UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea triennale in Ingegneria Informatica e dell'automazione

---

Progettazione e implementazione di un'app Android per l'organizzazione di party nelle abitazioni nell'era del Covid-19

Design and implementation of an Android app to organize home parties in the Covid-19 era

Candidato:
Andrea Lapolla

Relatore:
Prof. Domenico Ursino

Anno Accademico 2019-2020

Ringraziamenti

Arrivati al termine di questo lavoro ed anche a conclusione di questo mio primo traguardo universitario, considero necessario rivolgere un sentito “grazie” a quelle persone che, insieme a me, sono state protagoniste di questa avventura.
Primi fra tutti i miei genitori che, con il loro supporto e la loro vicinanza, hanno creato le giuste condizioni di serenità e tranquillità che mi hanno consentito di affrontare positivamente le difficoltà che, inevitabilmente, ho incontrato in questo cammino.
Un ringraziamento va, anche, a tutti gli amici che ho incontrato in questa prima fase di studi, condividendone la quotidianità delle lezioni e le “angosce” degli esami; anche se sono consapevole, con rammarico, che perderò inevitabilmente le tracce di alcuni di loro, dal momento che le nostre vite prenderanno probabilmente percorsi diversi, essi sono stati fondamentali per stemperare le difficoltà e le ansie di questi tre anni trascorsi da studente fuori sede lontano da casa e dagli affetti.
Una menzione particolare va al Prof. Domenico Ursino, relatore di questa tesi, ed al suo collaboratore Enrico Corradini che, con la loro incredibile disponibilità, mostrata sia durante il periodo delle lezioni, tenute regolarmente anche durante il lockdown, sia, successivamente, durante l’intero svolgimento di questo progetto, ne hanno reso possibile la realizzazione nonostante il difficile periodo che abbiamo vissuto.

Ancona, Agosto 2020

Andrea Lapolla

Indice

Elenco delle figure

Elenco dei listati

Introduzione

Sin dalla nascita dei primi smartphone si sente parlare della parola “app”. Si tratta di una semplice abbreviazione del termine anglosassone “application”, vale a dire “applicazione”.
La grande diffusione e il successo di questo nuovo modo di proporre tecnologia ha prodotto infinite app da poter installare sul nostro cellulare. Questi piccoli programmi sono stati ideati e progettati per poter essere utilizzati nei dispositivi mobili di ultima generazione, cioè smartphone e tablet, con diverse specifiche tecniche e sistemi operativi. Infatti, esistono app per il sistema operativo iOS di Apple, che è il “motore” degli iPhone e degli iPad. Allo stesso modo, esistono app specifiche per il sistema operativo Android, che è il motore installato sui dispositivi di altre aziende produttrici di telefoni: Samsung, HTC, LG, Huawei, e molte altre. Ormai le varie aziende hanno incluso nei propri dispositivi alcune applicazioni di base, dando poi ai clienti la facoltà di poterne scaricare altre dallo store, sia a pagamento che gratuite. Spesso si sente dire che c’è una applicazione per qualsiasi necessità e, a vedere dall’incredibile numero di app presenti negli store di Apple e Android, sembrerebbe proprio così.
Le app sono state ideate e realizzate per gli scopi più vari, dal semplice intrattenimento ludico a quello di facilitarci la comunicazione. Esistono applicazioni per finanza o fotografia; non mancano quelle per istruzione e lavoro e numerose sono quelle per libri e consultazioni. Sono tante quelle per uso multimediale e video. Troviamo, poi, le app di medicina e salute, meteo e notizie, trasporti e viaggi, e non mancano, ovviamente, le app di cucina che, sempre più aggiornate, permettono di condividere piatti all’altezza di veri chef stellati. Per non parlare, poi, delle app che permettono di organizzare grandi eventi, con la possibilità di gestire la vendita dei biglietti e di inviare comunicazioni ai partecipanti, o quelle per l’organizzazione di matrimoni, che gestiscono la lista degli invitati, la sistemazione dei tavoli, la spedizione degli inviti, la lista dei regali, fino a quelle per la semplice organizzazione di un compleanno, che consentono sia di risparmiare tempo per spedire gli inviti sia di conoscere il numero esatto di persone che parteciperanno alla festa.
L’idea alla base dell’applicazione oggetto della presente tesi vuole essere una sintesi delle funzionalità a volte già presenti su altre app specifiche, ma questa volta finalizzate alla gestione di eventi in ambito domestico, tra parenti ed amici che si conoscono e che, con piacere, si riuniscono per festeggiare insieme avvenimenti importanti, o anche solo per trascorrere una serata di svago. Certamente, il periodo di pandemia che stiamo vivendo ha contribuito a sollecitare l’esigenza di prevedere anche l’inserimento

di alcune funzionalità che consentissero di tutelare noi stessi e i nostri ospiti senza rinunciare al clima conviviale e spensierato di una riunione tra amici o parenti.
Dal 3 giugno 2020, e salvo ulteriori nuove restrizioni dovute all'andamento del contagio, infatti, la nostra vita è ritornata in parte alla normalità; siamo tutti liberi di uscire di casa e di spostarci liberamente, seppur rispettando le semplici, ma essenziali, regole che ancora impongono l'osservanza del distanziamento sociale e l'utilizzo della mascherina nei luoghi chiusi, o anche all'aperto, in determinati orari o condizioni di affollamento.
L'allentamento delle misure di lockdown ha portato tutti noi a poter riabbracciare i nostri amici e riavviare le nostre vite sociali. Ma il virus in Italia non è scomparso ed è essenziale non abbassare la guardia; dopo mesi di isolamento forzato come si può passare una serata a casa con gli amici, magari a cena, in sicurezza?
L' applicazione oggetto della presente tesi cerca, appunto, di dare risposta a questo interrogativo.
Attraverso l'utilizzo di questa app sarà, innanzitutto, possibile decidere se si vuole organizzare un evento con il rispetto delle norme anti COVID-19 e, quindi, osservando le regole sul distanziamento sociale dei partecipanti non conviventi.
In caso di risposta affermativa e di evento da svolgere "indoor", l'applicazione permette anche di calcolare il numero massimo degli invitati consentiti dall'ampiezza dei locali dove si svolgerà la riunione nonchè di determinare l'attribuzione dei posti a tavola, sempre nel rispetto delle regole di distanziamento richiesto tra persone conviventi e non.
L'applicazione si occupa, ancora, di memorizzare il menù proposto agli invitati in ogni occasione così da evitare eventuali, future ripetizioni delle stesse portate ai medesimi commensali; l'applicazione registra, altresì, tutte le intolleranze alimentari dichiarate dagli ospiti all'atto della loro registrazione, così che l'organizzatore possa tenerne conto nella predisposizione del menù.
Altra funzionalità dell'applicazione consiste, ovviamente, nell'inoltrare la notizia dell'evento tramite una notifica agli invitati rendendo, quindi, più semplice e veloce procedere alla distribuzione degli inviti ai vari ospiti e, conseguentemente, conoscere il numero esatto delle persone che parteciperanno.
La presente tesi verrà strutturata come di seguito specificato:

- Il primo capitolo farà un resoconto delle diverse fasi in cui si è sviluppata l'attuale pandemia di COVID-19, partendo dai primi focolai scoperti in Cina sino alla sua diffusione in Italia, descrivendo tutti i protocolli di sicurezza prescritti dal Governo alla cittadinanza e le restrizioni imposte, con riferimento specifico al settore della ristorazione.

- Il secondo capitolo tratterà la storia di Android, uno dei sistemi operativi più diffusi al mondo, descrivendone la struttura, le diverse versioni e la sua piattaforma per lo sviluppo di App, cioè Android Studio.

- Nel terzo capitolo verranno esposte le diverse funzionalità del progetto attraverso l'analisi dei requisiti, il diagramma dei casi d'uso e delle attività. La veste grafica verrà, poi, mostrata attraverso l'utilizzo di mockup.
- L'argomento del quarto capitolo riguarderà l'implementazione dell'applicazione; per i casi d'uso più rappresentativi verrà anche mostrato parte del corrispettivo codice. La sezione dedicata al manuale utente, poi, conterrà tutti gli screenshot relativi alle activity contenute nell'applicazione, con indicazione precisa e dettagliata del loro utilizzo.
- Il quinto capitolo presenterà le considerazioni e le valutazioni effettuate a fine lavoro, soprattutto in ordine alle conoscenze e competenze maturate; un'analisi SWOT effettuata sul progetto consentirà, poi, di evidenziarne fattori di successo ed eventuali criticità.
- Nel sesto capitolo, infine, verranno tratte le conclusioni finali sul progetto che si è realizzato, indicando, anche, possibili miglioramenti e sviluppi futuri.

Capitolo 1

Il Covid-19 e i protocolli di sicurezza

Mentre tutto il mondo si preparava a festeggiare l'arrivo del nuovo anno e tutti eravamo assolutamente inconsapevoli dell'emergenza sanitaria che si sarebbe di lì a poco creata, un nuovo virus altamente contagioso e completamente sconosciuto al nostro sistema immunitario aveva iniziato a circolare in una regione molto lontana da noi. Non avremmo mai pensato, all'epoca, che questo virus avrebbe potuto diffondersi e causare tanti problemi a livello individuale e collettivo, sia per la salute che per i sistemi sanitari ed economici. Ma in pochi mesi lo scenario globale è cambiato radicalmente e noi abbiamo dovuto adattarci e modificare completamente la nostra quotidianità e le nostre abitudini per far fronte alle nuove esigenze.

1.1 Cenni sulla pandemia di Covid 19 del 2019/20

Già a novembre 2019 - e forse anche prima - in Cina, in particolare a Wuhan, la città più popolata della parte orientale, nella provincia cinese di Hubei, inizia ad essere segnalata la presenza di un certo numero di polmoniti anomale, dalle cause non ascrivibili ad altri patogeni. La maggior parte dei casi presentava un legame epidemiologico con il mercato ittico di Huanan Seafood, nel sud della Cina, specializzato nella vendita all'ingrosso di frutti di mare e animali vivi.
La prima data ufficiale in cui inizia la storia di quello che impareremo tristemente a conoscere come Nuovo Coronavirus, o Sars-Cov 2 , o più semplicemente COVID-19, è il 31 dicembre 2019, data in cui le autorità sanitarie cinesi segnalavano all'Organizzazione Mondiale della Sanità (OMS) la presenza di questi casi insoliti di polmonite di origine sconosciuta. I pazienti accusavano sintomi simili all'influenza, come dermatiti, febbre, tosse secca, stanchezza, difficoltà di respiro. Nei casi più gravi, spesso riscontrati in soggetti già gravati da precedenti patologie, si sviluppavano forme di polmonite, insufficienza renale acuta fino ad arrivare anche al decesso. La città di Wuhan aveva riscontrato decine e decine di casi e centinaia di persone erano sotto osservazione tanto che i primi giorni di gennaio 2020 il virus veniva isolato e le autorità cinesi ne condividevano con il mondo la sequenza genetica, fondamentale in termini di ricerca.
Le autorità cinesi avevano, altresì, scoperto che il patogeno responsabile era un nuovo ceppo di coronavirus, della stessa famiglia dei coronavirus responsabili della Sars e

della Mers, le epidemie molto più gravi, ma sicuramente molto meno contagiose, già impostesi all'attenzione del mondo, rispettivamente, tra il 2002/2003 e il 2012/2013, ma diverso da tutti questi, nuovo appunto.
Il nuovo coronavirus, passato probabilmente dall'animale all'essere umano (un salto di specie, in gergo tecnico) viene, infatti, scoperto trasmettersi anche da uomo a uomo, motivo per il quale l'OMS , allo scopo di limitare la diffusione del contagio, cominciava a fornire tutte le istruzioni del caso, raccomandando di lavarsi le mani regolarmente per più di 20 secondi, anche eventualmente utilizzando soluzioni antisettiche a base di alcool o cloro, coprendo bocca e naso all'atto di tossire o starnutire, evitando altresì un contatto ravvicinato con chiunque mostrasse sintomi di malattie respiratorie, o consigliando l'utilizzo di mascherine nei casi in cui non potesse essere garantito il distanziamento sociale. A coloro che avevano recentemente viaggiato in un Paese con contagio diffuso, o che erano stati in diretto contatto con qualcuno a cui era stata diagnosticata la malattia da COVID-19, è stato raccomandato di mettersi in quarantena o praticare l'allontanamento sociale per 14 giorni dal momento dell'ultima possibile esposizione con limitazione di tutte le attività al di fuori di casa, ad eccezione delle cure mediche. Alla fine di gennaio, nonostante le precauzioni e le restrizioni imposte dalla Cina per evitare che l'epidemia si diffondesse, il rischio passava da moderato ad alto e, nella serata del 30 gennaio, l'OMS dichiarava l'"emergenza sanitaria pubblica di interesse internazionale".

1.2 Il COVID 19 in Italia: gestione della pandemia dal lockdown di marzo alla fase 3

Il giorno successivo alla dichiarazione dell'OMS, 31 gennaio 2020, l'Italia, con delibera del Consiglio dei Ministri, dichiarava lo stato di emergenza su tutto il territorio nazionale per il termine di 6 mesi, poi recentemente prorogato fino al 15 ottobre in base all'andamento del contagio. All'inizio, in Italia, i casi erano pochissimi e tutti provenienti dalla Cina. Intorno alla seconda metà del mese di febbraio, però, si manifestarono diversi casi di coronavirus nel lodigiano, in Lombardia, su persone non provenienti dalla Cina, determinando un nuovo focolaio di cui non si conosceva l'estensione. Alcuni dei paesi colpiti (Codogno, Castiglione d'Adda, Casalpusterlengo, ed altri, unitamente al paese di Vò Euganeo nel Veneto) sono stati, di fatto, messi in quarantena per cercare di arginare il diffondersi del contagio. Da quella data è stato tutto un susseguirsi di DPCM, Decreti legislativi, indicazioni e ordinanze regionali legate al diffondersi del contagio che cominciava ad espandersi prevalentemente al Nord, ma in misura diversa in ogni regione.
Per questo motivo il governo ha dapprima stabilito la chiusura delle scuole e delle università in tutta Italia per poi emettere, domenica 8 marzo, un decreto che prevedeva l'isolamento della Lombardia, in assoluto la regione più colpita, e di altre 14 province, che diventavano "zona rossa". L'11 marzo viene, poi, pubblicato il cosiddetto "Decreto

#IoRestoaCasa", il provvedimento che, dando inizio alla cosiddetta "Fase 1", di fatto estendeva a tutto il territorio nazionale quanto già previsto col decreto dell'8 marzo: venivano sospese le comuni attività commerciali al dettaglio, le attività didattiche, i servizi di ristorazione e venivano vietati gli assembramenti di persone in luoghi pubblici o aperti al pubblico. La regola, di fatto, è che si poteva uscire solo per comprovate ragioni di necessità, come per fare la spesa, per esigenze lavorative, per l'acquisto di farmaci, o per altri motivi di salute.
Successivamente, in data 22 marzo, veniva adottata una nuova ordinanza che vietava a tutte le persone fisiche di trasferirsi o spostarsi con mezzi di trasporto pubblici o privati in comuni diversi da quello in cui si trovavano, salvo che per comprovate esigenze lavorative, di assoluta urgenza, ovvero per motivi di salute. Veniva, altresì, pubblicata una lista relativa alla chiusura di tutte le attività ritenute non necessarie ed un'altra contenente, invece, l'indicazione di tutte quelle ritenute fondamentali e strategiche che, da quella data, venivano sottoposte, per poter operare, al rispetto di un rigido protocollo per il contrasto e il contenimento della diffusione del virus COVID-19 negli ambienti di lavoro.
Le misure adottate, con successive proroghe, rimanevano in vigore fino al 3 maggio. Un nuovo DPCM del 26 aprile 2020, in vigore dal 4 maggio 2020, infatti, prevedeva l'inizio della cosiddetta "Fase 2", ovvero un allentamento graduale delle precedenti misure di contenimento, essendo la curva epidemica in fase di discesa. Nelle prime due settimane il decreto aggiungeva agli spostamenti consentiti anche le visite ai congiunti all'interno del territorio regionale (sempre, però, mantenendo la distanza di almeno un metro e con l'uso obbligatorio di mascherine e guanti), permetteva l'apertura dei parchi pubblici, il servizio d'asporto per le attività di ristorazione, la riapertura di stabilimenti balneari e consentiva lo svolgimento di attività motorie, a prescindere dalla lontananza dal proprio domicilio.
Bar, ristoranti, parrucchieri e centri estetici hanno, invece, ripreso la loro attività in tutta Italia il 18 maggio, su disposizione del nuovo DPCM del 17 Maggio 2020, che ha disposto anche la riapertura di mostre e di luoghi culturali all'aperto e delle celebrazioni religiose (con ingressi contingentati).
Da questa data, dunque, l'Italia ha ripreso una pseudo-normalità durante la quale è, però, rimasto obbligatorio rispettare le misure varate per evitare il contagio, quali il distanziamento sociale, l'uso di mascherine e gel igienizzanti e la rilevazione della temperatura. Dal 3 giugno è stata, poi, permessa la libera circolazione fra regioni. Il DPCM dell'11 Giugno 2020, in vigore dal 15 giugno, le cui decisioni sono state poi ribadite e prorogate dal successivo DPCM del 14 luglio, ha dato inizio all'attesa "Fase 3" con la quale è stata prevista, ad esempio, la riapertura di cinema, teatri e concerti, ma con obbligo di mascherina e distanziamento, sono state consentite le competizioni sportive di interesse nazionale, ma a porte chiuse, è stato permesso il commercio al dettaglio, ma con l'obbligo del rispetto delle distanze e dei protocolli; sono stati, poi, riaperti palestre, piscine, circoli sportivi rispettando il distanziamento, sono stati consentiti l'accesso nei luoghi di culto e le funzioni religiose nel rispetto

dei protocolli siglati con le comunità religiose. Eccetto gli esami di maturità, sono rimaste sospese le scuole e le università di ogni ordine e grado, che hanno proseguito la didattica e gli esami universitari a distanza.
È, poi, stato previsto che i soggetti con febbre superiore ai 37,5 gradi debbano rimanere a casa mentre è stabilito l'obbligo della quarantena per chi proviene dall'estero, eccetto per i cittadini dei Paesi dell'Unione Europea, dell'area Schengen, del Regno Unito, di Andorra, del Principato di Monaco, di San Marino e del Vaticano. In attesa dell'auspicabile inizio di una Fase 4 post-pandemia, il nostro Governo ha scelto, quindi, la via della cautela, procedendo a riaperture graduali e sempre con l'obbligo del rispetto dei Protocolli per scongiurare nuovi picchi dei contagi.

1.3 Covid 19 e ristorazione

Tra le varie restrizioni che, di fatto, hanno praticamente blindato gli italiani nelle loro case e provocato uno stop forzato ad innumerevoli attività produttive, i vari DPCM che si sono susseguiti hanno, in un primo momento, imposto l'obbligo di chiusura anche alle attività di ristorazione, con la sola eccezione della consegna di cibo a domicilio. La situazione di emergenza ha, di conseguenza, cambiato drasticamente le dinamiche operative del settore: per molti ristoratori il lockdown è stato l'occasione per capire che il food delivery non riguardava solo patatine, hamburger, pizza e sushi. Migliaia di ristoranti, dalla trattoria allo stellato, per adeguarsi ai notevoli cambiamenti dettati dalla crisi sanitaria, hanno organizzato il servizio di delivery, spesso ricorrendo a soluzioni che molti ristoratori non avevano mai considerato prima, come, per esempio, l'utilizzo di applicazioni che consentono di prenotare cibo a domicilio da bar, ristoranti e pizzerie aderenti, come Glovo, Deliveroo e Uber Eats. Sia che il servizio di consegna venisse affidato ai cosiddetti "riders", sia che venisse svolto dal proprio personale, tutti si sono dovuti comunque adeguare alle rigorose raccomandazioni impartite dal Governo, affinchè, nel servizio della ristorazione, la consegna a domicilio avvenisse nel pieno rispetto delle norme igienico-sanitarie. Sono state, anche, emanate norme per l'attività di confezionamento e di trasporto, assicurando la distanza di sicurezza interpersonale di almeno un metro e l'assenza di contatto diretto con il cliente.

1.3.1 Fase 3 e protocolli di sicurezza nel settore della ristorazione

Con il DPCM del 17 maggio 2020, come già riferito, è stato deliberato il riavvio delle attività produttive e, quindi, in particolare per i pubblici esercizi, è stata disposta la ripresa di tutte le attività dei servizi di ristorazione, fra cui bar, pub, ristoranti, gelaterie e pasticcerie. Per disciplinare la fase della riapertura, e con l'obiettivo di garantire la salute e la sicurezza sia degli operatori che dei consumatori, l'INAIL, in collaborazione con l'Istituto Superiore di Sanità, ha realizzato un documento tecnico contenente una serie di regole al quale il Governo potesse ispirarsi tenendo

in considerazione le specificità e le modalità di organizzazione del lavoro, nonché le particolari criticità di gestione del rischio da contagio in un settore, quale quello della ristorazione, che, in Italia, conta circa 1,2 milioni di addetti, che durante il lockdown, sono rimasti per la maggior parte inattivi. Per garantire la sicurezza di clienti e lavoratori, e per contenere i nuovi contagi da coronavirus, le Regioni ed il Governo hanno trattato lungamente per stabilire di concerto le misure da osservare per la ripresa delle attività economiche e produttive predisponendo, alla fine, una serie di regole più flessibili e meno gravose di quelle proposte dall'INAIL nel suo documento tecnico. Tali regole, all'indomani della loro diffusione, avevano creato non pochi malumori tra gli operatori del settore, soprattutto per l'indicazione al rispetto di una distanza di 2 metri tra un tavolo e l'altro e 4 metri quadrati per ciascun cliente. Un'applicazione tassativa di tale principio avrebbe, di fatto, comportato la chiusura di moltissime attività, impossibilitate per la metratura del loro esercizio al rispetto di tali indicazioni; pertanto, il Governo, su sollecitazione delle Regioni, è stato costretto ad allentare tutte queste regole. Più specificatamente:

- Tra le novità principali c'è quindi la riduzione della distanza di sicurezza da assicurare tra i clienti: nel documento dell'INAIL si proponevano 2 metri tra un tavolo e l'altro, mentre le linee guida del governo parlano di 1 metro tra i clienti.

- Le regole invitano a fornire ai clienti informazioni adeguate sulle misure di prevenzione, che siano comprensibili anche per chi non parla l'italiano.

- All'ingresso dei locali potrà essere misurata la temperatura corporea dei clienti e dovrà essere impedito l'accesso a chi avrà una temperatura di 37.5 C o superiore.

- Ogni attività di ristorazione deve rendere disponibili prodotti igienizzanti per i propri clienti e per il personale in più punti del locale, in particolare all'entrata e in prossimità dei servizi igienici, che dovranno essere puliti più di una volta al giorno.

- Bisognerà cambiare frequentemente l'aria degli ambienti interni e, nel caso di uso di impianti di condizionamento, si dovrà escludere la funzione di ricircolo dell'aria.

- Ogni tavolo andrà disinfettato dopo il servizio e bisognerà evitare il più possibile l'uso di oggetti e contenitori riutilizzabili da diversi clienti, come saliere e oliere, se non igienizzati di volta in volta.

- Per quando riguarda i menù si consigliano alcune soluzioni: uso di menù cartacei usa-e-getta,adozione di menù in carta plastificata da poter disinfettare dopo ogni uso o, ancora meglio, uso di menù digitali, che i clienti possano consultare online sui propri cellulari.

- Il personale di servizio, che lavora a contatto con i clienti, dovrà portare mascherine e lavarsi frequentemente le mani con soluzioni idroalcoliche; in particolare le mani dovranno essere igienizzate prima di ogni servizio al tavolo.
- I clienti dovranno indossare le mascherine per tutto il tempo in cui non saranno seduti al tavolo o staranno consumando la propria ordinazione al banco.
- Per quanto riguarda la disposizione dei tavoli, dovranno essere organizzati in modo che i clienti, fatta eccezione per quelli che non devono rispettare il distanziamento fisico, ad esempio perché conviventi (Figura 1.1), siano seduti ad almeno 1 metro gli uni dagli altri (Figura 1.2). Questa distanza può essere minore nel caso in cui tra i tavoli vengano messe delle barriere fisiche adeguate a prevenire potenziali contagi attraverso goccioline di saliva (droplet).
- Dove possibile, inoltre, si dovrà privilegiare l'uso di spazi esterni, come giardini e terrazze, sempre rispettando, però, il distanziamento di almeno 1 metro.
- In generale nei locali con posti a sedere non potranno entrare più clienti di quanti siano i posti a sedere.
- La consumazione al banco sarà consentita solo nei casi in cui sia possibile mantenere distanze interpersonali di almeno 1 metro tra i clienti. Anche in questo caso vale l'eccezione per le persone che non devono rispettare il distanziamento fisico, ad esempio perché conviventi; ovviamente non sta ai ristoratori di verificare se due persone lo siano effettivamente: ognuno deve essere responsabile per se stesso in questo senso.
- Negli esercizi che non hanno posti a sedere potrà entrare nel locale un numero limitato di clienti per volta, da stabilire a seconda degli spazi dei singoli locali: l'importante è che sia sempre mantenuto il distanziamento di almeno 1 metro tra i clienti.
- Dovranno essere favoriti i pagamenti al tavolo e in modalità elettronica. Le casse potranno essere dotate di schermi che facciano da barriera fisica per il cassiere; in alternativa, le regole dicono che chi sta in cassa deve indossare la mascherina e avere a disposizione una riserva di gel igienizzante per le mani.
- Nei locali con posti a sedere bisognerà privilegiare l'accesso tramite prenotazione e il locale dovrà conservare per 14 giorni l'elenco delle persone che hanno prenotato.
- Il servizio a buffet, inizialmente assolutamente vietato, è stato parzialmente reso possibile dal DPCM del 14 luglio, introduttivo della Fase 3, che sostanzialmente ha ribadito e prorogato le regole già previste nel precedente DPCM. È, quindi, ora possibile organizzare una modalità a buffet mediante somministrazione da parte di personale incaricato, escludendo la possibilità per i clienti di toccare

quanto esposto e prevedendo, in ogni caso, per clienti e personale, l'obbligo del mantenimento della distanza e l'obbligo dell'utilizzo della mascherina a protezione delle vie respiratorie. La modalità self-service può essere eventualmente consentita per buffet realizzati esclusivamente con prodotti confezionati in monodose. In particolare, la distribuzione degli alimenti dovrà avvenire con modalità organizzative che evitino la formazione di assembramenti anche attraverso una riorganizzazione degli spazi in relazione alla dimensione dei locali; dovranno essere, altresì, valutate idonee misure (ad esempio segnaletica a terra, barriere, etc.) per garantire il distanziamento interpersonale di almeno un metro durante la fila per l'accesso al buffet.

Al di là delle norme stabilite dal Governo e delle indicazioni dei tecnici, comunque, quello che è difficile prevedere è quanti e quali clienti avranno voglia di sottostare a queste regole nel quadro di un contesto che dovrebbe essere di relax, come un'uscita a cena, ma, soprattutto, se quei clienti che, nonostante tutte le restrizioni comunque decideranno di andare al ristorante, renderanno sostenibili i costi delle attività di ristorazione, in questo periodo sicuramente incrementati da tutti gli oneri imposti.

