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

TESI DI LAUREA

Progettazione e implementazione di un'app in tecnologia SWIFT per il supporto nella ricerca di abitazioni: la componente di gestione delle abitazioni

Design and implementation of an app in SWIFT technology for housing search support: the housing management component

Relatore

Prof. Domenico Ursino

Candidato

Simone Sgalla

Correlatore

Dott. Enrico Corradini

ANNO ACCADEMICO 2022-2023

Il tempo che ti piace sprecare non è mai tempo sprecato

John Lennon

Sommario

Anno dopo anno la quantità di tempo che le persone passano davanti al proprio smartphone aumenta. Ciò è dovuto al rapido sviluppo delle applicazioni mobile che, giorno dopo giorno, semplificano la vita dei propri utenti. In questo contesto dominato dagli smartphone abbiamo pensato di realizzare una nostra applicazione mobile, che soddisfacesse le esigenze di una ristretta cerchia di utenti come quella degli studenti universitari. Questo, perchè, tra tutti i software che si occupano di immobili, non ne esiste ancora uno dedicato esclusivamente agli universitari, che ogni anno occupano una porzione importante del mercato immobiliare. Di conseguenza abbiamo realizzato un'applicazione mobile, che consentisse ai propri utenti l'affitto di proprietà per la durata dell'Anno Accademico e l'organizzazione in gruppi per l'occupazione di appartamenti in condivisione.

Keyword: Applicazione mobile, Swift, API, Ingegneria del software, Paradigma dichiarativo, GoogleMaps, SwiftUI, Firebase.

Indice

Introduzione **1**

1 Introduzione a Swift **4**
1.1 Nascita e sviluppo di Swift . . . 4
1.1.1 Origine di Swift . . . 4
1.1.2 Rilascio di Swift . . . 5
1.2 Generalità di Swift . . . 5
1.2.1 Modernità . . . 5
1.2.2 Sicurezza . . . 6
1.2.3 Open source . . . 6
1.2.4 Velocità . . . 7
1.3 Pro e contro del linguaggio Swift . . . 7
1.3.1 Pro . . . 7
1.3.2 Contro . . . 8
1.4 Specifiche per la programmazione . . . 10
1.4.1 Variabili semplici . . . 10
1.4.2 Operatori condizionali . . . 10
1.4.3 Metodi e funzioni . . . 11
1.4.4 Oggetti e classi . . . 11
1.4.5 Funzioni asincrone . . . 12
1.4.6 Gestione degli errori . . . 12
1.5 SwiftUI . . . 12
1.5.1 Introduzione . . . 12
1.5.2 Aspetti positivi di SwiftUI . . . 13
1.5.3 Aspetti negativi di SwiftUI . . . 15

2 Specifica e Analisi dei requisiti **18**
2.1 Introduzione . . . 18
2.1.1 Scopo . . . 18
2.1.2 Pubblico . . . 18
2.1.3 Riferimenti . . . 18
2.1.4 Obiettivo . . . 19
2.1.5 Acronimi e definizioni . . . 19
2.2 Descrizione generale . . . 20
2.2.1 Funzionalità del prodotto . . . 20

2.2.2 Caratteristiche utente . . . 21
2.2.3 Vincoli generali . . . 21
2.2.4 Ipotesi e dipendenze . . . 21
2.3 Requisiti e caratteristiche del sistema . . . 21
2.3.1 Requisiti funzionali . . . 21
2.3.2 Requisiti di interfaccia grafica . . . 22
2.3.3 Requisiti non funzionali . . . 25
2.3.4 Caratteristiche del sistema . . . 25

3 Progettazione **26**
3.1 Progettazione concettuale . . . 26
3.1.1 Diagramma entità relazione . . . 26
3.1.2 Analisi entità e relazioni . . . 27
3.2 Progettazione logica . . . 28
3.2.1 Eliminazione delle gerarchie . . . 28
3.2.2 Schema logico . . . 28
3.3 Progettazione delle applicazioni . . . 28
3.3.1 Diagramma delle classi . . . 28
3.3.2 Diagramma di deployment . . . 29
3.3.3 Diagramma dei casi d'uso . . . 29
3.3.4 Diagrammi di sequenza . . . 32
3.3.5 Diagrammi di attività . . . 33
3.4 Mockup . . . 34

4 Implementazione **56**
4.1 Architettura dell'applicazione . . . 56
4.2 Divisione in package . . . 56
4.3 Implementazione frontend . . . 58
4.3.1 ChatUserInfoView . . . 58
4.4 Implementazione backend . . . 60
4.4.1 Model . . . 60
4.4.2 ViewModel . . . 61

5 Manuale dell'utente **63**
5.1 Introduzione all'applicazione . . . 63
5.2 Accesso all'applicazione . . . 63
5.2.1 Registrazione . . . 64
5.2.2 Login . . . 64
5.3 Istruzioni per l'uso delle sezioni principali . . . 64
5.3.1 Mappa . . . 65
5.3.2 Lista . . . 65
5.3.3 Preferiti . . . 67
5.3.4 Chat . . . 67
5.3.5 Profilo . . . 69

6 Discussione **75**
6.1 Analisi SWOT . . . 75
6.1.1 Punti di forza . . . 75
6.1.2 Punti di debolezza . . . 76
6.1.3 Opportunità . . . 76
6.1.4 Minacce . . . 76

6.2 Confronto 77
6.2.1 Confronto con Idealista 77
6.2.2 Confronto con Immobiliare 78
6.2.3 Confronto con Booking 78
6.2.4 Confronto con Casa 78
6.2.5 Confronto con Trovacasa 79

7 Conclusioni **80**

Bibliografia **82**

Ringraziamenti **84**

Elenco delle figure

1.1 Chris Lattner 5
1.2 Risultati del sondaggio di StackOverflow sulla tecnologia più amata del 2015, fonte: 2015 StackOverflow Developer Survey 6
1.3 Lista dei linguaggi di programmazione più sviluppati su Github 2021 fonte: github.com 8
1.4 Lista dei linguaggi di programmazione più utilizzati nel 2022 fonte: StackOverflow 9
1.5 Dichiarazione di variabile utilizzando `let` e `var` 10
1.6 Diversi tipi di dichiarazione di variabile 10
1.7 Creazione e operazioni con vettori 11
1.8 Esempio di utilizzo dell'operatore `if` 11
1.9 Esempio utilizzo dell'operatore `for-in` 12
1.10 Esempio di utilizzo dell'operatore `while` e `repeat-while` 13
1.11 Esempio di dichiarazione di una funzione 13
1.12 Esempio di funzione che restituisce più valori 14
1.13 Esempio di dichiarazione di una classe senza costruttore 14
1.14 Esempio di chiamata ad una classe 15
1.15 Esempio di dichiarazione di una classe con costruttore, metodo e proprietà . 15
1.16 Esempio di funzione asincrona 16
1.17 Esempio di chiamata ad una funzione asincrona 16
1.18 Esempio di utilizzo di `async let` 17
1.19 Esempio di dichiarazione di un errore 17
1.20 Esempio di chiamata e dichiarazione di una funzione con errore 17
1.21 Esempio di utilizzo del costrutto `do-catch` 17

3.1 Diagramma E-R del database dell'applicazione 26
3.2 Diagramma logico del database dell'applicazione 29
3.3 Diagramma delle classi relativo alla gestione della mappa 29
3.4 Diagramma delle classi relativo alla gestione delle abitazioni 30
3.5 Diagramma delle classi relativo alla gestione della chat 31
3.6 Diagramma delle classi relativo alla gestione degli utenti 32
3.7 Diagramma di deployment 33
3.8 Caso d'uso Autenticazione 33
3.9 Caso d'uso Chat 34
3.10 Caso d'uso Modifica Profilo 34

3.11 Caso d'uso Ricerca Filtrata 35
3.12 Diagramma di sequenza Login 35
3.13 Diagramma di sequenza Registrazione 36
3.14 Diagramma di sequenza ricerca filtrata 36
3.15 Diagramma di sequenza modifica profilo 37
3.16 Diagramma di sequenza Chat 38
3.17 Diagramma di attività Login 38
3.18 Diagramma di attività Registrazione 39
3.19 Diagramma di attività ricerca filtrata 40
3.20 Diagramma di attività modifica profilo 41
3.21 Diagramma di attività Chat 42
3.22 Mockup: dettagli abitazione 43
3.23 Mockup: modifica della password 44
3.24 Mockup: mappa 45
3.25 Mockup: filtri 46
3.26 Mockup: login 47
3.27 Mockup: registazione 48
3.28 Mockup: Chat 49
3.29 Mockup: selezione dell'utente 50
3.30 Mockup: aggiornamento dell'e-mail 51
3.31 Mockup: aggiornamento delle informazioni utente 52
3.32 Mockup: visualizzazione del profilo 53
3.33 Mockup: lista dei preferiti 54
3.34 Mockup: lista delle abitazioni 55

4.1 Divisione in package 57

5.1 Analisi dell'interfaccia di registrazione 64
5.2 Analisi dell'interfaccia di login 65
5.3 Analisi dell'interfaccia della mappa 66
5.4 Analisi dell'interfaccia della lista di abitazioni 67
5.5 Analisi dell'interfaccia dei filtri 68
5.6 Analisi dell'interfaccia più informazioni relative ad un'abitazione 69
5.7 Analisi dell'interfaccia dei preferiti 70
5.8 Analisi dell'interfaccia della chat 71
5.9 Analisi dell'interfaccia dei messaggi 71
5.10 Analisi dell'interfaccia del profilo dell'utente con cui si sta chattando 72
5.11 Analisi dell'interfaccia del profilo 72
5.12 Analisi dell'interfaccia di modifica dell'e-mail 73
5.13 Analisi dell'interfaccia di modifica della password 73
5.14 Analisi dell'interfaccia di completamento del profilo 74

Introduzione

Negli ultimi anni l'importanza delle applicazioni mobile è aumentata vertiginosamente. Infatti, tutti i servizi dispongono, ormai, di una rispettiva versione online che ne aumenti l'accessibilità per tutti i clienti. Tutti noi sappiamo che, per soddisfare le nostre esigenze, risulta molto più rapido e pratico l'utilizzo di un software, piuttosto che il metodo tradizionale. Ciò è anche dovuto all'incremento della popolarità degli smartphone; infatti, chiunque dispone di almeno uno smartphone, che utilizza per svariate ore ogni giorno. Per questo motivo, ci ha sorpreso l'assenza di un software dedicato esclusivamente all'utenza universitaria, che ogni anno è costretta a ricercare un'abitazione con metodi non ottimali, come, ad esempio, gruppi su Facebook, agenzie immobiliari, etc.

A nostro parere risultano superati i servizi offerti dalle agenzie immobiliari, in quanto svolgono delle operazioni ormai accessibili a tutti tramite appositi siti web, esigendo una percentuale del costo del servizio offerto. Sicuramente questi servizi non sono accessibili a buona parte di coloro che frequentano l'università da fuori-sede, che, infatti, sono già costretti ad affrontare costi molto alti per l'affitto di abitazioni in comuni costosi. Molti di coloro che frequentano l'università non dispongono neanche di un reddito, in quanto risulta molto difficile riuscire a lavorare e proseguire gli studi universitari in contemporanea. Inoltre, la disponibilità economica degli studenti risulta già ridotta dalle tasse universitarie ed il materiale richiesto dai singoli corsi.

Per lo sviluppo dell'applicazione abbiamo scelto la tecnologia Swift, in quanto essa rispecchia l'idea di modernità da noi ricercata con la realizzazione di questo progetto. L'apprendimento di Swift, inoltre, è molto facile e sono disponibili vari testi online per un eventuale approfondimento. Inoltre, in Swift risulta immediata l'integrazione con il toolkit SwiftUI, che affronta il tema delle interfacce grafiche utilizzando un paradigma dichiarativo; questo permette di vedere la programmazione da un'altra prospettiva, poichè i linguaggi meno recenti sfruttano il paradigma imperativo. Il paradigma dichiarativo è, a nostro parere, un punto di svolta nella programmazione, in quanto permette di realizzare righe di codice in maniera molto più efficace e leggibile rispetto all'approccio classico. Per giunta, essendo più corte e leggibili, le righe di codice risultano anche più facili da correggere o modificare, permettendo di risparmiare molto tempo anche nella fase di debugging.

Tutti questi fattori ci hanno permesso di realizzare un'applicazione che soddisfacesse le nostre aspettative in breve tempo, nonostante non avessimo mai utilizzato la tecnologia Swift.

