UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e implementazione di un'app in tecnologia Flutter per la gestione di un'azienda di lampade votive

Design and implementation of an app in Flutter technology for the management of a votive lamp company

Relatore

Prof. Domenico Ursino

Candidato

Andrea Iasenzaniro

Correlatore

Dott. Enrico Corradini

ANNO ACCADEMICO 2021-2022

Sommario

Nell'ultimo decennio lo smartphone è diventato un strumento fondamentale nella vita dell'uomo; chi di noi non ne ha, o non ne ha avuto, almeno uno nel corso della propria esistenza? Il suo utilizzo ricopre gran parte del nostro tempo, che trascorriamo utilizzando una o più app tra quelle che ci vengono offerte dai diversi sviluppatori. In questa tesi vengono presentati il lavoro e lo studio che si celano dietro il processo di progettazione della maggior parte delle applicazioni per dispositivi mobili, evidenziando, per la specifica app progettata, non solo gli aspetti positivi, ma anche quelli che potrebbero essere migliorati con il passare del tempo.

Keyword: App Multipiattaforma, Dispositivi Mobili, Sfide, Progettazione, Prototipo, Configurazione, SWOT Analysis.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Da circa cinquant'anni stiamo vivendo un periodo storico che viene individuato come quello della *rivoluzione digitale*; con questo termine ci si vuole riferire ad un'epoca caratterizzata da un'ampia diffusione degli strumenti digitali, i quali stanno portando, a loro volta, a degli inevitabili cambiamenti non solo sul modo di pensare delle persone, ma anche nell'organizzazione stessa della società. Le innovazioni tecnologiche stanno interessando ogni ambito sociale, medico, economico, edilizio, ma anche tutto ciò che circonda il mercato degli smartphone.

Il primo passo nel mondo del mobile è stato compiuto nel 1994, quando IBM, un'azienda statunitense fondata nel 1911 e che risulta ancora oggi tra le più quotate nell'ambito informatico, riuscì a produrre, e ad immettere sul mercato, quello che possiamo considerare il progenitore degli attuali smartphone in commercio: il *Simon*, un cellulare privo di tastiera fisica, ma che era possibile comandare tramite un pennino da utilizzare su uno schermo tattile.

Il 2008 è stato, poi, un momento fondamentale per lo sviluppo del mercato moderno; quell'anno è avvenuto un vero e proprio salto qualitativo con la nascita di Android, uno dei primi sistemi operativi che, insieme ad iOS, lanciato sul mercato appena un anno prima , sarebbe diventato il sistema operativo protagonista sugli attuali dispositivi mobili. Basti pensare che solo Android è presente in quasi 7 dispositivi su 10 venduti dai maggiori produttori a livello mondiale.

Parallelamente al mercato mobile ha preso piede, inevitabilmente, anche quello delle applicazioni; queste altro non sono che dei software specificatamente progettati per funzionare sui dispositivi mobili. Le app fanno, ormai, parte della vita di tutti i giorni, sono capaci di renderci l'esistenza più agiata sotto innumerevoli aspetti, ma generano, a loro volta, ulteriori bisogni, innescando, così, un meccanismo che sembra non avere mai fine.

Ad oggi sugli Store Apple ed Android si contano circa 1,5 milioni [1] e 2,6 milioni [2] di applicazioni per dispositivi mobili, rispettivamente. Questi numeri aiutano a comprendere come lo smartphone, e quindi la pervasività della sua tecnologia, possa essere capace di insinuarsi in ogni aspetto della vita quotidiana, interessando sia adulti che bambini. Esso ha ormai raggiunto non solo la pubblica amministrazione, ma anche medie e grandi imprese, le quali riescono ad erogare grazie alle app, parte dei propri servizi, a tutto vantaggio della velocità.

Siamo, anche, reduci da un lungo periodo di pandemia; non sono trascorse molte settimane da quando è stata dichiarata la fine dell'emergenza da Covid-19. Dal 2019 ad oggi la

[1] `https://toucharcade.com/`
[2] `https://www.appbrain.com/`

popolazione italiana ha smesso di crescere, anzi, ha iniziato, seppur di soli 0.4 punti percentuali di media annua[3], a decrescere. Tale andamento non è dovuto solo ad un incremento delle morti, causate in parte, tra il 2020 ed il 2021, dal Covid-19, che ha colpito il Mondo con oltre 6 Milioni di vittime[4], ma anche ad una progressiva diminuzione delle nascite nel Paese, dovute ad una concomitanza di diversi fattori tra i quali, indubbiamente, quelli di tipo economico e sociale. Sin dagli anni Ottanta del 1800, l'Italia è stata interessata da continui flussi emigratori verso altri paesi, che hanno portato le persone a ricercare delle migliori condizioni economiche e nuove opportunità di lavoro e di studio; l'ISTAT stima che tra il 1876 e il 1976 siano emigrati all'estero circa 26 milioni di italiani.

Dall'unione di tutte queste informazioni che sono state raccolte, e tenendo conto anche del contesto sociale di una piccola regione del centro Italia, ovvero il Molise, dalla quale questa tesi prende ispirazione, nasce l'idea di sviluppare un'applicazione per dispositivi mobili dedita alla gestione di aziende che operano nel settore dell'illuminazione votiva.

Numerosi sono stati i molisani che hanno preso parte ai fenomeni migratori italiani nel corso della storia, ma una caratteristica che li contraddistingue è l'attaccamento alla propria terra di origine. È usanza, infatti, in questa zona, ma non solo, riportare, al termine della vita terrena, la salma di un parente defunto nel cimitero della propria terra natia.

È proprio da queste considerazioni, unitamente alla consapevolezza della persistenza degli smartphone nella società odierna che nasce l'idea di sviluppare quest'app che consenta a chiunque, da qualsiasi parte del Globo, di poter avere un contatto, seppur minimo, con un caro defunto.

Questo elaborato ha come fine ultimo quello di presentare in modo chiaro ed esplicito il susseguirsi delle fasi che consentono di condurre una corretta progettazione di un'applicazione mobile di tipo multipiattaforma. Per prima cosa è stato necessario fornire una breve introduzione sul mondo mobile, con alcuni numeri che possano dare un maggior peso al lavoro svolto. La fase successiva, ovvero quella iniziale, ha riguardato l'analisi dei requisiti dell'applicazione, dai quali è stato possibile definire le funzionalità che questa avrebbe dovuto avere. Sono seguite, poi, le fasi di progettazione della struttura, dell'organizzazione e dell'archiviazione dei dati, di implementazione, ovvero di realizzazione dell'app, nonchè di realizzazione di una guida utente a supporto dell'utilizzo dell'app. Per terminare, è stata condotta una completa analisi del lavoro svolto, volta ad evidenziare non solo gli aspetti positivi, ma anche quelli negativi che sarebbe possibile migliorare in eventuali sviluppi futuri.

La presente tesi è articolata in sette capitoli strutturati come segue:

- Nel Capitolo 1 viene introdotto il mondo delle applicazioni mobili, evidenziando le sfide che uno sviluppatore deve affrontare nel proprio lavoro.
- Nel Capitolo 2 si fornisce una descrizione dell'applicazione in linguaggio naturale, seguita, poi, dall'analisi dei requisiti e delle funzionalità.
- Nel Capitolo 3 vengono riportate le fasi della progettazione, ovvero quelle relative all'organizzazione delle classi di codice, all'interfaccia e all'archiviazione dei dati.
- Nel Capitolo 4 vengono specificati gli strumenti utilizzati per lo sviluppo e vengono illustrate alcune delle funzioni di maggior rilievo nel codice.
- Nel Capitolo 5 viene riportata un breve manuale utente.
- Nel Capitolo 6 viene effettuata l'analisi SWOT dell'app, evidenziando sia i punti di forza che quelli di debolezza.

[3]Fonte ISTAT, 12 Aprile 2023

[4]Ministero della Salute, 6 Aprile 2023

- Nel Capitolo 7 vengono tratte le conclusioni, proponendo alcuni possibili sviluppi futuri.

CAPITOLO 1

Descrizione del contesto di riferimento

Questo primo capitolo funge da apripista per ciò che sarà affrontato nelle pagine successive; esso vuole semplicemente introdurre il lettore al mondo delle mobile app, facendogli cogliere, tra le altre cose, l'importanza del tema affrontato nello sviluppo dell'applicazione. Si cercherà ,pertanto, di fornire una visione globale sulle problematiche che comporta lo sviluppo di un'applicazione mobile e di come questo sia ampiamente supportato da adeguati strumenti software.

1.1 Settore di riferimento dell'app

Quello cimiteriale è un settore in continua evoluzione che, nostro malgrado, non sembra avere una potenziale fine.

Esso comprende numerose attività legate non solo ai defunti, quali la sepoltura e la manutenzione della lapide, ma anche operazioni di manutenzione ordinaria e straordinaria, come la pulizia del sito cimiteriale o l'illuminazione dello stesso.

È ormai noto che, in conseguenza anche dello stile di vita frenetico e della lontananza geografica dal luogo di sepoltura, la gestione di un defunto, che va dalla più banale pulizia della lapide, all'aggiunta di fiori, fino ad arrivare ad un controllo del corretto funzionamento della lampada votiva, può rivelarsi un'attività complicata ed onerosa.

Proprio in questo contesto si colloca 'Appdila', che vuole offrire una soluzione rapida, intuitiva ed efficace per ciò che riguarda la totalità delle operazioni di gestione di un caro defunto nella sfera cimiteriale.

L'applicazione consente, mediante semplici operazioni che possono essere eseguite da qualsiasi categoria di utente, di visualizzare un defunto, richiedere un servizio, allacciare o distaccare la lampada votiva ed avere anche un resoconto di tutte le fatture ricevute, per una gestione contabile veloce ed efficiente.

Ferme restando le funzionalità descritte, 'Appdila' rappresenta un primo passo verso l'innovazione del settore cimiteriale proponendosi di abbattere difficoltà e distanze tra amici, parenti e defunti.

1.2 Il mercato delle app e le sfide di sviluppo

Il mercato delle applicazioni per dispositivi mobili è stato protagonista di una crescita esponenziale fino allo scorso anno. Le statistiche riportate in Figura 1.1 rivelano che nel 2021 un utente ha trascorso circa 4.8 ore davanti allo schermo ogni giorno e che nel 2020 sono stati

effettuati approssimativamente 218 Miliardi di download da Google PlayStore ed AppStore [1] con un investimento che ammonta a quasi 73 Miliardi di dollari [2].

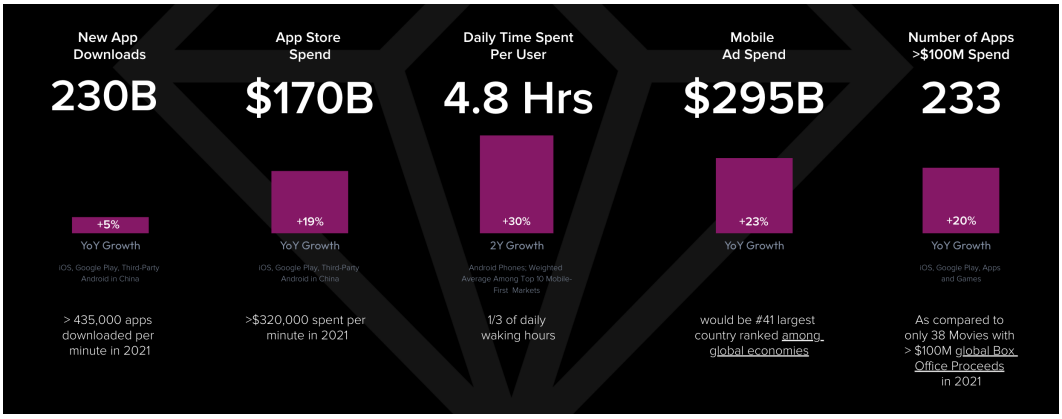


Figura 1.1: Panoramica del mercato mobile nel 2021

Il settore delle app è, allo stesso tempo, complicato e stimolante, frenetico per certi versi; è come se riflettesse appieno lo stile di vita dell'uomo.

Non c'è una fine allo sviluppo di un'app, a meno che non la si voglia portare al declino per poi dismetterla. È necessario stare al passo con i tempi rilasciando continui aggiornamenti con costanti migliorie sia dal punto di vista delle funzionalità che da quello dell'interfaccia utente.

Numerose sono le sfide che uno sviluppatore di applicazioni mobili deve affrontare.Esse si possono riassumere come segue:

- *Competitività del mercato.* Il mercato delle app è affollato, e sta diventando gradualmente più difficile rilasciare prodotti che piacciano agli utenti. Nel 2021, su Play Store, il numero di applicazioni era di circa 3,48 milioni, mentre su AppStore di circa 2.11 milioni. Con ciò si può ben comprendere come sia facile trovare della concorrenza; per ottenere una propria nicchia di mercato spesso non bisogna solo guardare alle funzionalità offerte quanto, invece, al modo in cui vengono rese disponibili all'utente.

- *Dimensioni e compatibilità del dispositivo.* Al giorno d'oggi ci sono centinaia di dispositivi sul mercato, dagli smartphone ai tablet agli smartwatch, che presentano differenti caratteristiche per ciò che concerne lo schermo, il processore ed il software. Diventa, quindi, una vera e propria sfida per lo sviluppatore fare in modo che l'app sia correttamente funzionante, visualizzabile e performante sulla maggior parte dei dispositivi in commercio.

- *Sicurezza delle app mobili.* Nessun utente utilizzerà mai un'app che raccoglie dati personali sensibili e che non riesce a garantire la loro sicurezza. Per questo motivo sia PlayStore che AppStore impongono severi requisiti di sicurezza a qualsiasi app che tratta le informazioni degli utenti. Per garantire ciò gli store non accettano le applicazioni che non rispettano gli standard richiesti.

- *Tecnologia di sviluppo scelta in modo errato.* È importante decidere se sviluppare un'app nativa o multipiattaforma, perché tale scelta può incidere sia sul tipo di utenza a cui è indirizzata l'app, che sulle prestazioni della stessa. Ad ogni modo rimandiamo un approfondimento sulla differenza tra le due modalità al paragrafo successivo.

[1] www.data.ai/en
[2] www.crunchbase.com

- *Funzionalità non necessarie o carenti.* Ogni app nasce per adempiere ad uno scopo ben preciso, e quindi per ricoprire una propria nicchia di mercato con una determinata categoria di utenti.

  Riempire eccessivamente l'app di funzionalità tanto che lo scopo della stessa possa non essere più soddisfatto, oppure complicare l'interfaccia in modo da non renderla più snella ed intuitiva, può spingere gli utenti a cercare un'app dello stesso genere ma più semplice, in quanto risulterebbe meno onerosa in termini di tempo speso per imparare ad utilizzarla.

1.3 App native e multipiattaforma

Molteplici sono i fattori che devono essere presi in considerazione prima di intraprendere lo sviluppo di un'applicazione: costo di sviluppo, tempo e funzionalità. Per di più, se si desidera scegliere come target sia il pubblico Android che iOS, potrebbe essere difficile scegliere quale approccio di sviluppo mobile sarà il migliore per un determinato progetto.

Indipendentemente dall'ambito in cui viene sviluppata l'applicazione, il tipo di approccio avrà inevitabilmente delle ripercussioni sul risultato finale prodotto.

È utile precisare, a tale proposito, che le applicazioni mobili non sono tutte uguali; potremmo sommariamente dividerle in app native ed app multipiattaforma.

1.3.1 Sviluppo di app native

La codifica di un'applicazione per una piattaforma specifica, come iOS o Android, è chiamata sviluppo di app nativa.

Sono app progettate per funzionare esclusivamente su un'unica piattaforma utilizzando il linguaggio e gli strumenti offerti dal sistema di proprietà, come, ad esempio, Kotlin per Android e Swift per iOS.