1.3.2 Consigli e buone pratiche nella ristorazione domestica

La convivialità e la possibilità di incontrare liberamente le persone a cui siamo affezionati sono alcune tra le rinunce più pesanti alle quali siamo stati sottoposti durante le settimane del lockdown. Adesso che siamo più liberi di muoverci rimane, comunque, l'obbligo di rispettare le regole prescritte per evitare la recrudescenza dell'epidemia. Per questo motivo molti sono ancora riluttanti a frequentare locali e ristoranti e preferiscono stare più tranquilli decidendo di incontrare i propri amici tra le mura domestiche, magari offrendo loro qualcosa da bere o mangiare.
Come prima misura di prevenzione potrebbe essere consigliato creare delle cosiddette "bolle sociali", cioè dei (piccoli) gruppi di amici che si impegnino a incontrarsi solo tra di loro: minore è il numero di persone con cui si sta a stretto contatto e sicuramente minore è il rischio, ma sarà anche più facile eventualmente eseguire un tracciamento dei contatti. Per il resto, come al ristorante, anche a casa sarà necessario attenersi a delle semplici regole per fare in modo di trascorrere piacevoli momenti con i propri ospiti in tutta sicurezza:

- Innanzitutto, dovrà sempre tenersi ben presente che, anche se il peggio è passato, la pandemia non è conclusa, e quindi sarà una regola fondamentale quella di adeguare il numero degli ospiti alla necessità di mantenere il distanziamento quando si è seduti a tavola e durante tutti i momenti.

- Saranno sicuramente da preferire gli eventi all'aperto, dove, se si dispone di uno spazio adeguato, potranno essere previsti più commensali, ovviamente sempre nel rispetto selle regole sul distanziamento sociale.

- Nel caso di eventi al chiuso il numero degli ospiti dovrà necessariamente essere regolato in funzione della dimensione della stanza in cui si svolgerà. Sul tema deve tenersi presente che la normativa antecedente al COVID-19 sull'organizzazione dei locali addetti alla ristorazione non prevede norme specifiche sul distanziamento, ma indicazioni molto flessibili, fino a uno spazio di superficie per cliente seduto pari a 1,20 metri quadrati. In casa, sempre rispettando la regola del distanziamento sociale di almeno 1 metro tra gli ospiti, potrebbe rivelarsi equo considerare uno spazio di almeno due metri quadri a persona.

- All'arrivo degli ospiti è opportuno evitare abbracci, strette di mano ed effusioni. Man mano che gli ospiti arrivano, occorre fornire loro il gel disinfettante e invitare tutti a lavare e igienizzare le mani.

- Per cercare di limitare al massimo i movimenti delle persone il padrone di casa dovrà personalmente provvedere alle varie necessità della tavola in modo che i commensali non debbano alzarsi dal loro posto.

- Agli invitati dovrà chiedersi di non appoggiare telefonini o altri oggetti personali sulla tavola, quantomeno fino a quando la stessa non venga sparecchiata.

- I buffet, i piatti in comune, i finger food sono decisamente da evitare, come pure le cene in piedi, per evitare che gli ospiti vaghino da una parte all'altra e tendano ad avvicinarsi più del dovuto.

- La tavola andrà predisposta mantenendo i posti a un metro di distanza l'uno dall'altro (Figura 1.2), facendo in modo che le persone si alzino eventualmente da tavola una alla volta. Solo per i congiunti potrà derogarsi alla regola di lasciare un posto libero tra gli invitati per garantirne il distanziamento. (Figura 1.1)

- Per quanto riguarda il cibo esso dovrà essere servito già impiattato, per evitare che persone diverse maneggino piatti di portata in comune e le stesse posate. Per lo stesso motivo, sarebbe opportuno che fosse sempre la stessa persona a versare acqua e vino.

- Il pane andrà servito già tagliato o in panini di piccole dimensioni: meglio farlo trovare già in tavola accanto a ciascun ospite nell'apposito piattino, mentre è da evitare il cestino in comune. Se non se ne può fare a meno, occorre predisporre l'apposita pinza per servirsi senza toccare il pane con le mani. In ogni caso, dovrà essere la padrona di casa a maneggiarlo e a servire gli ospiti.

- In bagno dovranno essere messi a disposizione degli ospiti asciugamani monouso, di carta o di spugna, e un luogo in cui porli dopo l'uso.

- Se la cena è al chiuso ed è in funzione un condizionatore, il flusso dell'aria dovrà essere orientato verso l'alto e non direttamente verso il tavolo.

Purtroppo la prudenza è ancora d'obbligo tanto più in situazioni di distensione e spensieratezza, come quelle vissute tra le mura domestiche ed insieme a persone amiche, con cui potrebbe rivelarsi normale tendere a ridurre le precauzioni. Nessuna esitazione, quindi, a richiamare eventualmente gli invitati al rispetto delle norme di prudenza e di distanziamento per stare insieme in modo piacevole, condividendo nuovamente momenti di convivialità in piena sicurezza.

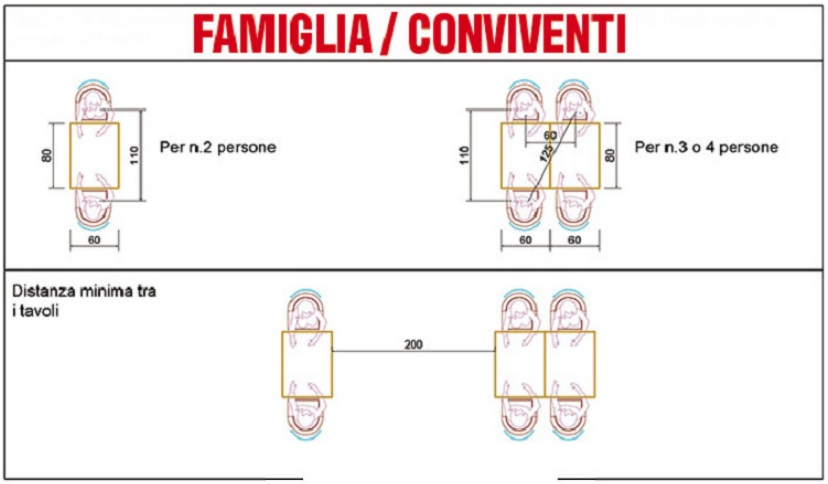

Figura 1.1: Esempio di tavolo conviventi

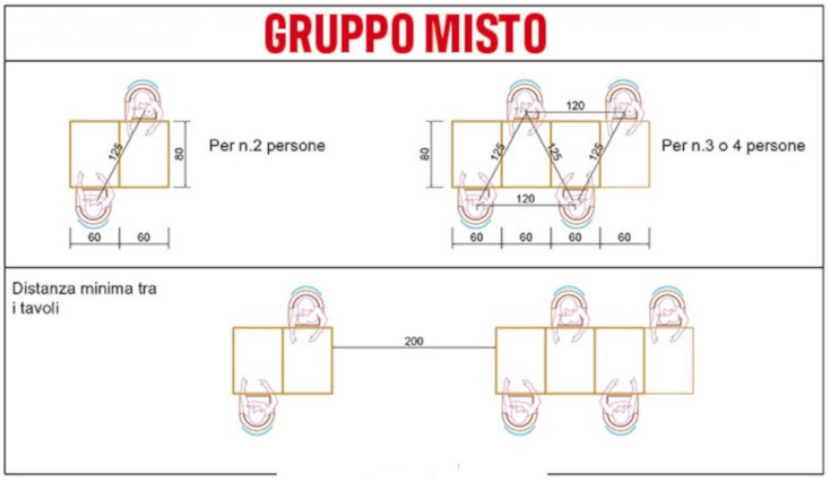

Figura 1.2: Esempio di tavolo con gruppo misto

Capitolo 2

Android

Android è indiscutibilmente il sistema operativo per il mobile più utilizzato e più importante al mondo. Stando alle statistiche, è presente su circa tre dispositivi su quattro lasciando lo spazio restante ad IOS di Apple. Le motivazioni per cui Android ha trovato tanto riscontro nel pubblico decretandone il suo successo deve certamente rinvenirsi nell'adozione di questo sistema da parte di dispositivi molto diversi tra loro, non solo per tipologia - smartphone piuttosto che tablet - ma soprattutto per fasce di prezzo, da poche decine di euro fino a cifre piuttosto significative. Ciò ne ha permesso una diffusione molto diversificata trasversalmente alle diverse categorie sociali ed ha causato di riflesso una frammentazione delle possibilità applicative, costringendo gli sviluppatori ad una particolare cura degli aspetti di adattamento alle caratteristiche del dispositivo ospite. Proprio in questo Android ha dimostrato la sua grande modernità offrendo tutto il supporto necessario per permettere all'applicazione in esecuzione di adeguarsi ad ogni circostanza.

2.1 Storia di Android

Le radici del sistema operativo mobile più diffuso al mondo si possono rintracciare nei primi anni del millennio, quando nel 2003, Andy Rubin, Rich Miner, Nick Sears e Chris White fondarono Android Inc. Nelle iniziali intenzioni dei suoi ideatori Android doveva servire da sistema operativo per fotocamere digitali, in modo che gli utenti potessero installare nuove app e aggiungere nuove funzionalità ai loro dispositivi. Ben presto, però, i quattro decisero di cambiare rotta e virare verso il mercato degli smartphone, convinti che ben presto i "cellulari intelligenti" avrebbero sopraffatto le fotocamere.
Nel 2005 l'azienda di Rubin venne acquistata da Google che, in cerca di un modo per entrare da protagonista nel mondo degli smartphone, aveva giustamente visto nel progetto una possibilità unica: permettere a moltissimi utenti di poter effettuare ricerche direttamente dal proprio terminale mobile.
Dopo oltre un anno di anonimato, Android tornò a far parlare di sé nel 2007 quando Google annunciò la costituzione dell'Open Handset Alliace (OHA), un consorzio che riuniva in un'unica sigla una lunga lista di produttori di smartphone e telefonini. Lo scopo era quello di gettare le basi per lo sviluppo di standard aperti (sia di

tipo software che hardware) in ambito mobile, realizzando una piattaforma open in grado di tenere il passo del mercato senza che vi fossero da pagare brevetti e diritti: in quell'occasione, inoltre, Google presentò al mondo il suo sistema operativo mobile, ovvero Android. Nello stesso anno venne anche messa a disposizione dei programmatori la prima versione del Software Development Kit (o "SDK"), un insieme di strumenti di sviluppo che consentiva la creazione di applicazioni per un determinato software package, software framework, piattaforme hardware, computer system, console per videogiochi, sistemi operativi o piattaforme simili, consentendo lo sviluppo e la realizzazione delle applicazioni sperimentali.
Il primo telefono basato su Android fu il famoso G1 prodotto da HTC (Figura 2.1), distribuito negli Stati Uniti nel 2008, anche noto come Dream (letteralmente " sogno").

Figura 2.1: G1 Dream

Rispetto agli smartphone che conosciamo oggi stiamo parlando di un antenato, ma era pur sempre il primo passo nella direzione delle grandi innovazioni che seguirono nel giro di pochissimo tempo. Non era certo l'hardware a determinare il successo del G1 quanto il software. Al suo interno erano già preinstallati diversi programmi di Google (ad esempio, Gmail, Google Calendar, YouTube, Google Maps, Google Talk, e altri). Ci volle del tempo per comprendere quali elementi fisici rimanevano essenziali per l'utente rispetto alle potenzialità offerte dal nuovo software: nel caso del G1 c'era una tastiera con lettere, numeri e tasti funzione, oltre ad un tasto per rispondere e riagganciare. Del resto, nel 2008, c'erano ancora moltissimi utenti che pensavano di non poter rinunciare ad una tastiera fisica.
Soltanto ad aprile 2009, con il rilascio della Versione 1.5 dell'SDK (Cupcake), venne inserita la gestione della tastiera virtuale liberando, di conseguenza, i costruttori di hardware dal vincolo di realizzare la tastiera fisica. Ma è stato solo nel 2010 che l'affare di Android si è rivelato su scala mondiale ed è coinciso con il rilascio del Nexus One (Figura 2.2), il primo telefono "Pure Google", che ha determinato il successo universale di Android.

Figura 2.2: Nexus One

2.2 Architettura di Android

Android ha un'architettura di tipo gerarchico, strutturata a layer a complessità crescente dal basso verso l'alto, dove i livelli inferiori offrono servizi ai livelli superiori garantendo un alto grado di astrazione (Figura 2.3).

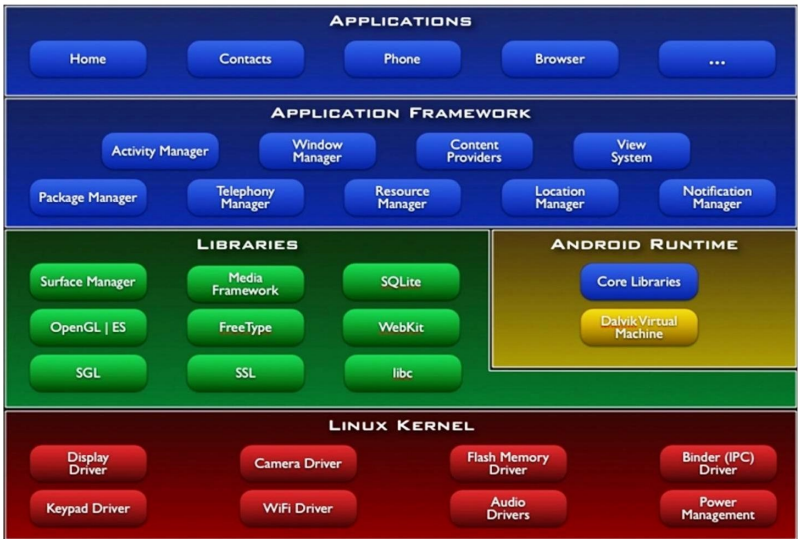

Figura 2.3: Architettura di Android

- Il livello più basso è rappresentato dal *kernel* Linux che fornisce un livello di astrazione tra l'hardware dei dispositivi ed i livelli superiori dell'architettura mettendo a disposizione driver per il corretto utilizzo da parte delle app di elementi quali display, fotocamera, audio, tastiera e tutti i componenti fondamentali di un dispositivo. I produttori di telefoni possono, quindi, intervenire già a questo livello per personalizzare i driver di comunicazione con i propri dispositivi. La scelta verso l'utilizzo di un kernel Linux si spiega attraverso la necessità di avere un sistema operativo mobile che fornisse tutte le feature di sicurezza, gestione della memoria, gestione dei processi, power management e che fosse, quindi, molto affidabile e ampiamente testato.

- Salendo nella gerarchia troviamo un insieme di *librerie* native realizzate in C e C++ e l'*Android Runtime*.
  L'ambiente di runtime è costituito dalle librerie core Java e dalla macchina virtuale Dalvik (DVM) che, insieme, costituiscono la piattaforma di sviluppo per Android. A tale proposito va sottolineato come la scelta di Google di utilizzare il linguaggio Java per lo sviluppo delle applicazioni, piuttosto che crearne uno nuovo, avrebbe potuto creare contrasti con la filosofia open di Android; per questo motivo, al fine di evitare il pagamento di royalty, Google ha deciso di creare una propria macchina virtuale, che prende, appunto, il nome di Dalvik Virtual Machine (DVM), ottimizzata per sfruttare la poca memoria presente nei dispositivi mobili. Prima di essere installati su un dispositivo i file `.class` con bytecode Java vengono convertiti in file `.dex` (Dalvik Executable) che sono poi eseguiti dalla DVM, comportando una sensibile riduzione della memoria richiesta dall'esecuzione delle applicazioni.
  Oltre alle librerie standard Java, l'ambiente di sviluppo Android ne include altre denominate, appunto, Android Libraries. Si tratta di un insieme di librerie Java specifiche per Android. Tra quelle più importanti troviamo:
  - `android.app` - Fornisce l'accesso al modello applicativo e costituisce la base di tutte le applicazioni Android.
  - `android.content` - Facilita l'accesso ai contenuti, consentendo la comunicazione tra applicazioni e componenti.
  - `android.database` - Permette di accedere ai dati pubblicati da un fornitore di contenuti e include classi per la gestione di database SQLite.
  - `android.os` - Fornisce funzionalità per l'utilizzo di servizi di base del sistema.
  - `android.media` - Fornisce le classi per abilitare la riproduzione di audio e video.

- Al livello successivo si trova l'*Application Framework*, costituito da un insieme di servizi che rappresenta l'ambiente in cui le applicazioni Android vengono eseguite. Esso è costituito dai seguenti componenti:

  - *Activity Manager* - Controlla i vari aspetti del ciclo di vita di un'applicazione e lo stack delle activity.
  - *Content Providers* - Consente alle applicazioni di pubblicare e condividere dati con altre applicazioni.
  - *Resource Manager*- Consente l'accesso a risorse quali stringhe, impostazioni e layout.
  - *Notifications Manager*- Consente alle applicazioni di far visualizzare agli utenti allarmi e notifiche.
  - *View System* - Un insieme di componenti utilizzati per creare le interfacce utenti.
  - *Package Manager*- Sistema tramite cui le applicazioni possono reperire informazioni sulle altre applicazioni attualmente installate su un dispositivo.
  - *Telephony Manager* - Fornisce informazioni alle applicazioni circa i servizi telefonici disponibili.
  - *Location Manager* - Fornisce l'accesso ai servizi di localizzazione da parte delle applicazioni.

- In cima all'architettura Android si trovano, infine, le *Application.* Esse comprendono sia le applicazioni native della versione di Android in uso sul dispositivo, sia le applicazioni di terze parti installate dall'utente sul medesimo dispositivo.

2.3 Versioni di Android

Android ha fatto il suo debutto ufficiale nel 2008 con la sua Versione 1.0, una versione così "antica" da non avere nemmeno un nome in codice distintivo. All'epoca le cose erano piuttosto basiche ma il software, come già detto in precedenza, includeva già una serie di prime app di Google, come Gmail, Maps, Calendar e YouTube.
Ne seguì a breve una nuova versione, la 1.1, uscita il 9 febbraio del 2009, quest'ultima successivamente anche soprannominata Petit Four, quattro piccoli dolcetti. Questo riferimento scherzoso non ha origini molto chiare; tuttavia, sembra che i primi sviluppatori Android si scambiassero spesso e-mail giocose, a causa del crescente entusiasmo in merito ai progressi del sistema operativo. Questa scelta dei dolci è rimasta e si è integrata poi con l'ordine alfabetico iniziato con la lettera C.
Con l'uscita di Android 1.5 Cupcake, che risale all'inizio del 2009, è infatti nata la tradizione per cui ogni aggiornamento della versione di Android viene contraddistinto da un numero, che si riferisce alla versione, e dal nome di un dolce.
Inizialmente i rilasci interni di Android erano contrassegnati con delle milestone, seguite da un numero progressivo. Così avevamo nomi come m3-rc20a e m5-rc15 . I problemi iniziarono quando gli sviluppatori nelle varie email facevano riferimento

solo al codice numerico (es. rc20a), creando confusione, in quanto non si riusciva a capire a quale milestone si facesse riferimento. È così che si decise di ricorrere a una catalogazione progressiva alfabetica (in uso ancora ora), iniziando dalla lettera C, in quanto terza release ufficiale di Android.
La Versione 1.5., Cupcake, introdusse numerosi perfezionamenti all'interfaccia Android, tra cui la prima tastiera su schermo, qualcosa che sarebbe stato sempre più necessario man mano che i telefoni si allontanavano dal modello di tastiera fisica un tempo onnipresente.
Di seguito vengono riportati tutti i nomi dei sistemi operativi Android succedutisi nel tempo fino ad oggi (Figura 2.4):

- Android 1.5 Cupcake (27 Aprile 2009);
- Android 1.6 Donut (15 Settembre 2009);
- Android 2.0 – 2.1 Eclair (26 Ottobre 2009);
- Android 2.2 – 2.2.3 Froyo (20 Maggio 2010);
- Android 2.3 – 2.3.7 Gingerbread (6 Dicembre 2010);
- Android 3.0 – 3.2.6 Honeycomb (22 Febbraio 2011);
- Android 4.0 – 4.0.4 Ice Cream Sandwich (18 Ottobre 2011);
- Android 4.1 – 4.3.1 Jelly Bean (9 Luglio 2012);
- Android 4.4 – 4.4.4 KitKat (31 Ottobre 2013);
- Android 5.0 – 5.1.1 Lollipop (12 Novembre 2014);
- Android 6.0 – 6.0.1 Marshmallow (5 Ottobre 2015);
- Android 7.0 – 7.1.2 Nougat (22 Agosto 2016);
- Android 8.0 – 8.1 Oreo (21 Agosto 2017);
- Android 9.0 Pie (6 Agosto 2018);
- Android 10 (3 settembre 2019).

Per la versione successiva a Marshmallow, Google scelse di lasciare che fossero gli utenti stessi a scegliere quale fosse il nuovo nome di Android. Il nome ufficiale di Android 7.0 N è quindi stato scelto in Android Nougat (ovvero mandorlato/torrone), mentre Android 8.0 O è stato, invece, denominato Oreo, dal nome del famoso biscotto americano. Nel 2018 Google si è trattenuta dal voler giocare con i propri utenti e senza grande sorpresa ha annunciato Android 9.0 Pie.
Android Q, poi rilasciata ufficialmente con il nome di Android 10, è la prima versione del sistema operativo a non utilizzare più nomi di dolci per l'identificazione a livello

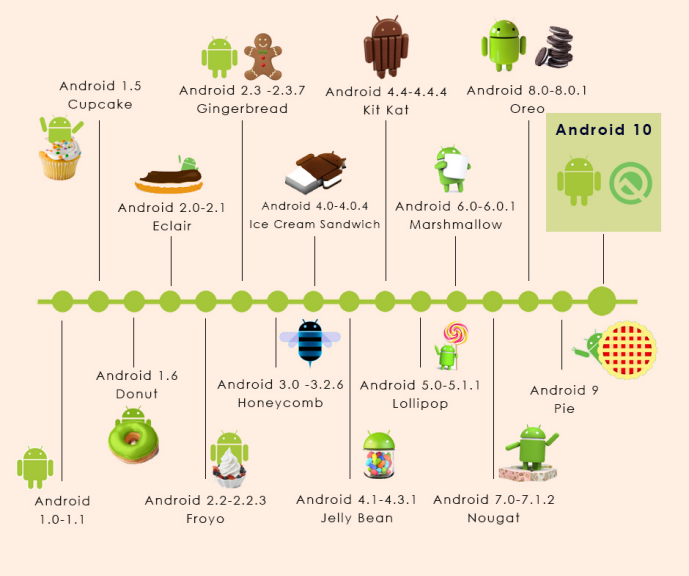

Figura 2.4: Evoluzione versioni di Android

commerciale. Tale scelta è stata motivata dal fatto che molti di questi dolciumi erano sconosciuti e difficili da pronunciare in alcune lingue.

È attualmente in fase di sviluppo la Versione 11 di Android che, presumibilmente, verrà rilasciata al pubblico a settembre 2020.

2.4 Sviluppo App Android

Il primo passo per iniziare a sviluppare app Android consiste nell'installare l'SDK (Software Development Kit), vale a dire un kit di sviluppo di software. Attraverso questo kit qualsiasi utente può procurarsi il codice sorgente di ogni versione Android e modificarlo o migliorarlo autonomamente. Il codice sorgente è un testo scritto in linguaggio di programmazione che, una volta elaborato, porta alla realizzazione di un programma eseguibile dal processore. I linguaggi di programmazione alla base della maggior parte delle applicazioni per Android sono Java e C/C++.
Quando si crea un nuovo progetto, una delle prime scelte che è necessario fare riguarda proprio il "Minimum SDK", ossia la versione del sistema operativo (API) per cui si intende sviluppare l'applicazione (MarhMallow, Lollipop, KitKat, Jelly bean, etc). Questa scelta influisce, infatti, sulla capacità di download dell'applicazione dallo store di Google. In sostanza, solo i possessori di dispositivi con quella versione di sistema operativo, o con una versione superiore, potranno vedere e installare l'applicazione. Uno dei problemi di Android è, infatti, quello della frammentazione delle versioni: purtroppo non tutti i device vengono aggiornati in tempi brevi o non vengono proprio più aggiornati. Questo complica le cose perchè, per sviluppare applicazioni complesse o con funzioni particolari, servono API di livello alto, ma queste non sono supportate da versioni vecchie di Android ancora abbastanza diffuse nei cellulari in commercio,

e quindi non abilitare tali versioni ridurrebbe il possibile bacino di utenti.
Per il progetto relativo alla presente tesi è stato scelto come mininum SDK la Versione 6.0, Marshmallow, corrispondente al livello di API 23.

2.4.1 Android studio

Nonostante la prima generazione di sviluppatori Android si sia formata su Eclipse, tra il 2013 ed il 2014 è stata progettata una piattaforma per lo sviluppo di App in Android che progressivamente ne ha preso il posto, diventando l'IDE primario di Android; tale piattaforma è Android Studio (Figura 2.5).
Questo ambiente di lavoro efficiente, intuitivo e multipiattaforma, ottenibile dal sito ufficiale di Android, è venuto al mondo all'insegna della flessibilità e della praticità. Innanzitutto è figlio di IntelliJ, un IDE molto intuitivo ed efficiente prodotto dalla società JetBrains. In secondo luogo è scaturito dalla stessa Google e nasce appositamente per Android, integrandosi con tutto il suo ecosistema. Esso permette di realizzare progetti per smartphone e tablet, nonchè per dispositivi indossabili, Android Auto e Android TV. Google ha a disposizione un universo di servizi cloud ed Android Studio si offre come ponte per creare app che dialoghino con essi.

Figura 2.5: Logo di Android Studio

Capitolo 3

Analisi dei requisiti e progettazione

Nel presente capitolo, dopo una breve esposizione degli obiettivi sottesi alla realizzazione di questo progetto, si procederà ad illustrarne in maniera dettagliata le funzionalità attraverso l'analisi dei requisiti ed il diagramma dei casi d'uso. Per quanto riguarda la progettazione, alcuni dei casi d'uso più rappresentativi verranno ulteriormente illustrati tramite il diagramma delle attività, mentre mediante i mockup si mostrerà la veste grafica che assumerà l'applicazione. In ultimo, si descriverà la componente dati dell'applicazione stessa, nello specifico il database impiegato.

3.1 Descrizione del progetto

Questo progetto è finalizzato alla realizzazione di un'applicazione per la gestione di eventi domestici, cene di compleanno, riunioni con amici e parenti nel delicato periodo del post-lockdown nel quale, gioco forza, il desiderio di riprendere i nostri contatti e riappropriarci della nostra quotidianità ci impone, comunque, di attenerci a delle semplici misure di sicurezza per tutelare noi stessi ed i nostri ospiti.
Attraverso l'utilizzo di questa app non solo sarà più semplice e veloce procedere alla distribuzione degli inviti ai vari ospiti e, conseguentemente, conoscere il numero esatto delle persone che parteciperanno (sempre all'interno del range massimo consentito dall'ampiezza dei locali in cui si svolgerà l'evento), ma sarà possibile per l'organizzatore stabilire la distribuzione dei posti a tavola nel rispetto delle regole del distanziamento sociale richiesto tra persone conviventi e non.
Altra caratteristica dell'applicazione è,inoltre, quella di consentire il controllo di eventuali intolleranze alimentari dichiarate dagli ospiti prima della predisposizione del menù da offrire durante l'evento ed, altresì, quella di tenere traccia di tutti gli eventi già tenuti, con indicazione delle persone e dei menù realizzati in ogni singola occasione, al fine di evitare di riproporre a breve distanza i medesimi piatti a identici commensali.