Per la realizzazione del progetto da noi ideato abbiamo iniziato effettuando un'analisi dei requisiti. Una volta stabiliti questi, ci siamo concentrati sul requisito più prioritario, ovvero l'individuazione di API adeguate alle nostre esigenze. Dopo varie richieste non andate a buon

fine, siamo riusciti ad ottenere un'accesso gratuito (anche se limitato) alle API di Idealista. Abbiamo proseguito la fase di progettazione realizzando i diagrammi relativi alla gestione dei dati (il diagramma entità-relazione ed il diagramma relazionale), adattandoli ai dati forniti dalle API da noi scelte. In seguito, abbiamo realizzato i diagrammi relativi ai casi d'uso, in modo da figurarci come avremmo voluto che la nostra applicazione si interfacciasse con gli utenti. Su questa base abbiamo poi creato i diagrammi delle classi, di deployment, di flusso e di attività.

In questo modo, abbiamo concluso la fase di progettazione e ci siamo, quindi, concentrati sull'implementazione, iniziando dalle interfacce grafiche. La realizzazione di queste è stata possibile grazie all'utilizzo del toolkit SwiftUI (che verrà approfondito in seguito nel capitolo dedicato), una tecnologia molto moderna creata da Apple che ci ha permesso di creare viste complesse in brevissimo tempo.

Successivamente siamo passati all'implementazione delle varie funzionalità, dividendole in sezioni in base al loro scopo per l'utente. Abbiamo quindi iniziato con la creazione di una barra di navigazione, che permettesse all'utente di muoversi agevolemente tra le varie interfacce dell'applicazione, per poi proseguire con l'implementazione di ciascuna delle sezioni da questa previste. Per la gestione delle informazioni generate dagli utenti, abbiamo scelto di utilizzare il database online Firebase, che dà gratuitamente accesso a vari servizi per la gestione di dati ad applicazioni mobile. Inoltre, per rendere più chiara all'utente la collocazione geografica delle abitazioni disponibili, abbiamo utilizzato le API di GoogleMaps per generare una cartina geografica interattiva. Infine, abbiamo implementato la parte dell'applicazione relativa alle conversazioni tra gli utenti, concludendo anche la parte di implementazione del nostro progetto.

In questo elaborato vengono trattate tutte le fasi che hanno portato al compimento del progetto da noi ideato, partendo dalla scelta del linguaggio di programmazione, fino ad arrivare ad un ipotetico rilascio del prodotto sul mercato. Vengono forniti anche tutti i diagrammi relativi alla fase di progettazione, i requisiti dell'applicazione ed i mockup grafici delle interfacce. In seguito vengono trattate nel dettaglio le fasi dell'implementazione dell'applicazione ed, infine, viene fornito un breve manuale per l'utente, che spiega nel dettaglio il funzionamento dell'applicazione facilitandone l'utilizzo.

La presente tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 viene analizzata la scelta del linguaggio di programmazione Swift, partendo dalla sua origine fino alle sue svariate funzionalità. Il capitolo si conclude con un approfondimento relativo a SwiftUI.
- Nel Capitolo 2 viene analizzata la parte di specifica ed analisi dei requisiti, spiegando questi utlimi singolarmente al lettore, a cui verrà anche fornito un elenco dei termini più utilizzati nella trattazione.
- Nel Capitolo 3 viene analizzata la parte di progettazione dell'applicazione, tramite la spiegazione di tutti i diagrammi previsti da questa fase.
- Nel Capitolo 4 viene analizzata la parte di implementazione dell'applicazione, analizzandone ogni classe significativa.
- Nel Capitolo 5 viene fornito un manuale per gli utenti dell'applicazione, che permette anche al lettore di figurarsi il funzionamento dell'applicazione.
- Nel Capitolo 6 viene discussa un'eventuale pubblicazione dell'applicazione, tramite l'analisi SWOT ed il confronto con applicazioni già esistenti con funzioni simili.

- Nel Capitolo 7 vengono tratte le conclusioni del nostro lavoro, ricapitolando i passi svolti in questo progetto e quelli ancora da svolgere in vista di un'eventuale pubblicazione.

CAPITOLO 1

Introduzione a Swift

In questo capitolo introduttivo vengono fornite al lettore delle informazioni sul linguaggio di programmazione Swift, in modo da rendere più chiari i capitoli successivi. In particolare, inizieremo con una breve introduzione storica sull'origine e lo sviluppo di Swift, seguita poi dalle caratteristiche principali di questo linguaggio. Successivamente analizzeremo gli aspetti positivi e quelli negativi per comprendere meglio i punti di forza e quelli di di debolezza di questa tecnologia. Infine, verranno fornite delle basi per la programmazione in Swift con degli esempi di codice, per avvicinare il lettore alla parte pratica di Swift.

L'ultima sezione del capitolo riguarda invece SwiftUI: il toolkit da noi utilizzato per implementare la parte grafica della nostra applicazione. In questa sezione, l'argomento verrà introdotto in breve spiegandone le principali funzionalità seguito poi dai pro e contro di tale tecnologia.

1.1 Nascita e sviluppo di Swift

Swift è un linguaggio di programmazione per iOS, macOS, watchOS e tvOS, in questo capitolo ne verranno spiegate le principali caratteristiche.

1.1.1 Origine di Swift

Partiamo dalla nascita: il creatore di Swift è Chris Lattner (vedi figura 1.1), capo del reparto di sviluppo della Apple, che ha iniziato lo sviluppo di Swift nel 2010.

> *"Initially, it was really just me messing around and nobody knew about it because it wasn't anything to know about. But eventually, it got a little bit more serious [...] So I started talking to my management and some of the engineers that were working on Clang, and they seemed excited about it. We got a couple people working on it part-time and I convinced my manager that it was interesting enough that we could have a couple of people work on it."*
>
> – Chris Lattner for Accidental Tech Podcast, January 2017

Prima di poter annunciare la creazione di un nuovo linguaggio, Apple dovette decidere come integrare Swift con il 'vecchio' linguaggio di programmazione per iOS, ovvero Objective-C, poichè si temeva che costringere i programmatori Apple a spostarsi su un nuovo linguaggio di programmazione potesse avere su questi un effetto negativo. Alla fine, nel 2013, Apple decise di continuare ad investire su Objective-C anche successivamente al rilascio di Swift lasciando così ai programmatori la scelta del linguaggio da utilizzare. L'anno seguente venne rilasciata la prima versione beta di Swift per i programmatori Apple.

Figura 1.1: Chris Lattner

1.1.2 Rilascio di Swift

Essendo rilasciato da Apple ed essendo annunciato come strumento principale di sviluppo iOS, le aspettative su Swift furono da subito molto alte e portarono i primi programmatori ad utilizzarlo appena dopo il rilascio. Swift fu, quindi, un grande successo: infatti, nel primo mese di rilascio come parte del kit di Xcode (IDE per programmare applicazioni Apple), Swift venne scaricato da 11 milioni di utenti. Le reazioni iniziali degli utenti furono discordi: qualcuno diceva che la semplicità e la flessibilità di Swift fossero un aspetto positivo, mentre altri le consideravano una pecca; la maggior parte dei programmatori erano, però, d'accordo sul fatto che fosse troppo presto per utilizzare Swift nella produzione di progetti complessi. Il linguaggio si stava, infatti, evolvendo rapidamente e, ad ogni suo aggiornamento, venivano introdotti grandi cambiamenti. Nell'anno successivo al suo rilascio Swift risultò, comunque, la tecnologia più amata dell'anno (vedi figuta 1.2).

Un altro importante punto di svolta nella crescita di Swift come linguaggio avvenne sempre nel 2015 quando Apple decise di rendere Swift 'Open Source'. Questo, infatti, portò ulteriori utenti ad avvicinarsi al linguaggio; nella prima settimana dall'annuncio oltre 60.000 persone scaricarono il progetto originale di Swift. Infine, nel 2020, Swift divenne accessibile anche agli utenti Windows e Linux permettendo, così, lo sviluppo anche da ambienti diversi da quelli iOS.

1.2 Generalità di Swift

Vediamo ora le principali caratteristiche del linguaggio di programmazione Swift per comprendere i motivi del suo successo.

1.2.1 Modernità

> *"Swift is the result of the latest research on programming languages, combined with decades of experience building Apple platforms."*
>
> – Fonte: developer.apple.com/swift

Come riportato sopra tra le prime caratteristiche di Swift viene sempre citata la modernità dovuta al suo recente sviluppo. Questo permette ai suoi programmatori di scrivere codice con

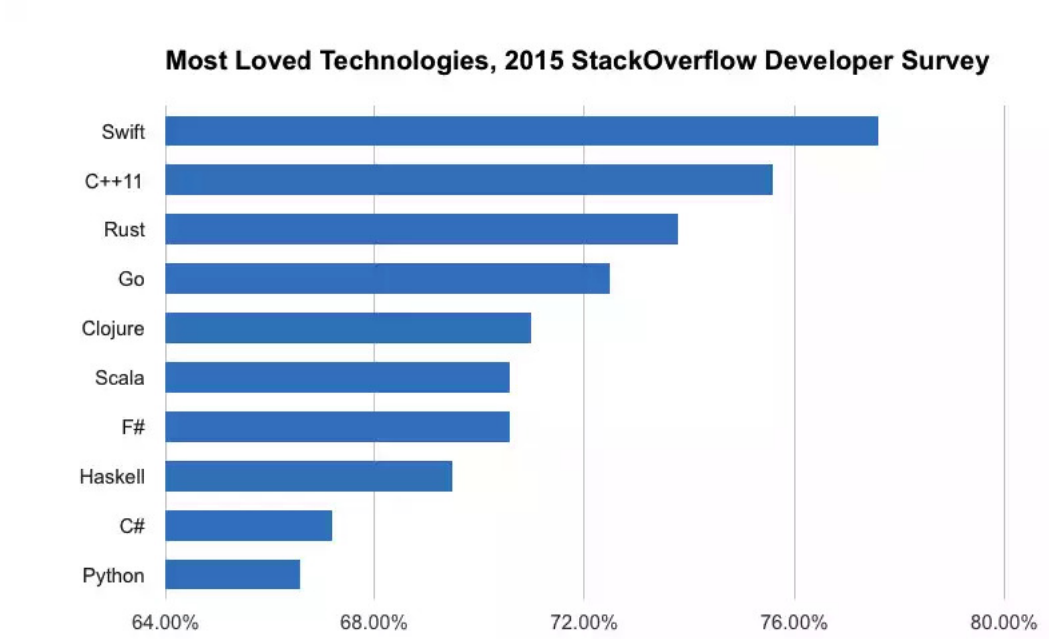

Figura 1.2: Risultati del sondaggio di StackOverflow sulla tecnologia più amata del 2015, fonte: 2015 StackOverflow Developer Survey

sintassi semplici e immediate permettendo una maggiore leggibilità del codice. La gestione della memoria avviene automaticamente senza l'utilizzo del *garbage collector*, grazie a dei riferimenti rigorosi alle variabii. Le stringhe seguono tutte la codifica UTF-8 e, per scrivere codice che esegue operazioni in maniera asincrona, è sufficiente utilizzare delle parole chiave definite dal linguaggio semplificando di molto il codice.

1.2.2 Sicurezza

Un'altra caratteristica che viene citata più volte quando si parla di Swift è la sua sicurezza: le variabili infatti devono sempre essere inizializzate prima dell'utilizzo, i vettori vengono sempre controllati per evitare l'*overflow* e vengono prese molte altre misure per evitare gli errori di programmazione. Per quanto riguarda la 'null-safety', in Swift gli oggetti non possono mai essere nulli: il compilatore impedirà di assegnare il valore `nil` ad un oggetto evitando così tutti gli errori dovuti a riferimenti a variabili nulle. La gestione di dati del tipo *nullable* avviene attraverso dei simboli appositi, utili proprio a gestire in maniera sicura i valori nulli.

1.2.3 Open source

Come detto nel capitolo precedente, Swift è un linguaggio *open source*: il codice sorgente può, infatti, essere scaricato dal sito *Swift.org*. Inoltre, nonostante sia molto "giovane" come linguaggio di programmazione, Swift ha già accumulato una community vasta, e sono presenti molte librerie create dagli utenti per facilitare il lavoro dei programmatori. Online è possibile trovare anche vari documenti e articoli che trattano l'argomento per approfondire la propria conoscenza di Swift. Il linguaggio viene, poi, continuamente aggiornato e migliorato dagli sviluppatori per migliorarne l'utilizzo da parte degli utenti.