Sono pubblicate nei relativi store, PlayStore o AppStore, e, qualora si volesse raggiungere allo stesso tempo dispositivi sia Android che Apple, è necessario creare e mantenere due progetti totalmente distinti, aumentando, così, i tempi di sviluppo, i costi e le risorse necessarie alla loro gestione in parallelo. Malgrado ciò godono di grande reattività e affidabilità; sono in grado di supportare il funzionamento offline garantendo ottime prestazioni sia nell'accesso ai sensori integrati dei dispositivi che all'hardware sottostante.

1.3.2 Sviluppo di App multipiattaforma

Lo sviluppo di app multipiattaforma è il processo di creazione di app mobili compatibili con diversi sistemi operativi.

Invece di creare applicazioni separate per iOS e Android, mediante l'utilizzo di strumenti software dedicati come React Native, Xamarin e Flutter, è possibile, a partire da un unico codice, distribuire la stessa app su entrambe le piattaforme.

In questo modo, le applicazioni saranno in grado di funzionare su più sistemi operativi, permettendo di risparmiare tempo e denaro sia in fase di creazione che in fase di manutenzione dell'app, in quanto sarà necessario aggiornare una sola versione dell'applicazione.

Vale la pena sottolineare che le prestazioni e l'esperienza di utilizzo non hanno ancora raggiunto del tutto il livello delle applicazioni native, anche se i nuovi Framework e la potenza dei nuovi dispositivi, permettono di creare soluzioni valide.

1.4 Tecnologie disponibili

Lo sviluppo di app per dispositivi mobili è in continua evoluzione, con nuove tecnologie e framework che emergono ogni anno. Esistono varie soluzioni sul mercato e spesso è difficile scegliere quale sia meglio utilizzare.

1.4.1 Tecnologia Flutter

Flutter è un framework open source di Google per la creazione di applicazioni multipiattaforma.

Nato nel 2018 come progetto open source, con contributi di Google ed altre aziende, si è poi sviluppato come software stabile a partire dal 2019; supportava inizialmente solo lo sviluppo di app mobili mentre ora è stato esteso ad ulteriori piattaforme, come iOS, Android, Web, Windows, MacOS e Linux.

Il punto di forza di Flutter è quello di poter creare, in modo performante, applicazioni per differenti tipi di sistemi operativi a partire da un'unica base di codice sorgente, realizzando un'interfaccia grafica identica per ciascun sistema operativo.

Flutter utilizza il linguaggio di programmazione Dart, sviluppato anche da Google ed ottimizzato per la creazione di interfacce utente, che è compilato in anticipo (AOT) in librerie native ARM e x86. In questo modo all'avvio viene caricata la libreria Flutter così che il dispositivo comprenda direttamente il codice in modo da poter gestire autonomamente il rendering e gli input.

Nella Figura 1.2 viene riportato uno schema che mostra come Flutter comunichi direttamente con l'hardware sottostante, senza l'utilizzo di alcun bridge o altro componente che funga da intermediario.

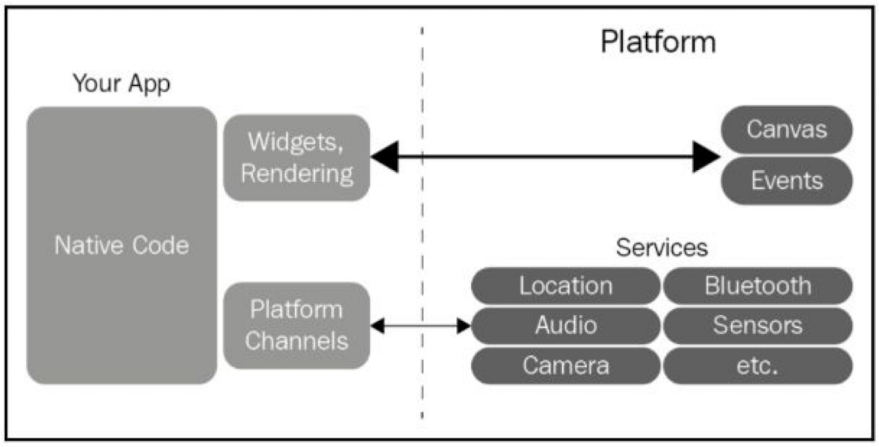


Figura 1.2: Diagramma di funzionamento Framework Flutter

Colossi come il New York Times, Alibaba e Baidu hanno già deciso di sviluppare la propria applicazione con Flutter proprio perchè è un framework indicato come veloce, flessibile e produttivo. Esso offre:

- *Prestazioni quasi native.* Utilizzando il linguaggio di programmazione Dart e compilando il codice macchina, i dispositivi host riescono a comprendere meglio il codice assicurando, così, prestazioni pressochè ottime.
- *Rendering rapido, coerente ed affidabile.* Invece di affidarsi a strumenti specifici, Flutter utilizza la libreria grafica Skia open source di Google per il rendering delle IU.

Durante la modalità di debug, esso utilizza una macchina virtuale (VM) per eseguire il proprio codice al fine di abilitare il ricaricamento rapido con stato (Hot Reload), una funzionalità che consente di apportare modifiche al codice in esecuzione senza ricompilazione. Inoltre, in Flutter, 'Everything is a Widget', ovvero il Widget è l'elemento fondamentale di costruzione: componenti di interfaccia, di layout, ma anche l'app stessa sono Widget che contengono, a loro volta, Widget.

CAPITOLO 2

Specifica ed analisi dei requisiti

Dopo aver introdotto il mondo 'mobile', affronteremo, in questo capitolo, l'insieme delle operazioni che precedono le fasi di progettazione ed implementazione, non solo di un'app, ma di qualsiasi altro tipo di sistema informatico. Verrà fornita, infatti, una descrizione ad alto livello delle funzionalità dell'app, che verranno approfondite poi nell'analisi e nella specifica dei requisti.

2.1 Descrizione ad alto livello dell'applicazione

'Appdila' è ideata per aziende che si occupano della gestione di siti cimiteriali. Nasce con lo scopo di fornire, ai clienti di queste aziende, un modo semplice ed efficace di gestire i cari defunti, che possono essere situati in cimiteri differenti tra di loro. Permette di compiere tutte le operazioni di ordinaria amministrazione, che, per una maggior chiarezza, presentano una netta separazione all'interno dell'applicazione.

Ogni utente dovrà, innanzitutto, essere in grado di registrarsi, autenticarsi e visualizzare esclusivamente le informazioni a lui relative.

La registrazione di un nuovo utente avviene inserendo i propri dati anagrafici, richiesti in un apposito form di registrazione; il login avviene tramite email e password precedentemente utilizzati in fase di registrazione.

La navigazione tra le schermate dell'app sarà gestita da una barra di navigazione posizionata nella parte inferiore dello schermo.

L'utente avrà quindi modo di concentrarsi sulla gestione dei defunti, visualizzati come elenco, oppure sulla contabilità. Dall'elenco dei defunti sarà possibile visualizzare la scheda anagrafica; da qui, sarà quindi possibile richiedere una servizio aggiuntivo come un portafiori, la pulizia della lapide, oppure allacciare o distaccare la lampada votiva del defunto. Da questa sezione sarà, inoltre, possibile richiedere la presa in carico di un nuovo defunto.

Nella parte relativa alla contabilità si potrà, invece, visualizzare l'elenco fatture ricevute dall'utente, che si potrà filtrare tramite la sezione apposita.

2.1.1 Glossario dei termini

È utile, prima di procedere, per una maggior chiarezza e completezza, evidenziare alcuni termini che saranno utilizzati nei paragrafi e nei capitoli successivi.

Nella Tabella 2.1 vengono riportati i termini, il loro significato ed il loro sinonimi utilizzati.

Termine	Significato	Sinonimo
Cliente	Persona fisica o giuridica che usufruisce, anche tramite l'app, dei servizi messi a disposizione dalla società di gestione dei siti cimiteriali.	Utente
Incaricato	Cliente che è stato associato almeno ad un defunto.	Cliente, Utente
Allaccio	Operazione con la quale un utente comunica la propria volontà di accendere la lampada votiva di un defunto.	-
Distacco	Operazione con la quale un utente comunica la propria volontà di spegnere la lampada votiva di un defunto.	-
Presa in carico	Operazione mediante la quale un utente, nonchè cliente, comunica la propria volontà di farsi carico della gestione di un nuovo defunto; sarà, da ora in poi, un incaricato per quel defunto.	Richiesta di allaccio

Tabella 2.1: Tabella dei termini e del loro significato.

2.2 Funzionalità dell'app

Alla base dello sviluppo dell'applicazione, antecedentemente alla sua progettazione ed implementazione, si svolgono, la raccolta, la specifica e l'analisi dei requisiti che il sistema dovrà possedere.

Nel termine 'requisito' si ingloba tutto ciò che concerne le funzionalità, i servizi, le caratteristiche e le prestazioni offerte dall'applicazione.

Se volessimo puntualizzare e formalizzare quanto descritto nel paragrafo precedente, potremmo dire che l'applicazione deve soddisfare dei requisiti, che possiamo classificare in *requisiti funzionali* e *requisiti non funzionali*.

2.2.1 Requisiti funzionali

I requisiti funzionali descrivono le funzionalità dell'applicazione; essi indicano, quindi, come un sistema dovrà comportarsi.

'Appdila' deve garantire, per un utente, le seguenti funzionalità:

1. *Registrazione di un nuovo utente;*
2. *Reset della password di accesso;*
3. *Login tramite email e password;*
4. *Logout;*
5. *Invio di una segnalazione di guasto;*
6. *Visualizzazione della scheda di un defunto;*
7. *Allaccio o distacco di un defunto*;
8. *Richiesta di un servizio per un defunto a carico;*
9. *Richiesta di presa in carico di un nuovo defunto;*
10. *Visualizzazione dell'elenco delle fatture ricevute;*

11. *Filtraggio delle fatture per data e per pagamento.*

2.2.2 Requisiti non funzionali

L'insieme delle caratteristiche che riguardano, invece, le prestazioni del sistema, la disponibilità, la qualità o la scalabilità, prende il nome di requisiti non funzionali.

I requisiti non funzionali di 'Appdila' riguardano:

1. *Gratuità del sistema.* Si prevede che l'app venga messa a disposizione di aziende che la offriranno, a loro volta, come servizio ai clienti dei siti cimiteriali.
2. *Modalità di accesso*. Le funzionalità dell'app devono essere disponibili solo agli utenti registrati.
3. *Sicurezza*. L'app deve proteggere i dati degli utenti e impedire l'accesso non autorizzato.
4. *Interfaccia grafica*. Si prevede che l'applicazione venga utilizzata da una vasta categoria di utenti; l'interfaccia grafica deve, quindi, necessariamente essere di tipo user frindly.

2.3 Attori e casi d'uso

Dal momento che 'Appdila' è destinata ad aziende che si occupano della gestione e della manutenzione di siti cimiteriali, l'unico attore che individuiamo è il cliente, nonchè l'utente.

Dall'analisi dei requisiti del paragrafo precedente emergono le azioni che possono essere compiute dall'attore. Esse possono essere suddivise in tre aree, ciascuna delle quali individua un caso d'uso specifico: la *gestione dell'utente*, la *gestione dei defunti* e la *gestione della contabilità*.

2.3.1 Gestione dell'utente

La gestione dell'utente, come schematizzato in Figura 2.1, comprende:

- la possibilità di registrazione dell'utente mediante la compilazione dei campi richiesti nell'apposita sezione;
- la possibilità di reimpostare la password mediante l'invio di un link sulla propria e-mail di registrazione;
- le operazioni di login e logout dall'app;
- la possibilità di pubblicare delle segnalazioni sull'apposita bacheca.

2.3.2 Gestione dei defunti

Così come mostrato in Figura 2.2, la gestione dei defunti è articolata come segue:

- visualizzazione della scheda anagrafica di un defunto.
- richiesta di presa in carico di un defunto, che avviene tramite la compilazione di un form pre-impostato.
- richiesta di uno specifico servizio per un defunto per il quale si risulta incaricati.
- allaccio o distacco della lampada votiva di un defunto.

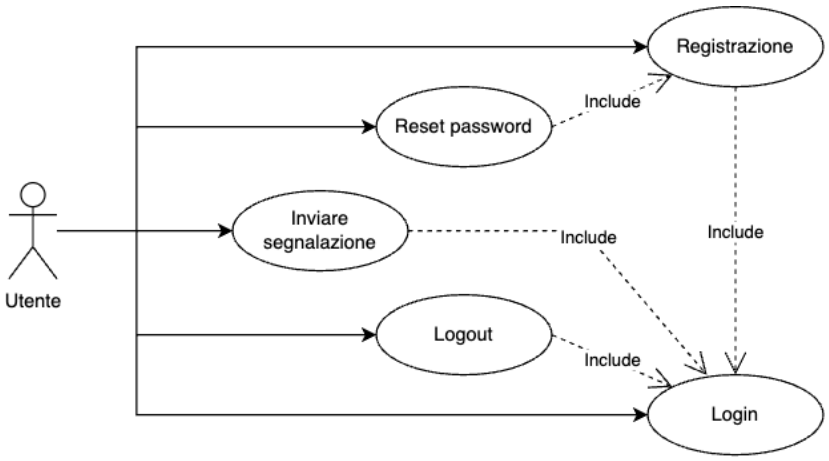


Figura 2.1: Caso d'uso: gestione dell'utente

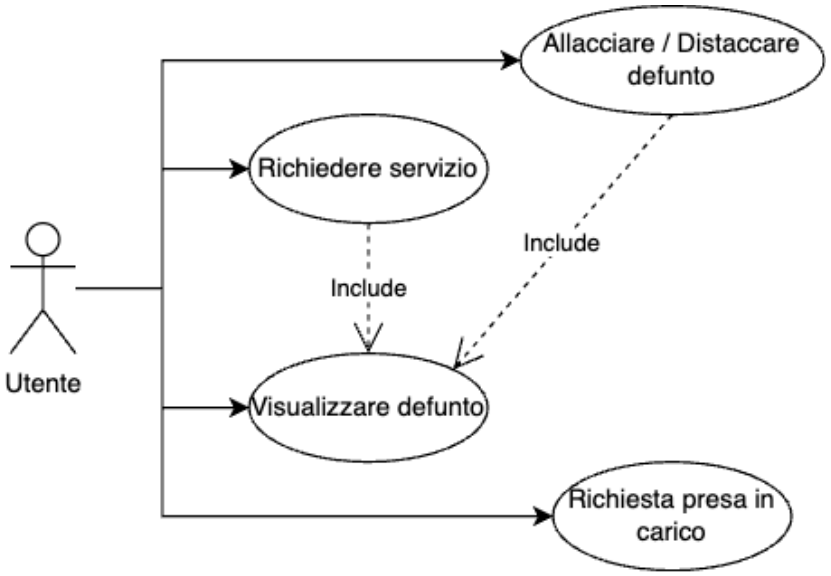


Figura 2.2: Caso d'uso: gestione dei defunti

2.3.3 Gestione della contabilità

La gestione della contabilità è, invece, riconducibile a due operazioni, che permettono sia di visualizzare l'elenco delle fatture ricevute dall'utente autenticato che di applicare all'elenco dei filtri sulla data di emissione delle stesse.

In Figura 2.3 troviamo una sua schematizzazione.

