal valore del contatore associato all'utente visto poco fa. Entrando ancora di più nello specifico, possiamo vedere un esempio della struttura interna al singolo documento che rappresenta una segnalazione da parte dell'utente, come mostrato nella Figura 6.4.

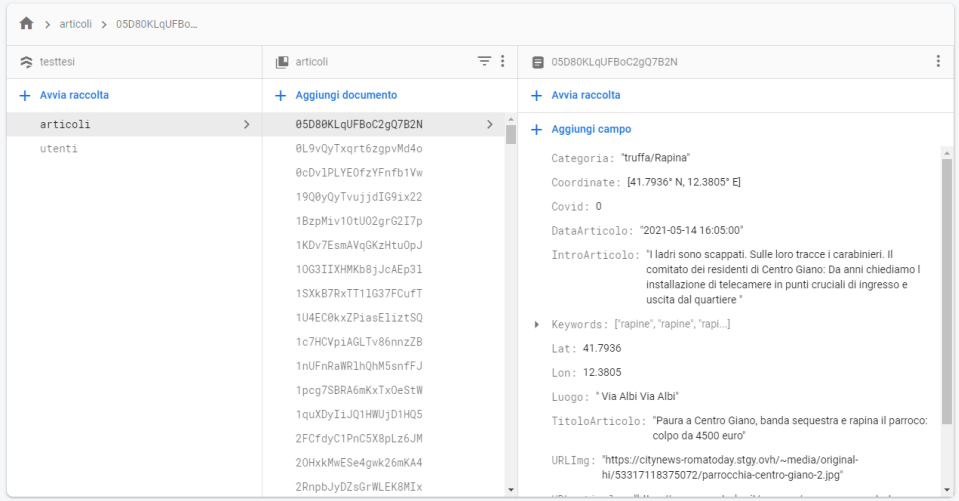

Figura 6.1. Struttura della raccolta "articoli" presente su Cloud Firestore

Passiamo, adesso, ad illustrare brevemente la struttura del database contenuta su Cloud Storage per l'archiviazione delle immagini degli eventi. Nella Figura 6.5 possiamo vedere la directory più esterna che costituisce la struttura del database di Storage chiamata "utenti"; al suo interno troviamo una serie di altre raccolte, ognuna denominata con l'UID dell'utente di cui contengono le immagini, così come mostrato dalla Figura 6.6.

Dando uno sguardo al contenuto di una di queste raccolte (Figura 6.7), possiamo vedere che contengono delle immagini, una rinominata "1", che rappresenta l'immagine originale dell'evento numero 1 inserito dall'utente con lo stesso UID che dà il nome alla raccolta. Oltre all'immagine originale, è presente anche l'immagine di thumbnail della stessa, chiamata "1thumb", che, come si intuisce dal nome, sta a identificare che si tratta della thumbnail dell'immagine numero 1 inserita dall'u-

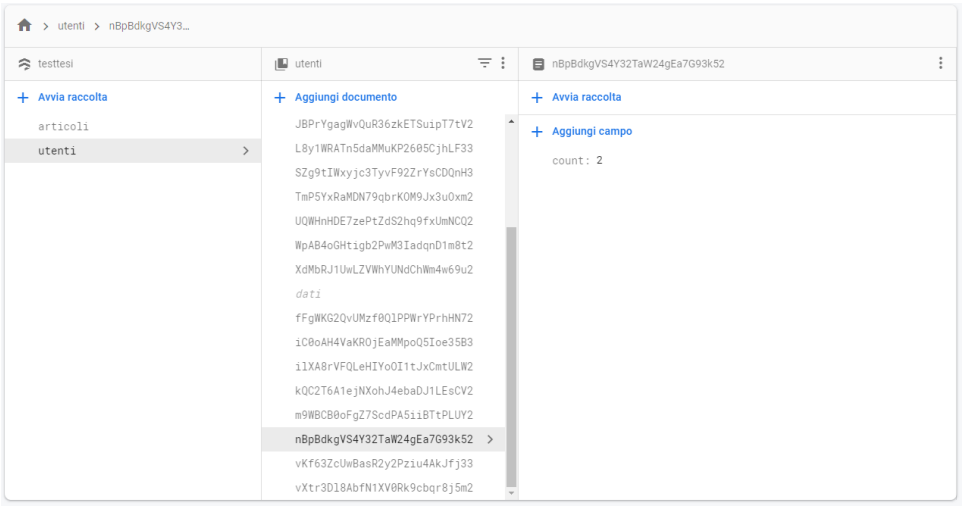

Figura 6.2. Struttura della raccolta "utenti" con all'interno i documenti contenenti il contatore (Cloud Firestore)