3.2 Analisi dei requisiti

In questa sede verranno illustrati i requisiti che caratterizzeranno il progetto. Verranno distinte le funzionalità che dovranno essere implementate nell'applicazione,

cioè i requisiti che descrivono ciò che l'app dovrà fare (requisiti funzionali), da quelle che, invece, esprimono le proprietà, oppure le qualità, che l'applicazione deve possedere, ovvero l'insieme di vincoli e regole di tipo organizzativo o tecnico che sottendono alla sua realizzazione (requisiti non funzionali).

3.2.1 Descrizione dei requisiti

Requisiti funzionali

L'applicazione dovrà garantire le seguenti funzionalità fondamentali:

- inserimento, durante il login, dei propri dati personali, quali nome, cognome, email, cellulare, immagine del profilo ed eventuali intolleranze alimentari;
- modifica della lista delle intolleranze e dell'immagine del profilo in qualunque momento, anche dopo la creazione dell'account;
- organizzazione di un evento, tra cui, preliminarmente, l'inserimento di nome, luogo, data e orario dello stesso, più altre operazioni successive;
- creazione di una lista invitati in grado di contenere sia persone che già sono autenticate, sia persone che non possiedono ancora un account e per cui è necessario l'inserimento manuale dei dati;
- visualizzazione delle eventuali intolleranze di tutti gli invitati;
- realizzazione di un menù che possa contenere sia piatti già proposti in eventi passati che altri del tutto nuovi;
- visualizzazione, al termine della realizzazione del menù e prima di completare la procedura di creazione dell'evento, di tutte le occasioni, con specifica indicazione della data in cui alcune pietanze sono state proposte ai medesimi invitati e di cui forse l'organizzatore non ricorda;
- visualizzazione degli eventi rifiutati, a cui si è stati invitati, in una sezione "storia";
- visualizzazione di tutti gli eventi correnti, cioè quelli che ancora non hanno luogo, in una sezione "attivi";
- visualizzazione di tutti i dati relativi ad un singolo evento;
- possibilità di accettare o rifiutare gli inviti ricevuti, anche dopo averli precedentemente accettati;
- possibilità per l'organizzatore di cancellare l'evento;
- possibilità di eliminare l'evento dalla lista degli eventi passati;

- possibilità di impostare il proprio stato di salute su “Sano” o “Malato” (nel secondo caso, l’utente non può essere invitato a nessun evento);
- possibilità di impostare lo stato del profilo su “Pubblico” (gli utenti possono vedere tutti i dati dell’utente) o “Privato” (gli utenti possono vedere solo il nome);
- possibilità di inserimento della metratura della stanza in cui si svolgerà l’evento e dei mq che si vogliono concedere a ciascuna persona con calcolo del tetto massimo di utenti che possono essere invitati;
- possibilità di scelta della disposizione di tavoli e invitati tramite un drag&drop in modo da posizionare gli invitati ad almeno 1 m circa di distanza l’uno dall’altro a meno che non appartengano ad uno stesso nucleo familiare;
- possibilità di apportare in un secondo momento modifiche all’evento già creato;
- visualizzazione, da parte degli invitati, di una notifica push che li informa della creazione dell’evento, della sua eventuale cancellazione o modifica e che, il giorno dell’evento, ne rammenta la ricorrenza;
- possibilità di impostare il tema scuro;
- possibilità di tenersi informati tramite una nuova tab dove possiamo trovare informazioni sui protocolli di sicurezza anti COVID-19 e sulle regole di sicurezza nel campo della ristorazione.

Requisiti non funzionali

- obbligo di registrarsi prima di accedere all’applicazione;
- obbligo dell’indicazione dei campi relativi al nome, all’email e al cellulare per la creazione di un account (non vi è un controllo di sicurezza su numeri di telefono e sulle email per verificare che i dati inseriti siano effettivamente esistenti o che si stia inserendo un’informazione “falsa”);
- obbligo di inserimento di una email valida (quindi contenente i caratteri “@” e “.”);
- obbligo di inserimento di una password di almeno 6 caratteri;
- obbligo per l’utente di scegliere le proprie intolleranze da un nutrito gruppo prestabilito, senza possibilità di inserimento manuale;
- obbligo di inserimento dei campi nome, indirizzo, data e ora per la creazione di un evento;
- impossibilità per l’utente di organizzare eventi in date antecedenti a quella corrente;

- obbligo che la lista degli invitati relativa ad un evento contenga almeno un invitato;
- obbligo per l'utente di creare un menù che contenga almeno una tipologia di portata;
- necessità dell'utilizzo di un database online per la memorizzazione dei dati relativi agli utenti e agli eventi, oltre che per consentire ad ogni utente l'accesso ai contatti che si desidera invitare e agli eventi di cui si vogliono visualizzare le informazioni;
- obbligo per l'utente, in fase di accettazione dell'evento, di compilare un'autocertificazione nella quale si attesti il buono stato di salute e il mancato contatto nei 15 gg. precedenti con persone risultate positive al COVID-19.

3.2.2 Diagramma dei casi d'uso

I casi d'uso rappresentano le funzionalità che un sistema possiede, ovvero le operazioni che un utente, dall'esterno, può eseguire. Condizione necessaria affinché una funzionalità venga considerata come un caso d'uso è che abbia una relazione diretta con l'utente. Il punto di partenza di un progetto è costituito normalmente dagli use case diagram, o diagrammi dei casi d'uso, che illustrano in maniera semplice quello che il sistema può fare. Gli use case diagram rappresentano, insomma, una vista esterna del sistema. Gli use case diagram sono composti da casi d'uso, attori e associazioni. Più specificatamente:

- i casi d'uso rappresentano sequenze d'azioni svolte dal sistema;
- gli attori (ruoli) rappresentano persone (o altri sistemi) che interagiscono con il sistema modellato;
- le associazioni tra attore e caso d'uso rappresentano il fatto che l'attore può "interagire" con il caso d'uso.

Nella Figura 3.1 viene mostrato il diagramma dei casi d'uso relativo al progetto che è stato realizzato.

3.3 Progettazione

3.3.1 Componente applicativa

Durante la fase di progettazione dell'applicazione devono tenersi presenti le funzionalità che il progetto dovrà soddisfare partendo dalla raffigurazione della mappa dell'applicazione, rappresentativa delle caratteristiche che la sua struttura dovrà presentare, passando per una descrizione dettagliata di alcuni dei casi d'uso più significativi, rappresentati dai relativi diagrammi delle attività, fino ad arrivare alla

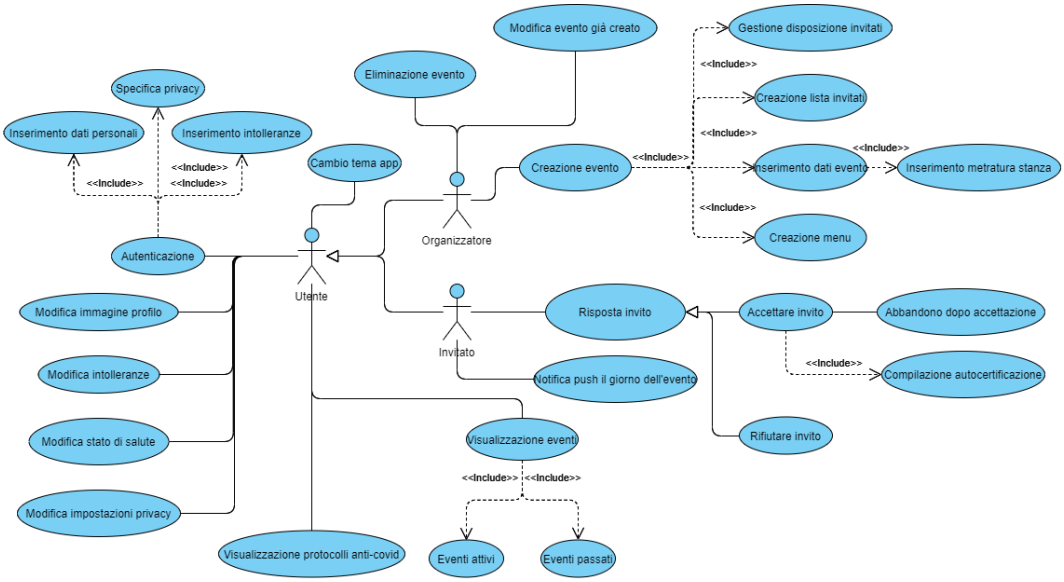

Figura 3.1: Diagramma dei casi d'uso

progettazione del suo aspetto grafico attraverso l'utilizzo di mockup.

Mappa dell'applicazione

Attraverso la mappa dell'applicazione (Figura 3.2) è stata rappresentata schematicamente la struttura del progetto al fine di consentire la visualizzazione dei suoi contenuti, distinti nelle diverse pagine dell'applicazione, con indicazione delle modalità di accesso per l'utente.

Diagramma delle attività

I diagrammi di attività descrivono il flusso delle operazioni che avvengono all'interno di uno specifico caso d'uso.
Le figure seguenti mostrano il diagramma delle attività relativo alle fasi di autenticazione (Figura 3.3), di creazione della lista invitati (Figura 3.4), di modifica dello stato di salute (Figura 3.5) e di risposta all'invito ricevuto (Figura 3.6).

Mockup

Per mezzo dei mockup viene mostrato, almeno in forma semplificata, l'aspetto che, al termine del lavoro, le singole pagine dell'applicazione dovranno presentare.
Esistono diversi livelli di mockup, ciascuno con un grado di complessità e di dettaglio superiori al precedente; di seguito vengono mostrati i mockup di livello 0 relativi all'applicazione nativa Android realizzati con il software Balsamiq riguardanti le diverse activity dell'applicazione. Più precisamente, vengono mostrati i mockup relativi alla fase di autenticazione (Figura 3.7 - Figura 3.11), alla main activity

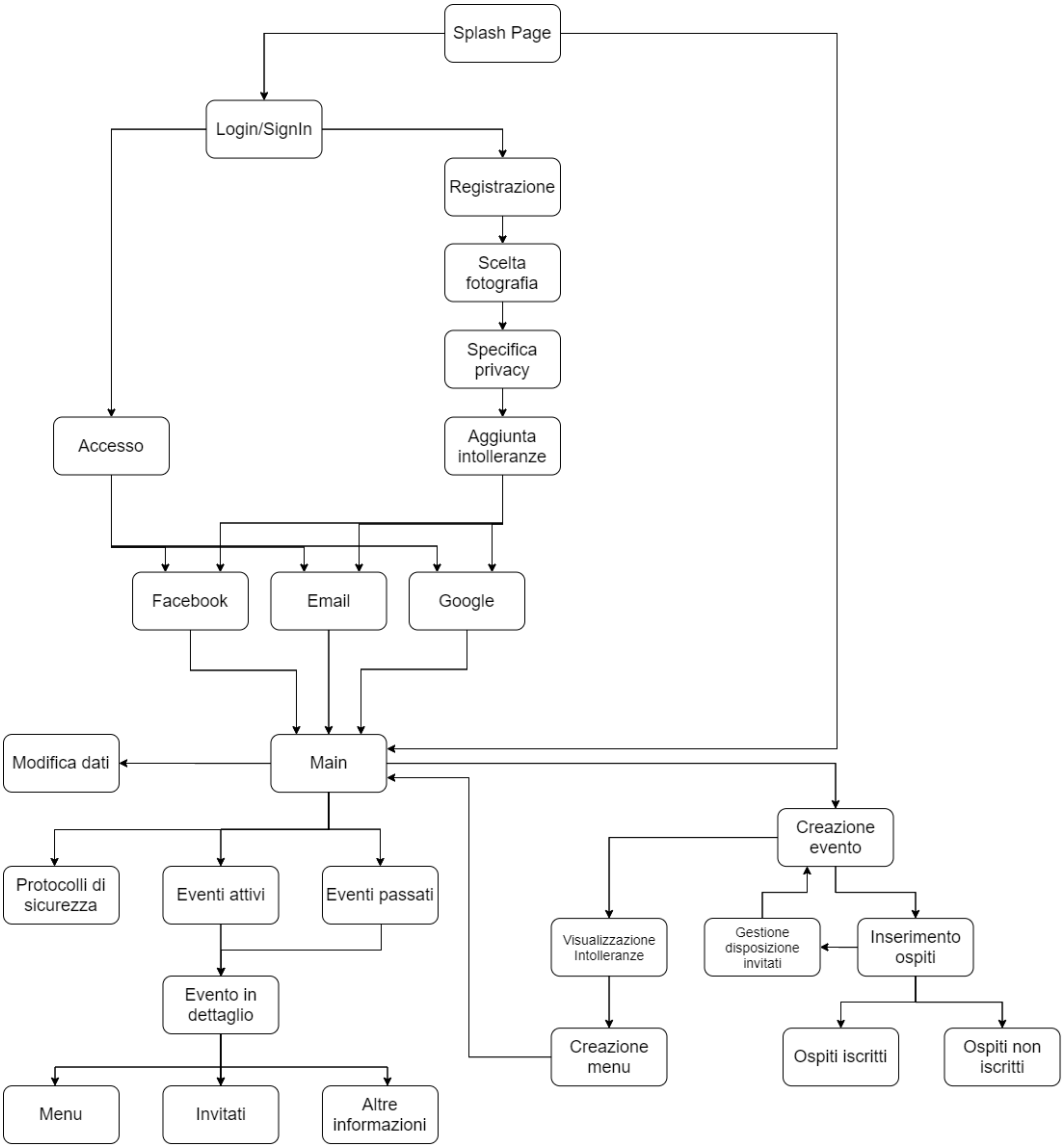

Figura 3.2: Mappa dell'applicazione

(Figura 3.12), ad un evento (Figura 3.13 - Figura 3.15) e alla creazione di quest'ultimo (Figura 3.16 - Figura 3.20).

3.3.2 Componente dati

Come detto in precedenza, questa applicazione necessita di un database online per poter funzionare: per questa ragione si è deciso di usufruire dei servizi forniti dalla piattaforma di sviluppo Firebase.

Firebase

Per ciò che concerne tale piattaforma, sono stati utilizzati i seguenti servizi:

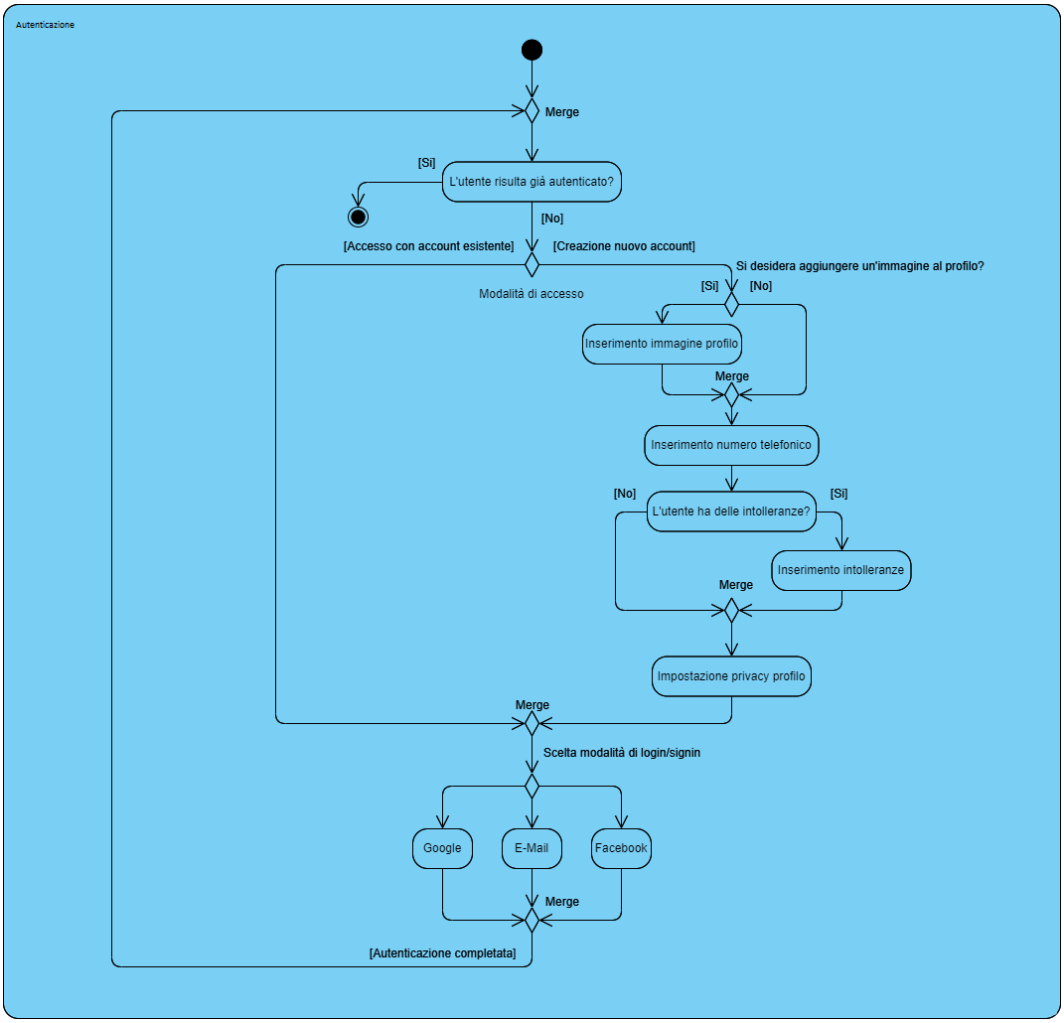

Figura 3.3: Diagramma delle attività relativo all'autenticazione

- *Authentication*, per semplificare l'implementazione della registrazione e dell'autenticazione, tramite email e password, oppure tramite account Google o Facebook;
- *Storage*, per memorizzare le immagini di profili inserite dagli utenti in fase di registrazione o in seguito, tutte identificate da un URL univoco;
- *Firebase Cloud Messaging (FCM)*, per inviare notifiche push a specifici utenti;
- *Functions*, per attivare l'FCM in risposta a delle chiamate HTTPS;
- *Cloud Firestore*, per immagazzinare e sincronizzare in un database NoSQL i dati sia dal lato client sia dal lato server.

In quanto NoSQL, Firestore è un DBMS non relazionale, cioè non possiede una struttura rigida dove i dati vengano salvati all'interno di tabelle prestabilite; al contrario, è costituito da collezioni: queste ultime presentano al loro interno documenti

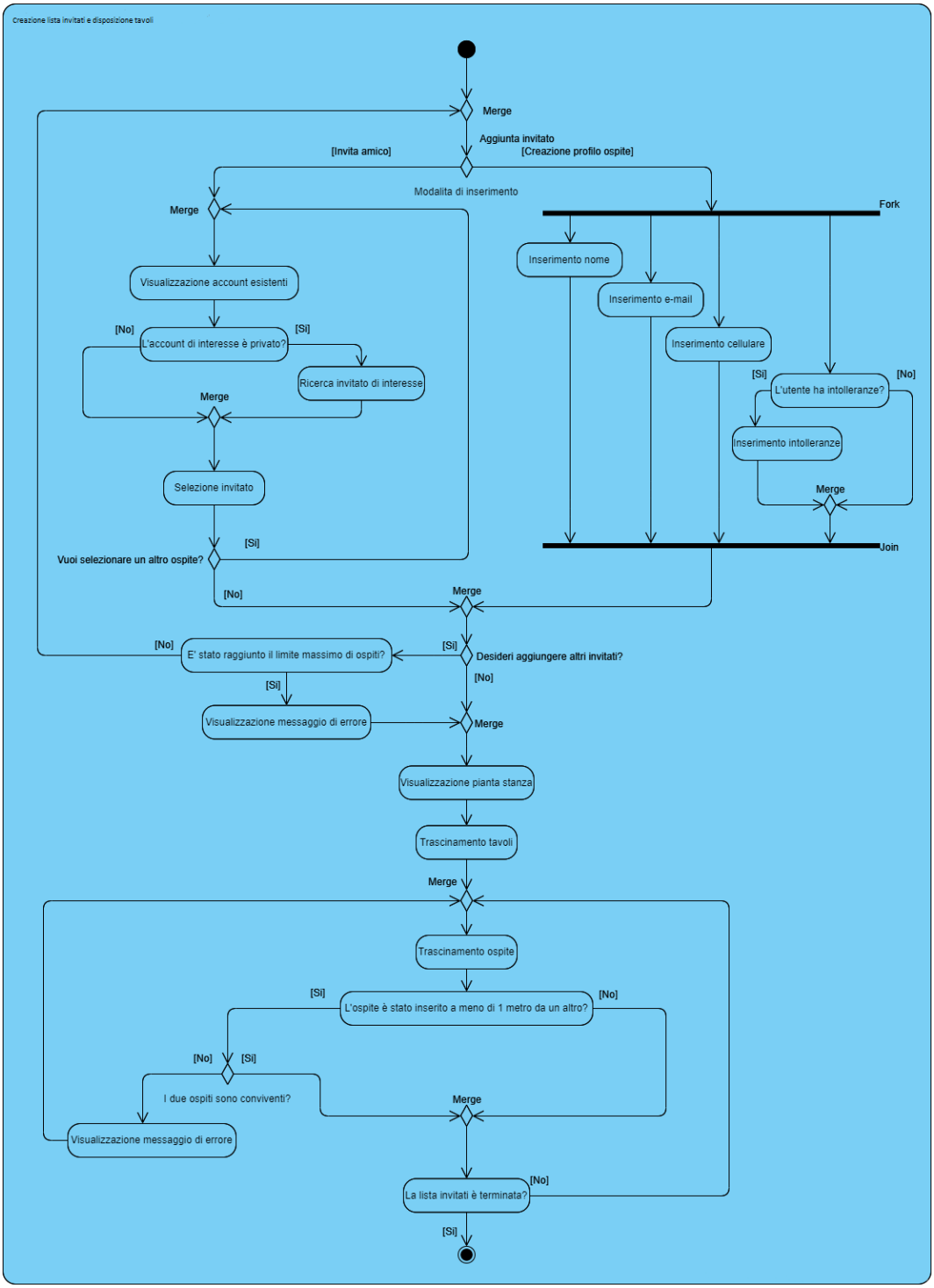

Figura 3.4: Diagramma delle attività relativo alla creazione della lista degli invitati

in formato JSON, ciascuno con un Id univoco per distinguerlo dagli altri, ed una serie di campi, di numero non fisso, ma variabile, che contengono i dati. A prima vista può sembrare che questa tipologia di database sia priva di qualunque regola o struttura, ma queste ultime, in realtà, sono determinate dall'implementazione del client (nel

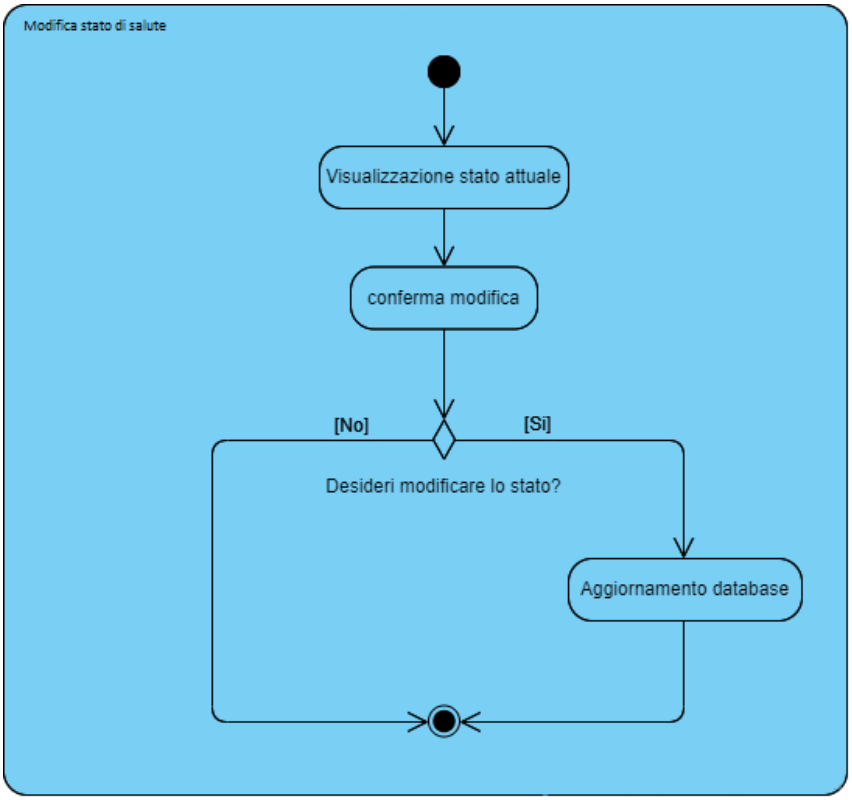

Figura 3.5: Diagramma delle attività relativo alla modifica dello stato di salute

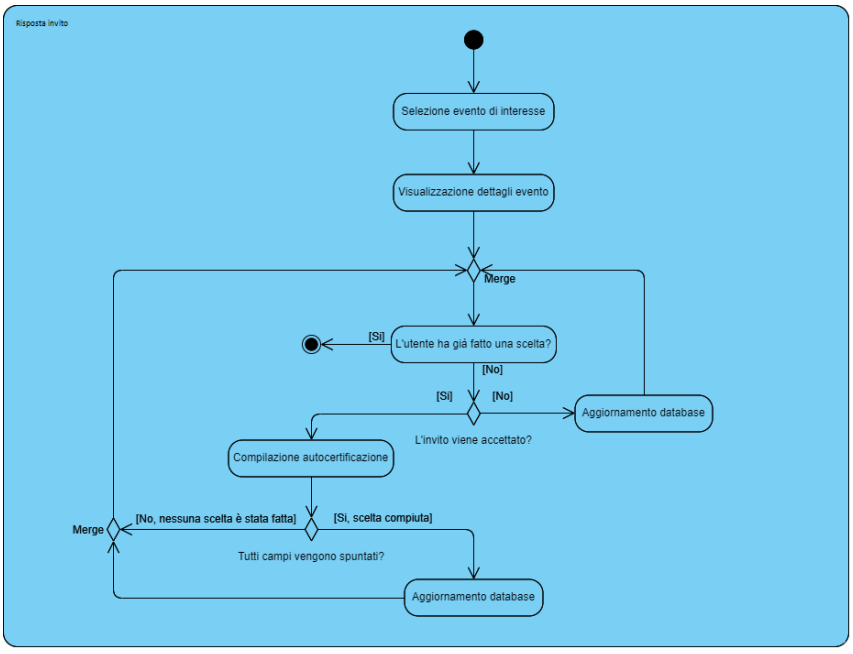

Figura 3.6: Diagramma delle attività relativo alla risposta all'invito

caso specifico, l'applicazione). I vantaggi rispetto ai database SQL si apprezzano soprattutto in termini di scalabilità e prestazioni: i DBMS SQL, infatti, per poter