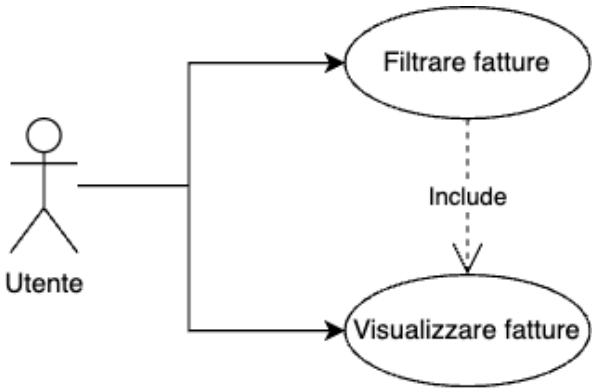


Figura 2.3: Caso d'uso: gestione della contabilità

CAPITOLO 3

Progettazione

In questo capitolo verranno affrontati tutti gli aspetti della progettazione delle funzionalità dell'applicazione delle quali seguirà l'implementazione in Flutter. Lo scopo di questa sezione è quello di definire l'architettura dell'app, la sua interfaccia e, infine, il criterio di archiviazione dei dati.

3.1 Progettazione della struttura dell'app

La fase di progettazione ricopre un ruolo fondamentale nello sviluppo di qualsiasi sistema informatico. Essa consiste non solo nella creazione di modelli del sistema, ma anche nella definizione dei dati e delle interfacce da utilizzare per implementarlo.

'Appdila' deve essere modulare, deve garantire una buona capacità di adattamento alle modifiche, ma anche alle migliorie future dovute all'evoluzione ed espansione del settore in cui sarà inserita.

Per rendere l'applicazione flessibile, è necessario far corrispondere, ad ogni sua funzionalità, un modulo; così facendo, sarà possibile aggiungere, modificare o rimuovere contenuti, agendo semplicemente sui moduli ad essi corrispondenti.

Nei paragrafi successivi saranno illustrati nel dettaglio i criteri di una corretta progettazione dell'applicazione, che saranno poi, decisivi nella fase di implementazione.

3.1.1 Mappa dell'applicazione

Per comprendere, da un punto di vista teorico, come strutturare l'applicazione, presentiamo di seguito la mappa concettuale che la descrive.

L'utilizzo di una mappa è utile, in generale, sia per avere una visione totale sulle funzionalità dell'applicazione, ma anche per comprendere il percorso, e la logica, da seguire, per far si che l'utente interagisca con le funzionalità che l'app mette a sua disposizione.

La mappa riportata in Figura 3.1, di tipo gerarchico, evidenzia una struttura lineare dell'applicazione, voluta proprio per garantire la sua semplicità di utilizzo.

Come possiamo notare, ciascuna funzionalità dell'applicazione è accessibile ai soli utenti che si sono, preventivamente, autenticati. Esaminando nel dettaglio i percorsi all'interno della mappa, possiamo fare delle considerazioni, che riportiamo qui di seguito.

- Dalla schermata Home è possibile visualizzare la *lista dei defunti*, la *contabilità*, oppure accedere all'area *utente*.

- Il dettaglio di un defunto è visualizzabile, solo ed esclusivamente, a partire dalla lista dei defunti; da qui, si possono, poi, eseguire le operazioni di allaccio o distacco del defunto, oltre alla richiesta di alcuni servizi per lo stesso.
- Per visualizzare una fattura è necessario accedere prima alla contabilità dell'utente.

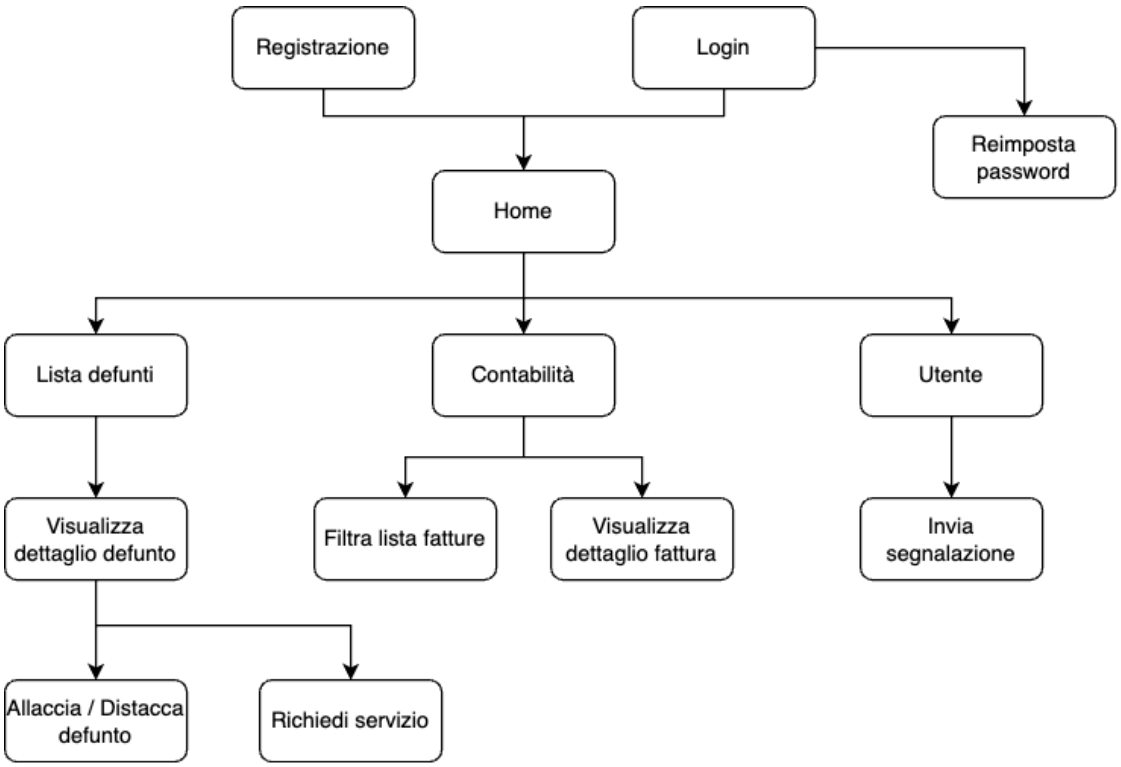


Figura 3.1: Mappa della struttura di Appdila

3.1.2 Organizzazione delle classi

Una volta stabilite, ad alto livello, quali saranno le funzionalità del sistema, è possibile compiere un ulteriore passo in avanti, definendo sia le classi da utilizzare per poterle implementare, che il modo in cui queste dovranno essere collegate tra di loro.

L'obiettivo di rendere l'applicazione modulare porta alla decisione di utilizzare:

- una classe *Model* che contenga gli attributi di un oggetto e che, quindi, funga da modello dei dati;
- una classe *Service* per tutte le funzioni che operano sui modelli dei dati;
- una classe *Screen* che ospiti la logica di visualizzazione dei dati, ovvero come questi devono essere organizzati nel layout delle schemrate.

La descrizione dettagliata delle classi, corredate dei relativi attributi, è fornita nei diagrammi illustrati a partire dalla Figura 3.2, fino alla Figura 3.5. Ci limitiamo, quindi, a fornire una breve descrizione di questi ultimi.

Il diagramma in Figura 3.2 mostra le classi utilizzate per la gestione dei defunti, ovvero:

- *Defunto*, che modella le informazioni relative ai defunti.

- *RichiestaAllaccio* e *Servizio* per modellare, rispettivamente, le informazioni relative alle richieste di allaccio da parte di un cliente, e quelle relative ai servizi richiesti per ciascun defunto.
- *DefuntoService*, *RichiestaAllaccioService* e *Servizioservice* che contengono, in ordine, i metodi di gestione dei defunti, delle richieste e dei servizi.
- *DefuntiScreen* che permette di visualizzare la lista dei defunti, e *DefuntoDetailScreen* che visualizza sullo schermo, invece, la scheda anagrafica di un defunto.

La gestione della contabilità, così come illustrato in Figura 3.3, necessita delle seguenti classi:

- *Fattura*, per modellare i dati relativi alle fatture emesse dalla società di gestione dei cimiteri.
- *FatturaService*, che contiene metodi relativi alla gestione delle fatture.
- *FattureScreen* e *FatturaDetailScreen* utilizzate, rispettivamente, per la visualizzazione dell'elenco delle fatture, con metodi di filtraggio, e per la visualizzazione del dettaglio di una fattura.

Per quanto riguarda la sezione relativa alle segnalazioni, in Figura 3.4 possiamo distinguere le seguenti classi:

- *Segnalazione*, per modellare le informazioni relative alle segnalazioni inviate da un utente.
- *SegnalazioneService*, per tutte le funzioni utili ad inserire e visualizzare le segnalazioni.
- *UtenteScreen*, necessaria per la visualizzazione sia delle informaizoni dell'utente che delle segnalazioni inviate.
- *UtenteService*, con tutte le funzioni di gestione del cliente.

L'ultimo diagramma, riportato in Figura 3.5, mostra, infine, come articolare la gestione dell'autenticazione da parte dell'utente. Per far ciò sono necessarie le seguenti classi:

- *Utente*, per modellare le informazioni relative agli utenti dell'app.
- *AuthService*, per contenere i metodi di accesso, registrazione, reimpostazione della password e uscita dall'applicazione.
- *LoginScreen*, *SignUpScreen* e *ResetScreen*, per visualizzare, rispettivamente, la schermata di accesso, di registrazione e quella per reimpostare la password.
- *HomeScreen*, per visualizzare la schermata home dell'applicazione una volta che viene eseguito l'accesso.

3.2 Design dell'interfaccia utente

Dopo aver progettato lo scheletro della struttura dell'app, bisogna studiare, ora, il suo aspetto, ovvero, il modo più vicino possibile al risultato finale, con cui le funzionalità dell'applicazione vengono esposte all'utente, in termini interfaccia grafica.

Il design dell'applicazione si avvale dell'utilizzo di strumenti software che permettono la realizzazione di mockup [1]. Un mockup altro non è che una rappresentazione di qualsiasi tipo di prodotto informatico, nel nostro caso dell'applicazione, con scopo illustrativo.

Mediante l'utilizzo di questi prototipi grafici è possibile comprendere il grado di usabilità del sistema, ma anche testare le componenti grafiche da utilizzare e verificare come esse interagiscono tra di loro.

A partire dalla Figura 3.6, fino alla Figura 3.12, vengono presentati i mockup di 'Appdila'. Dalla Figura 3.6 alla 3.8, sono mostrate, in ordine, la schermata di accesso all'app, quella di registrazione e quella di reimpostazione della password. Le successive Figure, dalla 3.9 alla 3.12, riguardano, invece, in ordine: la sezione relativa alla lista dei defunti, la scheda anagrafica del defunto, la sezione relativa alla contabilità e quella relativa all'area dell'utente da cui inviare segnalazioni.

3.3 Progettazione dell'archiviazione dei dati

Per sviluppare applicazioni robuste, ma anche di qualità, c'è bisogno di utilizzare, nel Backend, delle piattaforme, che non solo abbiano dei validi servizi, ma che siano soprattutto affidabili.

Firebase è una piattaforma di sviluppo di app mobili gestita da Google che possiede tutto ciò di cui si possa aver bisogno: è in grado di integrarsi facilmente con la maggior parte dei servizi mobili, offrendo un'ampia gamma di funzionalità e strumenti utili alla loro gestione.

Per l'archiviazione dei dati, utilizzati e generati da 'Appdila', ci serviremo di un servizio di Firebase, ovvero Cloud Firestore. La registrazione dell'utente all'app utilizzerà, invece, Firebase Authentication, un altro servizio Firebase che permetterà all'utente di registrarsi ed autenticarsi mediante email e password.

Cloud Firestore è un database NoSQL, flessibile e scalabile, che mantiene sincronizzati i dati tra tutte le app, mediante l'utilizzo di listener in tempo reale, offrendo, però, anche supporto offline.

Sfruttando il modello di Cloud Firestore, i dati vengono memorizzati in documenti che contengono, a loro volta, dei valori associati a dei campi. I documenti sono organizzati in raccolte, ovvero dei contenitori che permettono sia di raggrupparli, che di eseguire delle query per estrapolare, efficientemente, informazioni da essi.

3.3.1 Strutturazione delle raccolte Firestore

Seguendo le linee guida riportate nella documentazione Firebase, è possibile stabilire come salvare, al meglio, i dati di 'Appdila'. L'archiviazione dei dati sarà articolata in modo da utilizzare una raccolta per ogni modello illustrato nel Paragrafo 3.1.2. In particolare, avremo:

- Una raccolta *cliente*, per le informazioni relative agli utenti che effettuano la registrazione all'app. Sarà necessario salvare: *nome, cognome, codice fiscale, indirizzo, e-mail e photoUrl*. Per avere una corrispondenza tra l'utente in Firebase Authentication ed il documento Firestore corrispondente, è importante fare in modo che l'ID del documento del cliente corrisponda all'UID dell'utente registrato.

- Una raccolta *defunto*, per contenere le informazioni relative ai defunti, ovvero: *id, nome, cognome, data di nascita, data del decesso, luogo di nascita, cimitero, stato, incaricato ed imgProfilo*. Il campo *incaricato* è un riferimento al cliente che prende in carico il defunto; può quindi essere nullo. Lo stato del defunto è, invece, un valore booleano che indica se il defunto, in quel momento, è allacciato (true) oppure distaccato (false).

[1] `www.moqups.com`

- Una raccolta *servizio*, con tutti i servizi che sono stati richiesti per ciascun cliente. Ciascun servizio è rappresentato da: *tipologia, costo, defunto per il quale viene richiesto e stato*. Il campo 'stato' può assumere due valori, ovvero, Richiesto oppure Aggiunto.

- Una raccolta *fattura*, contenente tutte le informazioni sulle fatture che sono state emesse dalla società di gestione. Ciascuna fattura è caratterizzata da: *numero, data di emissione, data di scadenza, data di pagamento, defunto, cliente, descrizione e totale*. Il campo relativo ai defunti è un array di stringhe, ovvero un array che contiene gli identificativi dei defunti per il quale è stata emessa la fattura.

- Una raccolta *richiesta*, per contenere le nuove richieste di allaccio da parte di un cliente, per un dato defunto. Ogni richiesta è formata da: *nome e cognome del defunto, data di nascita e data del decesso del defunto, cimitero e stato*. Lo 'stato' indica se la richiesta di allaccio è già stata processata (true) oppure no (false).

- Una raccolta *segnalazione*, per le segnalazioni che sono state inviate dai clienti. Ciascuna segnalazione è identificata da: *utente, data, messaggio e cimitero*. Anche in questo caso, il campo utente risulta essere un riferimento all'utente che invia la segnalazione.

Bisogna, infine, sottolineare che ciascun campo di un documento, che viene utilizzato come riferimento ad un altro documento, contiene una stringa; questa altro non è che l'ID corrispondente al secondo documento al quale ci si vuole riferire.