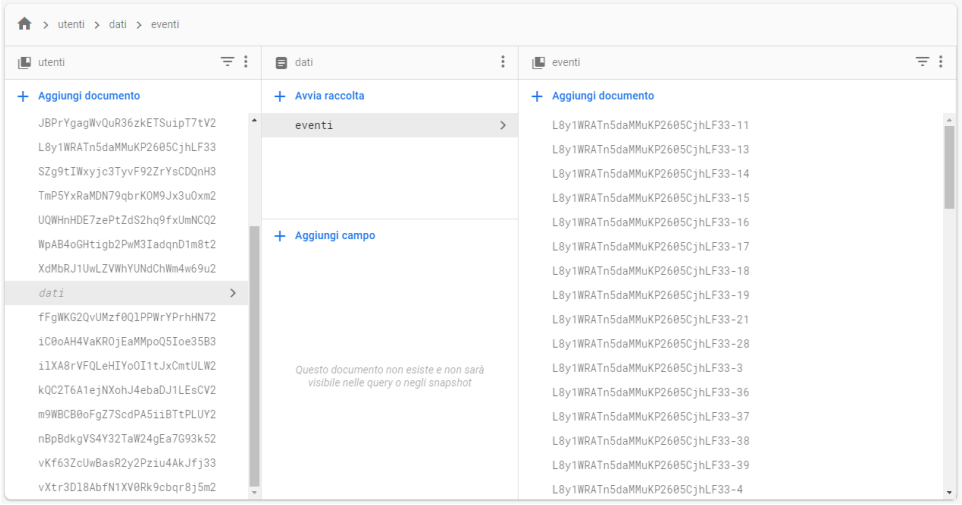

Figura 6.3. Struttura interna della raccolta "eventi" presente dentro il documento "dati" (Cloud Firestore)

tente. In fondo poi, possiamo vedere un'altra immagine, chiamata "pic"; questa è l'immagine di profilo impostata dall'utente dalla schermata delle opzioni dell'app.

Adesso che abbiamo terminato di esporre la struttura dei dati all'interno del database, passiamo a discutere delle API REST utilizzate per realizzare il codice necessario all'inserimento degli eventi estratti dal web. Le API che vedremo saranno espresse nel formato necessario a compiere una chiamata tramite lo strumento da linea di comando, *cURL*, usato per inviare o ricevere file attraverso la sintassi URL.

Incominciamo con l'osservare la prima delle API utilizzate, cioè l'API necessa-

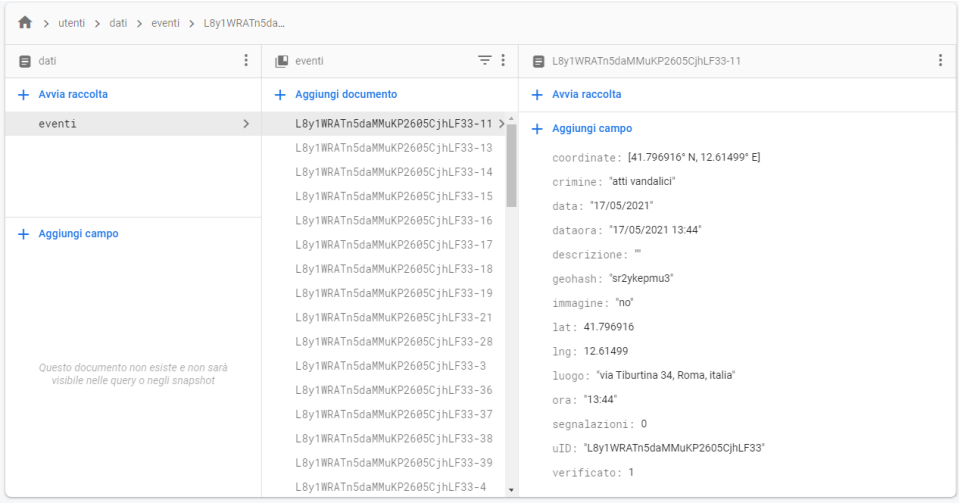

Figura 6.4. Struttura del documento di un evento di un utente salvato su Cloud Firestore