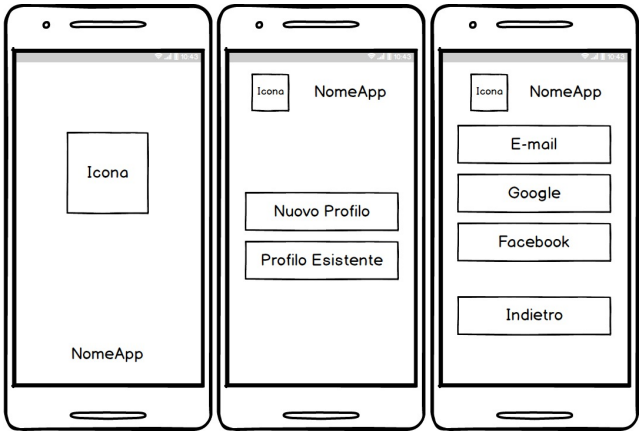

Figura 3.7: Mockup relativi all'accesso

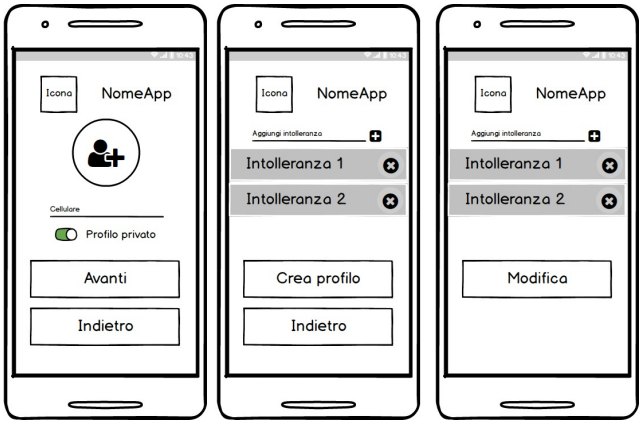

Figura 3.8: Mockup relativi alla crezione di un account

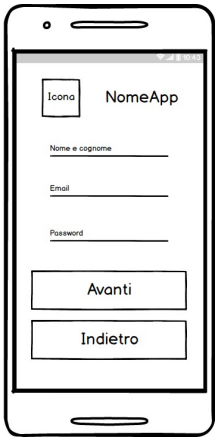

Figura 3.9: Mockup relativo all'accesso tramite email

eseguire qualunque modifica devono prima verificare che non vengano compromessi i vincoli (come ad esempio quelli di integrità dei dati) che ne determinano la correttezza, e quelli di integrità referenziale, che determinano i collegamenti tra le varie tabelle. Nella Figura 3.21 viene riportato l'insieme delle collezioni del database con l'insieme

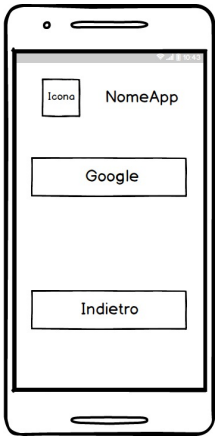

Figura 3.10: Mockup relativo all'accesso tramite account Google

Figura 3.11: Mockup relativo all'accesso tramite account Facebook

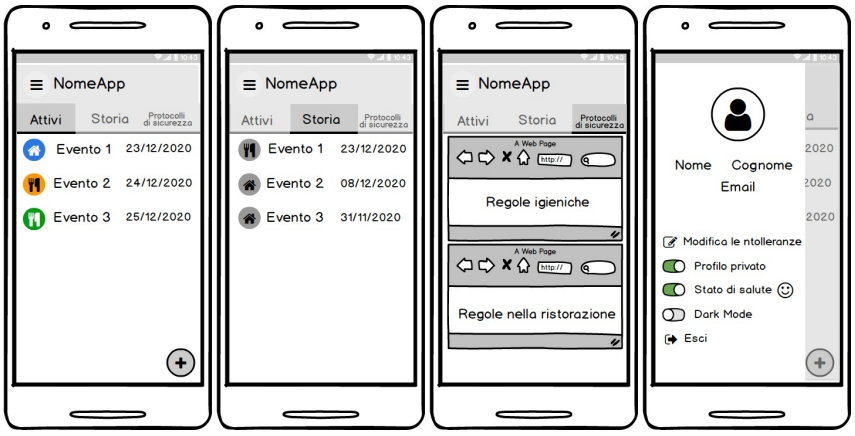

Figura 3.12: Mockup relativi alla main activity

dei campi che caratterizzano i documenti di ciascuna di esse.

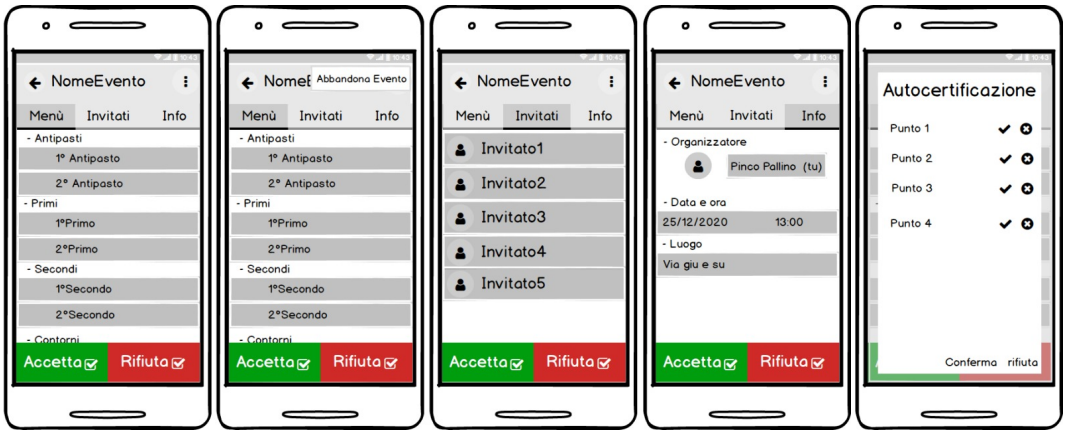

Figura 3.13: Mockup relativi ad un invito ad un evento in attesa di risposta

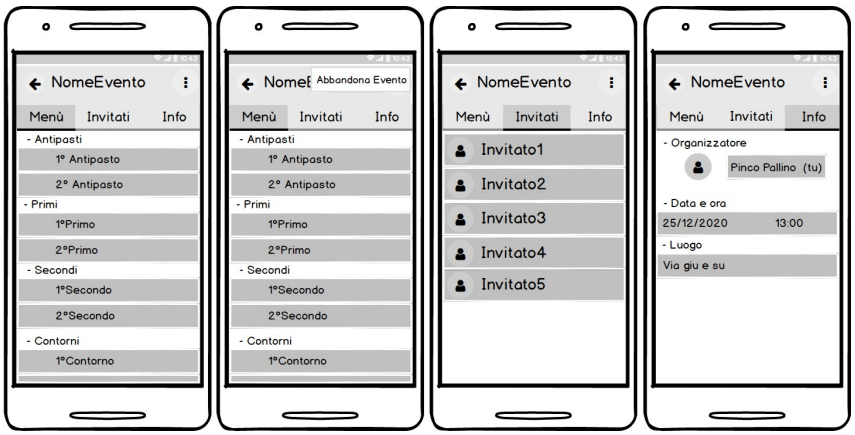

Figura 3.14: Mockup relativi ad un invito ad un evento accettato

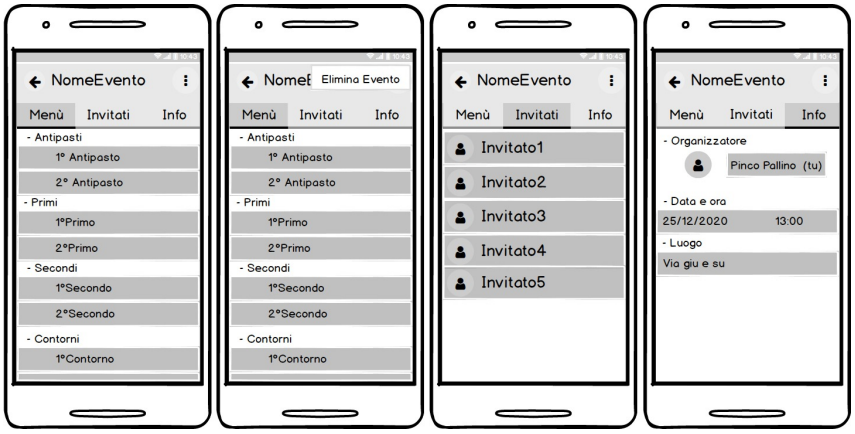

Figura 3.15: Mockup relativi ad un invito ad un evento rifiutato

Figura 3.16: Mockup relativo all'inserimento dei dati per un nuovo evento

Figura 3.17: Mockup relativo alla creazione della lista degli invitati

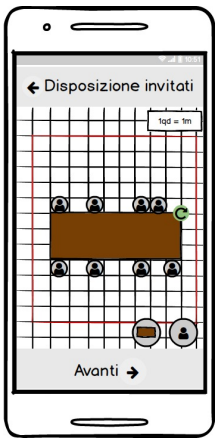

Figura 3.18: Mockup relativo alla disposizione degli ospiti nel tavolo

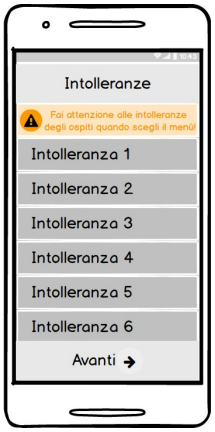

Figura 3.19: Mockup relativo alle intolleranze degli invitati

Figura 3.20: Mockup relativo alla creazione del menù

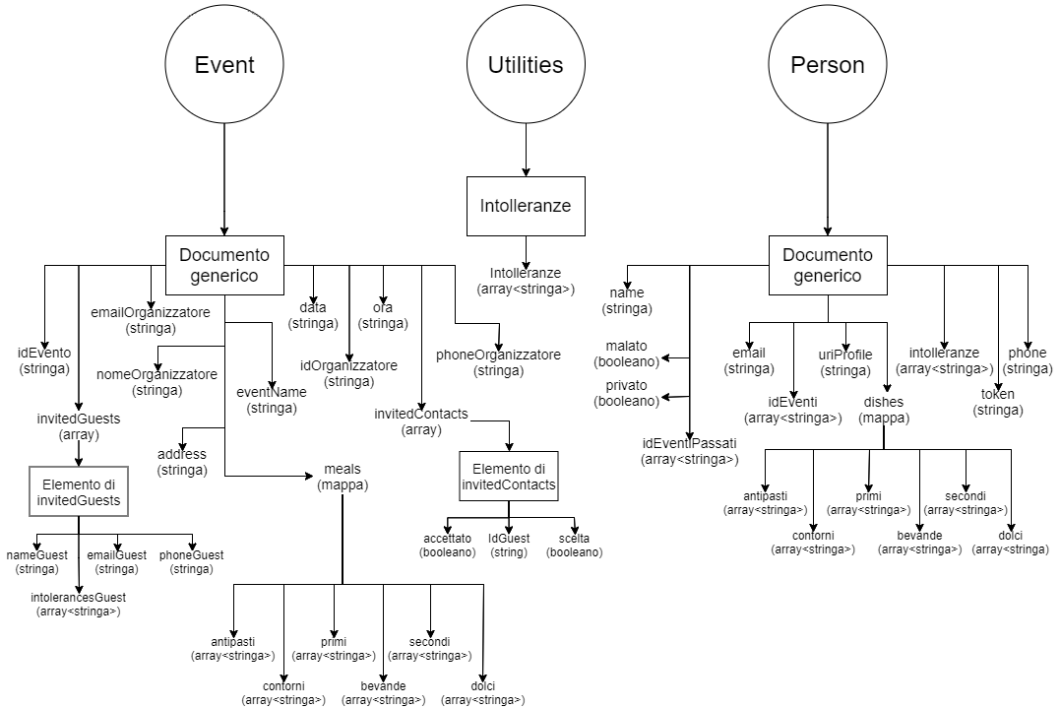

Figura 3.21: Schema del database Firestore

Capitolo 4

Implementazione dell'app e manuale utente

L'applicazione nativa Android, oggetto di questa tesi, è stata realizzata utilizzando l'IDE (Integrated Development Environment) Android Studio, il linguaggio XML per la definizione dei layout di ogni singola pagina (che in Android prendono il nome di Activity) ed il linguaggio di programmazione Java per definirne il comportamento. Come già riferito, è stato, poi, scelto come SDK (Software Development Kit) minima l'API (Application Programming Interface) 23, corrispondente ad Android 6 Marshmallow, al fine di trovare un giusto compromesso, in termini di compatibilità, non sempre facile da raggiungere in un mondo variegato come quello dei dispositivi Android.

4.1 Struttura del progetto

Il progetto dell'applicazione, oggetto della presente tesi, è stato strutturato come descritto nella figura Figura 4.1.

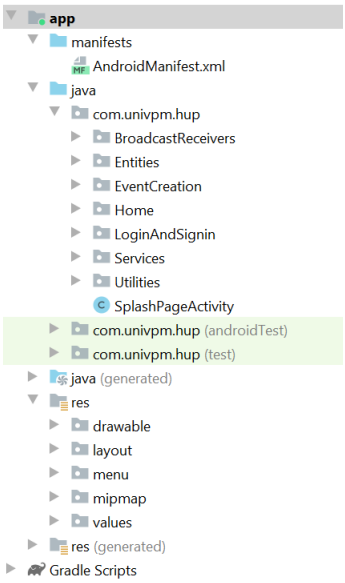

Figura 4.1: Struttura del progetto

Dal punto di vista implementativo, le componenti di particolare interesse sono le seguenti:

- `manifests:` si tratta di una cartella contenente `AndroidManifest.xml` quale unico file. Presente in ciascuna applicazione, `manifest` raccoglie informazioni basilari quali, ad esempio, il nome dell'app e la sua icona, i vari componenti della stessa (`Activity, Service, BroadcastReceiver, ContentProvider`), i permessi che la app richiede all'utente, etc.
- `BroadcastReceiver`: package realizzato per contenere tutte le classi Java che estendono la superclasse `BroadcastReceiver`.
- `Entities:` package contenente classi Java create per modellare i dati, in particolare, quelli relativi agli eventi (`Event`), agli account (`Person`), agli invitati agli eventi (`Guest`) e ai gruppi di invitati (`Group`).
- `EventCreation`: package contenente le classi Java relative alle Activity e ai Fragment che caratterizzano la fase della creazione dell'evento.
- `Home`: package contenente la classe Java corrispondente alla `MainActivity`, oltre ad altre classi.
- `LoginAndSignin`: package contenente le classi Java relative alle Activity ed ai Fragment che caratterizzano la fase di `Login` o `Signin`.
- `Services:` package realizzato per contenere tutte le classi Java che estendono la superclasse `Service`.
- `Utilities:` package contenente classi di supporto alle varie Activity (ad esempio, tutte le classi `Adapter` per le `RecyclerView`).
- `SplashPageActivity:` classe Java che rappresenta la prima Activity che viene lanciata dall'applicazione.
- `drawable:` si tratta di una cartella contenente tutte le immagini che l'utente potrà visualizzare durante l'utilizzo dell'applicazione.
- `layout:` si tratta di una cartella contenente i file XML che rappresentano l'aspetto delle Activity e dei Fragment; in pratica costituisce l'architettura grafica dei componenti dell'interfaccia utente.
- `menu:` cartella contenente file XML rappresentanti i menu impiegati nell'applicazione (come, ad esempio, nella NavigationView e nelle ToolBar).
- `mipmap:` cartella contenente l'icona dell'applicazione in varie risoluzioni.
- `values:` cartella contenente, a sua volta, file XML che rappresentano i colori (definiti in due file con e senza night mode), le dimensioni, le stringhe (definite in due file rappresentanti la lingua italiana e quella inglese) e gli stili usati nell'applicazione.

4.2 Implementazione

In questa sezione verranno analizzati i diversi componenti dell'applicazione e verranno descritte le loro funzioni.

4.2.1 SplashPageActivity

Prima Activity ad essere visualizzata all'avvio, la `SplashPageActivity` mostra il logo per alcuni secondi, dopo i quali effettua un controllo sulla presenza di una connessione Internet; in caso negativo mostra un Dialog di errore in quanto l'applicazione non può funzionare senza. Verificata la presenza di connessione, qualora l'utente sia già autenticato, si passa direttamente alla `MainActivity`, altrimenti si verrà indirizzati all'`Activity` di `LoginAndSignin` per la fase di autenticazione.

```
1  /*
2   * Questo è lo Splash screen che viene mostrato ogni volta che l'
       applicazione viene avviata.
3   * In questa schermata inoltre si controlla se è attiva la connessione
       ad internet
4   */
5  public class SplashPageActivity extends AppCompatActivity {
6
7      @Override
8      protected void onCreate(Bundle saveInstanceState) {
9          super.onCreate(saveInstanceState);
10         setContentView(R.layout.activity_splash_page);
11
12         new Handler().postDelayed(new Runnable() {
13             @Override
14             public void run() {
15                 if (!isOnline()) {
16                     buildDialog().show();
17                 } else {
18                     goNextActivity();
19                 }
20             }
21         }, 3000);
22     }
23
24     /*
25      * Questo metodo serve per stabilire se un utente si è già loggato
       nel dispositivo oppure se non ha ancora effettuato nessun accesso
26      */
27     private void goNextActivity() {
28         FirebaseUser myAuth = FirebaseAuth.getInstance().
       getCurrentUser();
29         if (myAuth == null) {
30             Intent i = new Intent(SplashPageActivity.this,
       LoginAndSigninActivity.class);
31             startActivity(i);
```

```
32              finish();
33          } else if (myAuth != null) {
34              Intent i = new Intent(SplashPageActivity.this,
    MainActivity.class);
35              startActivity(i);
36              finish();
37          }
38      }
39
40      /*
41       * Questo metodo serve per controllare se uno tra la connessione
    dati e il wifi è attivo quindi se si ha connessione ad internet
42       * e ritorna true se la connessione c'è, mentre ritorna false se
    non è connesso
43       */
44      private boolean isOnline() {
45
46          ConnectivityManager connectivityManager = (ConnectivityManager
    ) getSystemService(Context.CONNECTIVITY_SERVICE);
47          return connectivityManager.getActiveNetwork() != null &&
    connectivityManager.getActiveNetworkInfo().isConnectedOrConnecting
    ();
48      }
49
50      /*
51       * Questo è il dialog di errore nel caso non ci fosse nessuna
    connessione ad internet
52       */
53      private AlertDialog.Builder buildDialog() {
54          AlertDialog.Builder builders = new AlertDialog.Builder(this);
55          builders.setOnCancelListener(new DialogInterface.
    OnCancelListener() {
56              @Override
57              public void onCancel(DialogInterface dialog) {
58                  recreate();
59              }
60          });
61          builders.setTitle(getResources().getString(R.string.noInternet
    ))
62                  .setMessage(getResources().getString(R.string.
    retryConnection))
63                  .setPositiveButton(getResources().getString(R.string.
    retry), new DialogInterface.OnClickListener() {
64                      @Override
65                      public void onClick(DialogInterface
    dialogInterface, int i) {
66                          recreate();
67                      }
68                  });
69
70          return builders;
```

```
71      }
72  }
73
```

Listato 4.1: Listato relativo alla Splash Page

4.2.2 LoginAndSignin

Package che, tra le altre cose, contiene la classe `LoginAndSigninActivity` ed una serie di Fragment che si avvicendano per implementare le varie fasi dell'autenticazione. Per l'accesso con Facebook, tuttavia, è stato necessario creare un' Activity a parte, denominata `FacebookAccess`.

4.2.3 Home

Package che contiene la classe `MainActivity`. Da quest'ultima l'utente può visualizzare tutti gli eventi attivi, che siano da lui organizzati o ai quali sia stato invitato, gli eventi passati e i protocolli di sicurezza anti COVID-19. Dalla medesima Activity, tramite una NavigationView, è possibile visualizzare le informazioni relative all'account dell'utente ed anche effettuare modifiche alle stesse (ad esempio, relativamente all'immagine del profilo, allo stato di salute, all'impostazione della privacy e alle intolleranze alimentari). All'interno del package è contenuta, anche, l'Activity `EventInfo`, che mostra informazioni dettagliate relative all'evento. Nel caso di ricezione di un invito, sarà possibile accettarlo, rifiutarlo immediatamente o anche successivamente all'accettazione. L'organizzatore dell'evento, invece, avrà la possibilità di modificare le informazioni relative all'evento, oppure potrà annullarlo definitivamente.