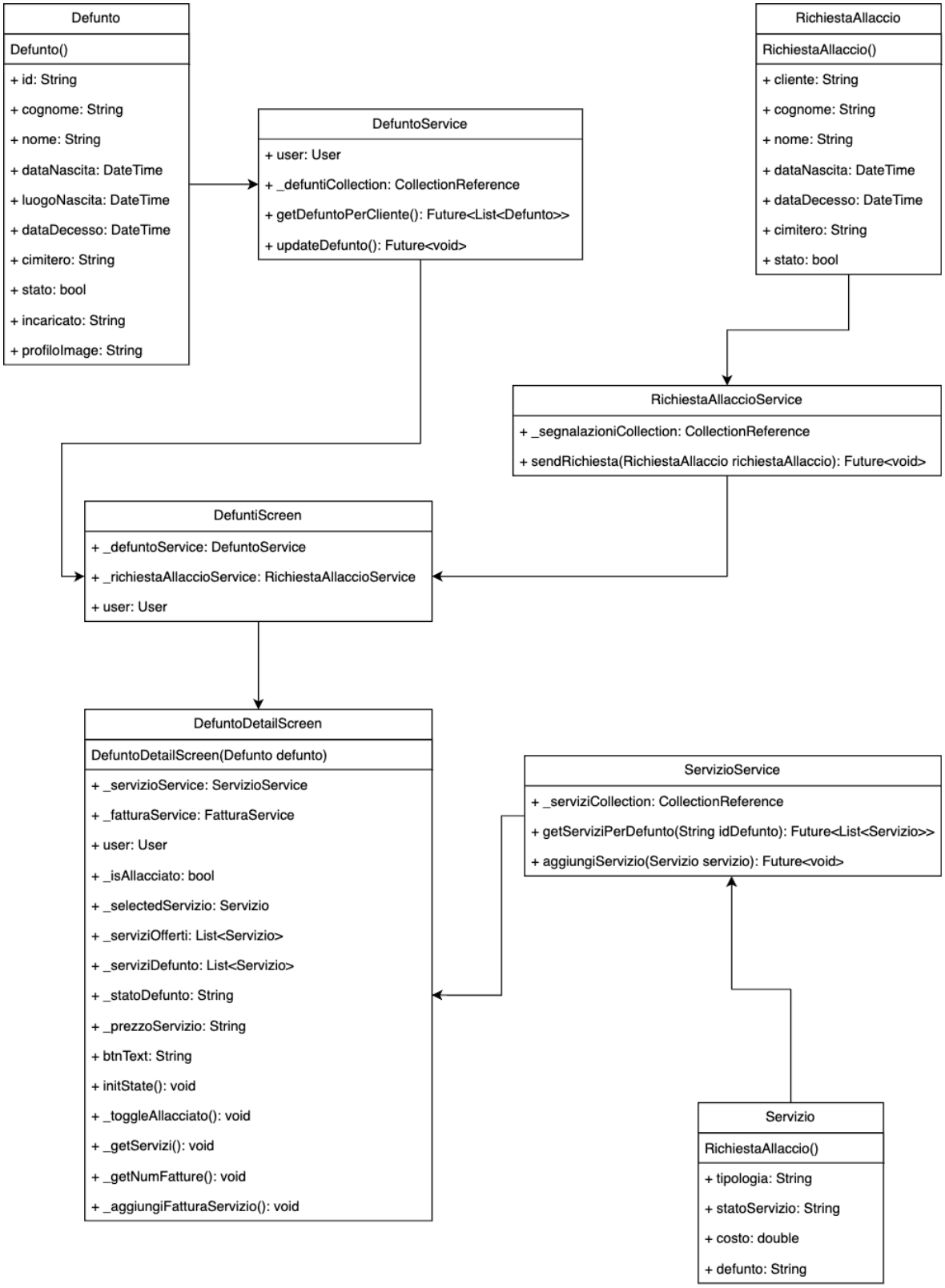


Figura 3.2: Diagramma delle classi relative alla gestione dei defunti

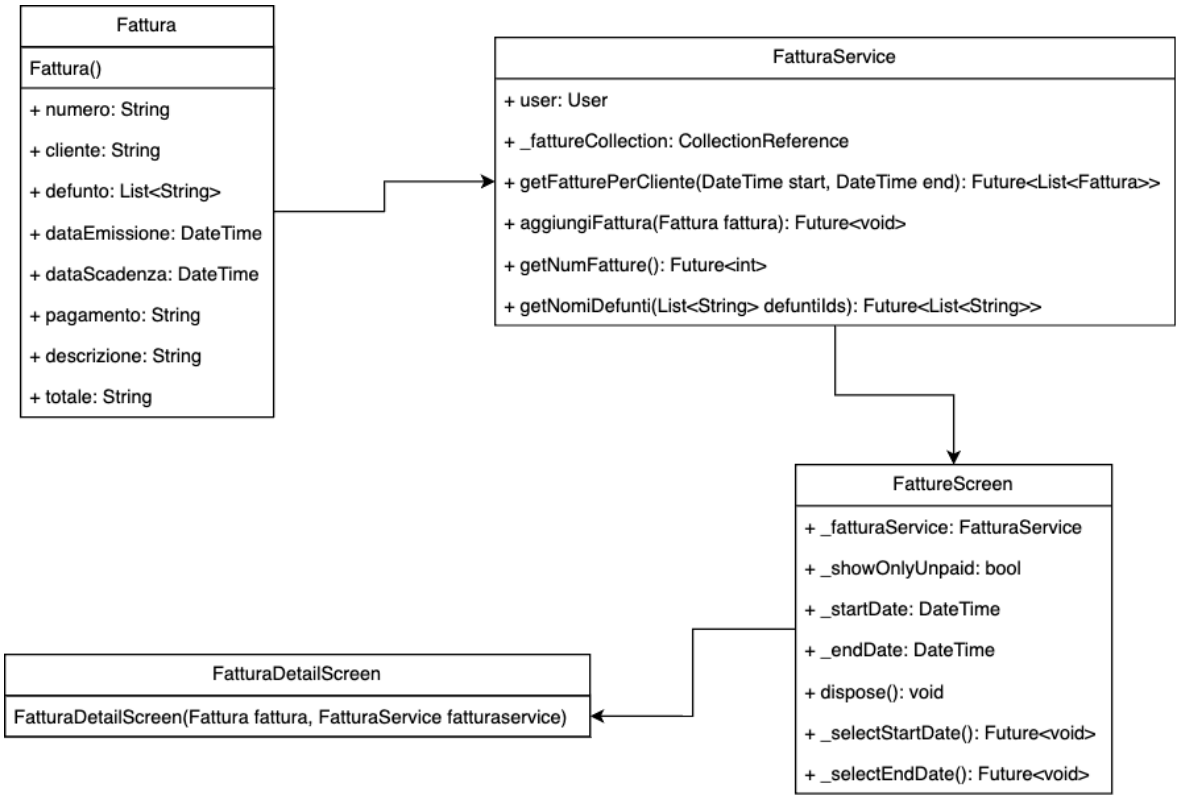


Figura 3.3: Diagramma delle classi utilizzate per la gestione della contabilità.

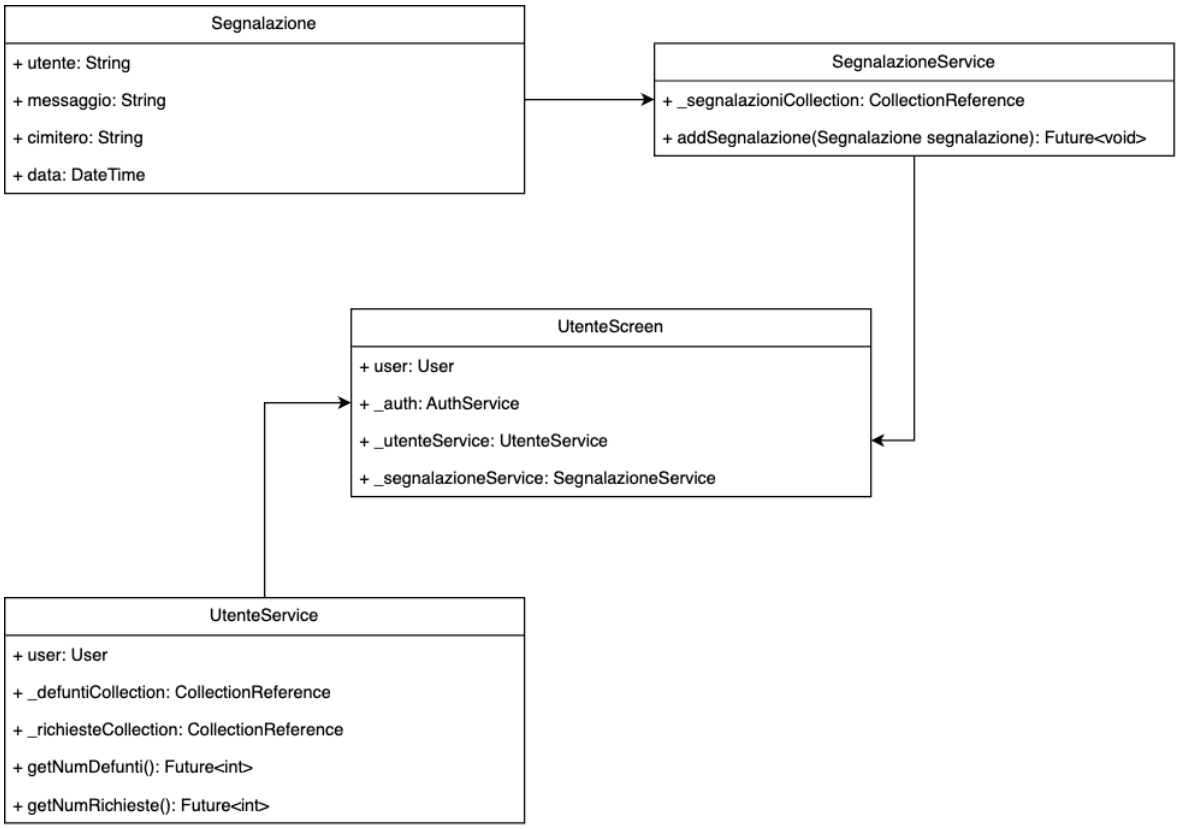


Figura 3.4: Diagramma delle classi relative alle segnalazioni.

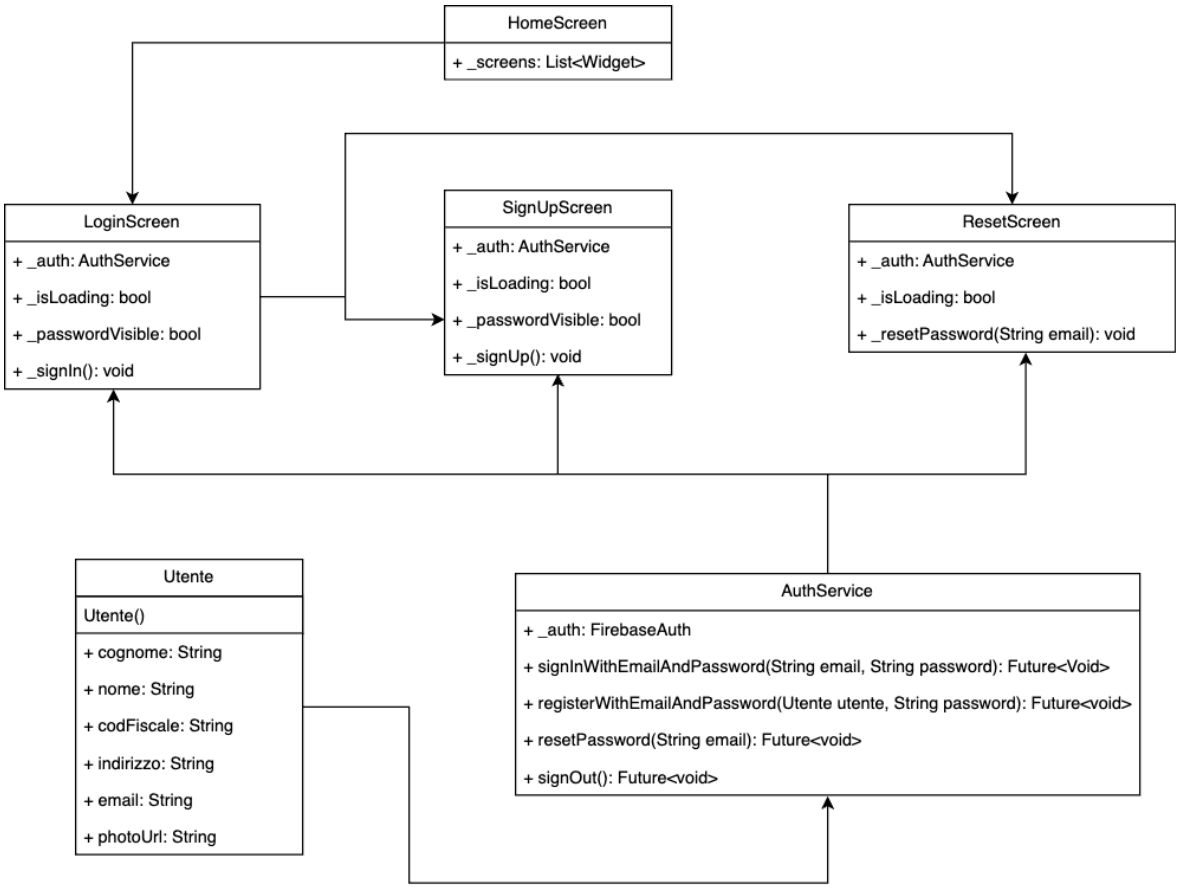


Figura 3.5: Diagramma delle classi utilizzate per l'autenticazione dell'utente

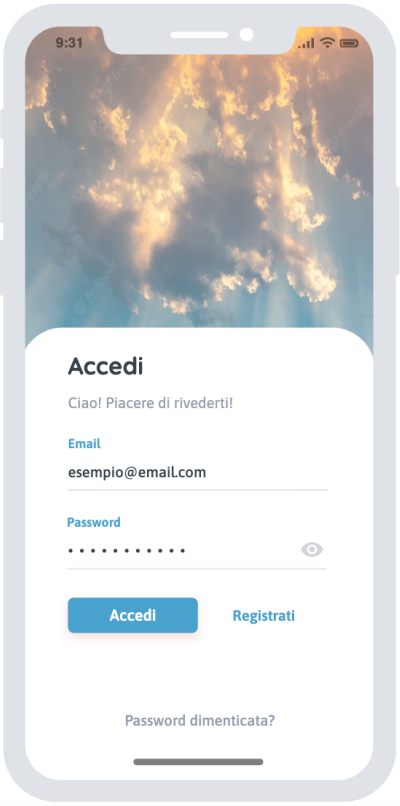


Figura 3.6: Mockup relativo alla schermata di login

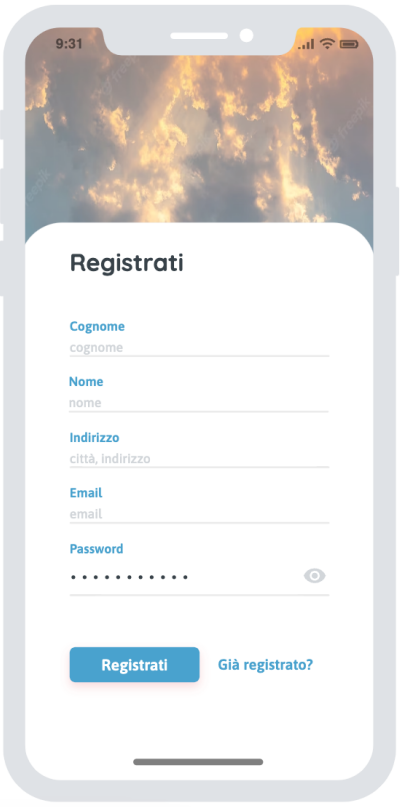


Figura 3.7: Mockup relativo alla schermata di registrazione

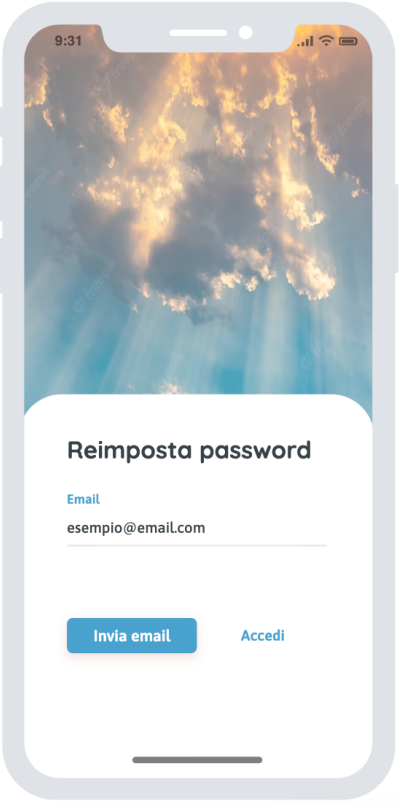


Figura 3.8: Mockup relativo alla schermata di reset della password

Figura 3.9: Mockup relativo alla schermata di visualizzazione della lista dei defunti

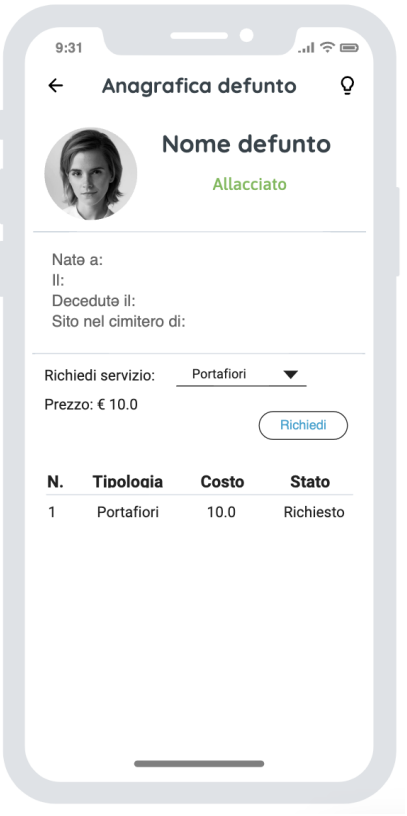


Figura 3.10: Mockup relativo alla schermata di dettaglio del defunto

Figura 3.11: Mockup relativo alla schermata di visualizzazione della lista delle fatture

Figura 3.12: Mockup relativo alla schermata a disposizione dell'utente per l'invio di delle segnalazioni

CAPITOLO 4

Implementazione

Questo capitolo è dedicato all'esposizione ed alla descrizione di alcuni degli aspetti implementativi che sono stati definiti e discussi nella fase di progettazione del capitolo precedente. Si tratta dell'ultimo dei passaggi da affrontare per ottenere un programma eseguibile, funzionante, scritto nel linguaggio di programmazione che meglio si adatta alle esigenze del caso. Si fornirà un breve accenno sugli strumenti necessari per sviluppare software, concludendo, poi, con la spiegazione di una parte del codice di 'Appdila'.