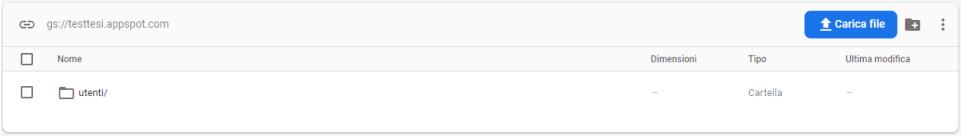

Figura 6.5. Raccolta esterna delle immagini degli eventi presente su Cloud Storage

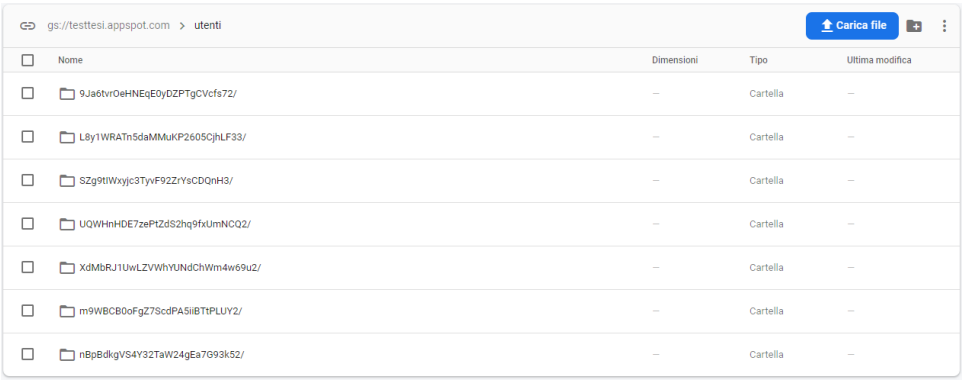

Figura 6.6. Struttura della raccolta "utenti" con all'interno altre raccolte divise per utenti (Cloud Storage)

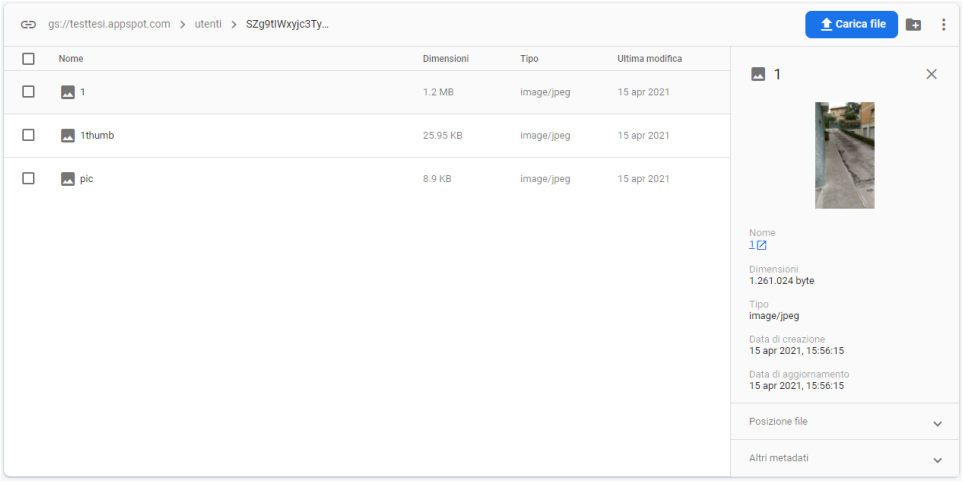

Figura 6.7. Elenco delle immagini di un utente salvate su Cloud Storage

ria a recuperare i documenti degli eventi degli utenti che soddisfano le condizioni imposte, utilizzando una query per operare un primo filtraggio sui dati restituiti. Tale query ha la seguente struttura:

curl –request POST "https://firestore.googleapis.com/v1/projects/testtesi/databases/(default)/documents/utenti/dati:runQuery?key=KEY" –header "Accept: application/json" –header "Content-Type: application/json" -d "@query10.json"

Come possiamo vedere, essa è composta da veri elementi; il primo è il metodo scelto, nel nostro caso *POST*, seguito dal path della raccolta nella quale vogliamo applicare la query stessa, nel nostro caso *"https://firestore.googleapis.com/v1/projects/testtesi/databases/(default)/documents/utenti/dati:runQuery?key=KEY"*.