1.2.4 Velocità

Durante il suo concepimento Swift venne ideato per essere rapido: sfruttando gli hardware moderni il compilatore molto potente di Swift permette di identificare gli errori all'interno del codice in maniera rapida. Le librerie e le varie sintassi, inoltre, sono scritte in modo da rendere la programmazione la più ovvia possibile e per garantire, comunque, prestazioni elevate quando il codice viene eseguito da qualsiasi dispositivo.

1.3 Pro e contro del linguaggio Swift

In questa sezione della tesi parleremo dei motivi che rendono Swift un eccellente linguaggio di programmazione e delle cose che potrebbero, invece, essere migliorate.

1.3.1 Pro

Di seguito verranno elencati gli aspetti positivi di Swift che spingono i programmatori ad utilizzarlo.

Facilità di gestione di progetti in team

Con Swift possono essere aggiunti più sviluppatori allo stesso progetto inserendoli nel team di programmazione. Inoltre, grazie alla grande leggibilità del codice, il lavoro in contemporanea risulta più semplice poichè questo risulta scritto simile al normale inglese.

Sintassi semplice e breve

Essendo simile all'inglese, come detto sopra, anche la codifica in Swift risulta più semplice, chiara e veloce; un altro vantaggio di questo linguaggio di programmazione è il fatto che la sintassi sia spesso concisa per effettuare anche operazioni molto complesse. Un esempio lampante è il caso dell'applicazione Lyft: l'applicazione, infatti, dopo essere stata scritta in Objective-C venne riprogrammata utilizzando Swift a seguito del suo rilascio. Mentre la prima programmazione dell'app richese intorno alle 75000 linee di codice, quella in Swift venne scritta con meno di 25000 ed implementò anche delle nuove funzionalità rispetto alla precedente. Infine, la prima programmazione di Lyft richiese più di un mese e vari ingegneri per essere ultimata, mentre la sua versione Swift venne conclusa in una settimana e da un solo ingegnere.

Ottima gestione della memoria

Come sappiamo, scrivere un'applicazione richiede spesso l'utilizzo di molto codice copiato da programmatori terzi e l'importazione di varie librerie. Per importare le librerie bisogna spesso compilarle insieme al codice e questo appesantisce e rallenta la nostra applicazione. Nel caso di Swift però, tutte le librerie standard sono già integrate in ogni dispositivo Apple e di conseguenza verranno richiamate dalla nostra applicazione solo quando e se necessarie risparmiando alla nostra applicazione tempo di calcolo. Inoltre, la gestione della memoria interna all'applicazione utilizza la tecnologia *ARC* (Automatic Reference Counting) che permette di tenere conto di quali istanze delle nostre variabili non sono più in uso e di eliminarle risparmiando memoria e *CPU*.

Maggior velocità nella scrittura e nell'esecuzione del codice

Quando Swift fu ideato venne creato per essere migliore del suo predecessore: al primo rilascio infatti venne affermato che le performance erano del 40 % superiori a Objective-C. Questo perchè Swift utilizza il compilatore *LLVM*, che premette di trasformare il linguaggio assembly in linguaggio macchina ottimizzando il codice. Un altro motivo che permette una maggior velocità nella scrittura del codice è il fatto che Swift abbia un ciclo di risposta più breve e permetta, quindi, di vedere gli errori direttamente nel codice senza bisogno di compilarlo consentendo una correzione più rapida.

Facilità di apprendimento

Uno dei motivi che spinge molti progrmmatori ad intraprendere l'uso di Swift è la facilità del suo apprendimento; infatti, la chiarezza e la semplicità della sintassi fanno sì che anche i programmatori inesperti riescano a produrre righe di codice in breve tempo. Inoltre, la community di Swift è molto attiva, e questo permette di trovare risposte o esempi di codice online con semplicità. Swift era infatti il quarto linguaggio più sviluppato su GitHub dopo Go, TypeScript e Rust nel 2021 (vedi figura 1.3).

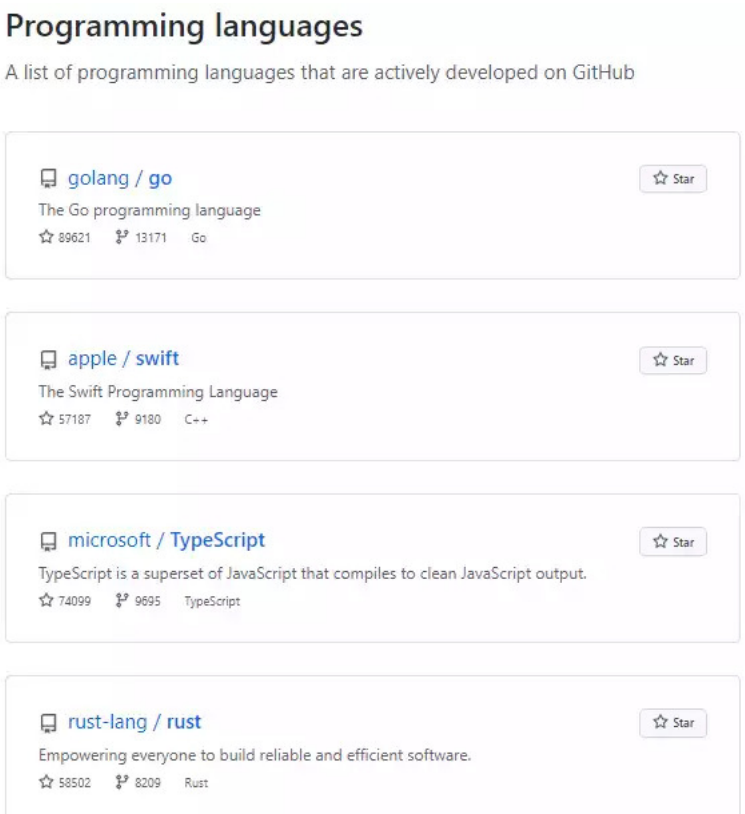

Figura 1.3: Lista dei linguaggi di programmazione più sviluppati su Github 2021 fonte: github.com

1.3.2 Contro

Di seguito verranno elencati gli aspetti negativi di Swift.

Linguaggio molto giovane

Essendo stato rilasciato nel 2014, Swift è ancora molto giovane; di conseguenza, capita spesso che nei frequenti aggiornamenti del linguaggio vengano cambiate alcune delle sue componenti. Questo, tuttavia rende incerta la programmazione in Swift poichè è possibile che del codice scritto oggi non sia più compatibile con le versioni future del linguaggio.

Comunity ristretta

Nonostante la sua popolarità ed il suo rapido sviluppo la community di Swift è ancora piccola rispetto a quelle dei più famosi linguaggi di programmazione (vedi figura 1.4). Risulta quindi più difficile trovare programmatori a cui chiedere aiuto o esempi di codice che svolgono le operazioni a noi utili.

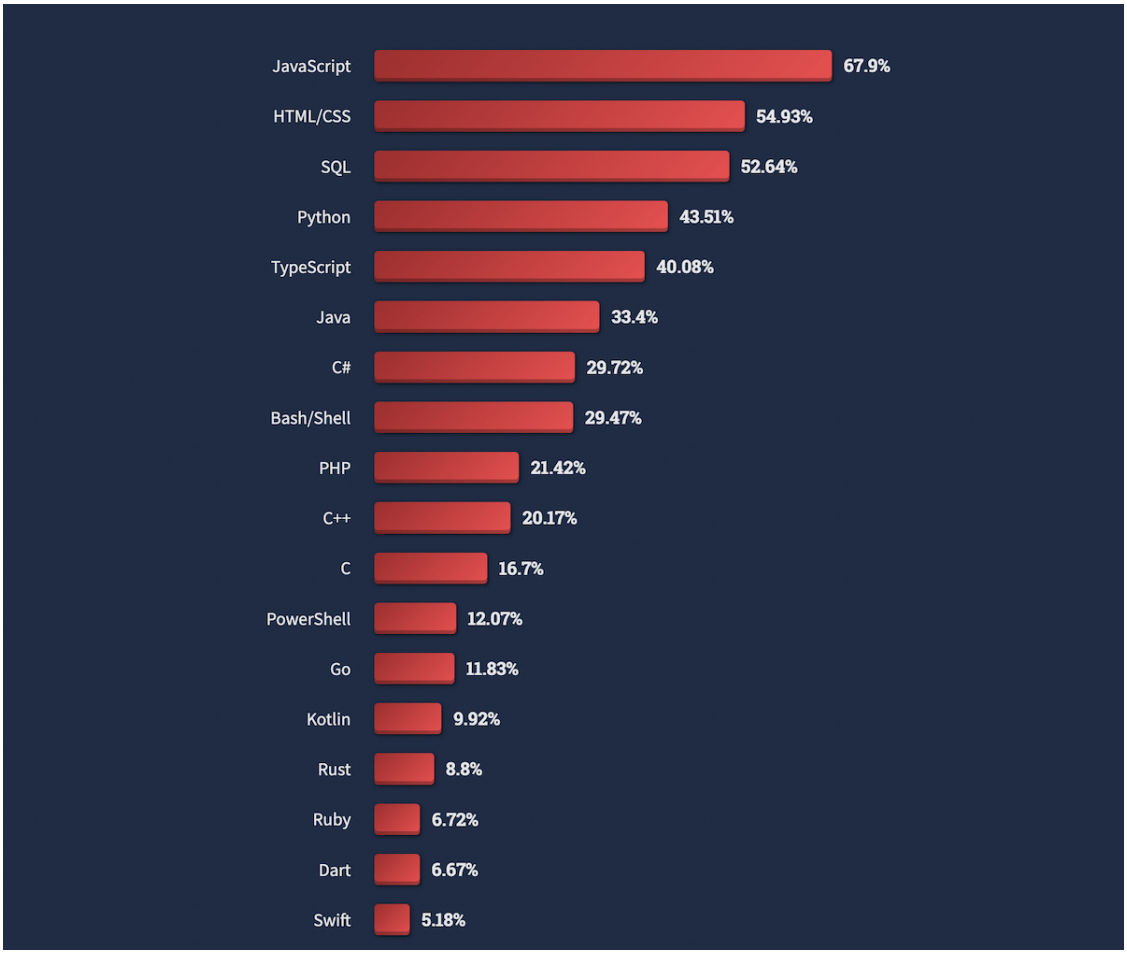

Figura 1.4: Lista dei linguaggi di programmazione più utilizzati nel 2022 fonte: StackOverflow

Bassa interoperatività con gli IDE

A causa dei numerosi e frequenti aggiornamenti di Swift capita spesso che la segnalazione degli errori o degli avvertimenti da parte dell'IDE ufficiale Apple XCode sia errata o difficilmente interpretabile. Inoltre, non sono ancora molti gli strumenti di supporto per Swift presenti all'interno di XCode.

Scarso supporto cross-platform

Nonostante Swift supporti tutte le piattaforme, le sue migliori performance si hanno nelle piattaforme Apple. Lo sviluppo su Linux e Windows è ancora molto difficile e macchinoso, risultando quindi preferibile ricorrere ad altri linguaggi di programmazione.

1.4 Specifiche per la programmazione

In questa sezione approfondiremo la parte pratica di Swift, riportando alcuni esempi di codice. Ovviamente, vengono riportate solo le funzionalità principali a scopo dimostrativo. Per approfondire qualsiasi delle seguenti sezioni è possibile consultare la documentazione di Swift al sito *swift.org*.

1.4.1 Variabili semplici

Per dichiarare una variabile abbiamo due scelte: la parola chiave `let` che permette di dichiarare una costante (variabile non modificabile) e la parola chiave `var` che permette, invece, di dichiarare una variabile modificabile (vedi figura 1.5). Durante la creazione di una variabile non occorre indicarne il tipo, ma è comunque possibile specificarlo tramite l'operatore : (vedi figura 1.6). Per dichiarare dei vettori basta inserire i valori degli elementi del vettore all'interno di parentesi quadre. È possibile dichiarare anche dei dizionari inserendo le chiavi ed i valori separati da : all'interno di parentesi quadre. Per accedere ad uno specifico valore di un vettore è necessario inserire tra parentesi quadre l'indice al quale il valore si trova, mentre per accedere ad un valore di un dizionario dobbiamo inserirvi la chiave corrispondente (vedi figura 1.7).