4.2.4 EventCreation

Package che contiene l'Activity `NewEvent`. Non appena questa diventa visibile sullo schermo, viene mostrato un Dialog per chiedere all'utente se vuole organizzare un evento seguendo le normative anti COVID-19. In caso positivo, si dovrà, anche, specificare se si tratta di evento organizzato all'aperto o al chiuso e, in quest'ultimo caso, si dovranno specificare le dimensioni della stanza e i metri quadrati che si vogliono concedere pro-capite, per consentire all'applicazione di calcolare il numero massimo degli invitati. Successivamente, sarà possibile inserire le informazioni, quali nome, data, ora ed indirizzo dell'evento.

```
1 //activity dedicata all'inserimento dei primi dati relativi all'evento
      che si intende creare
2 public class NewEvent extends AppCompatActivity {
3
4     //definizione degli oggetti che rappresenteranno le viste
5     private Button inviteGuestsButton;
6     private Button nextButton;
```

```
7     private Button eventDateButton;
8     private Button eventHourButton;
9     private TextInputEditText eventName;
10    private TextInputEditText eventLocation;
11    public TextInputEditText eventHour;
12    public TextInputEditText eventDate;
13    private MaterialToolbar newEventTopAppBar;
14    //definizione delle variabili che conterranno i valori immessi
    dall'utente
15    private String name;
16    private String location;
17    private String hour;
18    private String date;
19    private String length = "";
20    private String width = "";
21    private String distance = "";
22    private ArrayList<Guest> list = new ArrayList<>();
23    static boolean firstTime = true;
24    static boolean covid = false;
25    static boolean outside = false;
26    static boolean modify = false;
27    private Bundle bundle = new Bundle();
28    private Bundle menuBundle = new Bundle();
29    private Event currentEvent;
30
31    @Override
32    protected void onCreate(Bundle savedInstanceState) {
33        super.onCreate(savedInstanceState);
34        setContentView(R.layout.activity_new_event);
35        //collegamento degli oggetti alle viste dell'activity
36        inviteGuestsButton = (Button) findViewById(R.id.
    inviteGuestButton);
37        nextButton = (Button) findViewById(R.id.nextButton);
38        eventDateButton = (Button) findViewById(R.id.eventDateButton);
39        eventHourButton = (Button) findViewById(R.id.eventHourButton);
40        eventName = (TextInputEditText) findViewById(R.id.eventName);
41        eventLocation = (TextInputEditText) findViewById(R.id.
    eventLocation);
42        eventHour = (TextInputEditText) findViewById(R.id.eventHour);
43        eventDate = (TextInputEditText) findViewById(R.id.eventDate);
44        newEventTopAppBar = (MaterialToolbar) findViewById(R.id.
    topAppBarNewEvent);
45        //eventuale aggiornamento della lista
46        Intent intent = getIntent();
47        if(intent.getParcelableArrayListExtra("guestList") != null) {
48            list = intent.getParcelableArrayListExtra("guestList");
49        }
50        if(intent.getBundleExtra("roomDimensions") != null) {
51            bundle = intent.getBundleExtra("roomDimensions");
52        }
53        if(intent.getBundleExtra("meals") != null) {
```

```
54              menuBundle = intent.getBundleExtra("meals");
55          }
56          //eventuale recupero dei dati già immessi se l'activity viene
    distrutta
57          getEnteredData();
58          eventName.setText(name);
59          eventLocation.setText(location);
60          eventDate.setText(date);
61          eventHour.setText(hour);
62          //se si sta tornando all'activity per modificare i dati si
    recupereranno i dati dell'evento
63          //per rimostrarli sullo schermo
64          if(intent.getParcelableExtra("currentEvent") != null) {
65              modify = intent.getBooleanExtra("modify", false);
66              currentEvent = intent.getParcelableExtra("currentEvent");
67              assert currentEvent != null;
68              eventName.setText(currentEvent.getEventName());
69              eventLocation.setText(currentEvent.getAddress());
70              eventDate.setText(currentEvent.getData());
71              eventHour.setText(currentEvent.getOra());
72              FirebaseFirestore db = FirebaseFirestore.getInstance();
73              //recupero invitati
74              for(final HashMap<String, Object> member : currentEvent.
    getInvitedContacts()) {
75                  db.collection("Person").document((String) Objects.
    requireNonNull(member.get("idGuest"))).get().addOnSuccessListener(
    new OnSuccessListener<DocumentSnapshot>() {
76                      @Override
77                      public void onSuccess(DocumentSnapshot
    documentSnapshot) {
78                          if(documentSnapshot != null) {
79                              Guest guest = documentSnapshot.toObject(
    Guest.class);
80                              assert guest != null;
81                              guest.setAccettato((Boolean) member.get("
    accettato"));
82                              guest.setScelta((Boolean) member.get("
    scelta"));
83                              guest.setId(documentSnapshot.getId());
84                              list.add(guest);
85                          }
86                      }
87                  });
88              }
89              for(HashMap<String, Object> member : currentEvent.
    getInvitedGuests()) {
90                  Guest guest = new Guest(Objects.requireNonNull(member.
    get("nameGuest")).toString(), Objects.requireNonNull(member.get("
    emailGuest")).toString(), Objects.requireNonNull(member.get("
    phoneGuest")).toString(), (ArrayList<String>) member.get("
    intolerancesGuest"));
```

```
91                  list.add(guest);
92              }
93          }
94          //listener posto "in ascolto" del click del tasto indietro
    nella barra di navigazione
95          newEventTopAppBar.setNavigationOnClickListener(new View.
    OnClickListener() {
96              @Override
97              public void onClick(View v) {
98                  back();
99              }
100         });
101     }
102
103     //salvataggio dati da recuperare
104     @Override
105     public void onSaveInstanceState (Bundle savedInstanceState) {
106         savedInstanceState.putString("eventName", name);
107         savedInstanceState.putString("eventDate", date);
108         savedInstanceState.putString("eventHour", hour);
109         savedInstanceState.putString("eventLocation", location);
110         savedInstanceState.putParcelableArrayList("guestList", list);
111         super.onSaveInstanceState(savedInstanceState);
112     }
113
114     //recupero dati al ritorno in questa activity
115     @Override
116     public void onRestoreInstanceState(Bundle savedInstanceState) {
117         super.onRestoreInstanceState(savedInstanceState);
118         name = savedInstanceState.getString("eventName");
119         date = savedInstanceState.getString("eventDate");
120         location = savedInstanceState.getString("eventLocation");
121         hour = savedInstanceState.getString("eventHour");
122         list = savedInstanceState.getParcelableArrayList("guestList");
123     }
124
125     //recupero dati al ritorno a questa activity se non è già stata
    cancellata
126     @Override
127     public void onRestart() {
128         super.onRestart();
129         getEnteredData();
130     }
131
132     //alla prima apertura dell'activity si chiede all'utente se vuole
    applicare le regole anti-COVID
133     @Override
134     protected void onStart() {
135         super.onStart();
136         if(firstTime) {
137             new MaterialAlertDialogBuilder(NewEvent.this)
```

```
138                     .setMessage(getResources().getString(R.string.
    evento_covid))
139                     .setNegativeButton(getResources().getString(R.
    string.no), new DialogInterface.OnClickListener() {
140                         @Override
141                         public void onClick(DialogInterface dialog,
    int which) {
142                             covid = false;
143                         }
144                     })
145                     .setPositiveButton(getResources().getString(R.
    string.si), new DialogInterface.OnClickListener() {
146                         @Override
147                         public void onClick(DialogInterface dialog,
    int which) {
148                             covid = true;
149                         }
150                     })
151                     .show();
152         }
153     }
154
155     //metodo invocato per settare la data dell'evento
156     public void showDatePickerDialog(View v) {
157         DialogFragment newFragment = new DatePickerFragment();
158         newFragment.show(getSupportFragmentManager(), "datePicker");
159     }
160
161     //metodo invocato per settare l'orario dell'evento
162     public void showTimePickerDialog(View v) {
163         DialogFragment newFragment = new TimePickerFragment();
164         newFragment.show(getSupportFragmentManager(), "timePicker");
165     }
166
167     //metodo invocato al click del tasto invita ospiti
168     public void inviteGuests(View v) {
169         retrieveEventData();
170         //creazione intent esplicito in cui si passa la lista invitati
     da modificare
171         final Intent invite = new Intent(this, ContactEntry.class);
172         invite.putParcelableArrayListExtra("guestList", list);
173         //si controlla se è la prima volta che si va a ContactEntry
    per questo evento
174         if(firstTime && covid) {
175             //si chiede all'utente se l'evento sarà all'aperto oppure
    no
176             new MaterialAlertDialogBuilder(NewEvent.this)
177                     .setMessage(getResources().getString(R.string.
    evento_aperto))
178                     .setNegativeButton(getResources().getString(R.
    string.no), new DialogInterface.OnClickListener() {
```

```
179                         @Override
180                         public void onClick(DialogInterface dialog,
    int which) {
181                             outside = false;
182                             //vengono richieste le dimensioni della
    stanza in un altro dialog
183                             final Dialog dimensionsDialog = new Dialog
    (NewEvent.this);
184                             dimensionsDialog.setContentView(R.layout.
    dialog_dimensions_request);
185                             final TextInputEditText roomWidth = (
    TextInputEditText) dimensionsDialog.findViewById(R.id.roomWidht);
186                             final TextInputEditText roomLenght = (
    TextInputEditText) dimensionsDialog.findViewById(R.id.roomLenght);
187                             final TextInputEditText guestsDistance = (
    TextInputEditText) dimensionsDialog.findViewById(R.id.
    guestsDistance);
188                             Button yesButton = (Button)
    dimensionsDialog.findViewById(R.id.dimensionsYesButton);
189                             Button noButton = (Button)
    dimensionsDialog.findViewById(R.id.dimensionsNoButton);
190                             //listener per memorizzare le dimensioni
    quando inserite
191                             roomWidth.addTextChangedListener(new
    TextWatcher() {
192                                 @Override
193                                 public void beforeTextChanged(
    CharSequence s, int start, int count, int after) {
194
195                                 }
196                                 @Override
197                                 public void onTextChanged(CharSequence
     s, int start, int before, int count) {
198
199                                 }
200                                 @Override
201                                 public void afterTextChanged(Editable
    s) {
202                                     width = s.toString();
203                                 }
204                             });
205                             roomLenght.addTextChangedListener(new
    TextWatcher() {
206                                 @Override
207                                 public void beforeTextChanged(
    CharSequence s, int start, int count, int after) {
208
209                                 }
210                                 @Override
211                                 public void onTextChanged(CharSequence
     s, int start, int before, int count) {
```

```
212
213                             }
214                             @Override
215                             public void afterTextChanged(Editable
    s) {
216                                 length = s.toString();
217                             }
218                         });
219                         guestsDistance.addTextChangedListener(new
    TextWatcher() {
220                             @Override
221                             public void beforeTextChanged(
    CharSequence s, int start, int count, int after) {
222
223                             }
224                             @Override
225                             public void onTextChanged(CharSequence
     s, int start, int before, int count) {
226
227                             }
228                             @Override
229                             public void afterTextChanged(Editable
    s) {
230                                 distance = s.toString();
231                             }
232                         });
233                         //gestione click bottoni
234                         noButton.setOnClickListener(new View.
    OnClickListener() {
235                             @Override
236                             public void onClick(View v) {
237                                 dimensionsDialog.dismiss();
238                             }
239                         });
240                         yesButton.setOnClickListener(new View.
    OnClickListener() {
241                             @Override
242                             public void onClick(View v) {
243                                 //si controlla che i campi siano
    stati inseriti e si
244                                 //passano i valori alla prossima
    activity
245                                 if((!width.equals("")) && (!length
    .equals("")) && (!distance.equals(""))) {
246                                     dimensionsDialog.dismiss();
247                                     bundle.putString("roomWidth",
    width);
248                                     bundle.putString("roomLength",
     length);
249                                     bundle.putString("
    guestsDistance", distance);
```

```
250                                     insertcontacts(invite);
251                                 } else {
252                                     Toast.makeText(NewEvent.this,
    getResources().getString(R.string.check_mancanti), Toast.
    LENGTH_SHORT).show();
253                                 }
254                             }
255                         });
256                         dimensionsDialog.show();
257                     }
258                 })
259                 .setPositiveButton(getResources().getString(R.
    string.si), new DialogInterface.OnClickListener() {
260                     @Override
261                     public void onClick(final DialogInterface
    dialog, int which) {
262                         outside = true;
263                         //se la risposta è si non si procede
    normalmente in quanto non vengono
264                         //richieste la dimensioni della stanza
265                         bundle.putString("roomWidth", width);
266                         bundle.putString("roomLength", length);
267                         bundle.putString("guestsDistance",
    distance);
268                         insertcontacts(invite);
269                     }
270                 })
271                 .show();
272         } else {
273             insertcontacts(invite);
274         }
275     }
276
277     private void insertcontacts(Intent invite) {
278         firstTime = false;
279         putEnteredData();
280         invite.putExtra("roomDimensions", bundle);
281         invite.putExtra("meals", menuBundle);
282         startActivity(invite);
283         finish();
284     }
285
286     //metodo invocato al click del tasto avanti
287     public void nextPhase(View v) {
288         retrieveEventData();
289         //creazione intent esplicito
290         Intent next = new Intent(NewEvent.this, FoodIntolerance.class)
    ;
291         //creazione del Bundle che dovrà contenere i dati da inviare
    all'activity successiva
292         Bundle event = new Bundle();
```

```
293         //verifica che i campi siano tutti pieni e se manca qualcosa
    sarà inviato un apposito toast message
294         //il blocco try catch serve a gestire la NullPointerException
    causata da isEmpty() se la lista è vuota
295         try {
296             if ((!(name).equals("")) && (!(hour).equals("")) && (!(
    location).equals("")) && (!date.equals("")) && (!list.isEmpty()))
    {
297                 firstTime = false;
298                 //inserimento dei dati sull'evento nel bundle
299                 event.putString("eventName", name);
300                 event.putString("eventDate", date);
301                 event.putString("eventHour", hour);
302                 event.putString("eventLocation", location);
303                 next.putExtra("eventData", event);
304                 next.putExtra("roomDimensions", bundle);
305                 //aggiungere parte sugli invitati
306                 next.putParcelableArrayListExtra("guestList", list);
307                 putEnteredData();
308                 next.putExtra("meals", menuBundle);
309                 startActivity(next);
310                 finish();
311             } else if (name.equals("")) {
312                 Toast.makeText(this, R.string.messaggio_nome_mancante,
     Toast.LENGTH_SHORT).show();
313             } else if (date.equals("")) {
314                 Toast.makeText(this, R.string.messaggio_data_mancante,
     Toast.LENGTH_SHORT).show();
315             } else if (location.equals("")) {
316                 Toast.makeText(this, R.string.
    messaggio_posizione_mancante, Toast.LENGTH_SHORT).show();
317             } else if (hour.equals("")) {
318                 Toast.makeText(this, R.string.messaggio_ora_mancante,
    Toast.LENGTH_SHORT).show();
319             } else if (list.isEmpty()) {
320                 Toast.makeText(NewEvent.this, R.string.
    messaggio_lista_vuota, Toast.LENGTH_SHORT).show();
321             }
322         } catch(NullPointerException e) {
323             Toast.makeText(this, R.string.messaggio_lista_vuota, Toast
    .LENGTH_SHORT).show();
324         }
325     }
326
327     //metodo per recuperare i dati dalle viste
328     public void retrieveEventData() {
329         date = Objects.requireNonNull(eventDate.getText()).toString();
330         hour = Objects.requireNonNull(eventHour.getText()).toString();
331         name = Objects.requireNonNull(eventName.getText()).toString();
332         location = Objects.requireNonNull(eventLocation.getText()).
    toString();
```

```
333     }
334
335     //metodo per recuperare i dati immessi da una SharedPreference che
         viene successivamente svuotata
336     public void getEnteredData() {
337         SharedPreferences eventState = getSharedPreferences("eventData
    ",Context.MODE_PRIVATE);
338         name = eventState.getString("eventName", "");
339         location = eventState.getString("eventLocation", "");
340         date = eventState.getString("eventDate", "");
341         hour = eventState.getString("eventHour","");
342         SharedPreferences.Editor editor = eventState.edit();
343         editor.clear();
344         editor.apply();
345     }
346
347     //metodo per inserire i dati nelle SharedPreferences
348     public void putEnteredData() {
349         SharedPreferences eventState = getSharedPreferences("eventData
    ",Context.MODE_PRIVATE);
350         SharedPreferences.Editor editor = eventState.edit();
351         editor.putString("eventName", name);
352         editor.putString("eventLocation", location);
353         editor.putString("eventDate", date);
354         editor.putString("eventHour", hour);
355         editor.apply();
356     }
357
358     //metodo chiamato per tornare all'activity precedente
359     public void back() {
360         retrieveEventData();
361         //creazione intant esplicito
362         final Intent back = new Intent(NewEvent.this, MainActivity.
    class);
363         //qui si verifica se è già stato inserito qualche dato
364         try {
365             if ((!(name).equals("")) || (!(location).equals("")) ||
    (!(hour).equals("")) || (!date.equals("")) || (!list.isEmpty())) {
366                 //definizione di un alert per indicare che se si torna
     indietro i dati già immessi andranno persi
367                 new MaterialAlertDialogBuilder(NewEvent.this)
368                         .setTitle(getResources().getString(R.string.
    titolo_alert_indietro))
369                         .setMessage(getResources().getString(R.string.
    messaggio_alert_indietro))
370                         .setNegativeButton(getResources().getString(R.
    string.no), new DialogInterface.OnClickListener() {
371                             @Override
372                             public void onClick(DialogInterface dialog
    , int which) {
373
```

```
374                             }
375                         })
376                         .setPositiveButton(getResources().getString(R.
    string.si), new DialogInterface.OnClickListener() {
377                             @Override
378                             public void onClick(DialogInterface dialog
    , int which) {
379                                 SharedPreferences eventState =
    getSharedPreferences("eventData",Context.MODE_PRIVATE);
380                                 SharedPreferences.Editor editor =
    eventState.edit();
381                                 editor.clear();
382                                 editor.apply();
383                                 firstTime = true;
384                                 startActivity(back);
385                                 finish();
386                             }
387                         })
388                         .show();
389             } else {
390                 firstTime = true;
391                 startActivity(back);
392                 finish();
393             }
394         } catch(NullPointerException e) {
395             //si torna indietro in sicurezza perchè se scatta l'
    eccezione significa che la lista è vuota
396             //come tutti gli altri campi
397             firstTime = true;
398             startActivity(back);
399             finish();
400         }
401     }
402
403     //disabilita tasto indietro
404     @Override
405     public void onBackPressed() {
406         back();
407     }
408 }
409
```

Listato 4.2: Listato relativo all'Activity `NewEvent`

Nell'Activity `ContactEntry` sarà possibile visualizzare la lista degli invitati inseriti fino a quel momento, ricercare qualche nominativo o,anche, eliminarlo. L'Activity `ExistingContact` permette di aggiungere alla lista degli invitati persone che già possiedono un account, mentre `NewContact` consente di inserire manualmente i dati di un invitato. Dall'Activity `ContactEntry`, e qualora si sia scelto di seguire le normative anti COVID-19, verrà richiesto all'utente se vuole visualizzare un'anteprima della disposizione dei posti a tavola; in caso di risposta affermativa si verrà inviati

all'Activity `CohabitingGroupsActivity`, altrimenti si ritorna subito all'Activity `NewEvent`.
All'interno dell'Activity `CohabitingGroupsActivity` vengono definiti i gruppi di conviventi per i quali non è necessario osservare il distanziamento sociale: per la definizione di tali gruppi verrà richiesto di specificare un nome ed il numero dei componenti. Successivamente, si viene indirizzati all'Activity `GuestsLayoutActivity` la quale ha il compito di gestire la disposizione degli invitati tenendo conto del fatto che membri di gruppi diversi devono osservare il distanziamento sociale e, di conseguenza, verificare che le dimensioni del tavolo fornite alla creazione dell'Activity siano sufficienti per ospitare tutti gli invitati.

```
1  //activity per gestire la disposizione degli invitati
2  public class GuestsLayoutActivity extends AppCompatActivity {
3
4      private DrawerLayout drawerLayout;
5      private DrawerLayout.DrawerListener listener;
6      private MaterialToolbar toolbar;
7      private NavigationView navigationView;
8      private GridLayout tableContainer;
9      private ArrayList<Guest> list = new ArrayList<>();
10     private ArrayList<Group> groupsList = new ArrayList<>();
11     private Bundle bundle = new Bundle();
12     private Bundle menuBundle = new Bundle();
13     private Bundle groups = new Bundle();
14     private float roomWidth;
15     private float roomLength;
16     private int tableCellsLength;
17     private int tableCellsWidth;
18     private static final int CELL_MARGIN = 2;
19     private LinearLayout groupsContainer;
20     private int requestedSeats = 0;
21     private int currentSeats = 0;
22
23     @Override
24     protected void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         setContentView(R.layout.activity_guests_layout);
27         //inizializzazione viste
28         drawerLayout = (DrawerLayout) findViewById(R.id.
    guestsLayoutDrawer);
29         navigationView = (NavigationView) findViewById(R.id.
    tablesNavigationView);
30         toolbar = (MaterialToolbar) findViewById(R.id.guestsLayoutTB);
31         tableContainer = (GridLayout) findViewById(R.id.tableContainer
    );
32         setSupportActionBar(toolbar);
33         //recupero dati da altra activity
34         Intent intent = getIntent();
35         if(intent.getParcelableArrayListExtra("guestList") != null) {
36             list = intent.getParcelableArrayListExtra("guestList");
```

```
37          }
38          bundle = intent.getBundleExtra("roomDimensions");
39          menuBundle = intent.getBundleExtra("meals");
40          groups = intent.getBundleExtra("groups");
41          assert groups != null;
42          //inizializzazione lista dei gruppi
43          for(String name : groups.keySet()) {
44              Group group = new Group(name, groups.getInt(name));
45              groupsList.add(group);
46          }
47          if(!NewEvent.outside) {
48              roomWidth = Float.parseFloat(Objects.requireNonNull(bundle
    .getString("roomWidth")));
49              roomLength = Float.parseFloat(Objects.requireNonNull(
    bundle.getString("roomLength")));
50          }
51          setupHeader();
52          addTable();
53          //gestione dell'apertura della NavigationView
54          listener = new DrawerLayout.DrawerListener() {
55              @Override
56              public void onDrawerSlide(@NonNull View drawerView, float
    slideOffset) {
57
58              }
59              @Override
60              public void onDrawerOpened(@NonNull View drawerView) {
61                  drawerLayout.openDrawer(GravityCompat.START);
62              }
63              @Override
64              public void onDrawerClosed(@NonNull View drawerView) {
65                  drawerLayout.closeDrawers();
66              }
67              @Override
68              public void onDrawerStateChanged(int newState) {
69
70              }
71          };
72          drawerLayout.addDrawerListener(listener);
73          toolbar.setNavigationOnClickListener(new View.OnClickListener
    () {
74              @Override
75              public void onClick(View v) {
76                  drawerLayout.openDrawer(GravityCompat.START);
77              }
78          });
79      }
80
81      //metodo per inizializzare il menu laterale sulla base dei gruppi
    presenti
82      private void setupHeader() {
```

```
83          View header = navigationView.getHeaderView(0);
84          groupsContainer = header.findViewById(R.id.groupsContainer);
85          int viewIndex = 0;
86          for(Group group : groupsList) {
87              //si incrementa di 1 per ogni gruppo, piu' il numero di
    membri
88              requestedSeats ++;
89              requestedSeats = requestedSeats + group.getGroupNumber();
90              //creazione delle viste che saranno riportate per ogni
    gruppo
91              TextView textView = new TextView(GuestsLayoutActivity.this
    );
92              View divider = new View(GuestsLayoutActivity.this);
93              RecyclerView recyclerView = new RecyclerView(
    GuestsLayoutActivity.this);
94              RecyclerView.LayoutManager layoutManager = new
    GridLayoutManager(GuestsLayoutActivity.this, 2);
95              GroupMemberAdapter adapter = new GroupMemberAdapter(group,
     drawerLayout);
96              //setup delle viste e aggiunta al viewgroup contenitore
97              textView.setText(group.getGroupName());
98              textView.setTextColor(getColor(R.color.mainTextColor));
99              textView.setTextAppearance(R.style.
    TextAppearance_AppCompat_Large);
100             textView.setGravity(Gravity.CENTER);
101             divider.setBackgroundResource(R.color.subTextColor);
102             recyclerView.setTag(group.getGroupName());
103             recyclerView.setHasFixedSize(true);
104             recyclerView.setLayoutManager(layoutManager);
105             recyclerView.setAdapter(adapter);
106             LinearLayout.LayoutParams layoutParams = new LinearLayout.
    LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.
    LayoutParams.WRAP_CONTENT);
107             layoutParams.bottomMargin = 5;
108             layoutParams.leftMargin = 5;
109             layoutParams.rightMargin = 5;
110             layoutParams.topMargin = 75;
111             groupsContainer.addView(textView, viewIndex, layoutParams)
    ;
112             viewIndex ++;
113             groupsContainer.addView(divider, viewIndex, new
    LinearLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, 1))
    ;
114             viewIndex ++;
115             LinearLayout.LayoutParams layoutParams1 = new LinearLayout
    .LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.
    LayoutParams.WRAP_CONTENT);
116             layoutParams1.topMargin = 5;
117             layoutParams1.rightMargin = 5;
118             layoutParams1.leftMargin = 5;
119             layoutParams1.bottomMargin = 5;
```

```
120             layoutParams1.gravity = Gravity.CENTER;
121             groupsContainer.addView(recyclerView, viewIndex,
    layoutParams1);
122             viewIndex ++;
123         }
124     }
125
126     //metodo in cui si richiedono le dimensioni del tavolo per poi
    rappresentarlo come insieme di celle in un GridLayout
127     private void addTable() {
128         final Dialog dialog = new Dialog(GuestsLayoutActivity.this);
129         //creazione tavolo, si apre un dialog che richiede lunghezza e
     larghezza del tavolo da disegnare
130         dialog.setContentView(R.layout.
    dialog_ask_rectangular_table_dimensions);
131         final TextInputEditText tableLength = (TextInputEditText)
    dialog.findViewById(R.id.tableLenght);
132         final TextInputEditText tableWidth = (TextInputEditText)
    dialog.findViewById(R.id.tableWidth);
133         Button cancelButton = (Button) dialog.findViewById(R.id.
    cancelRectangularTableDimensionsButton);
134         Button nextButton = (Button) dialog.findViewById(R.id.
    nextRectangularTableDimensionsTableButton);
135         final float[] width = {0};
136         final float[] length = {0};
137         cancelButton.setOnClickListener(new View.OnClickListener() {
138             @Override
139             public void onClick(View v) {
140                 dialog.dismiss();
141                 onBackPressed();
142             }
143         });
144         nextButton.setOnClickListener(new View.OnClickListener() {
145             @Override
146             public void onClick(View v) {
147                 if(Objects.requireNonNull(tableWidth.getText()).
    toString().equals("") || Objects.requireNonNull(tableLength.
    getText()).toString().equals("")) {
148                     Toast.makeText(GuestsLayoutActivity.this,
    getResources().getString(R.string.check_mancanti), Toast.
    LENGTH_SHORT).show();
149                 } else {
150                     width[0] = Float.parseFloat(tableWidth.getText().
    toString());
151                     length[0] = Float.parseFloat(tableLength.getText()
    .toString());
152                     if(!NewEvent.outside && (width[0] > (roomWidth *
    100) || length[0] > (roomLength * 100))) {
153                         Toast.makeText(GuestsLayoutActivity.this,
    getResources().getString(R.string.dimensioni_errate), Toast.
    LENGTH_SHORT).show();
```

```
154                     } else {
155                         //creazione tavolo rettangolare
156                         tableContainer.removeAllViews();
157                         //approssimazione dimensioni tavolo
158                         if(length[0] % 60 >= 40) {
159                             length[0] = length[0] + (60 - (length[0] %
     60));
160                         }
161                         if(width[0] % 80 >= 55) {
162                             width[0] = width[0] + (80 - (width[0] %
    80));
163                         }
164                         tableCellsLength = (int) (length[0] / 60);
165                         tableCellsWidth = (int) (width[0] / 80);
166                         //si aggiungono due celle all'inizio e alla
    fine di ogni riga e colonna, su
167                         //questa "cornice" andranno disposti gli
    ospiti
168                         tableContainer.setColumnCount(tableCellsWidth
    + 2);
169                         tableContainer.setRowCount(tableCellsLength +
    2);
170                         //le celle che non fanno parte del tavolo
    saranno riempite con viste bianche
171                         for(int j = 0; j < tableContainer.getRowCount
    (); j++) {
172                             for(int i = 0; i < tableContainer.
    getColumnCount(); i++) {
173                                 if(i == 0 || i == tableContainer.
    getColumnCount() - 1 || j == 0 || j == tableContainer.getRowCount
    () - 1) {
174                                     //se la cella si trova sulla riga
    o colonna iniziale o finale si
175                                     //colora di bianco, altrimenti
    marrone
176                                     CardView cardView = new CardView(
    GuestsLayoutActivity.this);
177                                     cardView.setCardElevation(20);
178                                     cardView.setRadius(20);
179                                     cardView.setCardBackgroundColor(
    getColor(R.color.white));
180                                     String tag = j + "£" + i;
181                                     cardView.setTag(tag);
182                                     GridLayout.LayoutParams
    layoutParams = new GridLayout.LayoutParams();
183                                     layoutParams.width = (
    tableContainer.getWidth() / tableContainer.getColumnCount()) - (2
    * CELL_MARGIN);
184                                     layoutParams.height = (
    tableContainer.getHeight() / tableContainer.getRowCount()) - (2 *
    CELL_MARGIN);
```

```
185                                     layoutParams.bottomMargin =
    CELL_MARGIN;
186                                     layoutParams.leftMargin =
    CELL_MARGIN;
187                                     layoutParams.rightMargin =
    CELL_MARGIN;
188                                     layoutParams.topMargin =
    CELL_MARGIN;
189                                     layoutParams.setGravity(Gravity.
    CENTER);
190                                     //x è un indice che vale solo per
    le celle visibili e su cui si potranno posizionare ospiti
191                                     tableContainer.addView(cardView,
    layoutParams);
192                                     //se la cella si trova ad uno
    degli angoli non è visibile
193                                     //altrimenti potrà essere il
    target dell'operazione di drag&drop
194                                     if(((i == 0 && j == 0) || (i == 0
    && j == tableContainer.getRowCount() - 1) || (i == tableContainer.
    getColumnCount() - 1 && j == 0) || (i == tableContainer.
    getColumnCount() - 1 && j == tableContainer.getRowCount() - 1))) {
195                                         cardView.setVisibility(View.
    INVISIBLE);
196                                         cardView.setEnabled(false);
197                                     } else {
198                                         //ogni cella visibile è un
    posto disponibile
199                                         currentSeats ++;
200                                         //creazione di un DragListener
     da assegnare alle celle, in modo tale che siano capaci
201                                         //di rispondere ad un
    trascinamento di un ospite su di loro
202                                         GroupMemberDragEventListener
    groupMemberDragEventListener = new GroupMemberDragEventListener(
    GuestsLayoutActivity.this, tableContainer, groupsContainer,
    getResources().getString(R.string.messaggio_non_conviventi_vicini)
    );
203                                         cardView.setOnDragListener(
    groupMemberDragEventListener);
204                                         //gestione del click delle
    celle per mostrare, nel caso di quelle occupate,
205                                         //il nome del gruppo a cui
    appartiene l'ospite che è stato posizionato
206                                         cardView.setOnClickListener(
    new View.OnClickListener() {
207                                             @Override
208                                             public void onClick(View v
    ) {
209                                                 String[] strings = v.
    getTag().toString().split("£");
```

```
210                                                 if(strings.length ==
    3) {
211                                                     Toast.makeText(
    GuestsLayoutActivity.this, getResources().getString(R.string.
    appartenenza_gruppo) + " " + strings[2], Toast.LENGTH_SHORT).show
    ();
212                                                 }
213                                             }
214                                         });
215                                         //il longclick invece servirà
    a rimuovere l'invitato dal tavolo
216                                         cardView.
    setOnLongClickListener(new View.OnLongClickListener() {
217                                             @Override
218                                             public boolean onLongClick
    (final View v) {
219                                                 if((v.getTag().
    toString().split("£").length == 3) && (((CardView) v).
    getCardBackgroundColor() == ColorStateList.valueOf(Color.RED))) {
220                                                     new
    MaterialAlertDialogBuilder(GuestsLayoutActivity.this)
221                                                             .setTitle(
    getResources().getString(R.string.messaggio_rimozione_posto))
222                                                             .
    setNegativeButton(getResources().getString(R.string.no), new
    DialogInterface.OnClickListener() {
223
    @Override
224                                                             public
     void onClick(DialogInterface dialog, int which) {
225
226                                                             }
227                                                         })
228                                                         .
    setPositiveButton(getResources().getString(R.string.si), new
    DialogInterface.OnClickListener() {
229
    @Override
230                                                             public
     void onClick(DialogInterface dialog, int which) {
231
    CardView cardView = (CardView) v;
232
    String[] strings = cardView.getTag().toString().split("£");
233
    RecyclerView recyclerView = (RecyclerView) groupsContainer.
    findViewWithTag(strings[2]);
234                                                             if
    (recyclerView != null) {
235
      GroupMemberAdapter adapter = (GroupMemberAdapter) recyclerView.
```

```
    getAdapter();
236
      assert adapter != null;
237
      adapter.addOneElement();
238
      //annullare tutte le modifiche fatte in precedenza
239
      cardView.setCardBackgroundColor(Color.WHITE);
240
      cardView.setTag(strings[0] + "£" + strings[1]);
241
      cardView.invalidate();
242                                                                    }
243                                                                }
244                                                            })
245                                                            .show();
246                                                    }
247                                                    return false;
248                                                }
249                                            });
250                                    }
251                                } else if(i == 1 && j == 1) {
252                                    //il tavolo prende tutte le righe
    e le colonne tranne prima ed ultima
253                                    //ed è di colore marrone
254                                    CardView cardView = new CardView(
    GuestsLayoutActivity.this);
255                                    cardView.setCardElevation(8);
256                                    cardView.setRadius(20);
257                                    cardView.setCardBackgroundColor(
    getColor(R.color.brown));
258                                    cardView.setTag("table");
259                                    GridLayout.LayoutParams
    layoutParams = new GridLayout.LayoutParams();
260                                    layoutParams.columnSpec =
    GridLayout.spec(1, tableCellsWidth);
261                                    layoutParams.rowSpec = GridLayout.
    spec(1, tableCellsLength);
262                                    layoutParams.width = ((
    tableContainer.getWidth() / tableContainer.getColumnCount()) *
    tableCellsWidth);
263                                    layoutParams.height = ((
    tableContainer.getHeight() / tableContainer.getRowCount()) *
    tableCellsLength);
264                                    layoutParams.setGravity(Gravity.
    CENTER);
265                                    tableContainer.addView(cardView,
    layoutParams);
266                                }
267                            }
```

```
268                         }
269                         dialog.dismiss();
270                         //controllo per assicurarsi che gli ospiti
    entrino
271                         if(currentSeats < (requestedSeats - 4)) {
272                             String text = getResources().getString(R.
    string.messaggio_num_ospiti_1) + " " + currentSeats + " " +
    getResources().getString(R.string.messaggio_num_ospiti_2) + " " +
    requestedSeats + " " + getResources().getString(R.string.
    messaggio_num_ospiti_3);
273                             new MaterialAlertDialogBuilder(
    GuestsLayoutActivity.this)
274                                     .setTitle(getResources().getString
    (R.string.attenzione))
275                                     .setMessage(text)
276                                     .setNeutralButton(getResources().
    getString(R.string.indietro), new DialogInterface.OnClickListener
    () {
277                                         @Override
278                                         public void onClick(
    DialogInterface dialog, int which) {
279                                             onBackPressed();
280                                         }
281                                     })
282                                     .show();
283                         }
284                     }
285                 }
286             }
287         });
288         dialog.show();
289     }
290
291     //metodo per proseguire, tornando a NewEvent, attivato dal click
    del bottone in fondo
292     public void next(View v) {
293         Intent intent = new Intent(GuestsLayoutActivity.this, NewEvent
    .class);
294         intent.putParcelableArrayListExtra("guestList", list);
295         intent.putExtra("roomDimensions", bundle);
296         intent.putExtra("meals", menuBundle);
297         startActivity(intent);
298         finish();
299     }
300
301     //metodo per tornare indietro
302     public void cancel(View v) {
303         onBackPressed();
304     }
305
306     @Override
```

```
307     public void onBackPressed() {
308         Intent intent = new Intent(GuestsLayoutActivity.this,
        ContactEntry.class);
309         intent.putParcelableArrayListExtra("guestList", list);
310         intent.putExtra("roomDimensions", bundle);
311         intent.putExtra("meals", menuBundle);
312         startActivity(intent);
313         finish();
314     }
315 }
316
```