Come possiamo notare, il path punta alla raccolta interna "utenti" che, come abbiamo descritto in precedenza, contiene i documenti delle segnalazioni degli utenti; alla fine del path è presente una variabile da dover inserire, cioè il valore di *key*. Tale valore è il token associato al progetto da Firebase una volta che viene che viene creato; quindi, essendo univoco ad ogni progetto, va sostituito il placeholder *KEY* con la propria chiave.

Le opzioni specificate nella richiesta *cURL* in esame servono a definire che la chiamata all'API dovrà avere un file JSON di input contenente la query da dover sottoporre a Firestore. Vediamo, adesso, un esempio di codice contenuto nella query, così come mostrato dal Listato 6.1.

```
1  {
2    "structuredQuery": {
3      "select": {
4        "fields": [
5          {
6            "fieldPath": "data"
7          }
8        ]
9      },
10     "from": [
11       {
12         "collectionId": "eventi",
```

```
13          "allDescendants": false
14        }
15      ],
16      "where": {
17        "compositeFilter": {
18          "op": "AND",
19          "filters": [
20            {
21              "fieldFilter": {
22                "field": {
23                  "fieldPath": "crimine"
24                },
25                "op": "EQUAL",
26                "value": {
27                  "stringValue": "sparatoria"
28                }
29              }
30            },
31            {
32              "fieldFilter": {
33                "field": {
34                  "fieldPath": "coordinate"
35                },
36                "op": "GREATER_THAN_OR_EQUAL",
37                "value": {
38                  "geoPointValue": {
39                    "latitude": 41,
40                    "longitude": 14
41                  }
42                }
43              }
44            },
45            {
46              "fieldFilter": {
47                "field": {
48                  "fieldPath": "coordinate"
49                },
50                "op": "LESS_THAN_OR_EQUAL",
51                "value": {
52                  "geoPointValue": {
53                    "latitude": 42.05,
54                    "longitude": 15.05
55                  }
56                }
57              }
58            }
59          ]
60        }
61      },
62      "orderBy": [
63        {
64          "field": {
65            "fieldPath": "coordinate"
66          },
67          "direction": "ASCENDING"
68        }
69      ]
70    }
71  }
```

Listato 6.1. Codice del file JSON "query10"

Il file mostrato è composto da tre elementi principali: i campi restituiti, la query vera e propria e l'ordinamento dei risultati.

Partendo con ordine, abbiamo, all'inizio del file, la parte di codice che stabilisce quali campi ottenere come risultato dell'interrogazione del database (righe 3-9). Il codice successivo, invece, si occupa della realizzazione vera e propria della query; nel nostro caso è richiesto che vengano restituiti i soli documenti appartenenti alla categoria specificata (nel nostro codice, identificata dal campo "crimine") e che si trovino nell'area geografica racchiusa in un quadrato, di cui viene definito il vertice in alto a destra e quello in basso a sinistra (righe 16-61). Quindi per realizzare questa query geografica, si effettua un controllo sulla variabile "coordinate", che contiene le coordinate geografiche del punto in cui è avvenuto l'evento, in modo tale che il punto risulti minore del vertice in alto a destra maggiore del vertice in basso a sinistra dell'area in cui cercare.

L'ultima parte del codice, invece, serve a specificare l'ordinamento dei risultati restituiti; nel nostro caso, l'ordinamento è effettuato sul campo "coordinate", per

ordinare in maniera crescente i risultati in base a quel valore (righe 62-69).

Come è facile notare, la sola query non basta ad effettuare tutti i controlli necessari per identificare gli eventi simili; infatti, Firestore non permette di creare query così complesse. Per ovviare a tale inconveniente, il codice che si occupa dell'inserimento degli eventi ottenuti dagli articoli effettuerà l'ultimo controllo sulla data degli eventi ottenuti come risultato della query, per ottenere, così, la lista degli eventi simili.

Una volta effettuati tutti i controlli, il codice otterrà il nome dei documenti ottenuti come risultato per poter aggiornare il valore del campo "verificato" di questi eventi, portandolo al valore massimo, cioè 3. Di seguito è riportata l'API utilizzata per realizzare questa operazione:

curl –request PATCH "https://firestore.googleapis.com/v1/projects/testtesi/databases/(default)/documents/utenti/dati/eventi/nBpBdkgVS4Y32TaW24gEa7G93k52-1?updateMask.fieldPaths=verificato&key=KEY" –header "Accept: application/json" –header "Content-Type: application/json" -d "@request4.json"

Come possiamo vedere, essa è molto simile alla precedente, per cui descriveremo soltanto le differenze.