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

Figura 1.5: Dichiarazione di variabile utilizzando `let` e `var`

```
let implicitInteger = 70
let implicitDouble = 70.0
let explicitDouble: Double = 70
```

Figura 1.6: Diversi tipi di dichiarazione di variabile

1.4.2 Operatori condizionali

Per effettuare operazioni condizionali utilizziamo l'operatore `if` oppure l'operatore `switch` mentre per effettuare dei cicli posso usare gli operatori `for-in`, `while` o `repeat-while`. La condizione all'interno dell'`if` deve essere un'espressione booleana (vedi figura 1.8). L'operatore `for-in`, permette di iterare un'operazione per tante volte quanti sono gli oggetti all'interno di un vettore o di un dizionario (vedi figura 1.9), mentre l'operatore `while` o `repeat-while` itera finchè non viene soddisfatta la condizione specificata (vedi figura 1.10)

```
var fruits = ["strawberries", "limes", "tangerines"]
fruits[1] = "grapes"

var occupations = [
    "Malcolm": "Captain",
    "Kaylee": "Mechanic",
 ]
occupations["Jayne"] = "Public Relations"
```

Figura 1.7: Creazione e operazioni con vettori

```
let scoreDecoration = if teamScore > 10 {
    "🎉"
} else {
    ""
}
print("Score:", teamScore, scoreDecoration)
```

Figura 1.8: Esempio di utilizzo dell'operatore `if`

1.4.3 Metodi e funzioni

Per dichiarare una funzione utilizziamo la parola chiave `func` seguita dal nome della funzione e dalla lista degli argomenti richiesti da questa all'interno di parentesi tonde. Se la funzione deve restituire dei valori, questi vanno specificati di seguito preceduti dall'operatore `->` (vedi figura 1.11); se non viene specificato il tipo di valore da restituire la funzione non restituirà nessun valore. Se si vogliono ottenere più valori come risultato di una funzione, vanno utilizzate delle tuple (vedi figura 1.12). Si può dichiarare una funzione dentro un'altra, richiamare una funzione all'interno di un'altra e richiedere come parametro una funzione all'interno di un'altra.

1.4.4 Oggetti e classi

Per creare una classe va utilizzata la parola chiave `class` seguita dal nome della classe. La dichiarazione di proprietà all'interno della classe avviene come una normale dichiarazione di variabile e la dichiarazione di un metodo avviene come la normale dichiarazione di una funzione. All'interno di una classe si possono dichiarare il suo costruttore con la parola chiave `init` seguita da parentesi tonde all'interno delle quali devono essere inseriti gli argomenti richiesti dal costruttore (come una normale funzione). Per creare un'istanza di una classe basta richiamare il nome della classe seguito da parentesi tonde, nel caso in cui la classe abbia un costruttore; all'interno delle parentesi devono essere inserite le variabili da questo richieste. Per fare riferimento a delle proprietà della classe al suo interno va utilizzato `self`, mentre per accedervi dall'esterno deve essere richiamata l'istanza della classe seguita dall'operatore `.` seguito dal nome della proprietà che si vuole richiamare.

```swift
let interestingNumbers = [
    "Prime": [2, 3, 5, 7, 11, 13],
    "Fibonacci": [1, 1, 2, 3, 5, 8],
    "Square": [1, 4, 9, 16, 25],
]
var largest = 0
for (_, numbers) in interestingNumbers {
    for number in numbers {
        if number > largest {
            largest = number
        }
    }
}
print(largest)
// Prints "25"
```

Figura 1.9: Esempio utilizzo dell'operatore `for-in`

1.4.5 Funzioni asincrone

Per identificare una funzione come asincrona va utilizzata la parola chiave `async` (vedi figura 1.16); per richiamare una funzione di questo tipo va utilizzata la parola chiave `await` (vedi figura 1.17). Per far svolgere più funzioni contemporaneamente va utilizzato `async let` (vedi figura 1.18). Per richiamare del codice asincrono all'interno di codice sincrono va usato `Task` (in questo caso, il risultato delle funzioni asincrone non verrà atteso prima di proseguire col resto del codice).

1.4.6 Gestione degli errori

Per la rappresentazione degli errori vanno utilizzate classi che estendono il protocollo `Error` (vedi figura 1.19). Si usa la parola chiave `throw` per richiamare un errore e `throws` per segnalare che una funzione può richiamare un errore (vedi figura 1.20). Per la gestione degli errori va usato il costrutto `do...catch...`: all'interno del primo blocco va scritto il codice che potrebbe richiamare un errore, mentre all'interno del secondo blocco va scritto il codice che verrà eseguito solo in caso di errore (vedi figura 1.21). Prima del blocco `chatch` si può specificare il tipo di errore che farà entrare in funzione il codice all'interno del blocco (sono quindi possibili più `catch` per lo stesso `do`). Un altro modo per gestire gli errori è utilizzare `try?`: nel caso in cui si incorra in un errore il valore restituito dalla funzione preceduta dal `try?` viene assegnato a `nil`.

1.5 SwiftUI

1.5.1 Introduzione

Per creare la parte grafica della nostra applicazione abbiamo utilizzato SwiftUI. Questo è un kit di strumenti per creare delle interfacce utente per qualsiasi piattaforma Apple in maniera semplice e veloce. La peculiarità di SwiftUI è che le interfacce vengono create in maniera dichiarativa: il programmatore descrive l'interfaccia che vuole creare con le componenti fornite da SwiftUI che, poi, da queste crea l'interfaccia. Inoltre SwiftUI ci

```
var n = 2
while n < 100 {
    n *= 2
}
print(n)
// Prints "128"

var m = 2
repeat {
    m *= 2
} while m < 100
print(m)
// Prints "128"
```

Figura 1.10: Esempio di utilizzo dell'operatore `while` e `repeat-while`

```
func greet(person: String, day: String) -> String {
    return "Hello \(person), today is \(day)."
}
greet(person: "Bob", day: "Tuesday")
```

Figura 1.11: Esempio di dichiarazione di una funzione

fornisce anche vari strumenti per gestire le interazioni dell'utente con l'interfaccia. Di seguito parleremo degli aspetti positivi e di quelli negativi di SwiftUI senza entrare nel dettaglio con esempi di codice. Per approfondire questo aspetto, accedere al seguente link: `https://developer.apple.com/tutorials/swiftui`.

1.5.2 Aspetti positivi di SwiftUI

In questa sezione elencheremo le funzionalità che hanno spinto noi e molti programmatori ad utilizzare SwiftUI.

Bellezza delle componenti

Nel kit di SwiftUI sono presenti tutte le componenti importanti che permettono di costruire un'app completa. Inoltre, queste componenti offrono dei design molto belli e moderni già di default, e la loro interazione risulta molto semplice grazie all'utilizzo dell'interfaccia dichiarativa, come accennato sopra.

Estrema velocità e semplicità

In SwiftUI le interfacce grafiche vengono create in maniera dichiarativa, senza quindi, dover cambiare manualmente la grandezza dei 'frame' per le nostre viste e senza dover utilizzare i constraint per posizionare vari elementi dove li desideriamo. Di conseguenza, risulta molto semplice sia la creazione di interfacce che la loro gestione in seguito ad interazioni dell'utente.

```swift
func calculateStatistics(scores: [Int]) -> (min: Int, max: Int, sum: Int) {
    var min = scores[0]
    var max = scores[0]
    var sum = 0

    for score in scores {
        if score > max {
            max = score
        } else if score < min {
            min = score
        }
        sum += score
    }

    return (min, max, sum)
}
let statistics = calculateStatistics(scores: [5, 3, 100, 3, 9])
print(statistics.sum)
// Prints "120"
print(statistics.2)
// Prints "120"
```

Figura 1.12: Esempio di funzione che restituisce più valori

```swift
class Shape {
    var numberOfSides = 0
    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
```

Figura 1.13: Esempio di dichiarazione di una classe senza costruttore

Sviluppo Cross-Platform

Un altro aspetto positivo di SwiftUI è il fatto che, una volta progettata un'interfaccia, questa possa essere riutilizzata per tutti i dispositivi iOS. Purtroppo questo non avviene anche per i dispositivi Android e Windows.

Le interfacce sono completamente reattive

Grazie all'utilizzo di proprietà come `@State`, `@Binding` o `@Published` da assegnare alle nostre variabili queste possono essere osservate dall'interfaccia grafica. Essendo questa reattiva, una volta che le variabili variano, l'interfaccia cambia "magicamente" insieme a loro, anche con animazioni preimpostate, creando dei notevoli effetti grafici con poche e semplici righe di codice.

La community

Anche nel caso di SwiftUI la community è molto attiva e, quindi, risulta semplice venire introdotti ed inziare lo sviluppo con questo toolkit. Inoltre, è possibile trovare vari esempi online di interfacce da cui prendere ispirazione per realizzare le proprie.

```
var shape = Shape()
shape.numberOfSides = 7
var shapeDescription = shape.simpleDescription()
```

Figura 1.14: Esempio di chiamata ad una classe

```
class Shape {
    var numberOfSides = 0
    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
```

Figura 1.15: Esempio di dichiarazione di una classe con costruttore, metodo e proprietà

1.5.3 Aspetti negativi di SwiftUI

Essendo ancora molto "giovane" SwiftUI ha ancora varie pecche al suo interno che verranno sicuramente migliorate col tempo ma che, ad oggi, risultano come aspetti negativi.

Difficoltà del compilatore

Se viene creata una vista troppo complessa con degli errori al suo interno, questi smetteranno di essere segnalati correttamente e verranno, invece, forniti degli errori poco esplicativi su righe di codice funzionanti, rendendo difficile la correzione dell'errore.

Non tutte le componenti sono presenti

In SwiftUI non sono ancora presenti alcune delle componenti essenziali per costruire delle interfacce grafiche, e alcune delle componenti presenti sono troppo semplici o mancano di caratteristiche fondamentali per il loro utilizzo. Di seguito faremo un elenco di queste componenti.

Interazioni con le Gesture

In SwiftUI è incluso un set di Gesture che possono essere aggiunte alle viste per permettere all'utente di interagire in svariate maniere con le interfacce. A volte, però, queste possono andare in conflitto tra loro o con elementi grafici dell'interfaccia (ad esempio `DragGesture` con `ScrollView` impedisce di scorrere con il dito come sarebbe normale).

Gestione della navigazione

La gestione della navigazione tra le varie interfacce risulta molto complicata e macchinosa. Nel nostro caso, per esempio, abbiamo utilizzato il *NavigationLink*, metodo che risulta, però, deprecato dopo iOS 16. Questo perchè il nuovo metodo risulta molto più complesso del precedente e non è esente da errori di vario tipo. Con il *navigationLink* si incorre, comunque, in strani errori: per esempio, volendo passare da una vista con un titolo ad un'altra, nella vista successiva sarà comunque lasciato dello spazio vuoto per il titolo anche se questo non è presente.

```swift
func fetchUserID(from server: String) async -> Int {
    if server == "primary" {
        return 97
    }
    return 501
}
```

Figura 1.16: Esempio di funzione asincrona

```swift
func fetchUsername(from server: String) async -> String {
    let userID = await fetchUserID(from: server)
    if userID == 501 {
        return "John Appleseed"
    }
    return "Guest"
}
```

Figura 1.17: Esempio di chiamata ad una funzione asincrona

ScrollView

In generale la **ScrollView** serve per inserire vari elementi in un'interfaccia permetendoci di scorrere per visualizzare tutti i suoi contenuti. Purtroppo in SwiftUI si riscontrano vari problemi. Innanzitutto, nel caso si abbiano molti elementi all'interno di `ScrollView`, non è previsto un metodo per dividerli in più pagine. Occorre, quindi, ricorrere ad una `TabView` e dividere la lista dei nostri elementi in liste più piccole per visualizzarle in maniera adeguata. Inserendo un `LazyVStack` in una `ScrollView` si ottengono vari errori o strani comportamenti dell'interfaccia. Infine, non esiste un supporto per capire come l'utente sta scorrendo sulla schermata (ad esmempio, per capire se, sto utilizzando lo scroll, o sto trascinando).

```swift
func connectUser(to server: String) async {
    async let userID = fetchUserID(from: server)
    async let username = fetchUsername(from: server)
    let greeting = await "Hello \(username), user ID \(userID)"
    print(greeting)
}
```

Figura 1.18: Esempio di utilizzo di `async let`

```swift
enum PrinterError: Error {
    case outOfPaper
    case noToner
    case onFire
}
```

Figura 1.19: Esempio di dichiarazione di un errore

```swift
func send(job: Int, toPrinter printerName: String) throws -> String {
    if printerName == "Never Has Toner" {
        throw PrinterError.noToner
    }
    return "Job sent"
}
```

Figura 1.20: Esempio di chiamata e dichiarazione di una funzione con errore

```swift
do {
    let printerResponse = try send(job: 1040, toPrinter: "Bi Sheng")
    print(printerResponse)
} catch {
    print(error)
}
```

Figura 1.21: Esempio di utilizzo del costrutto `do-catch`

CAPITOLO 2

Specifica e Analisi dei requisiti

In questo capitolo ci concentreremo specificamente sui requisiti dell'applicazione che svilupperemo. In particolare, inizialmente, verranno brevemente introdotti lo scopo dell'applicazione, il pubblico alla quale è rivolta ed i riferimenti dai quali prenderemo spunto per la realizzazione. In seguito, analizzeremo gli obiettivi che ci siamo posti pensando all'applicazione e, successivamente, definiremo dei termini che ci risulternanno utili nel resto della trattazione per evitare ambiguità.