Listato 4.3: Listato relativo alla `GuestsLayoutActivity`

```
1  //classe che implementa l'interfaccia OnDragListener per gestire le
       varie fasi del trascinamento della vista
2  public class GroupMemberDragEventListener implements View.
       OnDragListener {
3
4      //contiene le recyclerview da cui si trascinano gli ospiti
5      private LinearLayout groupsContainer;
6      //contesto dell'activity
7      private Context context;
8      //messaggio da mostrare in un toast se stiamo posizionando un
       ospite vicino ad uno di un altro gruppo
9      private String dinstanceBrokenMessage;
10     private int columnCount;
11     private int rowCount;
12     //griglia contenente i posti a sedere
13     private GridLayout tableContainer;
14     //nome del gruppo da cui stiamo prendendo un membro
15     private static String groupName;
16
17     //costruttore
18     public GroupMemberDragEventListener(Context context, GridLayout
       tableContainer, LinearLayout groupsContainer, String
       dinstanceBrokenMessage) {
19         this.context = context;
20         this.tableContainer = tableContainer;
21         this.columnCount = tableContainer.getColumnCount() - 1;
22         this.rowCount = tableContainer.getRowCount() - 1;
23         this.groupsContainer = groupsContainer;
24         this.dinstanceBrokenMessage = dinstanceBrokenMessage;
25     }
26
27     @Override
28     public boolean onDrag(View v, DragEvent event) {
29         CardView cardView = (CardView) v;
30         //recupero coordinate della cella selezionata, in modo da
       poter conoscere quelle vicine
31         String[] array = cardView.getTag().toString().split("£");
32         int row = Integer.parseInt(array[0]);
```

```
33          int column = Integer.parseInt(array[1]);
34          String coordinates = row + "£" + column;
35          switch (event.getAction()) {
36              case DragEvent.ACTION_DRAG_STARTED:
37                  return event.getClipDescription().hasMimeType(
    ClipDescription.MIMETYPE_TEXT_PLAIN);
38              case DragEvent.ACTION_DRAG_ENTERED:
39                  if(cardView.getCardBackgroundColor() != ColorStateList
    .valueOf(Color.RED)) {
40                      cardView.setCardBackgroundColor(Color.GREEN);
41                      cardView.invalidate();
42                  }
43                  return true;
44              case DragEvent.ACTION_DRAG_LOCATION:
45                  return true;
46              case DragEvent.ACTION_DRAG_EXITED:
47                  if(cardView.getCardBackgroundColor() == ColorStateList
    .valueOf(Color.GREEN)) {
48                      cardView.setCardBackgroundColor(Color.WHITE);
49                      cardView.invalidate();
50                  }
51                  return true;
52              case DragEvent.ACTION_DROP:
53                  ClipData.Item item = event.getClipData().getItemAt(0);
54                  String dragData = item.getText().toString();
55                  groupName = (dragData.split("£"))[0];
56                  //se il tag è diverso dalle sole coordinate della
    cella allora questa è già occupata
57                  //nel caso in cui si stia posizionando l'ombra su una
    cella rossa che è già occupata
58                  if((!cardView.getTag().toString().equals(coordinates))
     && (cardView.getCardBackgroundColor() == ColorStateList.valueOf(
    Color.RED))) {
59                      break;
60                  } else {
61                      //controllo delle celle vicine
62                      //se la riga dove si trova la cella non è quella
    iniziale nè quella finale
63                      //dovremo controllare le righe posizionate sopra e
     sotto, altrimenti quelle laterali
64                      int upperRow = row - 1;
65                      int lowerRow = row + 1;
66                      int leftColumn = column - 1;
67                      int rightColumn = column + 1;
68                      String upperCoordinates = upperRow + "£" + column;
69                      String lowerCoordinates = lowerRow + "£" + column;
70                      String leftCoordinates = row + "£" + leftColumn;
71                      String rightCoordinates = row + "£" + rightColumn;
72                      if(row > 1 && row < rowCount - 1) {
73                          //si controlla sia sopra che sotto
74                          if(((tableContainer.findViewWithTag(
```

```
    upperCoordinates) != null) || (tableContainer.findViewWithTag(
    upperCoordinates + "£" + groupName) != null)) && ((tableContainer.
    findViewWithTag(lowerCoordinates) != null) || (tableContainer.
    findViewWithTag(lowerCoordinates + "£" + groupName) != null))) {
75                          completeDrop(cardView);
76                          return true;
77                      } else {
78                          Toast.makeText(context,
    dinstanceBrokenMessage, Toast.LENGTH_SHORT).show();
79                          break;
80                      }
81                  } else if(row == 1) {
82                      //caso in cui la lunghezza del tavolo sia
    costituita da solo una cella
83                      if(rowCount == 2) {
84                          completeDrop(cardView);
85                          return true;
86                      }
87                      //se la cella è sulla seconda riga avrà solo
    una cella sottostante
88                      else if((tableContainer.findViewWithTag(
    lowerCoordinates) != null) || (tableContainer.findViewWithTag(
    lowerCoordinates + "£" + groupName) != null)) {
89                          completeDrop(cardView);
90                          return true;
91                      } else {
92                          Toast.makeText(context,
    dinstanceBrokenMessage, Toast.LENGTH_SHORT).show();
93                          break;
94                      }
95                  } else if(row == rowCount - 1) {
96                      //se la cella si trova sulla penultima riga si
     dovrà controllare solo la superiore
97                      if((tableContainer.findViewWithTag(
    upperCoordinates) != null) || (tableContainer.findViewWithTag(
    upperCoordinates + "£" + groupName) != null)) {
98                          completeDrop(cardView);
99                          return true;
100                     } else {
101                         Toast.makeText(context,
    dinstanceBrokenMessage, Toast.LENGTH_SHORT).show();
102                         break;
103                     }
104                 } else if(column > 1 && column < columnCount - 1)
    {
105                     //controllo della cella di sinistra e di
    quella di destra con la stessa logica di prima
106                     if(((tableContainer.findViewWithTag(
    leftCoordinates) != null) || (tableContainer.findViewWithTag(
    leftCoordinates + "£" + groupName) != null)) && ((tableContainer.
    findViewWithTag(rightCoordinates) != null) || (tableContainer.
```

```
    findViewWithTag(rightCoordinates + "£" + groupName) != null))) {
107                         completeDrop(cardView);
108                         return true;
109                     } else {
110                         Toast.makeText(context,
    dinstanceBrokenMessage, Toast.LENGTH_SHORT).show();
111                         break;
112                     }
113                 } else if(column == 1) {
114                     //caso speciale in cui la larghezza è di una
    sola cella
115                     if(columnCount == 2) {
116                         completeDrop(cardView);
117                         return true;
118                     } else
119                         //solo a destra
120                         if((tableContainer.findViewWithTag(
    rightCoordinates) != null) || (tableContainer.findViewWithTag(
    rightCoordinates + "£" + groupName) != null)) {
121                             completeDrop(cardView);
122                             return true;
123                         } else {
124                             Toast.makeText(context,
    dinstanceBrokenMessage, Toast.LENGTH_SHORT).show();
125                             break;
126                         }
127                 } else if(column == columnCount - 1){
128                     //solo a sinistra
129                     if((tableContainer.findViewWithTag(
    leftCoordinates) != null) || (tableContainer.findViewWithTag(
    leftCoordinates + "£" + groupName) != null)) {
130                         completeDrop(cardView);
131                         return true;
132                     } else {
133                         Toast.makeText(context,
    dinstanceBrokenMessage, Toast.LENGTH_SHORT).show();
134                         break;
135                     }
136                 } else {
137                     break;
138                 }
139             }
140         case DragEvent.ACTION_DRAG_ENDED:
141             //tutte le celle devono tornare bianche, ad eccezione
    di quelle rosse
142             if(cardView.getCardBackgroundColor() != ColorStateList
    .valueOf(Color.RED)) {
143                 cardView.setCardBackgroundColor(Color.WHITE);
144                 cardView.invalidate();
145             }
146             return true;
```

```
147             default:
148                 break;
149         }
150         return false;
151     }
152
153     //la cardview in questione viene resa rossa e il suo tag viene
    modificato ad indicare che su quella cella è stato
154     //assegnato un invitato
155     public void completeDrop(CardView cardView) {
156         //recupero della recyclerview del gruppo da cui abbiamo
    rimosso un elemento
157         //tale recyclerview ha come tag il nome del gruppo
158         RecyclerView recyclerView = (RecyclerView) groupsContainer.
    findViewWithTag(groupName);
159         if(recyclerView != null) {
160             GroupMemberAdapter adapter = (GroupMemberAdapter)
    recyclerView.getAdapter();
161             assert adapter != null;
162             adapter.removeOneElement();
163             //dragData conteneva il nome del gruppo e la posizione
    dell'elemento trascinato
164             //il primo verrà aggiunto al contenuto del tag della vista
     ad indicare che c'è gia un membro assegnato
165             String newTag = cardView.getTag() + "£" + groupName;
166             cardView.setTag(newTag);
167             cardView.setCardBackgroundColor(Color.RED);
168         } else {
169             cardView.setCardBackgroundColor(Color.WHITE);
170         }
171         cardView.invalidate();
172     }
173 }
```

Listato 4.4: Listato relativo alla classe `GroupMemberDragEventListener`

Al termine della fase di realizzazione della lista invitati si viene indirizzati all'Activity `FoodIntolerance`, nella quale sono visualizzate tutte le intolleranze degli invitati inseriti nella lista. Infine, si arriva all'Activity `MenuCreation` nella quale è possibile comporre il menù: le varie portate di cui è composto il menù potranno essere totalmente nuove o riprese da menù precedenti.

4.2.5 Services

Questo package contiene come unica classe `EventFirebaseMessagingService`, che estende `FirebaseMessagingService`; il codice al suo interno viene eseguito ogni volta che arriva al dispositivo un messaggio proveniente dal `Firebase Cloud Messaging`: tale messaggio viene, poi, utilizzato per generare una notifica `push`.

```
1 exports.eventPushNotification = functions.https.onCall(async (data,
    context) => {
```

```
    //corpo della notifica e lista dei destinatari
    const text = String(data.text);
    const toSendTo = data.guestTokens;
    //creazione del messaggio
    const message = {
      notification: {
        title: "H.U.P.",
        body: text,
      },
      tokens: toSendTo,
      apns: {
        headers: {
          'apns-priority': '10',
        },
        payload: {
          aps: {
            sound: 'default',
          }
        }
      }
    };
    console.log(message);
    //gestione del caso in cui non ci sono invitati a cui mandare
    notifiche
    if(toSendTo.length === 0) {
      throw new functions.https.HttpsError("invalid-argument", "there
    are no contacts to send notifications");
    }
    //invio messaggio
    admin.messaging().sendMulticast(message);
})

```

Listato 4.5: Listato della function realizzata in linguaggio JavaScript

4.2.6 BroadcastReceiver

Questo package contiene come unica classe `DateReceiver`. La sua funzione è quella di generare una notifica `push` il giorno stesso dell'evento per rammentarne la ricorrenza.

```
//BroadcastReceiver chiamato quando la data attuale corrisponde a
    quella fornita nell'AlarmManager
public class DateReceiver extends BroadcastReceiver {

    public DateReceiver() {

    }

    @Override
    public void onReceive(Context context, Intent intent) {
```

```
10          String name = intent.getStringExtra("eventName");
11          String text = context.getResources().getString(R.string.
    messaggio_notifica_data_evento_1) + " " + name + " " + context.
    getResources().getString(R.string.messaggio_notifica_data_evento_2
    );
12          Uri defaultSoundUri = RingtoneManager.getDefaultUri(
    RingtoneManager.TYPE_NOTIFICATION);
13          NotificationCompat.Builder builder = new NotificationCompat.
    Builder(context, "reminder")
14                  .setSmallIcon(R.drawable.splash_logo)
15                  .setContentTitle("H.U.P.")
16                  .setContentText(text)
17                  .setSound(defaultSoundUri)
18                  .setPriority(NotificationCompat.PRIORITY_DEFAULT);
19          NotificationManagerCompat notificationManagerCompat =
    NotificationManagerCompat.from(context);
20          notificationManagerCompat.notify(4321, builder.build());
21      }
22  }
23
```

Listato 4.6: Listato relativo alla classe `DateReceiver`

4.3 Manuale utente

In questa sezione verranno mostrati gli screenshot relativi a tutte le Activity che si susseguono all'interno dell'applicazione.

4.3.1 SplashPage

L'Activity `SplashPage` presenta il logo dell'applicazione ed il suo nome, sia in acronimo sia in formato esteso (Figura 4.2a).
In essa, come già detto, viene effettuato un controllo sulla presenza di una connessione Internet, necessaria per poter utilizzare i servizi di Firebase; in caso negativo, non potendo l'applicazione funzionare senza connessione, viene mostrato un Dialog di errore (Figura 4.2b). Verificata la presenza di connessione, viene effettuato un ulteriore controllo sull'utente; qualora lo stesso sia già autenticato si passa direttamente alla `MainActivity`; altrimenti si verrà indirizzati all'Activity `LoginAndSignin`, per la fase di autenticazione.

In quest'ultima Activity si alternano diversi Fragment, che analizzeremo nel seguito in dettaglio per conoscerne struttura e funzioni.
Per tutte le Activity della registrazione viene mostrata un'interfaccia più pulita senza AppBar superiore, ma contenente logo e nome dell'applicazione.

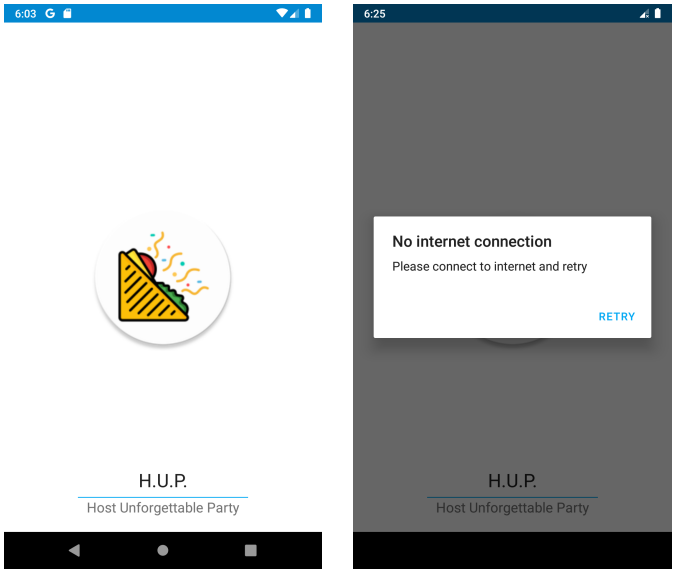

Figura 4.2: Screenshot della `SplashPageActivity`: a) logo dell'applicazione; b) dialog per connessione mancante

4.3.2 LoginAndSigninActivity - FirstChoiceFragment

Il Fragment presenta 2 pulsanti relativi alla possibilità di accedere come nuovo utente o come utente esistente, qualora si sia già effettuata la registrazione in precedenza (Figura 4.3).

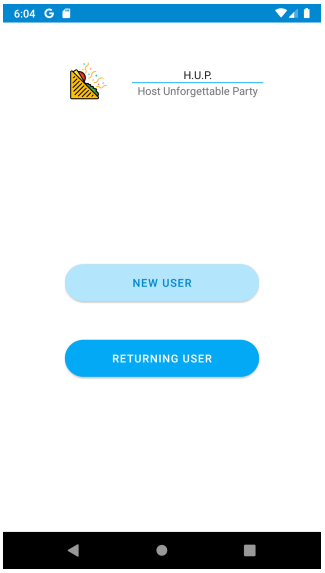

Figura 4.3: Screenshot del Fragment `FirstChoice`

4.3.3 LoginAndSigninActivity – AccountChoiceFragment

Il Fragment presenta 3 pulsanti relativi alle tre scelte possibili di Login o di Signin, ovvero tramite Google, Facebook o E-mail (Figura 4.4).

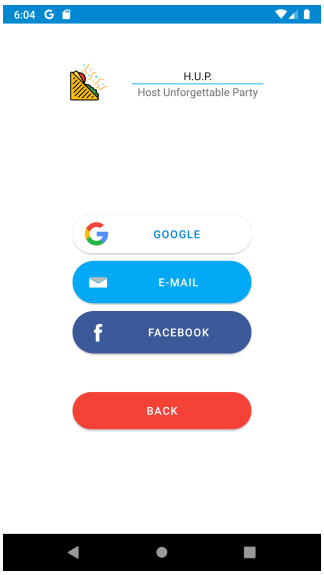

Figura 4.4: Screenshot del Fragment `AccountChoice`

4.3.4 LoginAndSigninActivity – PhotoChoiceFragment

Il Fragment `PhotoChoice` fa parte del "percorso" di registrazione dal quale si iniziano ad inserire i primi dati per la creazione del profilo. In particolare, in questo Fragment, possiamo inserire:

- il numero di telefono;
- l'impostazione della privacy;
- l'immagine del profilo (Figura 4.5).

L'immagine del profilo verrà salvata nello Storage di Firebase a fine registrazione. Con il pulsante *Back* si torna alla prima scelta `FirstChoiceFragment`, con la cancellazione di tutti i dati precedentemente inseriti; per questo motivo, prima di procedere, si apre un AlertDialog che chiede conferma dell'operazione prima di eseguirla.

4.3.5 LoginAndSigninActivity – AddIntolerancesFragment

Anche questo Fragment fa parte del percorso di registrazione; in questo caso l'utente andrà ad inserire le proprie intolleranze alimentari che, poi, saranno visibili agli organizzatori degli eventi che ne terranno conto nella creazione del menù.
Queste intolleranze vengono inserite tramite un'`AutoCompleteEditText` che racchiude nella sua lista tutte le intolleranze preinserite nel Database Firestore nella

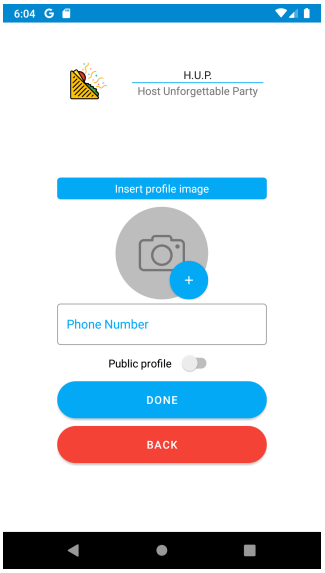

Figura 4.5: Screenshot del Fragment `PhotoChoice`

collection "`Utilities`". Quando ne viene selezionata una dalla lista, questa compare nella RecyclerView sottostante (Figura 4.6).
Questo Fragment può essere riutilizzato anche successivamente per modificare la lista delle intolleranze precedentemente definita.
Anche le intolleranze vengono salvate nel Database Firestore solo alla fine della registrazione.
Con il pulsante *Back* si torna alla prima scelta `FirtstChoiceFragment` con la cancellazione di tutti i dati precedentemente inseriti; per questo motivo, prima di procedere, si apre un AlertDialog che chiede conferma dell'operazione prima di eseguirla.

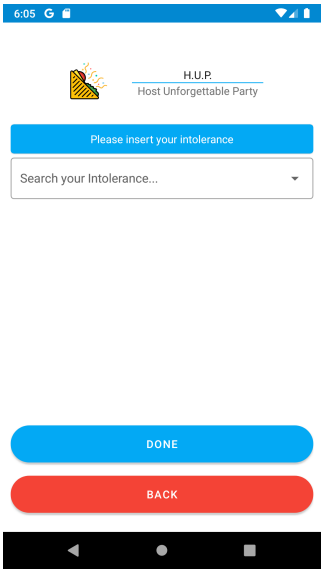

Figura 4.6: Screenshot del Fragment `AddIntolerances`

4.3.6 LoginAndSigninActivity – EmailFragment

Se all'inizio della registrazione è stato cliccato il pulsante relativo all'e-mail, alla fine dell'inserimento dei dati si arriverà a visualizzare questo Fragment dove verrà chiesto di inserire:

- Nome e cognome;
- E-mail;
- Password.

Come si può vedere dalle immagini, il Fragment si può mostrare con due configurazioni diverse, rispettivamente per la registrazione (Figura 4.7a) e per il login.(Figura 4.7b) Con il pulsante *Back* si torna alla prima scelta `FirtstChoiceFragment` con la cancellazione di tutti i dati precedentemente inseriti; per questo motivo, prima di procedere, si apre un AlertDialog che chiede conferma dell'operazione prima di eseguirla.

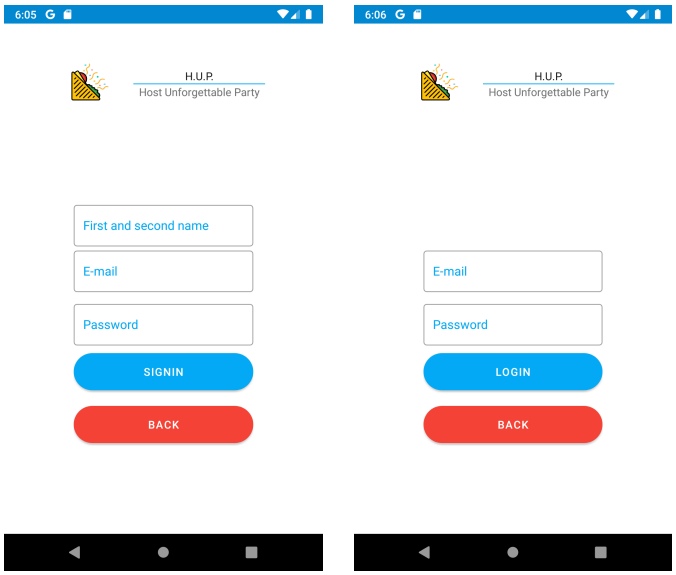

Figura 4.7: Screenshot relativo al Fragment `EmailFragment`: a) Fase di Signin; b) Fase di Login

4.3.7 LoginAndSigninActivity – AccountChoiceFragment (Google Authentication)

Se all'inizio della registrazione è stato cliccato il pulsante *Google*, alla fine dell'inserimento dei dati si arriverà a visualizzare questo Fragment che aprirà il Dialog predefinito di Google dove si potrà scegliere il profilo con cui registrarsi nell'applicazione (Figura 4.8).

Con il pulsante *Back* si torna alla prima scelta `FirtstChoiceFragment` con la cancellazione di tutti i dati precedentemente inseriti; per questo motivo, prima di procedere, si apre un AlertDialog che chiede conferma dell'operazione prima di eseguirla.

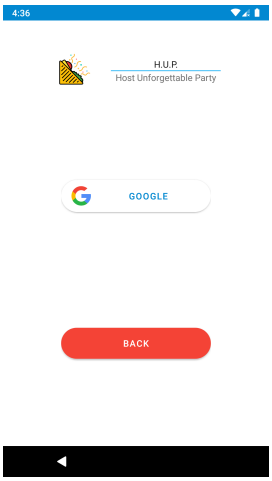

Figura 4.8: Screenshot dell'autenticazione con Google

4.3.8 FacebookAccess

Per l'accesso a Facebook non è stato possibile utilizzare un Fragment, come per l'e-mail e Google, in quanto il sito "Facebook Developers" richiedeva l'utilizzo di un'Activity (Figura 4.9).
Con il pulsante *Back* si torna alla prima scelta `FirtstChoiceFragment` con la cancellazione di tutti i dati precedentemente inseriti: per questo motivo, prima di procedere, si apre un AlertDialog che chiede conferma dell'operazione prima di eseguirla.