4.1 Strumenti utilizzati per lo sviluppo

Così come per lo sviluppo di un qualsiasi sistema eseguibile, anche la scrittura, ovvero la programmazione, di un'applicazione mobile, necessita dell'utilizzo di un editor di testo, oppure di un IDE[1].

Gli IDE sono dei software, appositamente progettati, che permettono di realizzare, a loro volta, dei sistemi software, e che inglobano al proprio interno specifici strumenti, come editor di testo, automatizzatori della creazione di build, debugger o, in alcuni casi, emulatori.

Per l'implementazione, in Flutter, di 'Appdila', si utilizzerà Android Studio. Si tratta di un ambiente di sviluppo tramite il quale è possibile creare non solo applicazioni che si basano sul sistema operativo Android, ma anche quelle multipiattaforma (Sezione 1.3.2); Android Studio è considerato il tool ufficiale per la realizzazione di app native Android.

Flutter può essere facilmente integrato in Android Studio seguendo i pochi passaggi che sono descritti, in modo completo, nell'apposita documentazione. Dedicheremo, a questo punto, il paragrafo successivo alla configurazione dell'ambiente di sviluppo, facendo riferimento al sistema operativo MacOS, predisponendolo, così, alla creazione dell'app in Flutter.

4.1.1 Configurazione di Android Studio

Il primo passo da compiere è installare l'SDK Flutter, scaricando la versione adatta al proprio sistema operativo dal sito ufficiale di Flutter [2].

Una volta terminata la procedura di configurazione indicata nella pagina stessa del download dell'SDK, si deve sia scaricare, dal sito di Developer Android [3], che installare, Android Studio, seguendo le indicazioni mostrate dall'IDE stesso.

[1] Integrated Development Environment, o ambiente di sviluppo integrato.

[2] `https://docs.flutter.dev/get-started/install`

[3] `https://developer.android.com/studio`

Completati i passaggi di installazione, ci troveremo, una volta avviato Android Studio, di fronte ad una schermata dello stesso tipo di quella riportata in Figura 4.1.

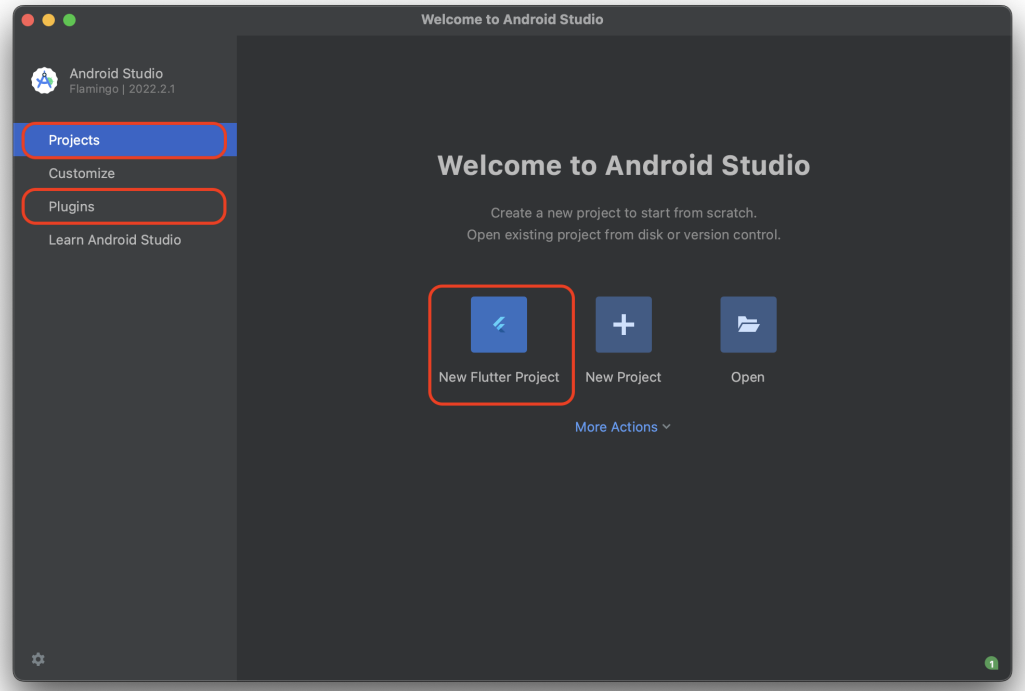


Figura 4.1: Schermata di Android Studio che viene visualizzata alla sua apertura

Prima di poter procedere con la creazione di un nuovo progetto, però, sarà necessario installare i plugin sia Flutter che Dart su Android Studio. Facendo sempre riferimento alla Figura 4.1, basterà spostarsi nella sezione *Plugins* situata nella parte sinistra, ricercare il nome di entrambi i plugin e cliccare sul pulsante *install*.

Terminata l'installazione dei plugin, si potrà tornare alla sezione *Projects* e creare un *New Flutter Project*. Si aprirà, a questo punto, una nuova finestra, del tipo mostrato in Figura 4.2.

Qui sarà necessario, come prima cosa, indicare, nel campo *Flutter SDK path*, il percorso assoluto della directory che contiene l'SDK Flutter scaricato precedentemente; completato questo passaggio, bisognerà cliccare su *next*, impostare il nome 'Appdila' nel campo relativo al *Project name* e cliccare, infine, su *create*.

Seguiti questi semplici passaggi, avremo il nostro ambiente di sviluppo configurato, e, quindi, pronto per la creazione della nostra app.

4.1.2 Integrazione di Firebase nel progetto

Una volta creato il progetto, è necessario integrare, al suo interno, il servizio Firebase, necessario ai fini dell'archiviazione dei dati dell'applicazione.

Si deve, prima di tutto, accedere al sito ufficiale di Firebase [4], recarsi nella sezione della console e creare un nuovo progetto su *Aggiungi progetto*, indicando sia il nome del progetto che l'account con cui esso dovrà essere creato.

Una volta creato il progetto, si potrà aggiungere, al suo interno, nella parte superiore della schermata visualizzata, una nuova applicazione. Come evidenziato in Figura 4.3, è sufficiente

[4] `https://firebase.google.com/`

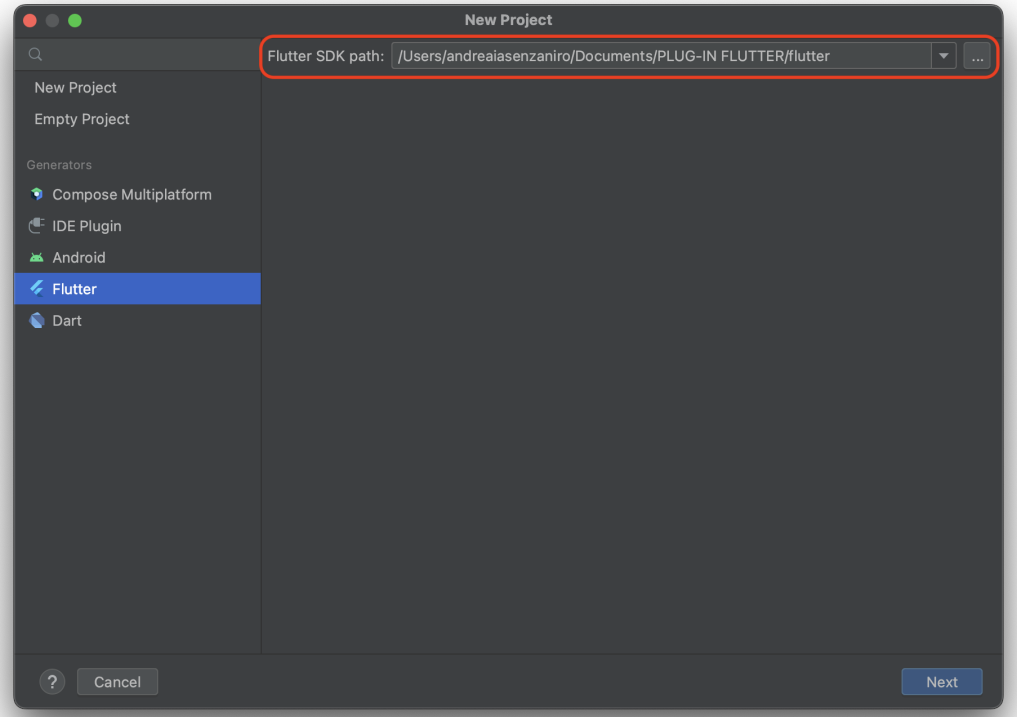


Figura 4.2: Schermata di Android Studio per la creazione di nuovo progetto Flutter

cliccare sull'icona '+' e selezionare il tipo di applicazione da aggiungere, Flutter in questo caso.

Adesso, aprendo la riga di comando, e seguendo le indicazioni della documentazione Firebase, saremo in grado di aggiungere le dipendenze alla nostra applicazione Flutter.

Una volta creato il progetto, e dopo aver correttamente configurato Firebase al suo interno, potremo procedere con l'implementazione delle classi che sono state oggetto di descrizione nella Sezione 3.1.2.

4.2 Organizzazione del codice

Sfruttando le informazioni descritte in fase di Progettazione, è possibile derivare il modo con cui strutturare il codice dell'applicazione in *file* con estensione `.dart`.

Per mantenere la stessa separazione logica indicata nella Sezione 3.1.2, è utile articolare il progetto in directory, ognuna delle quali adibita a contenere i file relativi ad un certo tipo di classe.

Ogni classe corrisponde ad un file, ed ogni file è contenuto in una directory, come mostrato in Figura 4.4, a seconda della tipologia di classe contenuta.

La directory utilizzate sono le seguenti:

- `data`, per contenere tutti i file che corrispondono alle classi `Model` dei dati dell'app.
- `screens`, che contiene delle sotto-directory; queste, a loro volta, sono suddivise per tipologia di funzionalità offerta, e contengono i file relativi alle classi di visualizzazione delle schermate Flutter.

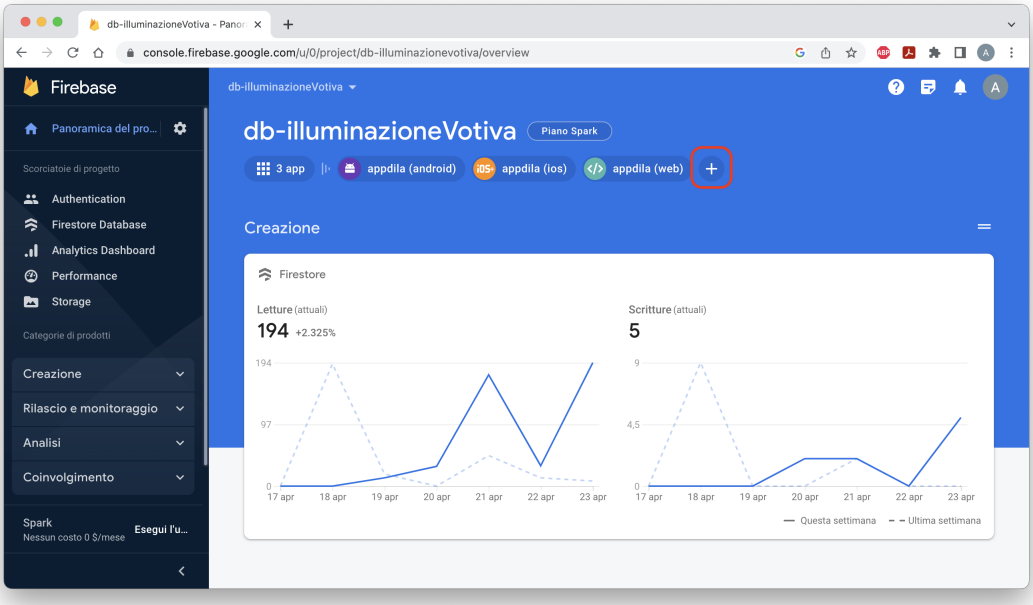


Figura 4.3: Schermata iniziale del progetto Firebase denominato *db-illuminazioneVotiva*

- `services`, per i file delle classi di tipo `Service`.

Ogni directory è contenuta, a sua volta, nella cartella `lib`, adibita ad ospitare il codice dell'applicazione. Al suo interno possiamo distinguere, oltre quanto appena elencato, un file `main.dart`, ovvero quello contenente la funzione `main()` di avvio dell'applicazione della quale verrà fornita una descrizione nel capitolo successivo.

4.3 Alcuni aspetti implementativi

Per meglio comprendere la logica implementativa dell'applicazione, vale la pena prendere in esame alcune delle caratteristiche del codice della stessa.

Per raggiungere una soddisfacente chiarezza espositiva, la sezione è articolata in sottosezioni, ciascuna delle quali dedicata alla descrizione di una specifica funzione dell'app.

4.3.1 Avvio dell'applicazione

La Figura 4.5 riporta un segmento di codice del file `main.dart`, all'interno del quale possiamo distinguere due componenti: una classe `MyApp` ed una funzione `main()`.

La funzione `main()`, asincrona, inizializza l'app tramite il metodo `Firebase.initializeApp()`, recupera poi l'utente corrente, mediante `FirebaseAuth.instance.currentUser`, ed esegue, infine, una verifica su di esso; se l'utente è autenticato, viene mostrata la schermata iniziale dell'app, *HomeScreen*, altrimenti, viene visualizzato quella di accesso, *LoginScreen*, per consentire l'autenticazione di un nuovo utente.