La prima differenza sta nel metodo, in questo caso è *PATCH*. La differenza sostanziale è, però, il path dell'API; infatti questa va più in profondità rispetto alla precedente, perchè punta ad uno specifico documento che, in questo esempio ha il nome *nBpBdkgVS4Y32TaW24gEa7G93k52-1*.

Un'altra differenza presente nel path sta nelle variabili alla fine dello stesso infatti, in questa, è presente la variabile *updateMask.fieldPaths* il cui valore indica il campo del documento che deve essere aggiornato. Osserviamo, adesso, il codice JSON del file *equest4.json* mostrato dal Listato 6.2.

```
1  {
2    "fields": {
3      "verificato": {
4        "integerValue": 3
5      }
6    }
7  }
```

Listato 6.2. Codice del file JSON "request4"

Dal codice vediamo che il file è molto piccolo ed ha lo scopo di specificare il valore da impostare per il campo "verificato".

Passiamo, adesso, all'ultima API utilizzata per la realizzazione dell'algoritmo per l'inserimento dei nuovi eventi ottenuti dal crawler. Essa ha la seguente forma:

curl –request POST "https://firestore.googleapis.com/v1/projects/testtesi/databases/(default)/documents/articoli?key=KEY" –header "Accept: application/json" –header "Content-Type: application/json" -d "@eve186.json"

Questa API si differenzia dalle precedenti per il path contenuto; infatti a differenza delle altre, essa ha il path che si conclude all'interno della raccolta "articoli" che, per l'appunto, contiene i documenti riguardanti gli eventi ottenuti a partire dagli articoli.

Il file JSON passato all'API, a differenza degli altri, non contiene un'interrogazione o un valore da modificare, bensì i campi ed i valori associati ad essi, che permettono di descrivere l'evento a cui fanno riferimento.

Con questo abbiamo concluso l'analisi delle varie API utilizzate per la realizzazione del codice che permette l'inserimento dei nuovi eventi ottenuti dagli articoli di cronaca. Prima di concludere questo capitolo, però, dobbiamo ancora discutere del controllo degli accessi effettuati al database. Infatti, sia Cloud Firestore che Storage permettono di realizzare un controllo sugli accessi abbastanza complesso, in modo da ridurre le possibilità di un uso improprio del database. È importante sottolineare, anche, il grande aiuto offerto in questo ambito dall'impiego del servizio di Authentication fornito da Firebase, in quanto questo servizio offre una grande sinergia con la configurazione di questi controlli degli accessi.

Incominciamo con l'esporre i controlli impostati per l'accesso al servizio Firestore, così come sono raffigurati nella Figura 6.8.

Figura 6.8. Controlli sugli accessi al servizio Cloud Firestore

Analizzando i controlli, possiamo vedere come il codice che va dalla riga 4 alla riga 7 serve ad impostare un controllo sulla raccolta "articoli", distinto in controlli sulla lettura e sulla scrittura di documenti. Infatti, si può vedere che la lettura è sempre permessa,purchè l'utente si sia autenticato usando il servizio di Authentication, prima di effettuare la richiesta di lettura. Invece, per quanto riguarda le operazioni di scrittura, non solo è richiesta l'autenticazione dell'utente, ma soltanto un particolare utente (l'amministratore), ovvero colui che possiede lo stesso valore dello UID impostato, può effettuare tali operazioni. Come abbiamo già accennato, l'UID è un codice univoco assegnato all'account degli utenti che si autenticano usando il servizio di Authentication offerto da Firebase.

Il codice che va dalla riga 8 alla 9, invece, permette l'accesso in lettura di tutti i documenti contenuti nel database, purchè l'utente sia autenticato. Vale la pena far notare che, in presenza di molteplici controlli sulle stesse operazioni e raccolte contenute nel database, il controllo più stringente viene applicato a seguito di tali operazioni.

Il codice che va dalla riga 12 alla 14 serve ad implementare il controllo sulle operazioni di scrittura sui documenti contenuti all'interno della raccolta "utenti" che, come abbiamo discusso in precedenza, contiene una serie di documenti identificati dallo UID degli utenti dell'app, con, al loro interno, un contatore. Questo controllo permette solo agli utenti autenticati di modificare il valore del contatore del documento associato al proprio UID.