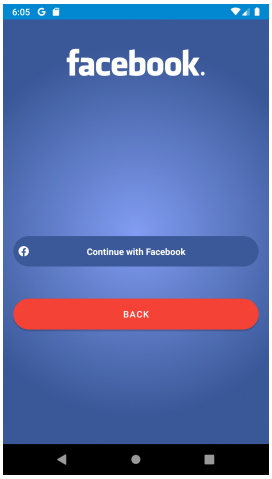

Figura 4.9: Screenshot dell'autenticazione con Facebook

4.3.9 MainActivity

Una volta terminata la fase di autenticazione, si può accedere alla `MainActivity`. Questa Activity è costituita da tre Tab: nella prima sono elencati gli eventi a cui l'utente è stato invitato, oppure di cui è l'organizzatore, ma che devono ancora avere luogo (*Active Events* - Figura 4.10a); nella seconda sono elencati gli eventi ai quali l'utente ha deciso di non partecipare (*Story* - (Figura 4.10b); infine, nell'ultima sono visibili delle immagini che mostrano i protocolli di sicurezza anti COVID-19 anche con riferimento al settore della ristorazione (*Security Protocols* - Figura 4.10c). Se l'utente è l'organizzatore del party, vedrà come immagine dell'evento l'icona della casa con sfondo blu mentre se è un invitato vedrà l'icona della forchetta con il colore degli sfondi differenti a seconda della propria decisione di partecipare o meno all'evento. Se lo sfondo è giallo vuol dire che la decisione è ancora da prendere, se lo sfondo è verde vuol dire che l'invito è stato accettato, mentre se lo sfondo è rosso l'evento è stato rifiutato (in questo caso si trova esclusivamente nella Tab *Story*).
Nella Tab *Active Events*, oltre alla lista degli eventi, è presente in basso a destra il pulsante tramite il quale si viene rimandati all'Activity di creazione dell'evento; inoltre, facendo swipe verso il basso si può aggiornare la lista stessa facendo apparire quindi la rotellina di caricamento, al termine del quale, se ci sono, verranno aggiunti nuovi eventi.

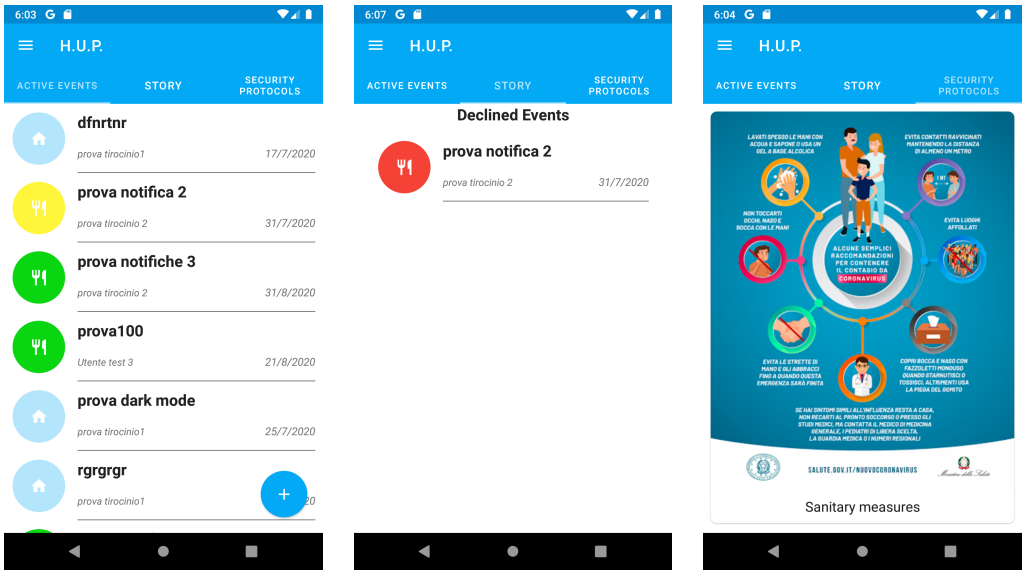

Figura 4.10: Screenshot della `MainActivity` (prima parte): a) Tab *Active Events*; b) Tab *Story*; c) Tab *Security Protocols*

Nella NavigationView, che diventa visibile o premendo sull'icona in alto a sinistra o facendo swipe verso destra, viene data all'utente anche la possibilità di cambiare i propri dati personali (Figura 4.11a). Questa sezione è costituita da un Header che comprende la propria immagine del profilo (modificabile premendo sull'immagine stessa); al di sotto, sono visibili nome, e-mail e numero di telefono. Sotto l'Header è

presente un menù costituito da cinque voci: tramite *Intolerances Modify* può essere modificata la lista delle intolleranze presenti nel Database; con *Modify Privacy Setting* si può decidere di rendere visibili o meno i propri dati; per mezzo di *Modify Health State* l'utente può dichiararsi "sano" o "malato" agli occhi degli altri utenti; con *Change Theme* l'utente può decidere di cambiare il tema dell'applicazione passando al dark mode e viceversa (Figura 4.11b), infine con *Exit* l'utente può chiudere la propria sessione di login, ed essere rimandato all'Activity che permette l'accesso o la registrazione.
Ogni evento è cliccabile e, una volta premuto su di esso, si viene reinviati all'Activity che ne mostra i dettagli.

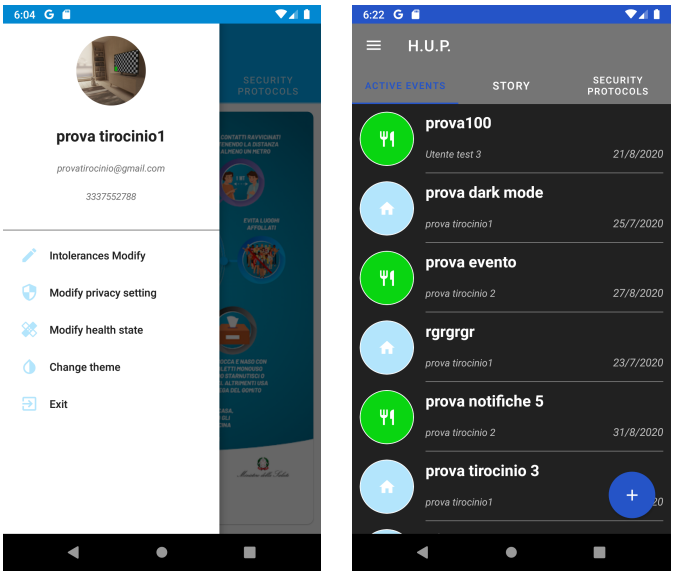

Figura 4.11: Screenshot della `MainActivity` (seconda parte): a) NavigationView; b) DarkMode

4.3.10 EventInfo

Questa Activity è costituita da tre Tab che mostrano le informazioni relative all'evento selezionato. Ogni Tab contiene informazioni di tipo diverso: la Tab *Menù* contiene il menù che verrà servito il giorno dell'evento (Figura 4.12a), la Tab *Positions and Contacts* contiene i dati relativi all'organizzatore (immagine del profilo, nome, e-mail e numero di telefono), nonché la data, l'ora e la posizione dell'evento (Figura 4.12b). La Tab *Guests* contiene, appunto, la lista degli invitati; per ognuno di essi vengono visualizzati l'immagine del profilo, il nome, l'e-mail, il numero di telefono e lo stato dell'invito, rappresentato con dei cerchi di colori differenti. Con il colore giallo viene indicato che la decisione non è stata ancora presa, con il colore verde viene indicato che l'invito è stato accettato, mentre con il rosso che l'utente non parteciperà all'evento (Figura 4.12c). All'atto dell'accettazione dell'invito, inoltre, ciascun invitato dovrà compilare un'autodichiarazione (Figura 4.13) con la quale

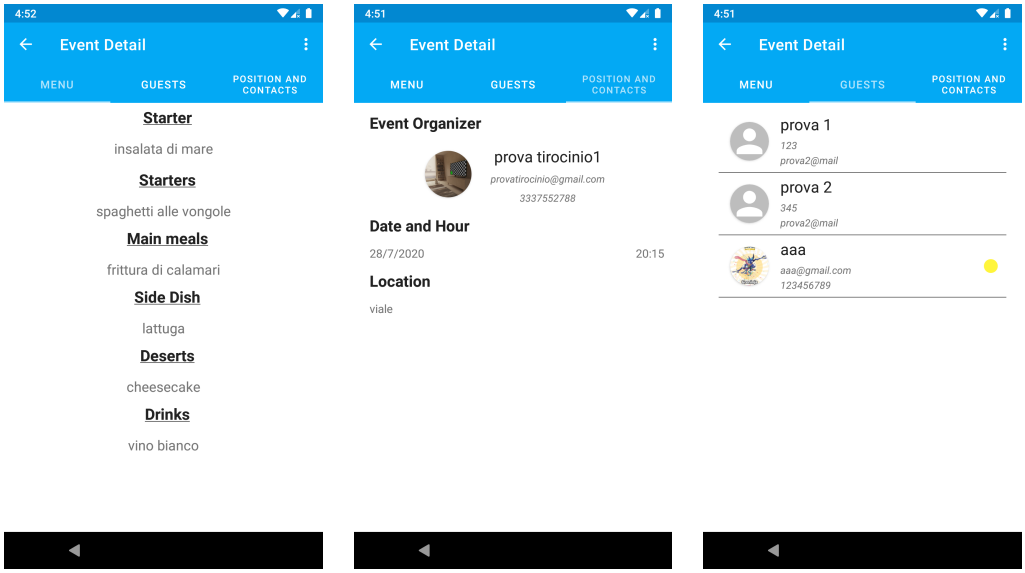

Figura 4.12: Screenshot dell'Activity `EventInfo`: a) Tab Menù; b) Tab Positions and Contacts; c) Tab Guests

attesta di non essere stato sottoposto a rischio di contagio nelle due settimane precedenti all'evento.
Una notifica Push il giorno stesso dell'evento ne ricorderà la ricorrenza a tutti gli ospiti che hanno accettato l'invito.

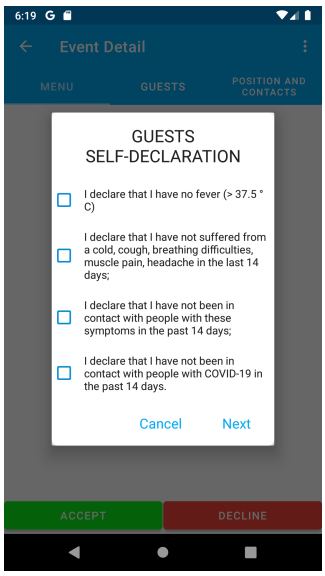

Figura 4.13: Tab relativa all'autodichiarazione

Varianti dell'Activity

La visualizzazione e le funzionalità di questa Activity cambiano a seconda del ruolo dell'utente all'interno dell'evento organizzato. Per questo motivo, di seguito, verrà descritta l'Activity distinguendo i vari casi e analizzandoli nel dettaglio:

1. Utente organizzatore
   Se il ruolo dell'utente è quello di "organizzatore", verrà visualizzato il menù con le voci *Delete* e *Modify* (Figura 4.14). Alla pressione dei rispettivi pulsanti si aprirà un Dialog con il quale si può confermare o annullare la scelta. Scegliendo la prima opzione l'evento non sarà più accessibile e verrà eliminato definitivamente dal Database. Attraverso la seconda opzione, invece, si verrà reinviati alle Activity di creazione dell'evento per modificarne i campi già riempiti.

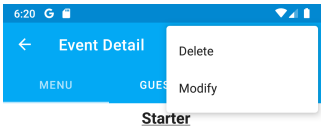

Figura 4.14: Opzioni relative all'utente organizzatore

2. Utente invitato
   Se il ruolo dell'utente è quello di "invitato" allora abbiamo più azioni e menù diversi.
   - Decisione non ancora presa
     Per prima cosa è necessario accettare o rifiutare l'invito all'evento. Questa possibilità è data da due pulsanti presenti in fondo allo schermo, uno di colore verde, che permette all'utente di accettare l'invito, ed uno di colore rosso, per rifiutarlo (Figura 4.15).

Figura 4.15: Opzione di accettazione o rifiuto dell'invito

   - Invito accettato
     Nel caso in cui l'utente accetti l'invito all'evento verrà visualizzato un ContextMenu con l'opzione *Exit* che dà la possibilità all'utente di ritirarsi dall'evento pur avendone precedentemente accettato l'invito (Figura 4.16). Alla pressione di questa voce viene visualizzato, anche in questo caso, un Dialog per confermare o meno la propria volontà e in caso di abbandono l'evento verrà visualizzato nella lista degli eventi della Tab *Story* con un'icona con sfondo rosso.
   - Invito rifiutato
     Nel caso in cui l'utente rifiuti l'invito all'evento, quest'ultimo verrà visualizzato nella sezione *Story*. Da qui, alla pressione dell'evento, si potranno

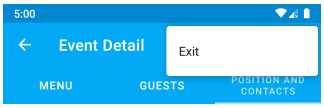

Figura 4.16: Opzione di abbandono dell'evento

ancora visualizzare i dati che lo caratterizzano; in più, sarà presente un ContextMenu contenente al suo interno la voce *Delete from List* (Figura 4.17). Premendo su quest'ultima si darà la possibilità all'utente di poter eliminare definitivamente l'evento dalla lista contenuta nella sezione *Story* evitando, quindi, un eventuale accumulo di eventi rifiutati.

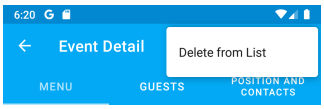

Figura 4.17: Opzione di eliminazione dalla lista

4.3.11 NewEvent

All'apertura dell'Activity un Dialog richiede all'utente se vuole organizzare l'evento osservando le normative anti COVID-19 (Figura 4.18a). L'Activity `NewEvent` presenta, poi, quattro EditText per l'inserimento di nome, data, ora ed indirizzo dell'evento (Figura 4.18b). In particolare, per quanto riguarda la data e l'ora, al fine di evitare

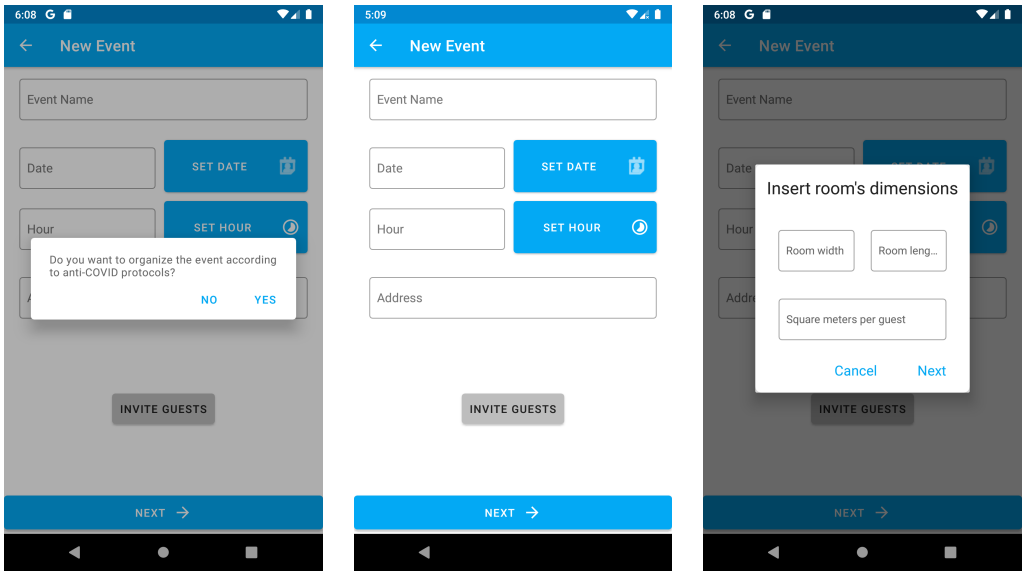

Figura 4.18: Screenshot dell'Activity `NewEvent` (prima parte): a) Opzione osservanza norme anti COVID-19; b) Inserimento dati evento; c) Opzione inserimento dimensioni stanza

che l'utente possa digitarli in maniera errata, sono state disabilitate le EditText

relative, e sono stati inseriti due pulsanti che, al click, aprono, rispettivamente, un DatePickerDialog (Figura 4.19a) ed un TimePickerDialog(Figura 4.19b).

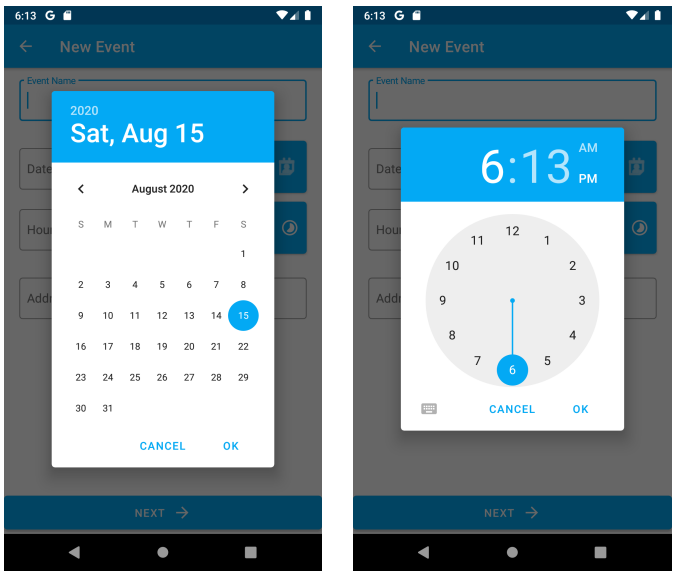

Figura 4.19: Screenshot dell'Activity `NewEvent` (seconda parte): a) Inserimento data evento; b) Inserimento ora evento

L'AppBar contiene, poi, in alto a sinistra, un pulsante di navigazione per tornare all'Activity precedente; qualora fossero stati già inseriti dei dati, si aprirà un Dialog che chiede di confermare il comando. Qualora si sia data risposta positiva alla domanda riguardante l'osservanza delle norme anti COVID-19, al click del pulsante *Invite guests* verrà, poi, richiesto se trattasi di evento da organizzarsi all'aperto o al chiuso; solo in quest'ultimo caso verrà anche richiesto di specificare le dimensioni della stanza e i metri quadrati che si vogliono concedere a ciascun invitato, così da consentire all'applicazione di calcolare il numero massimo degli ospiti (Figura 4.18c). In ogni caso, al click del pulsante, avviene il passaggio all'Activity `ContactEntry` mentre il pulsante *Next* permette il passaggio all'Activity `FoodIntolerance`, non prima che i quattro campi siano stati riempiti e che la lista degli invitati contenga almeno un elemento. Per evitare la perdita dei dati inseriti nelle quattro EditText al ritorno all'Activity `NewEvent`, questi sono stati memorizzati all'interno delle SharedPreferences.

4.3.12 ConctactEntry

L'Activity `ContactEntry` è costituita principalmente da una RecyclerView dove ogni riga corrisponde ad un invitato all'evento. Chiaramente, al primo accesso, la lista sarà vuota (Figura 4.20a). Dopo l'inserimento degli invitati, ogni riga sarà costituita da tre TextView relative a nome, e-mail e cellulare, un'ImageView contenente l'immagine del profilo (se esistente) ed, infine, un ImageButton per

l'eliminazione del contatto dalla lista.(Figura 4.20b)

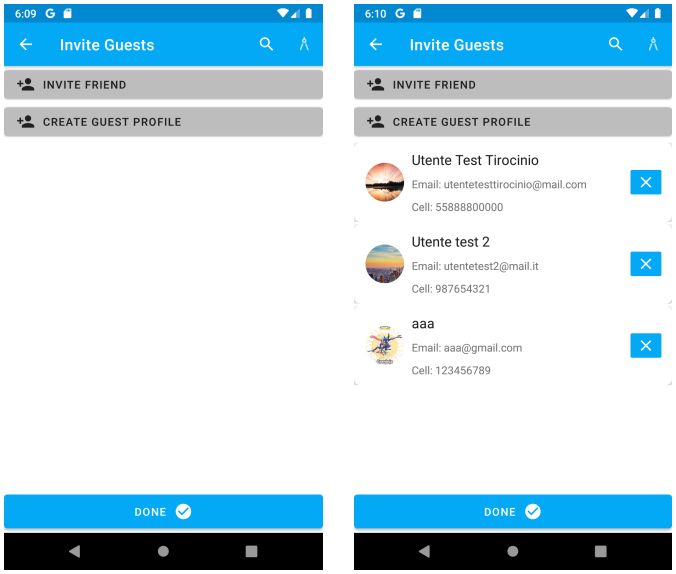

Figura 4.20: Screenshot dell'Activity `ContactEntry` (prima parte): a) lista vuota al primo accesso; b) riempimento lista

All'interno dell'AppBar, oltre al tasto per tornare indietro, è presente una SearchView che ci permette di filtrare gli elementi della lista degli invitati per nome, e-mail oppure numero di cellulare. A fianco della SearchView, se è stato scelto di organizzare l'evento seguendo le normative anti COVID-19, sarà presente un altro MenuItem, contrassegnato dall'icona del compasso, per modificare, eventualmente, la scelta delle dimensioni della stanza. Sulla base delle dimensioni fornite, infatti, viene calcolato e mostrato all'organizzatore il numero massimo di invitati che si possono ancora inserire (Figura 4.21a) e quanti, invece, superano il limite massimo (Figura 4.21b).
I pulsanti *Invite friend* e *Create guest profile* mandano, rispettivamente, alle Activity `ExistingContact` e `NewContact` ; il pulsante *Done*, posto in fondo all'Activity, consente, invece, di ritornare a `NewEvent` passando ad esso la lista degli invitati.
Qualora, però, l'organizzatore abbia deciso di osservare le normative anti COVID-19, la pressione del pulsante mostrerà un Dialog in cui viene richiesto se si vuole avere un'anteprima della disposizione dei posti a tavola. In caso di risposta positiva, si verrà indirizzati all'Activity `CohabitingGroups`.

4.3.13 ExistingContact

L'Activity `ExistingContact` è costituita anch'essa da una RecyclerView dove ogni riga rappresenta uno degli account registrati sull'applicazione. Le righe, oltre a contenere le tre TextView relative a nome, e-mail e cellulare, e l'ImageView per l'eventuale immagine del profilo, presentano una CheckBox per la selezione del contatto da aggiungere alla lista invitati (Figura 4.22a). Anche in questa Activity, oltre al

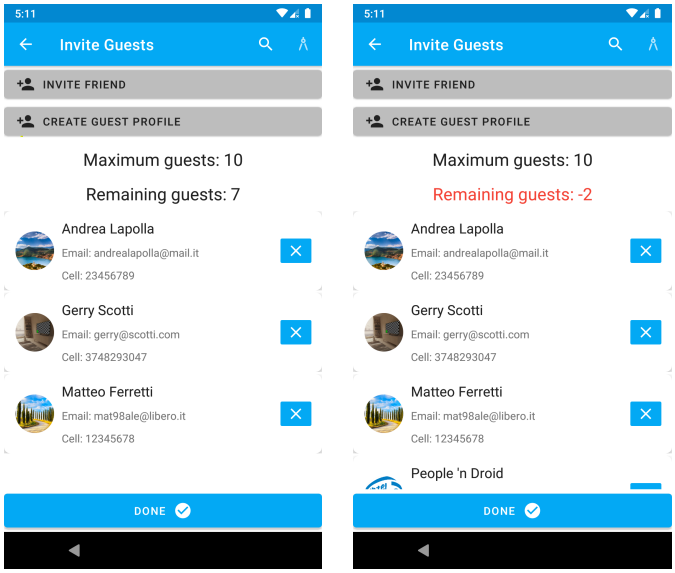

Figura 4.21: Screenshot dell'Activity `ContactEntry` (seconda parte): a) numero massimo invitati evento COVID; b) invitati in eccesso

pulsante per tornare indietro, all'interno dell'AppBar è presente una SearchView che ci permette di filtrare gli elementi della lista degli invitati per nome, e-mail oppure numero di cellulare.
Come si può vedere dalla Figura 4.22b, alcuni contatti non presentano dati visibili, al di fuori del nome e, soprattutto, non è presente una CheckBox per poterli invitare; ciò è dovuto al fatto che questi utenti hanno scelto di rendere "privato" il loro account. L'unico modo per visualizzare i loro dati e poterli, quindi, invitare è quello di inserirne correttamente e per intero o il numero di cellulare o la mail.
Gli utenti che, invece, sono evidenziati in rosso e presentano l'icona di un cerotto al posto dell'immagine del profilo non potranno essere invitati in quanto hanno dichiarato di essere malati (Figura 4.22c).

Alla creazione dell'Activity è necessario effettuare il download dei contatti dal Database Firestore: poiché, in presenza di molteplici contatti, l'operazione potrebbe richiedere un allungamento dei tempi di attesa, per segnalare l'effettivo svolgimento del download è stata inserita una ProgressBar che scompare al termine dell'operazione (Figura 4.23).

La spunta di una CheckBox rende visibile sul fondo dell'Activity il pulsante *Add guests* che consente l'aggiunta dei contatti selezionati alla lista degli invitati (Figura 4.22c). Quando tutti gli elementi della lista vengono selezionati il bottone torna invisibile.

4.3.14 NewContact

L'Activity `NewContact` consente di aggiungere alla lista degli invitati persone che non hanno scaricato l'applicazione e che, di conseguenza, non possiedono un account.

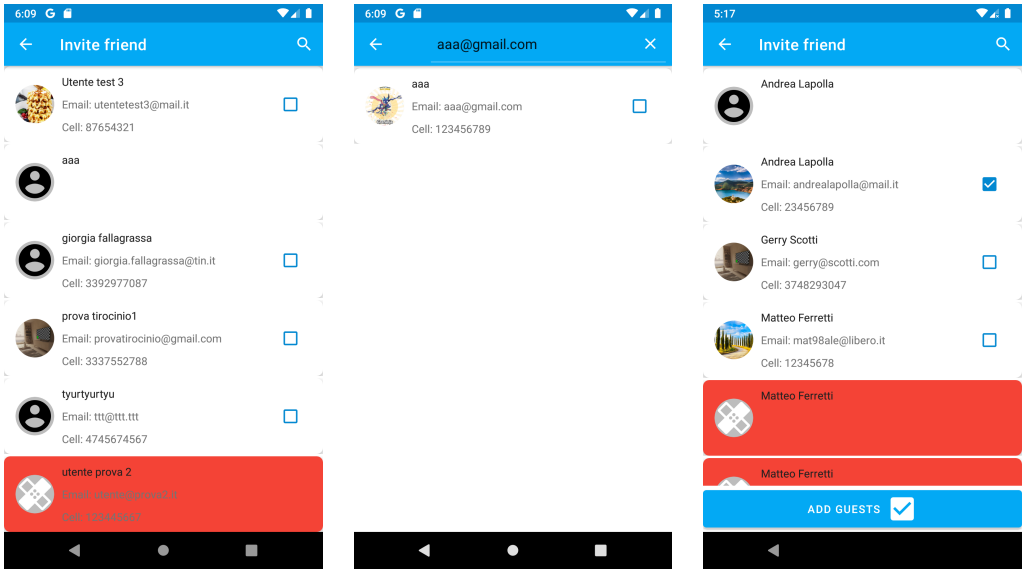

Figura 4.22: Screenshot dell'Activity `ExistingContact` prima parte): a) Selezione invitati; b) Invitati con account "privato", c) aggiunta contatti alla lista

Figura 4.23: Screenshot dell'Activity `ExistingContact` (seconda parte) Progress Bar

L'interfaccia utente dispone di tre EditText per l'inserimento dei tre campi obbligatori relativi a nome, e-mail e cellulare (Figura 4.24a), ed una AutoCompleteTextView, per selezionare da una lista salvata sul Database Firestore le intolleranze alimentari segnalate dagli invitati che andranno ad aggiungersi ad una RecyclerView sottostante (Figura 4.24b).
Al termine dell'inserimento dei dati del nuovo ospite il pulsante *Add Guests* ci permette di aggiungere il suo nominativo alla lista invitati.