La classe `MyApp` è uno StatelessWidget; essa definisce sia la struttura principale dell'applicazione, utilizzando un `MaterialAppwidgetper` per impostare il titolo e la schermata iniziale tramite il parametro `_defaultHome`, che delle rotte, per consentire la navigazione tra le schermate, a partire da quella di login.

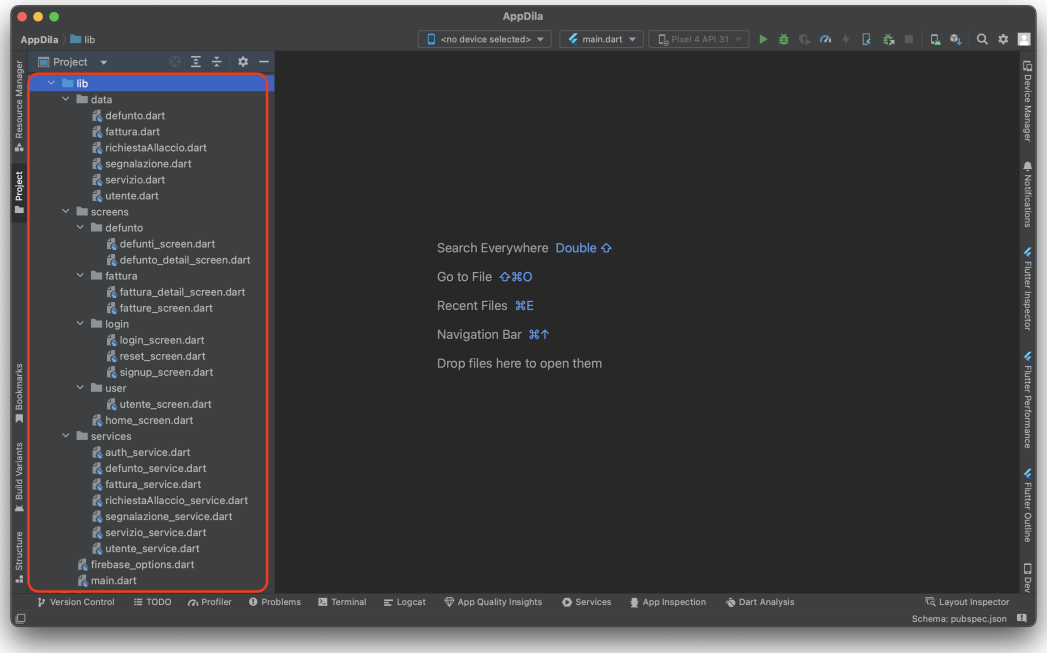


Figura 4.4: Organizzazione delle classi di 'Appdila' in Android Studio

4.3.2 Registrazione dell'utente su Firebase Auth

Le registrazione di un nuovo utente è un'operazione articolata in due passaggi, che sfrutta due metodi distinti, ovvero `_signUp()` della classe `SignUpScreen` e `registerWithEmailAndPassword()` della classe `AuthService`.

La funzione `_signUp()`, mostrata in Figura 4.6, è di tipo asincrono e viene invocata quando l'utente clicca sul pulsante 'Registrati' della schermata di registrazione. Questo metodo richiama, al suo interno, la funzione `registerWithEmailAndPassword()`; prima di far ciò, però, verifica che la password inserita dall'utente corrisponda alla sua ripetizione. Se le due password coincidono, allora il metodo crea un oggetto *Utente* con le informazioni inserite nei campi dell'interfaccia e lo passa come argomento al secondo metodo, unitamente alla password; altrimenti, se le password non corrispondono, viene mostrata una notifica di errore.

La Figura 4.7, riporta, invece, il secondo dei metodi di registrazione. Come già anticipato, essa accetta due argomenti in ingresso, ovvero un oggetto di tipo *Utente* per archiviare le informazioni relative all'utente da registrare ed una stringa *Password* per impostare la password di accesso all'account utente.

La funzione verifica, come prima cosa, che i campi inseriti per la registrazione dell'utente non siano nulli; successivamente, controlla il formato dell'email inserita: se il formato non è valido viene lanciata un'eccezione di tipo `FirebaseAuthException`, altrimenti viene chiamato il metodo `createUserWithEmailAndPassword()` per creare un nuovo account Firebase Authentication.

Terminati i controlli sulle informazioni immesse dall'utente, se la creazione dell'account va a buon fine, la funzione recupera l'UID dell'utente appena creato e lo utilizza per salvare le sue informazioni in un file nella collection 'cliente' di Firestore, il cui ID corrisponde all'UID dell'utente, altrimenti, genera un'eccezione di tipo `FirebaseAuthException`, annullando la creazione del file per l'utente.

```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();

  final FirebaseAuth _auth = FirebaseAuth.instance;
  // verifico che l'utente sia autenticato prima di inizializzare l'app
  User? user = _auth.currentUser;
  // se già autenticato allora mostro la HomeScreen
  Widget _defaultHome = user == null ? LoginScreen() : HomeScreen();

  runApp(MyApp(_defaultHome));
}

class MyApp extends StatelessWidget {
  final Widget _defaultHome;

  MyApp(this._defaultHome);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'My App',
      home: _defaultHome,
      routes: {
        '/login': (context) => LoginScreen(),
        '/home': (context) => HomeScreen(),
        '/register': (context) => SignUpScreen(),
        '/reset': (context) => ResetScreen(),
      },
    );  // MaterialApp
  }
}
```

Figura 4.5: Segmento di codice del file `main.dart`

4.3.3 Richiesta di un servizio

Tra le principali funzionalità offerte dall'applicazione, vi è la possibilità, da parte di un utente, di richiedere un servizio per uno dei defunti a proprio carico.

La richiesta di un servizio, da un punto di vista implementativo, si suddivide in più passaggi, ed utilizza le seguenti componenti: un `DropdownButton` nella classe `DefuntoDetailScreen`, che permette all'utente di selezionare uno dei servizi, una variabile `_selectedServizio` di tipo `Servizio`, una funzione `aggiungiServizio()` della classe `ServizioService` ed una funzione `_aggiungiFatturaServizio()` della classe `DefuntoDetailScreen`.

La variabile `_selectedServizio` memorizza le informazioni del servizio selezionato dall'utente. Al momento dell'istanziazione della classe `DefuntoDetailsScreen`, il metodo `initState()` imposta `_selectedServizio` con i valori del primo dei servizi offerti dalla società di gestione del sito cimiteriale.

Il `DropdownButton`, mostrato in Figura 4.8, riceve come valore una lista di oggetti di tipo `Servizio` e ne mostra il nome in un menù a discesa. La funzione di callback `onChanged` viene utilizzata per aggiornare sia la variabile `_selectedServizio` con il valore selezionato dall'utente che la variabile `_prezzoServizio` per mostrare nello schermo il prezzo del servizio selezionato.

Una volta impostato il valore del servizio corrente, la funzione di Figura 4.9 mostra il codice di abilitazione del click del pulsante di richiesta del servizio. Tramite la funzione di callback `onPressed` è possibile invocare prima `aggiungiServizio()`, poi `_aggiungiFatturaServizio()`, ed infine, aggiornare la lista dei servizi del defunto.

La funzione `aggiungiServizio()`, mostrata in Figura 4.10, asincrona, riceve come parametro un oggetto di tipo `Servizio`, in questo caso quello selezionato, e, utilizzando la libreria Firebase, crea un nuovo documento nella raccolta `servizio`, assegnando ai suoi campi i valori corrispondenti a quelli del servizio passato come argomento.

La Figura 4.11 mostra, invece, il metodo `_aggiungiFatturaServizio()`. Si tratta, anche in questo caso, di un metodo asincrono che crea un oggetto di tipo `Fattura` impostando i campi con i valori del servizio selezionato, richiama la funzione della libreria Firebase

```dart
void _signUp() async {
  setState(() {
    _isLoading = true;
  });
  if(_passwordController.text == _ripetiPasswordController.text){
    try{
      Utente utente = Utente(
          cognome: _cognomeController.text,
          nome: _nomeController.text,
          codFiscale: _codFiscaleController.text,
          indirizzo: _indirizzoController.text,
          email: _emailController.text,
          photoUrl: '');
      await _auth.registerWithEmailAndPassword( utente, _passwordController.text);
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Registrazione avvenuta con successo')),);
      Navigator.pop(context);
      Navigator.pushReplacementNamed(context, '/home');
    }on FirebaseAuthException catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Assicurati che l\'indirizzo email sia corretto')),);
    }
  }else{
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Le password inserite non corrispondono')),);
  }
  setState(() {
    _isLoading = false;
  });
}
```

Figura 4.6: Funzione di registrazione di un nuovo utente della classe `SignUpScreen` nel file `signup_screen.dart`

che permette di aggiungere il file della fattura nella raccolta `fattura` Firestore ed imposta opportunamente i campi del documento creato.

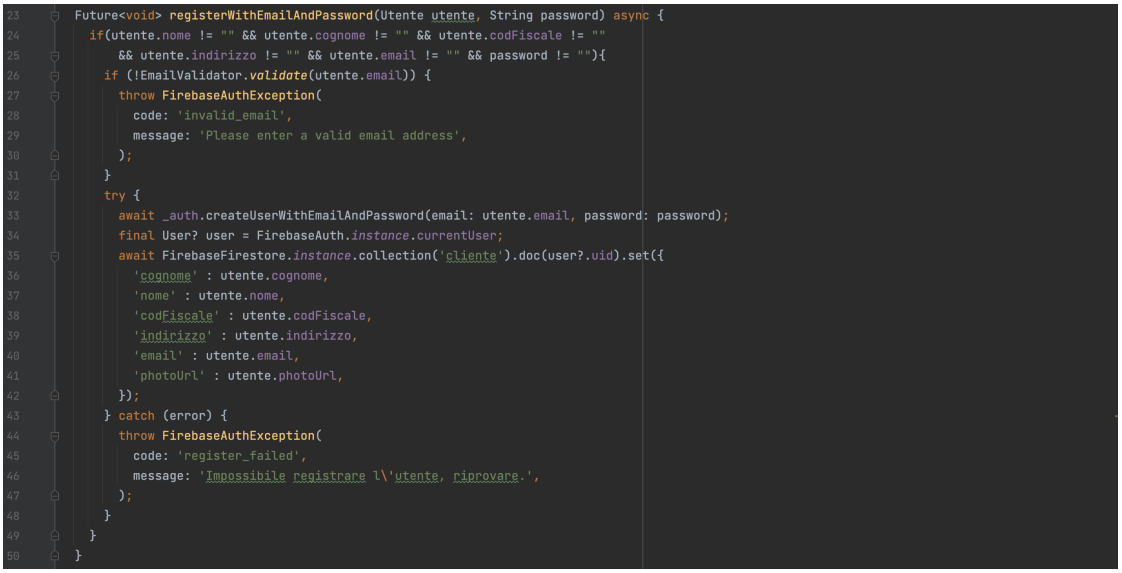
4.3.4 Filtraggio delle fatture

Un'ulteriore funzionalità rilevante, da prendere in esame, è quella che permette all'utente di applicare dei filtri alla lista delle fatture visualizzate nella sezione della contabilità. La schermata di visualizzazione dell'elenco delle fatture, come si può vedere dalla Figura 4.12, utilizza un `FutureBuilder` con due parametri, ovvero un `future` che viene impostato sul metodo `getFatturePerCliente()` della classe `FatturaService`, per recuperare la lista delle fatture, ed un `builder`, ovvero una funzione di callback che si occupa di visualizzare sullo schermo la lista delle fatture recuperate.

I dati recuperati dalla funzione di callback vengono prima filtrati in base al parametro booleano `_showOnlyUnpaid`, che determina se visualizzare tutte le fatture oppure solo quelle che non sono state pagate; i dati filtrati vengono, poi, visualizzati in un `ListView` in appositi widget.

La Figura 4.13 mostra l'implementazione del metodo `getFatturePerCliente()` che restituisce l'elenco di fatture di un dato cliente, ordinandolo per data di emissione.

La funzione accetta in ingresso due parametri, opzionali, che indicano la data di inizio e di fine del periodo entro il quale ricercare le fatture del cliente. I documenti recuperati da Firestore, dopo un'appropriata operazione di parsing delle date, vengono aggiunti ad una lista di elementi di tipo `Fattura`, alla quale sono applicati, poi, i filtri delle date indicate come argomento della funzione. Se la data di emissione della fattura rientra nel periodo di tempo selezionato, allora la fattura viene aggiunta ad una nuova lista filtrata, che sarà poi restituita dalla funzione; se , invece, non viene indicato l'arco temporale da filtrare, vengono restituite tutte le fatture del cliente.

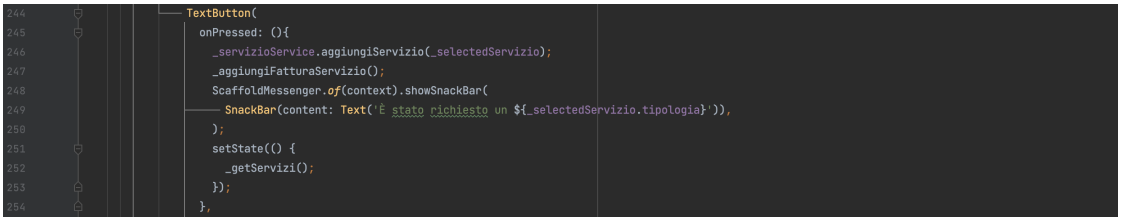


```
Future<void> registerWithEmailAndPassword(Utente utente, String password) async {
  if(utente.nome != "" && utente.cognome != "" && utente.codFiscale != ""
      && utente.indirizzo != "" && utente.email != "" && password != ""){
    if (!EmailValidator.validate(utente.email)) {
      throw FirebaseAuthException(
        code: 'invalid_email',
        message: 'Please enter a valid email address',
      );
    }
    try {
      await _auth.createUserWithEmailAndPassword(email: utente.email, password: password);
      final User? user = FirebaseAuth.instance.currentUser;
      await FirebaseFirestore.instance.collection('cliente').doc(user?.uid).set({
        'cognome' : utente.cognome,
        'nome' : utente.nome,
        'codFiscale' : utente.codFiscale,
        'indirizzo' : utente.indirizzo,
        'email' : utente.email,
        'photoUrl' : utente.photoUrl,
      });
    } catch (error) {
      throw FirebaseAuthException(
        code: 'register_failed',
        message: 'Impossibile registrare l\'utente, riprovare.',
      );
    }
  }
}
```

Figura 4.7: Funzione di registrazione di un nuovo utente della classe `AuthService` nel file `auth_service.dart`

```
DropdownButton<Servizio>(
  value: _selectedServizio,
  hint: Text('Selezionare'),
  items: _serviziOfferti.map((Servizio value) {
    return DropdownMenuItem<Servizio>(
      value: value,
      child: Text(value.tipologia),
    );  // DropdownMenuItem
  }).toList(),
  onChanged: (value){
    setState(() {
      _selectedServizio = value!;
      _prezzoServizio = "€ ${_selectedServizio.costo.toString()}";
    });
  },
),  // DropdownButton
```

Figura 4.8: Codice che implementa il menù a discesa che permette di selezionare un servizio nella classe `DefuntoDetailScreen` nel fine `defunto_detail_screen.dart`



```
TextButton(
  onPressed: (){
    _servizioService.aggiungiServizio(_selectedServizio);
    _aggiungiFatturaServizio();
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('È stato richiesto un ${_selectedServizio.tipologia}')),
    );
    setState(() {
      _getServizi();
    });
  },
```

Figura 4.9: Codice di implementazione del clic del pulsante di richiesta di un servizio della classe `DefuntoDetailScreen` nel file `defunto_detail_screen.dart`

```
Future<void> aggiungiServizio(Servizio servizio) async {
  // Creazione di un nuovo documento nella collezione "servizio"
  await _serviziCollection.add({
    'tipologia': servizio.tipologia,
    'statoServizio': servizio.statoServizio,
    'costo': servizio.costo,
    'defunto': servizio.defunto,
  });
}
}
```