Il codice che va dalla riga 15 alla 18, invece, permette all'utente di effettuare le operazioni di creazione o modifica di un documento all'interno della raccolta interna "eventi". Tali operazioni sono permesse solo agli utenti autenticati, in particolare, per le operazioni di creazione di un nuovo documento, viene concesso all'utente il permesso di creare un nuovo documento associato al proprio account, quindi un utente non può creare documenti associati ad account diversi dal proprio.

Adesso passiamo, invece, a mostrare i controlli impostati per l'accesso al servizio di Storage di Firebase, così come sono raffigurati dalla Figura 6.9.

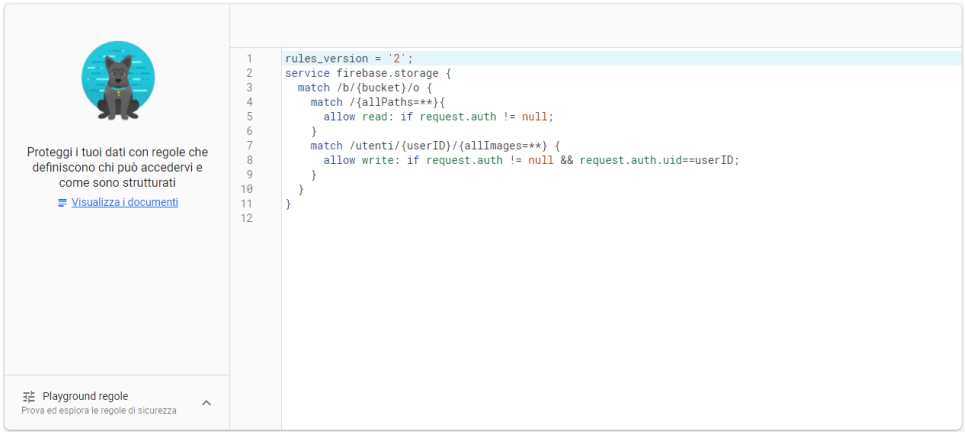

Figura 6.9. Controlli sugli accessi al servizio Cloud Storage

Analizzando i controlli, possiamo vedere come il codice alla riga 5 serve ad impostare un controllo sulla lettura delle immagini contenute nel database. Tale controllo permette la lettura delle immagini a tutti gli utenti che si sono prima autenticati.

Il codice che va dalla riga 7 alla 9 serve ad impostare la scrittura di nuove immagini all'interno della raccolta che possiede lo stesso UID dell'utente autenticato. Esso impedisce, quindi, che l'utente possa inserire le proprie immagini all'interno della raccolta di un altro utente.

Con questo ultimo argomento, si conclude sia l'analisi dei controlli effettuati sugli accessi al database che il capitolo stesso.

7

Discussione in merito al lavoro svolto

Nel capitolo corrente verrà proposto un riepilogo di quanto svolto nel periodo di tirocinio, per poi passare alle considerazioni personali.

7.1 Considerazioni riguardo l'esperienza di tirocinio

Il tirocinio extracurriculare presso la Res On Network ha avuto una durata complessiva di sei mesi. Il periodo precedente al tirocinio è servito allo studio dei fondamenti dello sviluppo di un'app Android in Java. Mentre, durante il periodo di tirocinio, soprattutto i primi due/tre mesi, il tempo a disposizione è stato impiegato nella ricerca delle soluzioni da utilizzare per soddisfare i requisiti concordati in fase di progettazione, e altrettanto tempo è stato impiegato nell'applicazione pratica di queste soluzioni. Sempre nello stesso periodo, un tempo congruo è stato dedicato alla realizzazione del materiale necessario ad illustrare agli Stakeholder le funzionalità e l'obiettivo del progetto. La parte restante del tirocinio è stata impiegata per l'implementazione di alcune aggiunte al progetto, la realizzazione, a scopo dimostrativo, di altro materiale concernente il lavoro svolto man mano che si procedeva nello sviluppo (delle demo dell'app), il miglioramento delle soluzioni precedentemente trovate e la pulizia del codice comprendente la rimozione dei bug riscontrati. In particolare, gli argomenti trattati durante il tirocinio sono stati i seguenti:

- lo studio del software di sviluppo utilizzato, ovvero Android Studio;
- la ricerca dei servizi necessari al soddisfacimento dei requisiti del progetto;
- l'implementazione dei servizi scelti, Cloud Firestore, Cloud Storage e Authentication;
- la programmazione Java avanzata, finalizzata allo sviluppo di un'app Andorid;
- lo studio delle API necessarie alla creazione del codice necessario per l'alimentazione del database con gli eventi ottenuti dal crawler;
- la realizzazione di demo e prototipi dell'app.