Nella sezione successiva verrà invece fornita una descrizione più dettagliata dell'applicazione Unifind, comprendendone le funzionalità, le caratteristiche che gli utenti devono avere, i vincoli presenti e le ipotesi richieste per utilizzare l'applicazione in maniera adeguata.

Infine, elencheremo e descriveremo i requisiti dividendoli in requisiti funzionali, non funzionali e di interfaccia. Concludiamo, sottolineando e spiegando alcune caratteristiche dell'applicazione.

2.1 Introduzione

2.1.1 Scopo

Il proposito che ci poniamo è quello di sviluppare un'applicazione per facilitare la ricerca e l'affitto di abitazioni da parte di studenti universitari. Per farlo, intendiamo fornire agli utenti non solo la possibilità di avere informazioni sull'immobile, ma anche quella di chattare con altri studenti interessati allo stesso, così da rendere più semplice l'interazione tra individui con le stesse esigenze.

2.1.2 Pubblico

Il pubblico al quale è rivolta questa applicazione è maggiormente quello universitario fuori sede interessato all'affitto di case per la durata dell'anno accademico. Ovviamente, l'applicazione può essere utilizzata anche da utenti che non siano studenti universitari, ma che cercano una casa in affitto per almeno 9 mesi, come, per esempio, giovani lavoratori fuori sede.

2.1.3 Riferimenti

Come riferimenti per l'applicazione ci siamo ispirati a vari siti immobiliari, in particolare vogliamo citare Idealista che, oltre a fornirci un'ispirazione sulle funzionalità ed il design della nostra applicazione, ci ha anche fornito un accesso alle proprie API per effettuare la ricerca di abitazioni in affitto.

2.1.4 Obiettivo

Il nostro obiettivo è la creazione di un'applicazione con le seguenti funzionalità:

1. *Ricerca di abitazioni tramite barra di ricerca su una mappa:* l'utente può ricercare delle abitazioni digitando un luogo nella barra di ricerca.
2. *Visualizzazione di abitazioni sulla mappa:* le abitazioni che soddisfano i parametri di ricerca verranno segnalate sulla mappa con dei *marker* appositi.
3. *Ricerca di abitazioni tramite il click sulla mappa:* l'utente può ricercare delle abitazioni utilizzando il suo click sulla mappa.
4. *Filtraggio sulle ricerche:* l'utente può filtrare le proprie ricerche.
5. *Visualizzazione della lista delle abitazioni:* l'utente può visualizzare i tutti i risultati della sua ricerca su una lista.
6. *Lista dei preferiti:* l'utente può salvare delle abitazioni che potrà visualizzare in seguito nella sua lista dei preferiti.
7. *Visualizzazione dei dettagli delle abitazioni:* l'utente può visualizzare i dettagli di tutte le abitazioni.
8. *Chat:* l'utente può chattare con altri utenti che abbiano almeno un'abitazione salvata a lui comune.
9. *Visualizzazione delle informazioni dell'utente:* l'utente può visualizzare le informazioni degli altri utenti con cui sta chattando.
10. *Gestione delle informazioni dell'utente:* ogni utente può gestire le proprie informazioni (modifica e visualizzazione).
11. *Registrazione di un utente:* l'utente deve registrarsi prima di poter utilizzare l'applicazione (e-mail e password).
12. *Login dell'utente:* l'utente deve effettuare la procedura di login quando avvia l'applicazione prima di utilizzarla (e-mail e password).

2.1.5 Acronimi e definizioni

- *abitazione:* immobile che può essere affittato dall'utente qualsiasi esso sia (casa, appartamento, villa...)
- *Ricerca:* Richiesta API ad Idealista che restituisce delle abitazioni.
- *Preferiti:* lista delle abitazioni salvate dal singolo utente.
- *Utente:* chiunque acceda all'applicazione.
- *Mappa:* interfaccia che permette di visualizzare la cartina geografica con le apposite funzionalità.
- *Lista:* elenco di abitazioni visualizzate in formato di lista.
- *Profilo:* insieme delle informazioni di un utente.

- *Centro:* centro della ricerca che l'utente effettua da cui vengono prese le coordinate geografiche di partenza per limitare la massima distanza entro la quale vengono restituite delle abitazioni.
- *Marker:* segnaposto sulla mappa che rappresenta un'abitazione.

2.2 Descrizione generale

2.2.1 Funzionalità del prodotto

L'applicazione *Unifind* deve includere le seguenti funzionalità:

- salvataggio dei dati di ricerca di un utente;
- visualizzazione delle abitazioni sulla mappa;
- visualizzazione delle informazioni generali delle abitazioni dalla lista;
- visualizzazione della totalità delle informazioni dall'apposita interfaccia;
- visualizzazione della lista totale dei preferiti per utente;
- salvataggio dei preferiti per utente;
- rimozione dei preferiti per utente;
- filtraggio per prezzo massimo sulle abitazioni;
- filtraggio per numero minimo di posti letto sulle abitazioni;
- filtraggio per distanza massima dal centro;
- ordinamento delle abitazioni per prezzo (crescente/decrescente);
- reindirizzamento alla casa selezionata dopo il click su un marker;
- modifica e-mail dal profilo utente;
- modifica password dal profilo utente;
- inserimento e modifica dell'immagine di profilo;
- inserimento e modifica della descrizione dal profilo utente;
- inserimento e modifica del nome e del cognome dal profilo utente;
- inserimento e modifica del numero di telefono dal profilo utente;
- possibilità di chattare tra tutti gli utenti con almeno un'abitazione in comune;
- possibilità di visualizzare tutte le abitazioni in comune con l'utente con cui si sta chattando;
- visualizzazione dell'abitazione dalla quale si è avviata la chat (quindi, quella di cui teoricamente si è parlato all'inizio della chat);
- login utente all'avvio dell'app;
- registrazione utente al primo utilizzo dell'app.

2.2.2 Caratteristiche utente

L'applicazione *Unifind* è rivolta ad un'utenza con età compresa tra i 18 ed i 30 anni e provvista di dispositivi Apple per il suo utilizzo. L'utenza sopracitata non deve, ovviamente, avere nessun tipo di conoscenza informatica per poter utilizzare l'applicazione.

2.2.3 Vincoli generali

I vincoli dell'applicazione sono i seguenti:

- l'utente deve inserire i parametri per una ricerca prima di poter visualizzare delle abitazioni nella sezione lista;
- la password inserita dall'utente deve essere di almeno 8 caratteri; l'email deve essere valida.
- le abitazioni visualizzate dall'utente sono tutte disponibili per l'affito per almeno 9 mesi.

2.2.4 Ipotesi e dipendenze

L'applicazione Unifind dovrà essere utilizzata su dispositivi iOS con versione successiva ad iOS 13.0.0. oppure su dispositi tvOS o watchOS con l'ultima versione disponibile installata. L'utente deve disporre di una connessione internet.

2.3 Requisiti e caratteristiche del sistema

2.3.1 Requisiti funzionali

- *Verifica della password :* la password deve contenere almeno 8 caratteri per essere valida e va inserita 2 volte in maniera identica.
- *Verifica dell'e-mail:* l'indirizzo e-mail inserito deve essere valido.
- *Gestione del numero di telefono per ogni utente:* ogni utente può inserire, rimuovere e modificare il proprio numero di telefono (che deve comunque essere valido).
- *Gestione del nome e del cognome:* ogni utente può inserire, rimuovere e modificare il proprio nome e cognome.
- *Gestione della descrizione:* ogni utente può inserire, rimuovere e modificare la propria descrizione.
- *Gestione dell'immagine di profilo:* ogni utente può inserire, rimuovere e modificare la propria immagine di profilo.
- *Inserimento del centro per la ricerca delle abitazioni:* l'utente deve inserire le coordinate geografiche del centro del cerchio (che ha come raggio la distanza massima selezionata sui filtri) all'interno del quale vuole ricercare le abitazioni. Questo può avvenire tramite un click sulla mappa o tramite la ricerca sull'apposita barra.
- *Inserimento dei parametri di ricerca:* l'inserimento degli altri parametri di ricerca è facoltativo per l'utente: nel caso in cui questi non vengano inseriti, saranno assegnati di default dall'applicazione. Nel caso in cui l'utente voglia modificarli, potrà farlo dalla sezione dei filtri (i parametri di ricerca sono: prezzo massimo, numero minimo di inquilini,

distanza massima dal centro ed ordinamento di prezzo crescente o decrescente delle abitazioni visualizzate).

- *Visualizzazione delle abitazioni su mappa:* l'utente, dopo aver effettuato una ricerca, deve poterne visualizzare il risultato sulla mappa.
- *Visualizzazione delle abitazioni su lista:* l'utente, dopo aver effettuato una ricerca, deve poterne visualizzare il risultato sotto forma di lista di abitazioni.
- *Salvataggio dei risultati di ricerca:* l'utente, dopo aver effettuato una ricerca, potrà visualizzarne il risultato senza doverla svolgere nuovamente. I dati saranno salvati sulla cache del dispositivo dove si sta utilizzando l'applicazione.
- *Rating per le abitazioni:* l'utente può inserire un rating ad un'abitazione (da 0 a 5).
- *Visualizzazione del rating medio:* l'utente può visualizzare la media dei rating inseriti dagli altri utenti su un'abitazione.
- *Possibilità di memorizzare delle abitazioni:* l'utente può salvare delle abitazioni che può visualizzare in seguito da un'apposita lista.
- *Gestione della lista dei preferiti:* l'utente può visualizzare ed eliminare abitazioni dalla sua lista dei preferiti, tramite la sezione apposita.
- *Visualizzazione dettagliata di ogni abitazione:* l'utente può visualizzare nel dettaglio le informazioni relative ad ogni abitazione presente sulla lista, sulla mappa e sui preferiti.
- *Possibilità di chat con altri utenti del sito:* l'utente può chattare con altri utenti del sito, purchè questi abbiano tra i preferiti almeno un'abitazione, fra quelle che l'utente ha salvato.
- *Visualizzazione delle abitazioni in comune dalla chat:* dalla schermata della chat, dati due utenti, si possono vedere tutte le abitazioni salvate che questi hanno in comune.
- *Visualizzazione informazioni utente dalla chat:* dalla schermata della chat, un utente può visualizzare le informazioni relative all'utente con cui sta chattando.
- *Autenticazione all'avvio dell'app:* quando avvia l'app, l'utente deve accedere con il proprio account, tramite l'inserimento di e-mail e password, o registrarsi mediante l'inserimento di e-mail e password con reinserimento di quest'ultima. I dati inseriti verranno validati secondo le specifiche sopra indicate.

2.3.2 Requisiti di interfaccia grafica

- *MapView*
  - visualizzazione della mappa scorribile;
  - visualizzazione dei marker relativi alle abitazioni;
  - visualizzazione dei marker (differenti dai precedenti) relativo al centro geografico della ricerca.

- *SignInView*
  - `TextField` per inserire l'email;
  - `TextField` per inserire la password.

- *SignUpView*
  - `TextField` per inserire l'email;
  - `TextField` per inserire la password;
  - `TextField` per confermare la password.

- *ListView*
  - pulsante per aprire la vista dei filtri.
  - una Card per ogni abitazione, contenente:
    * immagine dell'abitazione;
    * rating medio dell'abitazione;
    * città in cui si trova l'abitazione;
    * indirizzo dell'abitazione;
    * prezzo mesile dell'abitazione;
    * pulsante per salvare un'abitazione tra i preferiti.

- *FilterView*
  - Barra per selezionare il prezzo massimo.
  - Barra per selezionare la distanza massima dal centro.
  - Scelta sull'ordinamento per prezzo (ascendente/decrescente);
  - Menù a tendina per scelta del numero di coinquilini.