Figura 4.10: Funzione che aggiunge un servizio in Firestore della classe `FatturaService` nel file `servizio_service.dart`

```dart
void _aggiungiFatturaServizio() async{
  final fattura = Fattura(
      numero: "${await _getNumFatture()}",
      cliente: user!.uid,
      defunto: [widget.defunto.id],
      dataEmissione: DateTime.now(),
      dataScadenza: DateTime.now().add(Duration(days: 30)),
      pagamento: "",
      descrizione: "Richiesto servizio ${_selectedServizio.tipologia}",
      totale: _selectedServizio.costo.toString());  // Fattura

  _fatturaService.aggiungiFattura(fattura);
}
```

Figura 4.11: Funzione che crea la fattura relativa ad un servizio e richiama il metodo di aggiunta della fattura della classe `FatturaService` nel file `fattura_service.dart`

```dart
child: FutureBuilder<List<Fattura>>(
  future: _fatturaService.getFatturePerCliente(_startDate, _endDate),
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return Center(child: CircularProgressIndicator());
    } else if (snapshot.hasError) {
      return Center(child: Text('Errore: ${snapshot.error}'));
    } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
      return Center(child: Text('Nessuna fattura trovata'));
    } else {
      final fatture = snapshot.data!
          .where((f) => !_showOnlyUnpaid || f.pagamento == "")
          .toList();
      return ListView.builder(
        itemCount: fatture.length,
        itemBuilder: (context, index) {
          final fattura = fatture[index];
          return InkWell(
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => FatturaDetailScreen(fattura: fattura, fatturaService: _fatturaService,),
                ),  // MaterialPageRoute
              );
            },
            child: Card(
              elevation: 1,
```

Figura 4.12: Parte del codice che mostra a schermo l'elenco delle fatture nella classe `FattureScreen` nel file `fatture_screen.dart`

```dart
Future<List<Fattura>> getFatturePerCliente(DateTime? startDate, DateTime? endDate) async {
  final QuerySnapshot<Object?> querySnapshot = await _fattureCollection
      .where('cliente', isEqualTo: user?.uid)
      .orderBy('dataEmissione', descending: true)
      .get();
  List<Fattura> fatture = [];
  for (final document in querySnapshot.docs) {
    final dataEmissionePars = document['dataEmissione'].split('-');
    final dataEmissione = DateTime(int.parse(dataEmissionePars[2]), int.parse(dataEmissionePars[1]), int.parse(dataEmissionePars[0]));
    final dataScadenzaPars = document['dataScadenza'].split('-');
    final dataScadenza = DateTime(int.parse(dataScadenzaPars[2]), int.parse(dataScadenzaPars[1]), int.parse(dataScadenzaPars[0]));
    final fattura = Fattura(
      numero: document['numero'],
      cliente: document['cliente'],
      defunto: List<String>.from(document['defunto']),
      dataEmissione: dataEmissione,
      dataScadenza: dataScadenza,
      pagamento: document['dataPagamento'],
      descrizione: document['descrizione'],
      totale: document['totale'].toString(),
    );  // Fattura
    if(startDate != null){
      if (endDate != null) {
        if (fattura.dataEmissione.isAfter(startDate.subtract(Duration(days: 1))) && fattura.dataEmissione.isBefore(endDate.add(Duration(days: 1)))){
          fatture.add(fattura);
        }
      }else{
        if(fattura.dataEmissione.isAfter(startDate.subtract(Duration(days: 1)))){
          fatture.add(fattura);
        }
      }
    }
    if(startDate == null){
      if(endDate != null){
        if(fattura.dataEmissione.isBefore(endDate.add(Duration(days: 1)))){
          fatture.add(fattura);
        }
      }else{
        fatture.add(fattura);
      }
    }
  }
  return fatture;
}
```

Figura 4.13: Codice della funzione che recupera le fatture di un cliente, della classe `FatturaService`, nel file `fattura_service.dart`

CAPITOLO 5

Manuale utente

Portata a termine la parte di implementazione dell'applicazione, indirizzata ad un pubblico più specifico, struttureremo questo capitolo come una breve guida che sarà utile agli utenti si che approcciano per la prima volta all'utilizzo dell'app.

5.1 Guida all'installazione dell'app

A differenza di qualsiasi altra applicaizone presente sugli Store Android ed Apple, 'Appdila' non è disponibile al loro interno.

Essendo inquadrata come un servizio che viene offerto dalle società di gestione dei siti cimiteriali, si è ritenuto sufficiente rendere accessibile il download dell'app solo tramite il sito web di tali società; tuttavia, una versione completa dell'app può essere scaricata ed installata dal seguente link `https://drive.google.com/drive/u/0/folders/1mfEnFSATu-YpLJwleYadiwLm-NW3jjg46?lfhs=2` [1], che rimanda ad una cartella Google Drive, al cui interno è presente l'APK di 'Appdila'.

5.2 Guida all'utilizzo dell'app

Come ribadito più volte, il pubblico a cui si rivolge l'applicazione è molto vasto. Potrebbe essere utile, a tal proposito, per quella categoria di utenti meno avvezza all'utilizzo della tecnologia, stilare una semplice guida di introduzione alle funzionalità offerte dall'applicazione.

5.2.1 Accesso e registrazione all'applicazione

Alla prima apertura dell'app, dopo averla installata, viene visualizzata la schermata di accesso riportata in Figura 5.1.

Se non si è ancora in possesso di un account, è possibile crearne uno cliccando sul pulsante '*Registrati*' della schermata di login. Si verrà reindirizzati verso una nuova schermata, mostrata in Figura 5.2, dalla quale, una volta inserite correttamente tutte le informazioni richieste per la registrazione, sarà eseguito automaticamente l'accesso all'applicazione.

[1] Per visualizzare correttamente la pagina è possibile utilizzare il seguente link: `https://drive.google.com/drive/u/0/folders/1mfEnFSATuYpLJwleYadiwLm-NW3jjg46?lfhs=2`

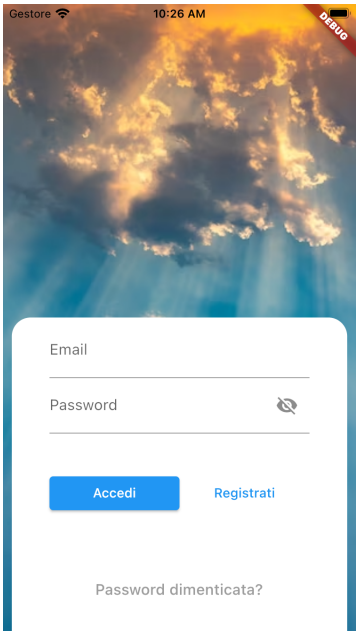


Figura 5.1: Screenshot della schermata di accesso ad 'Appdila'

Qualora l'utente fosse già in possesso di un account, ma non dovesse ricordarne la password di accesso, è possibile, sempre a partire dalla schermata di login, reimpostarla.

Cliccando in basso sulla scritta *'Password dimenticata?'* si navigherà verso la schermata mostrata in Figura 5.3 nella quale sarà possibile, inserendo l'email del proprio account, ricevere un link, dal quale reimpostare la password di accesso; completata questa operazione, l'utente sarà in grado di effettuare nuovamente l'accesso all'app utilizzando la nuova password.

5.2.2 Visualizzazione della lista dei defunti ed invio di una richiesta di allaccio

Dopo avere eseguito il login all'app, la prima schermata utile è quella che, come mostrato in Figura 5.4, visualizza l'elenco dei defunti a carico dell'utente. Da questa schermata è possibile sia effettuare una nuova richiesta di allaccio che visualizzare la scheda anagrafica di un defunto.

Per inviare una richiesta di allaccio basterà cliccare sull'icona **+** nella parte in alto a desta dello schermo; si aprirà, in questo modo, un form da compilare con i dati della richiesta di allaccio; in Figura 5.5 viene riportato un esempio. Una volta compilato il form, per confermare la richiesta, basterà cliccare sul pulsante *'Invia'*; se, invece, si vuole annullare la richiesta, si potrà cliccare sul pulsante *'Annulla'*.

5.2.3 Visualizzazione di un defunto con le relative operazioni annesse

Dalla schermata di visualizzazione della lista dei defunti è possibile, cliccando su uno dei defunti mostrati, accedere alla sua scheda anagrafica; ad esempio, se volessimo visualizzare il dettaglio del primo defunto, cliccando su di esso si aprirà una nuova schermata come quella mostrata in Figura 5.6.

Nella parte in alto a destra è presente un'icona di una lampadina, che varia a seconda dello stato attuale del defunto; se il defunto è attualmente allacciato, la lampadina sarà vuota, e, cliccandoci, permetterà di distaccare la lampada votiva del defunto; al contrario, se il defunto non dovesse essere allacciato, la lampadina sarà piena, ad indicare la possibilità di riallacciare la lampada del defunto.

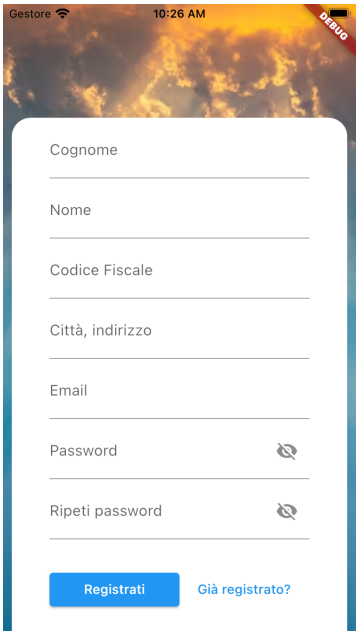


Figura 5.2: Screenshot della schermata di registrazione

Al di sotto delle informazioni del defunto che vengono visualizzate sullo schermo, è situata una sezione che permette di selezionare, tramite un menù a discesa, un servizio per il defunto selezionato; una volta effettuata la scelta del servizio, sarà sufficiente cliccare sul pulsante *'Richiedi'* per richiederlo effettivamente alla società.

Ad ogni richiesta di servizio corrisponde una emissione, in automatico, della fattura relativa a quel servizio, da parte della società di gestione.

5.2.4 Visualizzazione della contabilità e filtraggio delle fatture

Sfruttando la barra di navigazione posta nella parte inferiore dello schermo si accede alla sezione relativa alla contabilità dell'utente.

La schermata di visualizzazione delle fatture riportata in Figura 5.7 presenta, in alto, sia la possibilità di visualizzare, cliccando sul pulsante switch a destra, solo le fatture non ancora pagate, che la possibilità di filtrare le fatture in base al loro periodo di emissione.

Per filtrare le fatture in base alla loro data si potrà impostare un periodo di tempo, indicando sia la data di inizio che di fine, oppure specificando solo una delle due; per far ciò basterà cliccare sull'icona del calendario di fianco alle scritte *'Inizio'* o *'Fine'* per far apparire un calendario dal quale selezionare la data desiderata.

Una volta applicati i dovuti filtri alla lista delle fatture, è possibile visualizzare il dettaglio di una di esse cliccandoci sopra.

5.2.5 Invio di una segnalazione

Cliccando sull'ultima voce a destra della barra di navigazione inferiore dell'app, si raggiunge la sezione riservata dell'utente per inviare delle segnalazioni alla società di gestione del sito cimiteriale.

La schermata relativa alle segnalazioni, come mostrato in Figura 5.8, presenta, nella parte bassa sulla destra, un *Floating Action Button*, al cui click, corrisponde l'apertura di un form da compilare con la propria segnalazione.

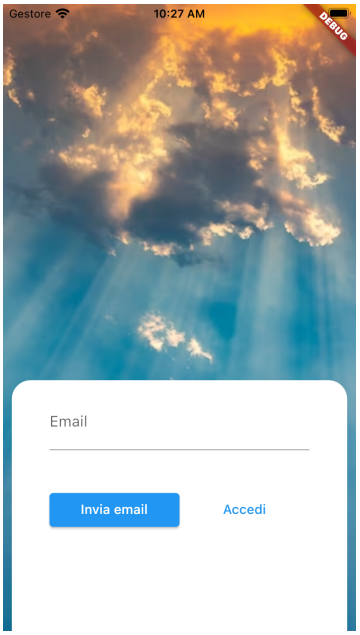


Figura 5.3: Screenshot della schermata di reimpostazione della password

5.2.6 Uscita dall'applicazione

La schermata riservata all'utente per la visualizzazione delle segnalazioni mostra, inoltre, in alto a destra, un'icona con 3 puntini; come si può vedere dalla Figura 5.9, cliccando sull'icona viene visualizzato un pulsante rosso *'Esci'* che consente all'utente di eseguire il logout dall'applicazione.

Va sottolineato che, qualora l'utente dovesse chiudere l'applicazione senza premere il pulsante *'Esci'*, alla sua riapertura non sarà necessario rieseguire nuovamente l'accesso; in caso contrario sarebbe nuovamente necessario eseguire l'autenticazione tramite email e password.

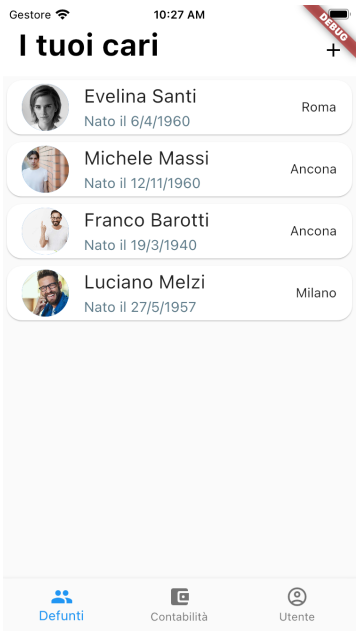


Figura 5.4: Screenshot della schermata di visualizzazione dell'elenco dei defunti

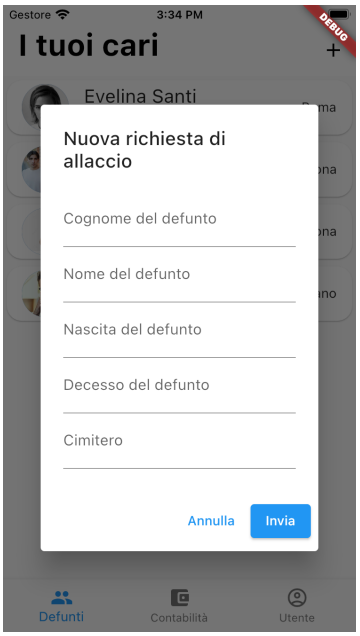


Figura 5.5: Screenshot della schermata visualizzata per effettuare una richiesta di allaccio

Figura 5.6: Screenshot della schermata di visualizzazione dell'anagrafica di un defunto

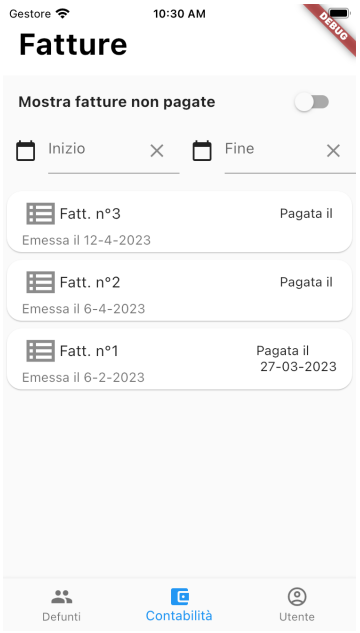


Figura 5.7: Screenshot della schermata di visualizzazione dell'elenco delle fatture

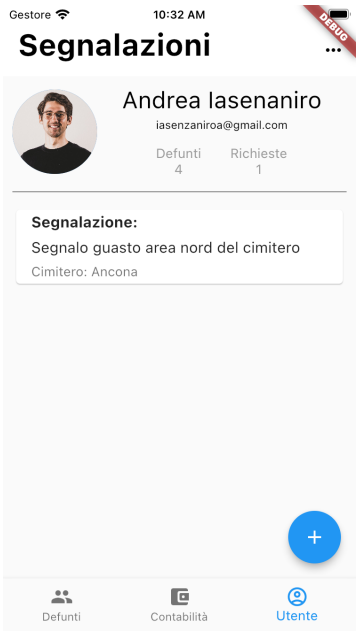


Figura 5.8: Screenshot della schermata utente che permette di visualizzare ed inviare le segnalazioni

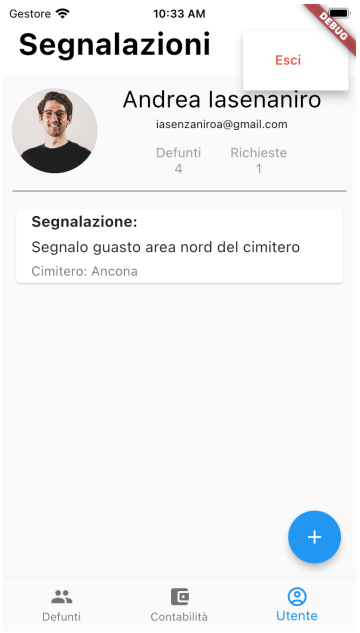


Figura 5.9: Screenshot della schermata utente che mostra il pulsante per eseguire il logout dall'app

CAPITOLO 6

SWOT analysis

In questo penultimo capitolo, il primo dei due che porteranno il lettore alla conclusione di questa tesi, si cercherà di analizzare il lavoro che è stato svolto fino a questo punto, mostrando uno spirito critico verso di esso e proponendo, così, una discussione circa gli argomenti trattati, che sia oggettiva ed imparziale.