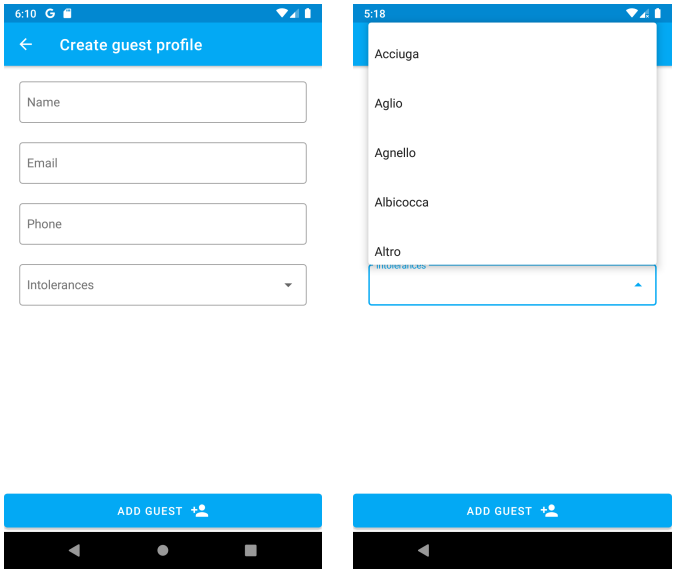

Figura 4.24: Screenshot dell'Activity `NewContact`: a) inserimento dati invitato; b) selezione intolleranze

4.3.15 CohabitingGroupsActivity

L'Activity `CohabitingGroups` ha l'obiettivo di definire gruppi di conviventi tra i membri della lista degli invitati, al fine di esonerarli dal rispetto delle regole sul distanziamento sociale. Essa è costituita da tre pulsanti posti sul fondo dello schermo: il tasto tondo con l'immagine del "+" serve ad aggiungere un nuovo gruppo del quale andranno, poi, definiti nome e numero di membri (Figura 4.25b) Il pulsante *Cancel* serve a tornare indietro all'Activity `ContactEntry`, mentre il tasto *Next* verifica, innanzitutto, la correttezza dell'inserimento dei gruppi e, se il numero dei membri tra i gruppi è inferiore a quelli contenuti nella lista degli invitati, provvede automaticamente a generare gruppi da un solo membro. Al termine dell'operazione, si viene indirizzati all'Activity `GuestsLayout`.

4.3.16 GuestLayoutActivity

Alla creazione di questa Activity, viene richiesto all'utente, tramite un Dialog, quali sono le dimensioni del tavolo che ospiterà gli invitati (Figura 4.26a). Le dimensioni dichiarate verranno utilizzate per realizzare una griglia contenente celle da 60 cm di lunghezza per 80 cm di larghezza rappresentanti i posti a sedere (Figura 4.26b).

All'interno di una NavigationView, accessibile premendo il pulsante in alto a sinistra o facendo swipe verso destra, si trova la lista di tutti i gruppi di conviventi, già definiti precedentemente, contenenti il numero di membri di ciascuno (Figura 4.27a). Le icone dei singoli membri di un gruppo potranno essere trascinate all'interno della griglia e posizionate nelle celle bianche, che assumeranno, una volta occupate, il colore rosso. È importante notare che non è possibile trascinare membri di gruppi

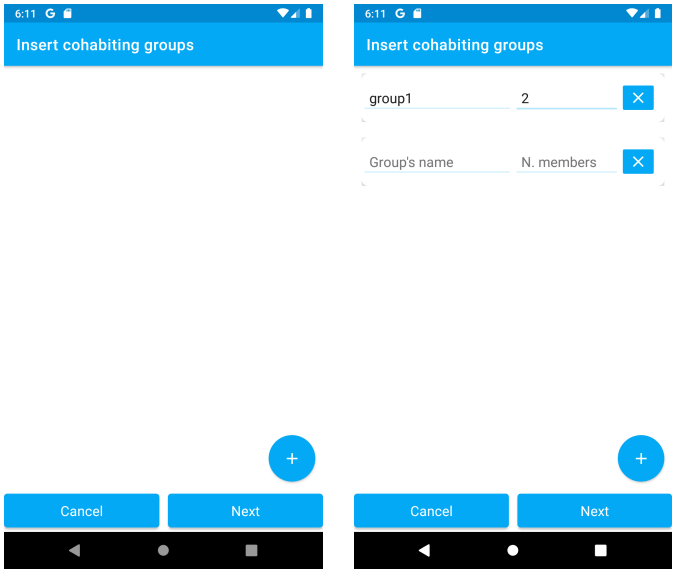

Figura 4.25: Screenshot dell'Activity `CohabitingGroups`: a) activity all'accesso; b) activity dopo l'inserimento di un gruppo

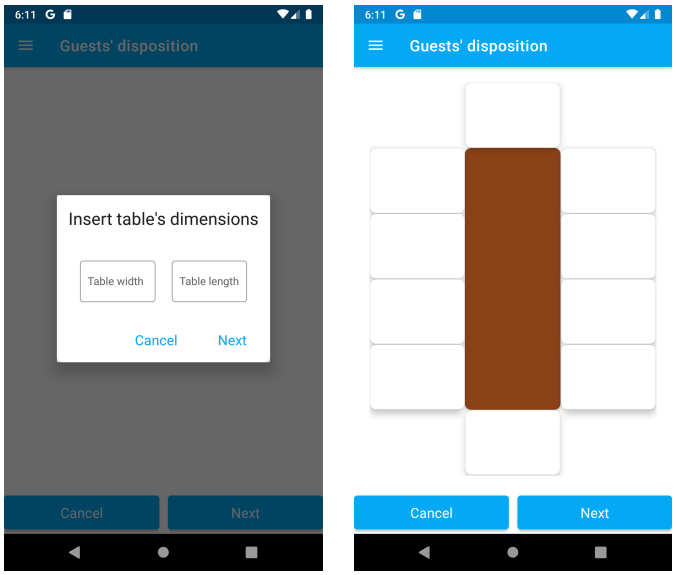

Figura 4.26: Screenshot dell'Activity `GuestsLayout` (prima parte): a) inserimento dimensioni tavolo; b) griglia dei posti

diversi su celle contigue, a meno che non si tratti del capotavola che già garantisce il distanziamento sufficiente con gli altri commensali (Figura 4.27b).

Sul fondo, il pulsante *Cancel* consente di tornare indietro all'Activity `ContactEntry` mentre il pulsante *Next* indirizza all'Activity `NewEvent`.

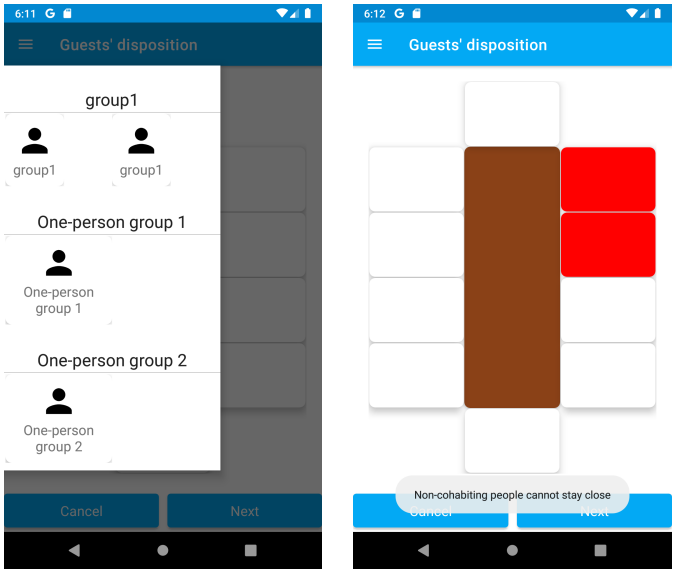

Figura 4.27: Screenshot dell'Activity `GuestsLayout` (seconda parte): a) elenco dei gruppi; b) disposizione posti al tavolo

4.3.17 FoodIntolerance

L'Activity `FoodIntolerance` mostra tutte le intolleranze alimentari dichiarate dai membri della lista degli invitati, specificandone anche il numero, all'interno di una RecyclerView. Al di sopra di essa, è presente una CardView il cui aspetto varia a seconda della presenza (Figura 4.28a) o meno (Figura 4.28b) di intolleranze alimentari fra gli ospiti .
Sul fondo dell'Activity un pulsante *Next* ci indirizza all'Activity `MenuCreation`.

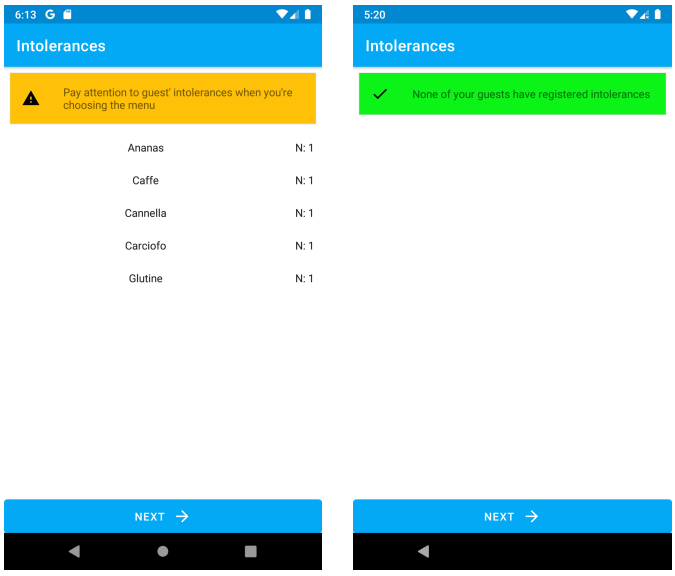

Figura 4.28: Screenshot dell'Activity `FoodIntolerance`: a) elenco intolleranze degli ospiti; b) assenza di intolleranze

4.3.18 MenuCreation

Una volta inserite tutte le informazioni riguardo l'evento e formata la lista degli invitati, il passo successivo e conclusivo è quello della creazione del menù.
Questa Activity presenta un'eventuale CardView in presenza di intolleranze alimentari (Figura 4.29a) fra gli ospiti, mostrate nuovamente da una finestra di Dialog che si apre al suo click (Figura 4.30a).
Il corpo dell'Activity, dovendo contenere strutture ripetitive relative all'inserimento delle varie portate, è stato realizzato attraverso l'utilizzo di un Fragment ripetuto sei volte in corrispondenza di ogni singola tipologia di portata (antipasti, primi, secondi, contorni, dessert, bevande).

Il Fragment è costituito da una AutoCompleteTextView che, per ogni portata,

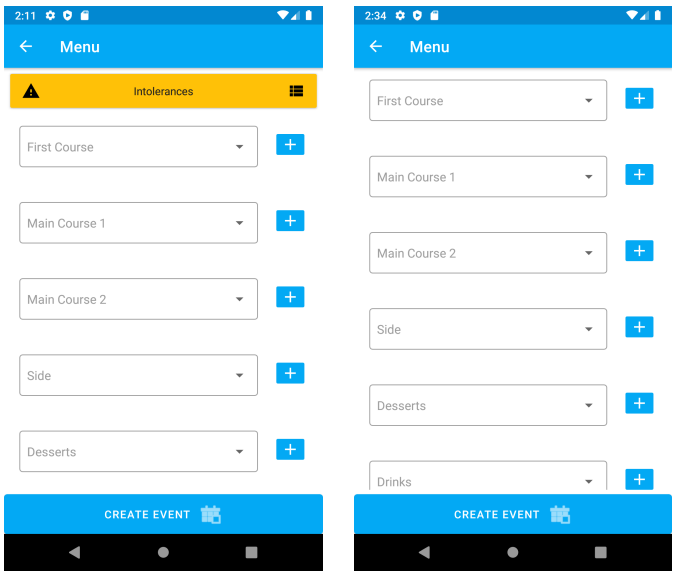

Figura 4.29: Screenshot dell'Activity `MenuCreation` (prima parte): a) creazione menù con intolleranze; b) creazione menù senza intolleranze

contiene come suggerimenti i piatti già realizzati in eventi passati e già salvati nel Database. In ogni caso, sia che si sia deciso di riproporre un piatto già presente nel Database, sia che si sia deciso di inserirne uno completamente nuovo, sarà necessario premere il pulsante "+" sulla destra al fine di salvare il nuovo piatto nella RecyclerView sottostante.
Terminata la creazione del menù e al click del pulsante *Create Event*, potrà aprirsi un'eventuale finestra di dialogo per avvertirci se un piatto scelto è stato già proposto allo stesso invitato in un precedente evento, con indicazione della data relativa (Figura 4.30b). Una volta deciso di continuare nonostante l'avvertimento, l'evento verrà salvato nel Database unitamente ai nuovi piatti creati per l'occasione che, quindi, verranno inseriti tra i suggerimenti delle AutoCompleteTextView per i prossimi eventi.

Come ultimo atto, viene inviata una notifica push a tutti gli invitati all'evento.
Terminata la creazione dell'evento. si torna automaticamente alla `MainActivity`.

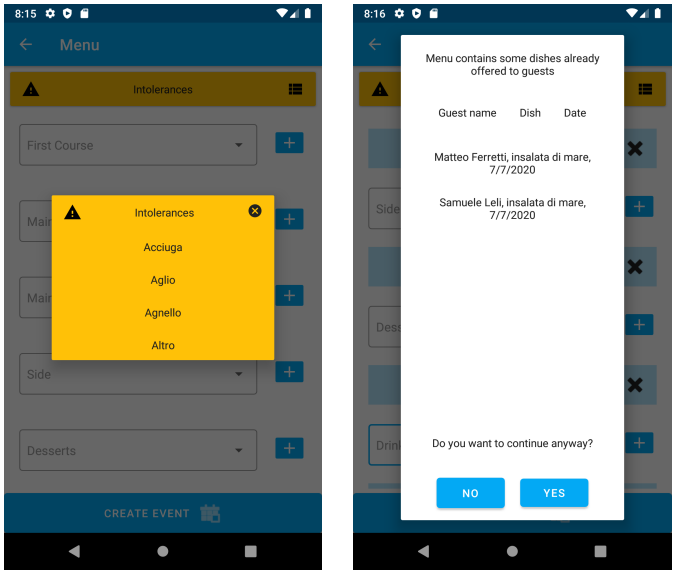

Figura 4.30: Screenshot dell'Activity `MenuCreation`(seconda parte):a) lista intolleranze ospiti; b) elenco piatti già proposti

Capitolo 5

Considerazioni sul lavoro svolto

In questo capitolo verranno preliminarmente svolte alcune considerazioni e valutazioni sul progetto che si è portato a termine, soprattutto sotto l'aspetto del know-how che è scaturito dalla sua realizzazione. Successivamente, verrà svolta un'analisi SWOT, cioè una valutazione strategica del progetto realizzato, al fine di valutarne i punti di forza e le eventuali criticità.

5.1 Lezioni Apprese

La realizzazione pratica di un progetto costituisce un' importante fase di accrescimento e di acquisizione di competenze. Sperimentare concretamente ciò che si è studiato risulta sicuramente stimolante sotto vari aspetti, rappresentando l'occasione per testare le proprie capacità organizzative, operative, decisionali, ma soprattutto per rapportarsi, per la prima volta, con una situazione reale che ci proietta, anche se in minima parte, in quello che potrebbe essere il nostro prossimo futuro professionale. La progettazione, e la successiva implementazione, dell'applicazione oggetto di questa tesi hanno, sicuramente, comportato l'approfondimento di alcuni argomenti già trattati in corsi di studio precedenti, quali lo studio del paradigma ad oggetti e l'utilizzo dei diagrammi UML per la fase di progettazione. La realizzazione di alcune funzionalità dell'applicazione ha comportato, però, la necessità di affrontare anche argomenti del tutto nuovi che, non essendo stati affatto trattati in precedenza, hanno determinato una fase di ricerca e di studio individuale. Ciò, ad esempio, si è verificato in occasione dell'implementazione delle seguenti funzionalità:

- **Dark Mode**
  Questa funzionalità è stata aggiunta in un momento successivo, dopo che gli stili erano stati già definiti e realizzati. Per questo motivo e per non pregiudicare il lavoro già svolto, è stata presa la decisione di creare una seconda versione del file `colors.xml` definendo, al proprio interno, colori con il medesimo identificativo, ma con diverso valore esadecimale. Così facendo, sono state realizzate due versioni dei medesimi colori, una utilizzata nella modalità chiara, l'altra nella dark mode, alla quale si passa mediante una specifica istruzione.

- **Notifiche Push**
  Questa funzionalità si è resa necessaria per notificare a ciascun invitato la notizia dell'invito ad un evento. Il metodo utilizzato per dare soluzione al problema è stato quello di effettuare, all'atto della creazione di un evento, una chiamata HTTPS per attivare una Function scritta in linguaggio JavaScript e caricata sulla console di Firebase che, a sua volta, invia a tutti gli invitati all'evento un messaggio per mezzo del Firebase Cloud Messaging. Il messaggio, una volta ricevuto dal dispositivo dell'invitato, viene utilizzato da un apposito Service per generare una notifica Push.

- **Drag&Drop**
  Per l'implementazione del Drag&Drop, utilizzato per la gestione dei posti a tavola, è stato necessario realizzare due classi, una che estendesse `DragShadowBuilder`, per creare l'ombra dell'oggetto trascinato, ed un'altra che estendesse `OnDragListener`, per gestire, appunto, tutte le varie situazioni che si possono verificare quando l'oggetto viene trascinato su una delle celle che rappresentano i posti a tavola.

5.2 SWOT Analysis del sistema realizzato

Con l'acronimo "SWOT" (Strengths, Weakness, Opportunities, Threats), in ambito di pianificazione strategica aziendale, si identificano le variabili intrinseche ed estrinseche di cui un'impresa tiene conto prima di avviare un progetto o di prendere una decisione per il raggiungimento di un obiettivo. Le quattro variabili prese in esame vengono rappresentate attraverso una matrice nella quale i primi due punti (forze e debolezze) descrivono fattori interni sui quali è di solito possibile esercitare un certo controllo, mentre gli altri due (opportunità e minacce) sono fattori esterni totalmente indipendenti che, quindi, sfuggono alla possibilità di una nostra gestione. (Figura 5.1).
Un'analisi SWOT, anche detta Matrice SWOT, analizza, quindi, un progetto o un business focalizzandosi su ciascuno di questi fattori, mettendone a fuoco le caratteristiche distintive nel tentativo di valorizzare quelli che sono gli aspetti positivi del progetto e di minimizzare, invece, quelli negativi.

La nascita di questa metodologia di analisi (nota anche come SWOT Analysis) si deve all'economista statunitense Albert Humphrey che, nell'ambito di studi condotti per lo Stanford Research Institute, intorno agli anni '60, cercò di analizzare i motivi per cui la pianificazione aziendale falliva. La metodologia di cui si fece portatore consente uno studio razionale e scientifico dei contesti interni ed esterni, rendendo sistematiche, e facilmente fruibili, tutte le informazioni che potrebbero influenzare il sistema.

Vediamo i risultati che possono scaturire da un'analisi del progetto di applicazione descritto in questa tesi.

Figura 5.1: Matrice SWOT

5.2.1 Punti di forza

I punti di forza, come già detto, costituiscono quei fattori interni positivi che incidono sul progetto e contribuiscono a determinarne il successo. I punti di forza individuati per l'applicazione oggetto della presente tesi , sono:

- La *versatilità*, in quanto l'applicazione, oltre a consentire una gestione degli eventi nell'attuale periodo contrassegnato dalla presenza del Covid, presenta anche altre funzionalità che la rendono fruibile anche quando, si spera, l'attuale situazione di pandemia sarà solo un ricordo.

- L'*unicità*, in quanto sul mercato non sembrano esserci app uguali. Le applicazioni che trattano della gestione di eventi per lo più riguardano la disposizione dei posti a tavola in occasione di eventi con tanti invitati, come, ad esempio, nei matrimoni, mentre quelle che trattano di eventi domestici riguardano principalmente il nascente fenomeno degli "Home Restaurant".

- Il peso dell'APK generato è di circa 8 megabyte, quindi l'applicazione risulta essere abbastanza leggera.

5.2.2 Punti di debolezza

I punti di debolezza, al contrario dei precedenti, rappresentano quei fattori interni negativi che possono determinare l'insuccesso del progetto e, pertanto, dovrebbero essere minimizzati quanto più possibile. Nel caso in questione:

- L'applicazione per funzionare ha bisogno della connessione dati che, seppure ormai utilizzabile da quasi tutti i dispositivi, potrebbe tuttavia mancare, determinando l'impossibilità di utilizzare l'applicazione stessa.
- L'applicazione è, poi, strutturata su tante Activity, il che potrebbe rendere la navigazione un pò pesante per l'utente.

5.2.3 Opportunità

Un'eventuale opportunità per lo sviluppo di questa applicazione potrebbe scaturire dal timore che sempre più persone hanno nel frequentare locali pubblici (ristoranti, bar) dove è più facile la possibilità che si creino assembramenti, e quindi che aumentino i rischi di contagio. Questo potrebbe contribuire ad incrementare le occasioni in cui amici o parenti decidano di incontrarsi con maggiore tranquillità tra le mura domestiche costituendo, quindi, un vantaggio per la diffusione dell'applicazione.

5.2.4 Minacce

Un fattore esterno che, invece, potrebbe rappresentare un limite per l'applicazione potrebbe essere costituito dal fatto che quest'ultima è stata strutturata sulla normativa attualmente vigente che, se dovesse venir cambiata in maniera sensibile dal Governo, potrebbe comportare la necessità di procedere ad un aggiornamento dell'applicazione stessa.

Capitolo 6

Conclusioni e possibili sviluppi futuri

Il presente elaborato ha come oggetto la descrizione delle fasi di progettazione e successiva implementazione di un'applicazione Android per la gestione di eventi domestici con piccoli gruppi di amici o parenti, con particolare attenzione al delicato momento di pandemia che stiamo vivendo, dove la necessità di evitare occasioni di eccessivo affollamento e, di conseguenza, rischi di contagio, comporta l'esigenza di osservare determinati accorgimenti e regole anche fra le mura domestiche.
Preliminarmente è stata fatta una descrizione delle fasi in cui si è sviluppata la pandemia, partendo dai primi focolai scoperti in Cina tra dicembre 2019 e gennaio 2020 fino ad arrivare alla sua diffusione in Italia. Sono state, quindi, descritte tutte le fasi del lockdown imposto dal Governo alla nazione nel tentativo di arginare il contagio, a partire dal mese di marzo, con specifica indicazione dei protocolli di sicurezza prescritti alla cittadinanza, anche con riferimento al settore della ristorazione.
Nel secondo capitolo è stata effettuata una breve descrizione del sistema operativo Android, dalle sue origini fino ad arrivare al successivo e veloce sviluppo, documentato dalle innumerevoli versioni rilasciate negli anni, con particolare riferimento alle caratteristiche della sua struttura e, soprattutto, alla piattaforma per lo sviluppo di App che ne costituisce l'IDE primario, cioè Android Studio.
Nel terzo capitolo, dopo aver esposto brevemente gli obiettivi del progetto, ne sono state descritte le funzionalità attraverso l'analisi dei requisiti ed il diagramma dei casi d'uso, alcuni dei quali ulteriormente illustrati mediante il diagramma delle attività. Tramite i mockup, poi, è stata mostrata la veste grafica dell'applicazione ed, in ultimo, ne è stata descritta la componente dati, nello specifico il database impiegato.
Il quarto capitolo ha trattato, infine, l'implementazione; è stata mostrata la struttura dell'app e sono stati illustrati e spiegati alcuni dei listati riguardanti le fasi ritenute più rappresentative; è stato, poi, dedicato ampio spazio al manuale utente, cioè ad una guida dettagliata all'utilizzo dell'applicazione, e sono stati mostrati gli screenshot relativi a tutte le Activity che si susseguono al suo interno.
Nel quinto capitolo, in conclusione, sono state svolte alcune considerazioni e valutazioni sul progetto che si è portato a termine e sulle nozioni che sono state apprese con la sua realizzazione; è stata, altresì, svolta un'analisi SWOT, al fine di valutare i punti di forza o eventuali criticità del progetto.
La realizzazione di questo progetto si è rivelata molto interessante perchè, come già

detto, ha consentito di mettere in pratica e di cimentarsi con argomenti e problematiche che difficilmente sarebbero stati affrontati se il loro studio si fosse limitato al solo aspetto teorico. Per non parlare, poi, dell'entusiasmo e della soddisfazione che ha comportato il raggiungimento di ogni singolo obiettivo, magari dopo giorni di tentativi falliti e scoraggiamento.
Ovviamente, poichè ogni autore è il primo critico di se stesso, anche in questo caso non si può negare che esistano alcuni aspetti dell'applicazione che potrebbero essere migliorati; ad esempio, andrebbe rivista la logica di alcuni aspetti, per migliorarne la funzionalità; si potrebbe, inoltre, tentare di modificare la distribuzione ed il percorso delle Activity per migliorare l'esperienza utente.

Bibliografia

[1] Android. https://it.wikipedia.org/wiki/Android, 2020.

[2] Android Developers. https://developer.android.com, 2020.

[3] Camera dei Deputati - Misure sull'emergenza Covid. https://www.camera.it/temiap/documentazione/temi/pdf/1203754.pdf?_1598438210763, 2020.

[4] Documento INAIL e ISS. https://www.inail.it/cs/internet/docs/alg-pubbl-doc-tec-ipotesi-rimod-misure-cont-ristoraz-covid-2_6443147014458.pdf, 2020.

[5] Firebase. https://firebase.google.com/docs?hl=it, 2020.

[6] M. Bonifazi. *Sviluppare applicazioni per Android in 7 giorni.* Edizioni LSWR, 2016.

[7] M. Carli. *Android 9 - Guida completa per lo sviluppo di applicazioni mobile.* Apogeo, 2019.

[8] F. Collini, M. Bonifazi, A. Martellucci, and S. Sanna. *Android: Programmazione avanzata.* Edizioni LSWR, 2015.

[9] C. De Sio Cesari. *Manuale di Java 9: Programmazione orientata agli oggetti con Java standard edition 9.* Hoepli, 2018.

[10] P. Deitel, H. Deitel, A. Deitel, and M. Morgano. *Sviluppare App per Android.* Pearson, 2012.

[11] G. Grandinetti. *Android Studio 2: sviluppare vere applicazioni Android partendo da zero.* Edizioni Futura, 2016.

[12] C. Haseman. *Creare App per Android. Progettazione e sviluppo.* Sperling & Kupfer, 2015.

[13] C.S. Horstmann. *Concetti di informatica e fondamenti di Java (settima edizione).* Apogeo, 2020.

[14] R. Meier. *Professional Android 4 Application Development.* Wrox Pr Inc, 2012.

[15] W. Savitch. *Programmazione di base e avanzata con Java.* Pearson, 2018.