- *FavouritesView*
  - una Card per ogni abitazione, contenente:
    * immagine dell'abitazione;
    * rating medio dell'abitazione;
    * città in cui si trova l'abitazione;

    * indirizzo dell'abitazione;
    * prezzo mensile dell'abitazione;
    * pulsante per eliminare una abitazione dai preferiti.

- *MoreInformationView*

  - Immagine dell'abitazione.
  - Prezzo dell'abitazione.
  - Città della abitazione.
  - Indirizzo dell'abitazione.
  - Numero di camere da letto dell'abitazione.
  - Numero di bagni dell'abitazione.
  - Descrizione dell'abitazione.
  - Pulsante per inserire il rating all'abitazione.
  - Nel caso in cui l'abitazione sia salvata, pulsante per chattare con altri utenti che hanno salvato la stessa abitazione.

- *ChatView*

  - contatto selezionabile per ogni utente che abbia la casa tra i preferiti.

- *MessageView*

  - una Card per ogni abitazione, contenente:
    * immagine dell'abitazione;
    * città in cui si trova l'abitazione;
    * indirizzo della abitazione;
    * numero di altre abitazioni in comune con l'utente con cui si sta chattando.
  - `TextField` per inserire il messaggio prima di inviarlo.
  - Pulsante per l'invio del messaggio.
  - E-mail dell'utente con cui si sta chattando.
  - `TextView` contenente messaggi ricevuti ed inviati (di colore diverso e posizionata diversamente sullo schermo).

  .

- *ProfileView*

  - Immagine di profilo dell'utente.
  - E-mail dell'utente.
  - Nome e cognome dell'utente.
  - Numero di telefono dell'utente.
  - Breve descrizione dell'utente.
  - Pulsanti per modificare tutti i campi del profilo.

- *ChatUserProfileView*

  - Immagine di profilo dell'utente.
  - E-mail dell'utente.
  - Nome e cognome dell'utente;
  - Numero di telefono dell'utente;
  - Breve descrizione dell'utente.

2.3.3 Requisiti non funzionali

- *Sviluppo dell'applicazione in Swift:* lo sviluppo della parte di backhand dell'applicazione avverrà utilizzando il linguaggio di programmazione Swift.
- *Utilizzo di Firebase per la gestione dei dati:* la gestione dei dati dell'applicazione avverrà utilizzando Firebase (in particolare Firebase Storage, Firebase Database e Firebase Authentication).

2.3.4 Caratteristiche del sistema

- *Multiutenza:* l'applicazione può gestire l'accesso di utenti da diversi terminali contemporaneamente. L'accesso ai dati dell'applicazioni è anch'esso garantito a più dispositivi contemporaneamente consentendo ad ogni utente di visualizzare le informazioni relative alle operazioni da lui svolte.
- *Sicurezza:* l'accesso ad ogni profilo utente è protetto dalla funzione di Login obbligatoria quando viene avviata l'applicazione. È quindi possibile accedere alle informazioni di un utente solo conoscendone la password.

CAPITOLO 3

Progettazione

Nel capitolo corrente tratteremo la parte di progettazione dell'applicazione, partendo dalla struttura del database fino ad arrivare alle interfacce grafiche. Nella parte iniziale, si trovano il diagramma entità-relazione e quello logico dell'applicazione seguiti da una descrizione degli stessi. Nella parte successiva, si trovano, invece, tutti i diagrammi riguardanti la parte applicativa della progettazione: il diagramma delle classi, quello di deployment, il diagramma dei casi d'uso, il diagramma di sequenza e quello di attività. Il capitolo si chiude, infine, con i mockup dell'applicazione.

3.1 Progettazione concettuale

3.1.1 Diagramma entità relazione

In questa sezione viene rappresentato il diagramma entità-relazione del database (Figura 3.1) della nostra applicazione, che verrà implementato all'interno di Firebase Database. Nella sezione successiva, descriveremo nel dettaglio tale diagramma, spiegando il motivo delle scelte effettuate.

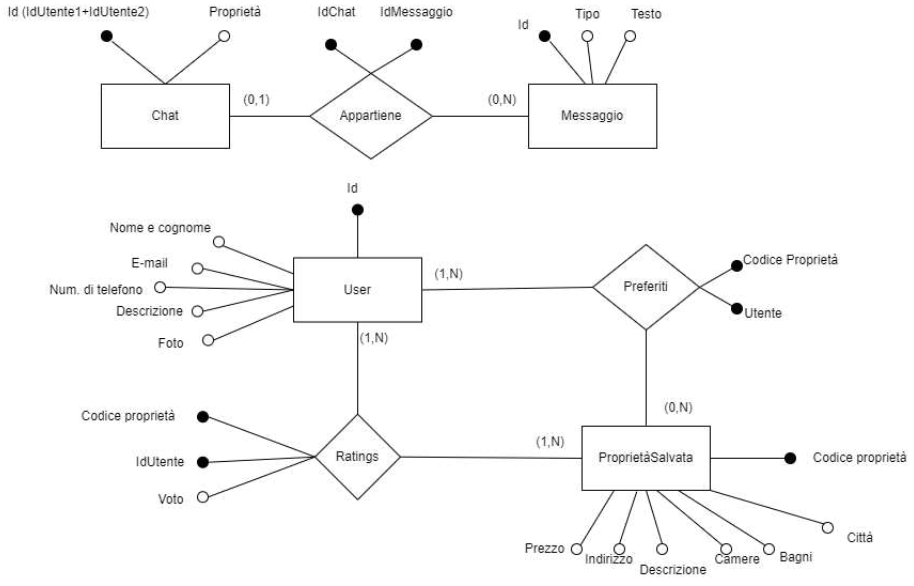

Figura 3.1: Diagramma E-R del database dell'applicazione

3.1.2 Analisi entità e relazioni

Il database rappresentato dal diagramma entità-relazione è semplice, poichè al suo interno non vengono salvate tutte le abitazioni disponibili nell'applicazione (queste, infatti, sono già salvate all'interno del database di Idealista che le fornisce a seguito delle richieste), ma soltanto quelle salvate da almeno un utente. L'entità per memorizzare le abitazioni è *ProprietàSalvata*, al cui interno sono salvati tutti gli attributi utili alla visualizzazione da parte degli utenti. In particolare, abbiamo i seguenti attributi:

- textitCodice Proprietà: il codice che identifica univocamente tutte le abitazioni. Questo ci viene fornito direttamente da Idealista e viene utilizzato come chiave primaria poichè, così facendo, è possibile salvare le abitazioni prese da Idealista direttamente sul nostro database. In questo modo, per operare sulle abitazioni da noi salvate e su quelle fornite da Idealista possiamo fare riferimento allo stesso codice univoco, semplificando molto le operazioni che le coinvolgono.
- *Prezzo:* il prezzo mensile dell'abitazione.
- *Indirizzo:* l'indirizzo dell'abitazione contenente via e numero civico.
- *Descrizione:* una descrizione dell'abitazione fornita dal suo proprietario.
- *Camere:* il numero di camere da letto presenti nell'abitazione.
- *Bagni:* il numero di bagni presenti nell'abitazione.
- *Città:* la città nella quale si trova l'abitazione.

Quest'entità è coinvolta nella relazione *Preferiti*, che serve a rappresentare quali abitazioni sono state salvate e da quali utenti. Infatti, all'interno di *Preferiti* troviamo solamente gli identificatori dell'utente e dell'abitazione salvata.

L'altra entità coinvolta in questa relazione, è l'entità *User*, che contiene i dati appartenenti ai singoli utenti. Al suo interno sono presenti i seguenti attributi:

- *Id:* il codice identificativo degli utenti, questo viene fornito da Firebase nel momento della creazione di un utente. Anche in questo caso, l'utilizzo del codice fornito da Firebase, ci permette di svolgere operazioni riguardanti Database e Authentication di Firebase utilizzando lo stesso codice identificativo.
- *Nome e Cognome:* nome e cognome dell'utente. Questo dato è facoltativo e, nel caso in cui l'utente non fornisca queste informazioni, rimarrà nullo.
- *E-mail:* l'indirizzo e-mail dell'utente che deve essere inserito al momento della registrazione.
- *Num. di telefono:* il numero di telefono dell'utente. Anche questo dato è facoltativo.
- *Descrizione:* la descrizione che l'utente fornisce di se stesso. Anche questo dato è facoltativo.
- *Foto:* L'immagine di profilo che l'utente inserisce. Anche questo dato è facoltativo.

Un'altra relazione che coinvolge entrambe le entità viste finora è *Ratings*, al cui interno sono presenti i dati riguardanti le recensioni che gli utenti lasciano alle abitazioni. Questa relazione, è separata dalla precedente poichè non tutte le abitazioni salvate da un utente devono essere recensite. Analizzando nel dettaglio tutti gli attributi abbiamo:

- *Codice proprietà:* il codice dell'abitazione che viene valutata.
- *IdUtente:* l'id dell'utente che valuta l'abitazione.
- *Voto:* il voto attribuito all'abitazione (può assumere valori da 0 a 5).

Separatamente dalle precedenti vi sono altre due entità coinvolte in una relazione. Una delle due entità è *Chat*, la quale si trova in relazione con l'entità *Messaggio*. All'interno di chat abbiamo soltanto 2 attributi:

- *Id:* identificatore univoco formato dagli id dei due utenti che stanno chattando tra loro concatenati.
- *Proprietà:* il codice dell'abitazione dalla quale si è aperta la chat.

La relazione *Appartiene* al proprio interno contiene solamente il riferimento alle due chiavi primarie delle entità coinvolte.
L'entità *Messaggio* al suo interno contiene 3 attributi:

- *Id:* identificatore univoco del messaggio.
- *Tipo:* identifica il messaggio come ricevuto o inviato da parte dell'utente che ha creato la chat.
- *Testo:* contiene il testo del messaggio.

3.2 Progettazione logica

3.2.1 Eliminazione delle gerarchie

Non essendo presenti gerarchie all'interno del nostro diagramma E-R non troviamo niente da riportare in questa sezione.

3.2.2 Schema logico

In questa sezione riporteremo lo schema logico del database, (Figura 3.2) che non attua cambiamenti dal diagramma E-R, poichè questo è normalizzato. Per lo stesso motivo, non riporteremo una descrizione, poichè risulterebbe molto simile a quella dello schema E-R.

3.3 Progettazione delle applicazioni

3.3.1 Diagramma delle classi

In questa sezione riportiamo i diagrammi delle classi dell'applicazione, dividendoli per categoria.

- *Gestione della Mappa:* il diagramma di gestione della mappa (Figura 3.3).
- *Gestione abitazioni:* il diagramma che rappresenta come vengono gestite le abitazioni (Figura 3.4).
- *Gestione chat:* il diagramma che rappresenta come vengono gestite le chat (Figura 3.5).
- *Gestione utenti:* il diagramma che rappresenta come vengono gestiti gli utenti (Figura 3.6).

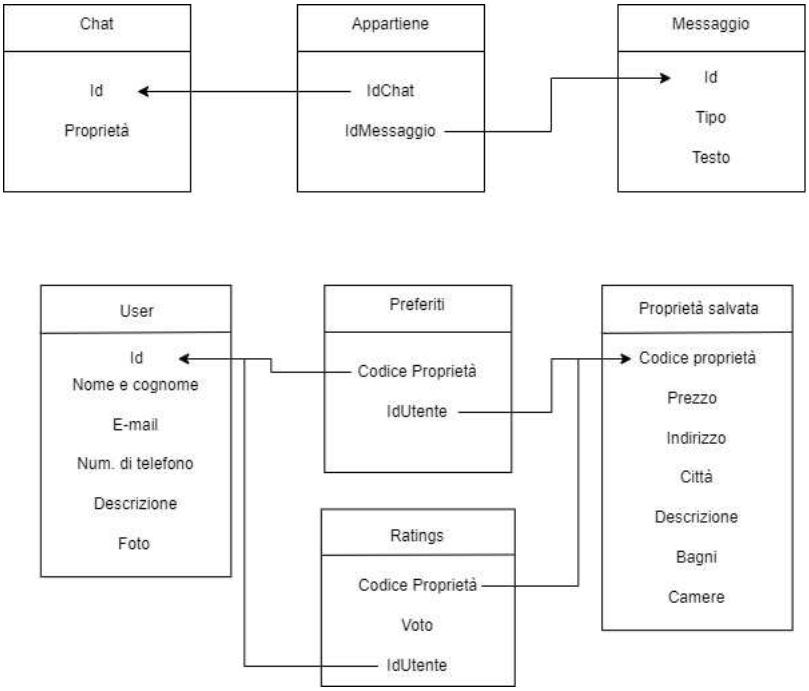

Figura 3.2: Diagramma logico del database dell'applicazione

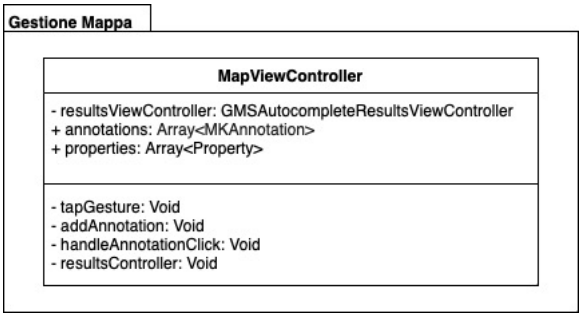

Figura 3.3: Diagramma delle classi relativo alla gestione della mappa

3.3.2 Diagramma di deployment

In questa sezione viene riportato il diagramma di deployment dell'applicazione (Figura 3.7).