Alcune di tali nozioni sono state fornite dai professori con cui ho collaborato durante la realizzazione del progetto, il professore Faramondi Luca e il professore Bonifazi Gianluca. Mentre il resto delle nozioni sono state acquisite sul campo, andando

avanti con lo sviluppo dell'app, approfondendo sempre di più le mie conoscenze in materia.

Gli ultimi due mesi, soprattutto, sono stati dedicati all'attività di integrazione dell'app con la piattaforma web del progetto e per questo motivo la collaborazione con il team del Campus Bio-Medico di Roma, che si è occupato della parte web di SafeLife e dell'estrazione degli articoli di giornali web tramite l'utilizzo del crawler, si è fatta più stretta. Grazie a tale collaborazione ed interazione tra le due parti, siamo riusciti nella realizzazione del codice che ha permesso l'integrazione tra le due parti del progetto.

7.2 Considerazioni personali

Lavorare all'argomento del tirocinio e di tesi è stato molto interessante, in quanto parte di una serie di sviluppi del tutto nuovi all'interno dell'applicativo. Inoltre, questa prima esperienza in un ambiente lavorativo è stata per me valorizzante, in quanto mi ha consentito di accrescere le conoscenze tecniche legate al mondo dell'ingegneria del software. Oltre a ciò, ho avuto modo di sperimentare cosa significa far parte di un'azienda, sia dal punto di vista lavorativo che dal punto di vista umano, anche nelle occasioni in cui sono stato costretto al lavoro a distanza dovuto all'attuale situazione della pandemia.

8

Conclusioni

In questo ultimo capitolo saranno tratte le conclusioni riguardanti il lavoro svolto; verranno, inoltre, esposte i possibili miglioramenti da apportare in futuro.

8.1 Conclusioni sul lavoro svolto

In questa tesi sono stati presentati lo sviluppo dell'app Android del progetto SafeLife e la realizzazione di una soluzione che consentisse l'integrazione con la parte web del progetto.

Nella prima parte si è descritto il progetto SafeLife in esame, distinguendo le due parti che lo compongono ovvero, la piattaforma web e l'app Android collegata. Abbiamo descritto la necessità che ha portato all'ideazione del progetto ed anche dell'obiettivo che si intende raggiungere tramite la sua realizzazione; inoltre, abbiamo discusso le varie caratteristiche e funzionalità previste, da realizzare per entrambe le piattaforme.

Un altro argomento discusso ha riguardato il crawler web, che permette il recupero delle informazioni degli eventi dal web, tramite l'analisi dei giornali di cronaca online, consentendo di accedere alle informazioni disponibili sul web. L'app è stata ideata con lo scopo principale di fornire uno strumento che permetta all'utente di effettuare delle segnalazioni in maniera semplice ed immediata. Al tempo stesso, esso fornisce la possibilità di visualizzare gli eventi pericolosi presenti in zona.

Terminata la parte introduttiva si è passati all'analisi dell'argomento centrale del lavoro di tesi, iniziando dalla stesura dei requisiti, effettuata tenendo conto degli obiettivi prefissati e del tempo a disposizione, per poi proseguire con la progettazione delle classi che compongono il codice dell'app, grazie anche all'ausilio dei diagrammi utilizzati nell'ingegneria del software. È stata, quindi, descritta l'implementazione di quanto progettato, basata sull'utilizzo dell'IDE Android Studio per lo sviluppo e il test del codice che realizza il funzionamento dell'app lato frontend.

Infine, abbiamo affrontato il problema dell'integrazione tra le due parti del progetto ed abbiamo descritto la soluzione trovata illustrando i servizi di Firebase utilizzati per realizzare il backend che ha permesso di soddisfare i requisiti associati a questa problematica.