6.1 Cos'è la SWOT analysis

La SWOT analysis, o analisi SWOT, è una delle tecniche di pianificazione, nonché di valutazione, maggiormente utilizzate in ambito organizzativo, imprenditoriale, progettuale, ma anche personale.

Si tratta di un framework molto valido che trova spazio nella progettazione moderna, adoperato nell'ambito del project management per analizzare non solo i punti di forza e di debolezza di un progetto (*Strengths* and *Weaknesses*, SW), ma anche le opportunità e le minacce le quali potrebbero derivare da questo (*Opportunities* and *Threats*, OT); è utilizzato per individuare i problemi ed i limiti al raggiungimento degli obiettivi prefissati, ma anche soluzioni e direzioni da seguire per il loro raggiungimento.

Come possiamo quindi comprendere, gli aspetti su cui si concentra l'analisi non derivano soltanto da fattori interni all'organizzazione, ma anche dall'esterno; i punti di forza e di debolezza sono interni al progetto o all'organizzazione, rappresentano, pertanto, quei fattori sui quali si può agire direttamente; le opportunità e le minacce sono esterne al contesto considerato e, quindi, non sono direttamente controllabili.

Uno dei metodi utilizzati per la rappresentazione della SWOT analysis fa uso di una matrice denominata matrice SWOT. Come mostrato in Figura 6.1 la matrice è organizzata come una tabella a doppia entrata suddivisa in quattro quadrati separati, ognuno dei quali dedicato ad uno dei fattori da analizzare.

6.2 SWOT analysis dell'app

Come anticipato nella sezione precedente, l'utilizzo della SWOT analysis non è ristretto alle sole imprese ed organizzazioni, ma può essere ampliato a qualsiasi altro ambito, con lo scopo di raggiungere un obiettivo prefissato.

Effettuiamo, a questo punto, l'analisi nel contesto delle applicazioni mobili, riferendoci nello specifico alla progettazione di 'Appdila'. Ponendoci con un adeguato spirito critico

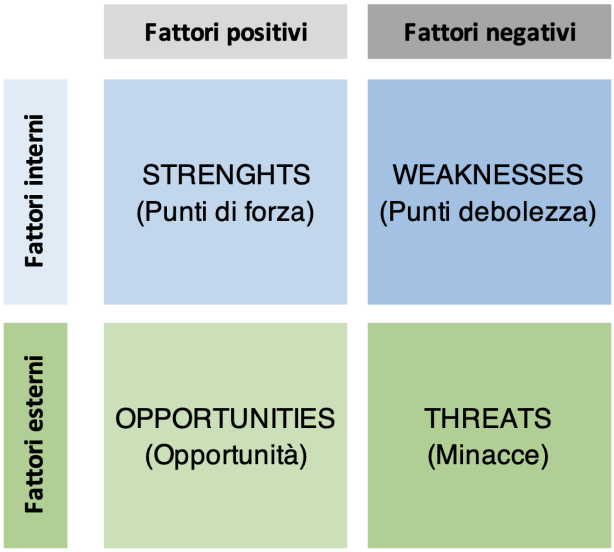


Figura 6.1: Matrice SWOT

ricaviamo delle informazioni da essa, che, per una maggior comprensione, riportiamo sotto forma di elenco, tralasciando la matrice SWOT.

6.2.1 Punti di forza

I punti di forza individuati dall'analisi dell'applicazione sviluppata possono essere riassunti come segue:

- *Interfaccia user friendly*: l'interfaccia dell'applicazione è stata progettata per risultare estremamente semplice ed intuitiva per tutte le categorie di utenti che si approcceranno al suo utilizzo.
- *Distanza geografica*: l'utilizzo dell'app permette di far fronte a quello che, in altri casi, risulterebbe un ostacolo; essa, infatti, garantisce l'abbattimento di ogni genere di difficoltà dovuta alla distanza geografica che potrebbe separare un cliente da un suo caro parente defunto.

6.2.2 Punti di debolezza

'Appdila' non sembra mostrare apparentemente dei lati estremamente negativi; possiamo individuare due punti di debolezza, che, come vedremo, con le opportune considerazioni, potrebbero essere ridotti ad uno soltanto. I punti di debolezza individuati sono i seguenti:

- *Connessione ad internet*: l'applicazione, per poter funzionare, richiede una connessione ad internet stabile da parte del dispositivo dell'utente; solo in questo modo può garantire una corretta gestione delle informazioni sia in lettura che in scrittura.
- *Compatibilità dei dispositivi*: data la grande varietà di dispositivi presenti sul mercato, l'interfaccia progettata per 'Appdila' potrebbe risultare non correttamente visualizzabile su alcune tipologie di questi ultimi; ciò potrebbe indurre gli utenti proprietari di quella categoria di dispositivi a non utilizzare l'app, o, addirittura, a disinstallarla.

Analizzando il primo dei punti sopra elencati possiamo affermare che l'utilizzo di una piattaforma come Firebase è in grado di garantire un'ottima user experience dell'app, anche in presenza di una connessione ad internet poco stabile.

Le app Firebase, infatti, sono in grado gestire automaticamente le interruzioni temporanee della rete; i dati vengono memorizzati nella cache del dispositivo per rimanere disponibili anche in assenza di connessione; sarà poi Firebase ad inviare nuovamente i dati quando viene ripristinata la connettività di rete.

Ciò permette di non considerare, a tutti gli effetti, la connessione ad internet un punto di debolezza di 'Appdila'.

6.2.3 Opportunità

Le opportunità che scaturiscono dall'utilizzo dell'applicazione sono da attribuire alla sua capacità di attirare, nel tempo, una buona fetta del mercato della gestione delle lampade votive; sarebbe possibile, in questo, modo pensare ad una *gestione a livello nazionale*, instaurando collaborazioni con agenzie funebri, ma anche con altri enti connessi al mondo della gestione dei siti cimiteriali.

Il tutto, ovviamente, deve essere corredato di un appropriato aggiornamento delle funzionalità dell'applicazione.

6.2.4 Minacce

Per quanto riguarda le minacce che sono state rilevate nell'utilizzo di 'Appdila', esse riguardano principalmente l'*utilizzo dei 'dati sensibili'* degli utenti e dei defunti, quali codice fiscale, indirizzo di residenza e numero di telefono.

Non va dimenticato che l'archiviazione dei dati in sistemi cloud, anche qualora questi fossero dotati di opportuni sistemi di prevenzione, protezione e ripristino da attacchi informatici, rappresenta una potenziale minaccia ai danni degli utenti ai quali essi appartengono.

CAPITOLO 7

Conclusioni

Giunti all'ultimo capitolo che conduce alla conclusione di questa tesi, possiamo riportare brevemente gli step fondamentali che ci hanno condotti ad ottenere i risultati voluti, mostrando anche quali potrebbero essere degli sviluppi futuri per l'applicazione.

7.1 Uno sguardo al lavoro svolto

Se volessimo motivare il lavoro che è stato svolto fino a questo punto potremmo dire che, al giorno d'oggi, qualsiasi tipo di sistema, informatico e non, che viene utilizzato nel tempo e che è studiato per evolversi nel corso della sua vita utile, necessita di un'adeguata progettazione, che permetta, a chi verrà dopo di noi, di continuare il lavoro che è stato svolto fino a quel momento. Detto ciò, anche un'applicazione mobile necessita di adeguati metodi di ingegneria del software, come quelli utilizzati nella stesura di questa tesi.

Siamo partiti con l'obiettivo di realizzare un'applicazione mobile cross platform, che fosse di ausilio alla gestione di società che operano nell'ambito delle lampade votive.

È stato prima di tutto utile introdurre il mondo del mercato mobile, evidenziando le sue caratteristiche, ma anche le 'sfide' che uno sviluppatore deve affrontare per poter ottenere un prodotto che risulti 'riuscito' sotto tutti i punti di vista.

Il secondo passaggio è stato quello di capire cosa effettivamente si sarebbe dovuto sviluppare; è stato necessario, a tal proposito, eseguire un'accurata analisi dei requisiti, i quali sarebbero andati poi a definire le funzionalità che l'applicazione avrebbe dovuto offrire ai suoi utenti.

La fase successiva alla definizione delle funzionalità è stata la loro progettazione. Questa fase ha interessato ogni aspetto dell'app: la sua struttura, l'organizzazione delle classi del codice, il modo di archiviare i dati da essa utilizzati e prodotti, ma anche la sua interfaccia grafica.

Solo dopo aver concluso la fase di progettazione è stato possibile implementare l'applicazione vera e propria. Durante questa fase sono stati definiti gli strumenti utilizzati per il suo sviluppo, e come questi debbano interagire tra di loro; è stato infine generato il file eseguibile dell'app.

7.2 I possibili sviluppi futuri

Osservando 'Appdila' si può intuire come questa sia un'applicazione ancora in uno stato prototipale, ma che, con le dovute migliorie, potrebbe avere ancora molto da offrire ai suoi utenti; nelle versioni successive a quella rilasciata per il completamento di questa tesi, infatti, è in previsione l'aggiunta di ulteriori funzionalità.

Il primo obiettivo è quello di introdurre il pagamento in-app nel momento in cui venga richiesto un servizio per uno dei defunti a carico; in questo modo si accelererebbe di molto l'attività di gestione ordinaria dell'azienda.

Un altro aspetto riguarda le fatture; si vuole dare modo all'utente di scaricare autonomamente, dall'applicazione, una copia delle fatture ricevute in formato pdf, qualora queste dovessero venire smarrite.

Come già discusso nella Sezione 6.2.3, l'evoluzione dell'applicazione dipenderà molto dal grado di successo che riscuoterà tra gli utenti; si prevede, a tal proposito, di inserire una sezione che possa raccogliere dei feedback, in modo da venire incontro alle loro esigenze.

Bibliografia

ALESSANDRIA, S. (2020), «Crea la Tua App con Flutter: Guida Pratica per Creare App per Android, iOS e il Web con Dart e Flutter», *Independently published*.

BURD, B. (2020), «Flutter For Dummies», *John Wiley Sons Inc*.

CHOPRA, D. (2023), «Flutter and Dart: Up and Running: Build native apps for both iOS and Android using a single codebase», *BPB Publications*.

GIACCHINA, V. (2023), «Sviluppare applicazioni con Flutter», *Apogeo*.

LEE, R. (2023), «Dart: la guida completa per lo sviluppo di app web», *Independently published*.

RAMEZ A. ELMASRI, S. B. N. (2017), «Sistemi di basi di dati. Fondamenti e complementi», *Pearson*.

SOMMERVILLE, I. (2017), «Introduzinoe all'ingegneria del software», *Pearson Education Italia*.

ZACCAGNINO, C. (2019), «Flutter. Guida allo sviluppo di app performanti e cross-platform», *Hoepli*.

Siti web consultati

- Ministero della Salute – `www.salute.gov.it/portale/home.html`
- ISTAT – `www.istat.it/`
- Data.ai Intelligence – `www.data.ai`
- Kotlin.org – `www.kotlinlang.org`
- Flutter.dev – `https://docs.flutter.dev/resources/faq`
- Firebase | Firestore – `https://firebase.google.com/`
- RedHat – `www.redhat.com/it`
- NTTData Trusted Global Innovator – `www.nttdata.com`

- Amazon AWS – `www.aws.amazon.com`
- back4app – `www.back4app.com/`
- Weelorum.com – `www.weelorum.com`
- Web Marketing Pro – `www.webmarketingpro.it`
- AlmaLaboris – `www.almalaboris.com/`

Ringraziamenti

Desidero ringraziare innanzitutto il Prof. Domenico Ursino, mio relatore, per il paziente, costante e validissimo supporto che mi ha fornito nella stesura di questa tesi, senza il quale il risultato ottenuto non sarebbe stato altrettanto di qualità. Ringrazio il Dott. Enrico Corradini non solo per i preziosi consigli forniti riguardo l'implementazione dell'applicazione, ma anche per la sua grandissima disponibilità.

Ringrazio, in particolar, modo i miei genitori, per il bene che mi volete, e che mi dimostrate ogni giorno, e per tutti gli sforzi che avete sempre sostenuto per non farmi mancare mai nulla. Mi auguro che questo traguardo, oltre a darmi uno slancio verso un futuro di crescita personale e professionale, sia anche un piccolo modo per ringraziarvi di tutto ciò che avete sempre fatto e che, so, continuerete a fare, finché potrete.

Voglio ringraziare mia sorella, perché, nonostante le nostre strade si siano divise da ormai cinque anni a questa parte, con la tua determinazione ed ambizione, rappresenti un mio costante punto di riferimento, ispirandomi e motivandomi a guardare sempre a quanto di meglio la vita possa offrire.

Ringrazio enormemente Fabiana. Quegli interminabili mesi estivi di preparazione al diploma sono serviti da riscaldamento per gli anni di università che ci avrebbero attesi. È stato un lungo periodo, segnato da alti e bassi, ma, nonostante ciò, non posso far altro che ringraziarti per l'impegno e la pazienza che ci metti ogni giorno per ascoltarmi, sopportarmi e sostenermi. Sono fiero di cosa siamo diventati e di come siamo cresciuti insieme. Ci auguro nient'altro che il futuro migliore al quale possiamo ambire.

Ringrazio i miei coinquilini, ma soprattutto amici, Carlo e Lorenzo, per aver vissuto con me questo viaggio indimenticabilmente unico; non avrei potuto condividere le mie giornate, tra gioie e dolori, con persone migliori di voi.

Ringrazio, infine, i miei colleghi universitari, nonché amici. In particolare ringrazio Davide. Ti ringrazio, anzi, ci ringrazio, per essere stati presenti nei nostri rispettivi percorsi universitari, per aver condiviso sia i momenti felici che quelli un po' meno, ma, soprattutto, per esserci dati forza a vicenda quando tutto poteva sembrare perduto. Spero che questi anni abbiano rappresentato solo l'inizio di una grande amicizia.