3.3.3 Diagramma dei casi d'uso

Nella sezione corrente riportiamo alcuni dei casi d'uso principali che avvengono nell'applicazione Unifind. Non tratteremo tutti i casi d'uso, poichè molti risulterebbero ripetitivi o estremamente brevi e semplici.

Autenticazione

Questo diagramma descrive il caso d'uso dell'autenticazione (Figura 3.8). Esso prevede due diverse alternative, una nel caso in cui l'utente debba collegarsi e l'altra nel caso in cui l'utente debba registrarsi.

Partendo dal primo, vediamo come, in questo caso, siano coinvolti due diversi attori, ovvero l'utente e Firebase. L'utente è, ovviamente, colui che intende effettuare l'accesso al

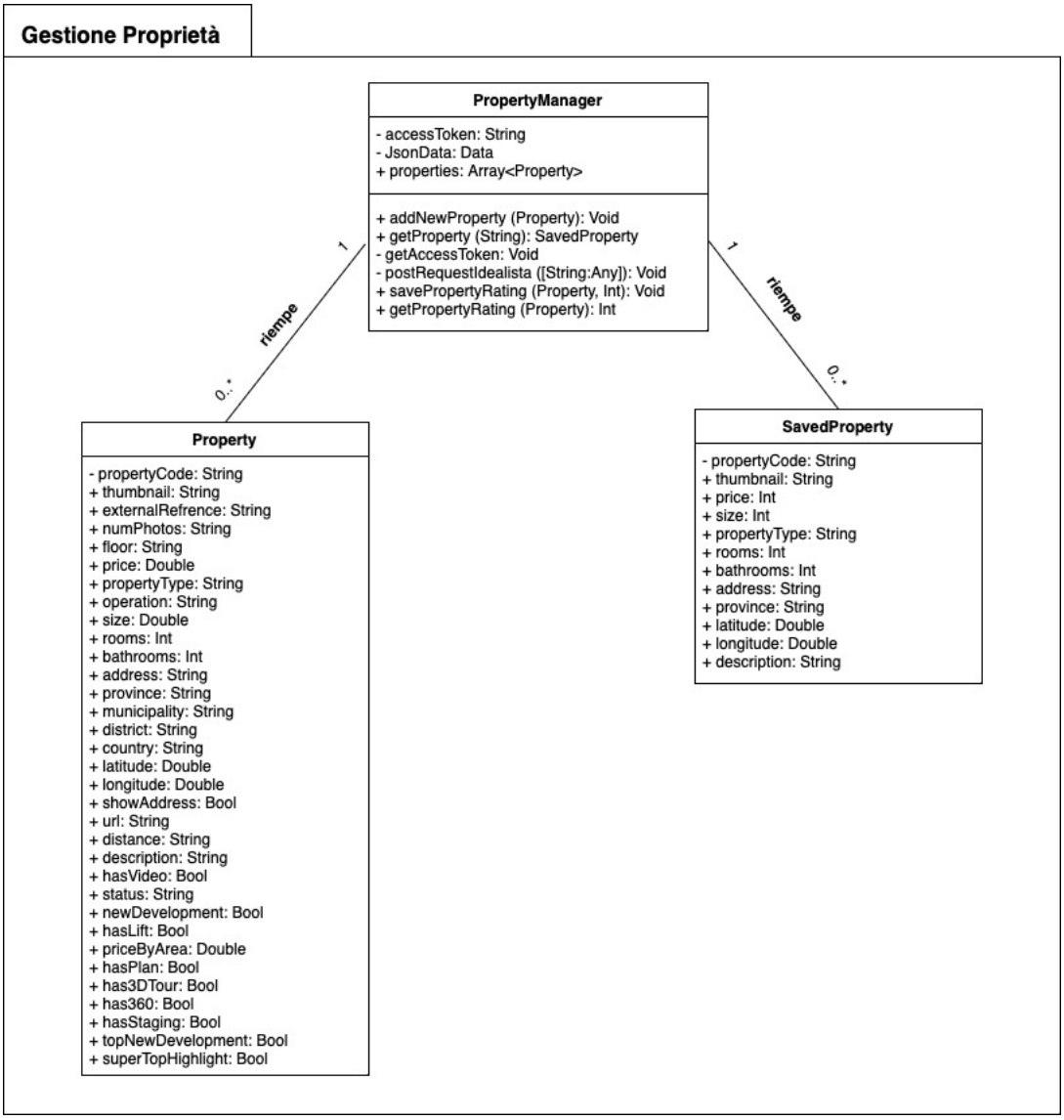

Figura 3.4: Diagramma delle classi relativo alla gestione delle abitazioni

proprio profilo per poter proseguire con l'utilizzo dell'applicazione. Per poter accedere, egli deve inserire una e-mail ed una password nei rispettivi campi. Quando questi sono stati inseriti e l'utente preme il pulsante di login, entra in campo il secondo attore (se entrambi i dati inseriti sono validi). Firebase, infatti, tramite la procedura di autenticazione, controlla che la password e l'e-mail inserita siano presenti all'interno del suo database *Authentication*. Se il controllo ha successo, l'utente accede al proprio profilo e può proseguire con l'applicazione, altrimenti questi visualizza un messaggio di errore e deve ripetere il tentativo di login.

Nel caso in cui l'utente debba effettuare una registrazione la procedura è simile: anche in questo caso l'utente deve inserire e-mail e password, ma la differenza è che la password va confermata (quindi inserita 2 volte). Nel caso in cui le password inserite siano diverse tra loro, quando l'utente preme il tasto registrati, visualizza un messaggio di errore e deve riprovare la registrazione. Nel caso in cui le password siano uguali (ed entrambe valide), l'attore Firebase crea un nuovo utente nel suo database *Authentication* al quale l'utente accede.

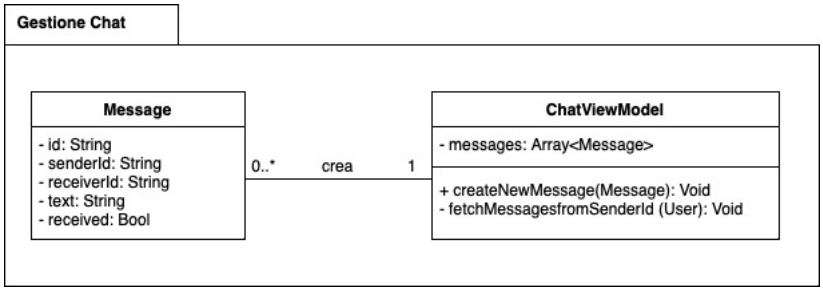

Figura 3.5: Diagramma delle classi relativo alla gestione della chat

Chat

In questa sezione analizzeremo il diagramma del caso d'uso Chat (Figura 3.9). Questo caso d'uso si verifica quando un utente ne seleziona un altro con cui chattare. La prima cosa necessaria perchè questo accada è che l'utente effettua una ricerca e salva tra i preferiti un'abitazione. Da ciascuna delle sue abitazioni salvate, l'utente può aprire l'elenco delle persone che hanno la suddetta abitazione tra i preferiti e, selezionandone uno, accede alla schermata della chat. I messaggi presenti nella chat e le informazioni ad essa relative (abitazioni evidenziata, e-mail dell'utente con cui si sta chattando, etc.) dovranno essere scaricati dal *Database* di Firebase. Dalla schermata di chat sarà possibile scrivere ed inviare messaggi, che saranno anch'essi salvati su Firebase così da essere visibili da entrambi gli utenti partecipanti alla chat (dividendoli tra inviati e ricevuti).

Modifica Profilo

Il caso d'uso Modifica Profilo avviene quando l'utente vuole modificare le proprie informazioni personali (Figura 3.10). Nel caso in cui l'utente voglia modificare le proprie credenziali di accesso (e-mail e password) le modifiche avvengono singolarmente con interfacce dedicate; l'utente dovrà, quindi, inserire delle informazioni valide per fare in modo che la modifica abbia successo (nel caso della password questa dovrà essere inserita 2 volte in maniera identica). Se l'utente vuole modificare altre informazioni, potrà farlo da un'interfaccia specifica che permetta di modificarle tutte contemporaneamente. Ovviamente, l'utente può scegliere quali campi modificare, semplicemente lasciando invariati gli altri. Nel diagramma non è presente l'utente Firebase, poichè non ha nessun ruolo attivo in questo caso d'uso; le informazioni di accesso, infatti, vengono modificate sulla sezione *Authentication* di Firebase, mentre le altre vengono modificate nella sezione *Database*.

Ricerca Filtrata

Il caso d'uso che affronteremo in questa sezione riguarda le ricerche effettuate dall'utente (Figura 3.11). Per effettuare una ricerca, egli deve obbligatoriamente inserire una località da utilizzare come centro. Questo può avvenire in due modi: cliccando sulla località desiderata direttamente dalla mappa, oppure ricercandola dalla barra di ricerca. Per rendere più precisa la propria ricerca, l'utente può anche inserire varie informazioni aggiuntive tramite i filtri (ma non è obbligatorio). Se l'utente ricerca una località tramite la barra di ricerca, la stringa digitata viene passata a Google Maps che restituirà le coordinate del primo risultato ottenuto. Dopo che ciò avviene, le coordinate ottenute verranno passate ad Idealista tramite richiesta API. Dalla richiesta API otterremo una lista di abitazioni che soddisfano i parametri selezionati dall'utente. I risultati della richiesta saranno visualizzabili, sia sotto forma di lista che sotto forma di marker, sulla mappa. Se si inseriscono dei filtri dopo aver effettuato una ricerca,

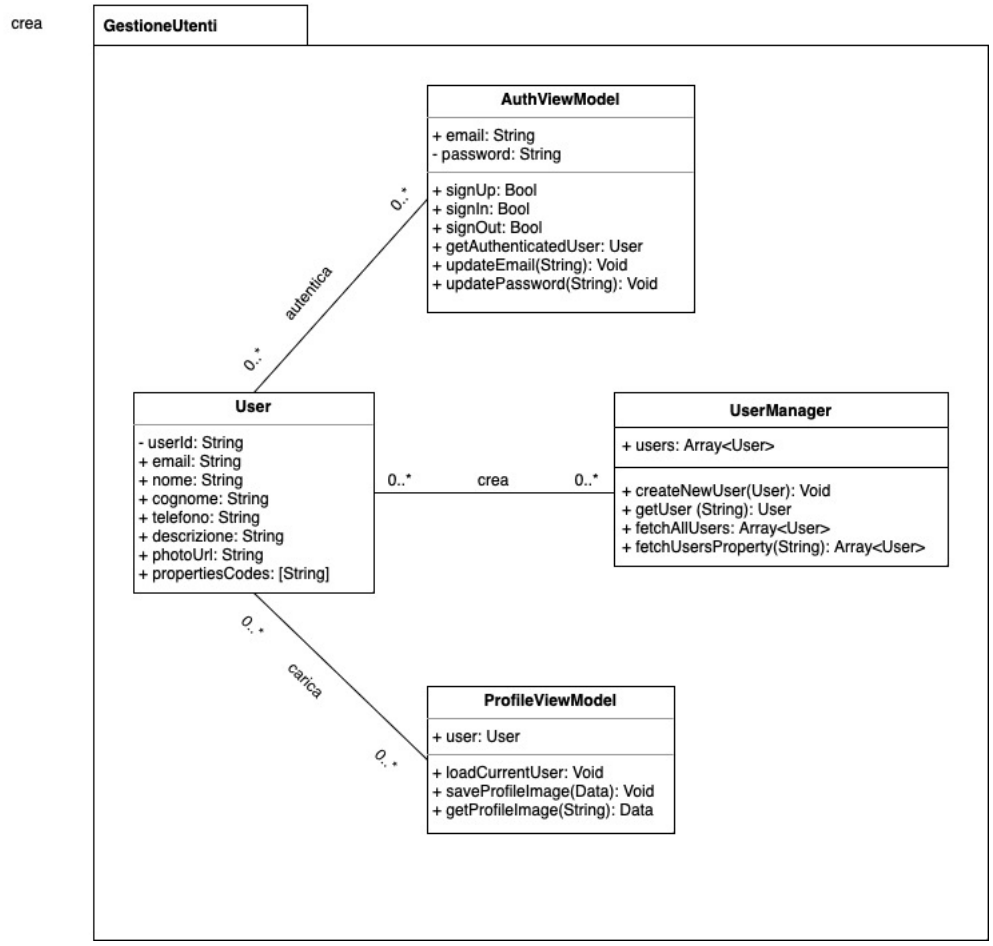

Figura 3.6: Diagramma delle classi relativo alla gestione degli utenti

i risultati di quest'ultima verranno sovrascritti da quelli ottenuti, effettuando una nuova ricerca con i dati aggiornati.