8.2 Sviluppi futuri

Il lavoro svolto sull'app finora, benchè rappresenti un buon punto di partenza, non è ancora completo di tutte le caratteristiche previste in fase di definizione degli obiettivi del progetto; infatti, la scelta delle caratteristiche da sviluppare è dovuta alla durata del tirocinio. Il progetto è, infatti, ambizioso e si pone un obiettivo molto complesso, che lascia anche spazio ad aggiornamenti futuri per l'aggiunta di eventuali altre caratteristiche.

Per questi motivi, gli obiettivi che sono stati fissati per un successivo sviluppo, che permetta il completamento del progetto, prevedono:

- in primo luogo, l'aggiunta della feature di pianificazione di un percorso sulla mappa, a partire da una destinazione scelta dall'utente, che consenta di avere informazioni riguardanti gli eventi pericolosi che si incrociano durante il tragitto ed, eventualmente, suggerisca dei percorsi alternativi.
- in secondo luogo, il passaggio del database (adesso realizzato tramite i servizi di Firebase) su server privati di proprietà della Res On Network stessa, in modo da ridurre i costi di gestione.

Riferimenti bibliografici

1. Enrique Estellés-Arolas, Fernando González-Ladrón-de-Guevara. *Towards an integrated crowdsourcing definition.* Journal of Information Science, 2012.
2. Daren C. Brabham. *Crowdsourcing as a Model for Problem Solving: An Introduction and Cases.* Convergence: The International Journal of Research into New Media Technologies, 2008.
3. ASSOCIAZIONE CONTROLLO DEL VICINATO - ACDV. `https://www.acdvevents.it/`, 2018.
4. Caccetta F. *Il controllo di vicinato. Manuale pratico per la creazione e gestione dei gruppi di controllo di vicinato.* MGC Edizioni, 2019.
5. Caccetta F. *L'occasione fa bene al ladro. Il controllo del vicinato.* MGC Edizioni, 2015.
6. Conversione in legge, con modificazioni, del decreto- legge 20 febbraio 2017, n. 14, recante disposizioni urgenti in materia di sicurezza delle città. `https://www.gazzettaufficiale.it/eli/id/2017/06/20/17G00102/sg`, 2017.
7. European Neighbourhood Watch Association. `https://eunwa.org/`, 2014.
8. Robert C. Martin. *UML for Java Programmers.* Addison Wesley, 2003.
9. Firebase Cloud Firestore. `https://firebase.google.com/docs/firestore`, 2019.
10. Firebase Cloud Storage. `https://firebase.google.com/docs/storage`, 2012.
11. Firebase Authentication. `https://firebase.google.com/docs/auth`, 2012.
12. Cloud Firestore API. `https://firebase.google.com/docs/firestore/reference/rest`, 2019.
13. Android Studio. `https://developer.android.com/studio/intro`, 2014.
14. T. Hagos. *Learn Android Studio 3 – Efficient Android Development.* Apress, 2018.
15. IntelliJ IDEA. `https://www.jetbrains.com/idea/`, 2001.
16. Maps SDK for Android. `https://developers.google.com/maps/documentation/android-sdk/overview`, 2015.
17. MapQuest Developer open geocoding API. `https://developer.mapquest.com/documentation/open/geocoding-api/`, 1996.
18. Harvey M. Deitel, Paul J. Deitel. *Programmare in Java.* Pearson, 2020.

Ringraziamenti

Il primo ringraziamento va alla mia famiglia, che mi ha dato la possibilità di intraprendere e portare a termine questo percorso universitario.

Vorrei ringraziare il prof. Ursino, per aver accettato di essere il mio relatore, per essersi speso nell'aiutarmi a trovare il percorso di tirocinio più adatto alle mie esigenze e per la straordinaria disponibilità dimostrata nel corso della stesura di questa tesi.

Ringrazio il prof. Faramondi e Martina del Campus Bio-Medico di Roma per avermi assistito durante il progetto, in particolar modo durante la fase di integrazione dell'app con la piattaforma web. Li ringrazio ancora una volta per avermi fornito il materiale a loro disposizione che mi ha permesso di realizzare la stesura dei primi due capitoli di questa tesi.

Infine, ringrazio Marco Santarelli, Direttore Scientifico della Res On Network, per avermi offerto l'opportunità di lavorare nell'ambito della "Security and Global Defence" durante questi ultimi sei mesi e ringrazio Valentina e Noemi della Fondazione Margherita Hack per avermi aiutato durante il mio tirocinio a Giulianova.