3.3.4 Diagrammi di sequenza

In questa sezione riporteremo i diagrammi di sequenza riguardanti le operazioni spiegate nella sezione precedente. Non sarà presente una spiegazione per i singoli diagrammi poichè ripeterebbe ciò che viene detto nella sezione dei casi d'uso. Di seguito un elenco dei diagrammi di sequenza:

- *Login:* il diagramma di sequenza del login è rappresentato nella figura 3.12.
- *Registrazione:* il diagramma di sequenza della registrazione è rappresentato nella figura 3.13.
- *Ricerca filtrata:* il diagramma di sequenza della ricerca è rappresentato nella figura 3.14.
- *Modifica profilo:* il diagramma di sequenza della modifica del profilo è rappresentato nella figura 3.15.
- *Chat:* il diagramma di sequenza della chat è rappresentato nella figura 3.16.

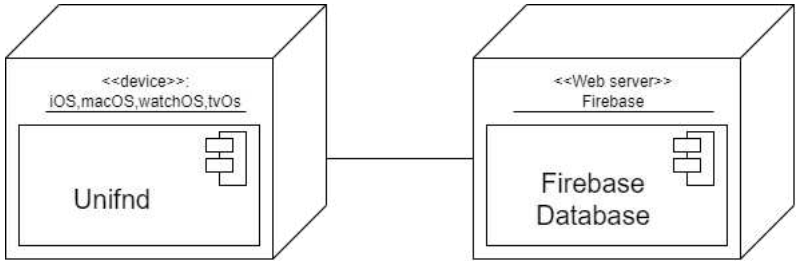

Figura 3.7: Diagramma di deployment

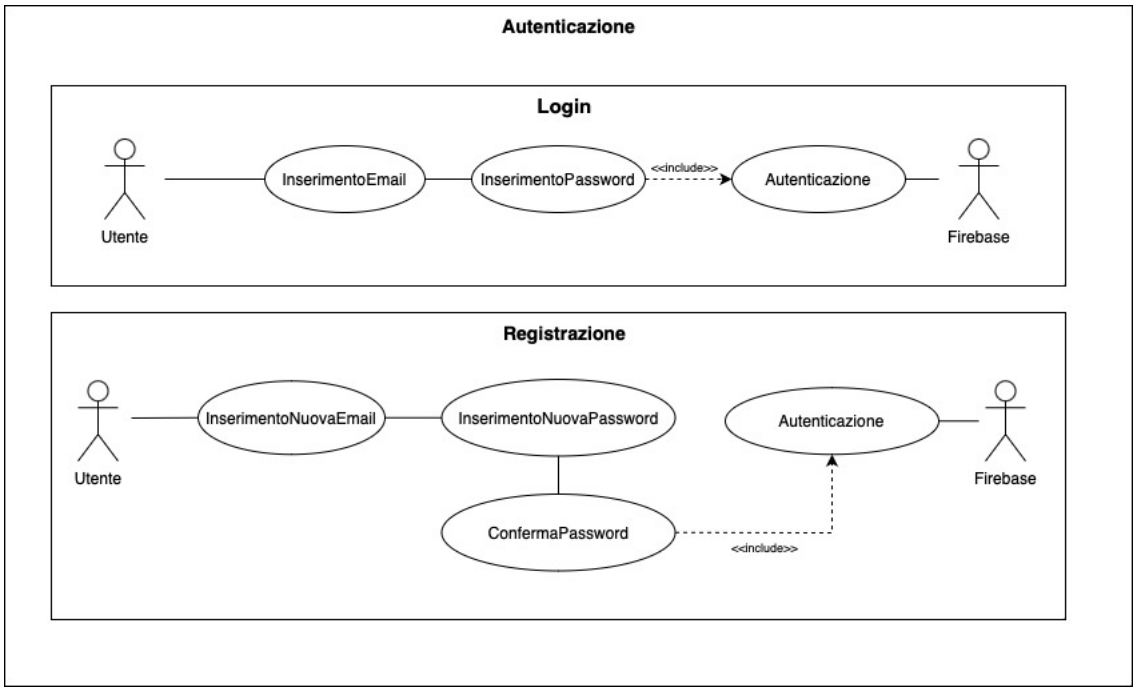

Figura 3.8: Caso d'uso Autenticazione

3.3.5 Diagrammi di attività

In questa sezione riporteremo i diagrammi di attività riguardanti le operazioni descritte nella sezione precedente. Non sarà presente una spiegazione per i singoli diagrammi, poichè ripeterebbe ciò che viene detto nella sezione dei casi d'uso. Di seguito un elenco dei diagrammi di attività:

- *Login:* il diagramma di attività del login è rappresentato nella figura 3.17.
- *Registrazione:* il diagramma di attività della registrazione è rappresentato nella figura 3.18.
- *Ricerca filtrata:* il diagramma di attività della ricerca è rappresentato nella figura 3.19.
- *Modifica profilo:* il diagramma di attività della modifica del profilo è rappresentato nella figura 3.20.
- *Chat:* il diagramma di attività della chat è rappresentato nella figura 3.21.

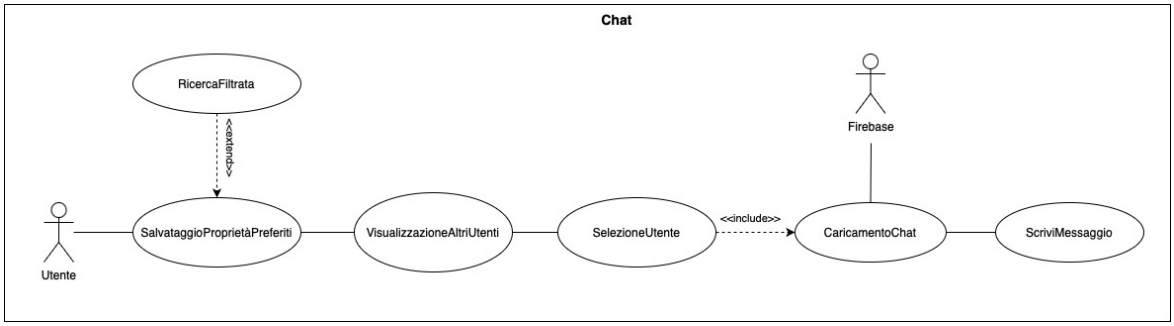

Figura 3.9: Caso d'uso Chat

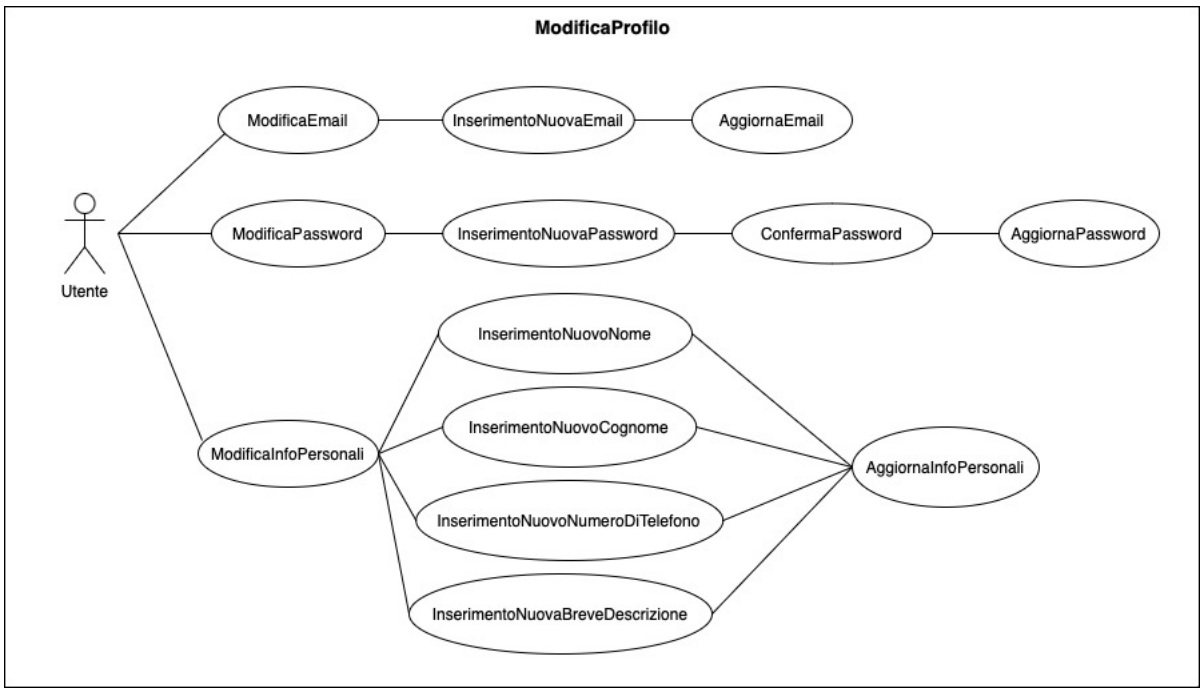

Figura 3.10: Caso d'uso Modifica Profilo

3.4 Mockup

In questa sezione finale, riporteremo i mockup delle interfacce grafiche presenti all'interno dell'applicazione. Più specificatamente i mockup effettuati sono i seguenti:

- *Mockup dei dettagli dell'abitazione:* riportato in Figura 3.22
- *Mockup di modifica della password:* riportato in Figura 3.23
- *Mockup della mappa:* riportato in Figura 3.24
- *Mockup dei filtri:* riportato in Figura 3.25
- *Mockup del login:* riportato in Figura 3.26
- *Mockup della registrazione:* riportato in Figura 3.27
- *Mockup della chat:* riportato in Figura 3.28
- *Mockup di selezione dell'utente:* riportato in Figura 3.29

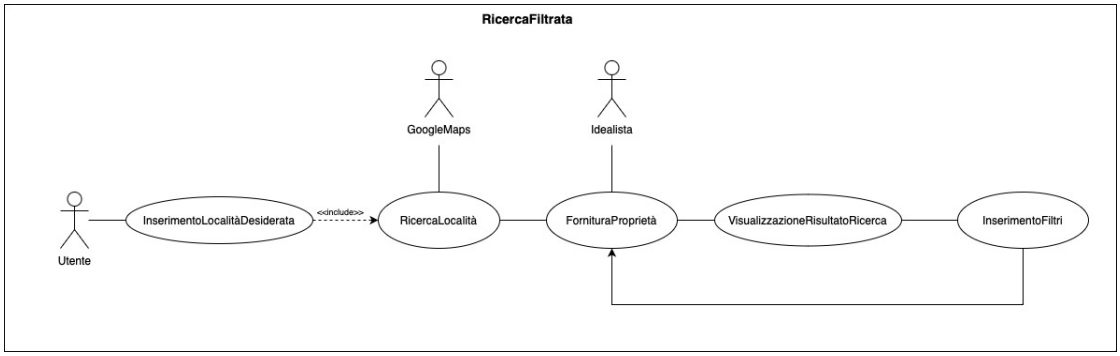

Figura 3.11: Caso d'uso Ricerca Filtrata

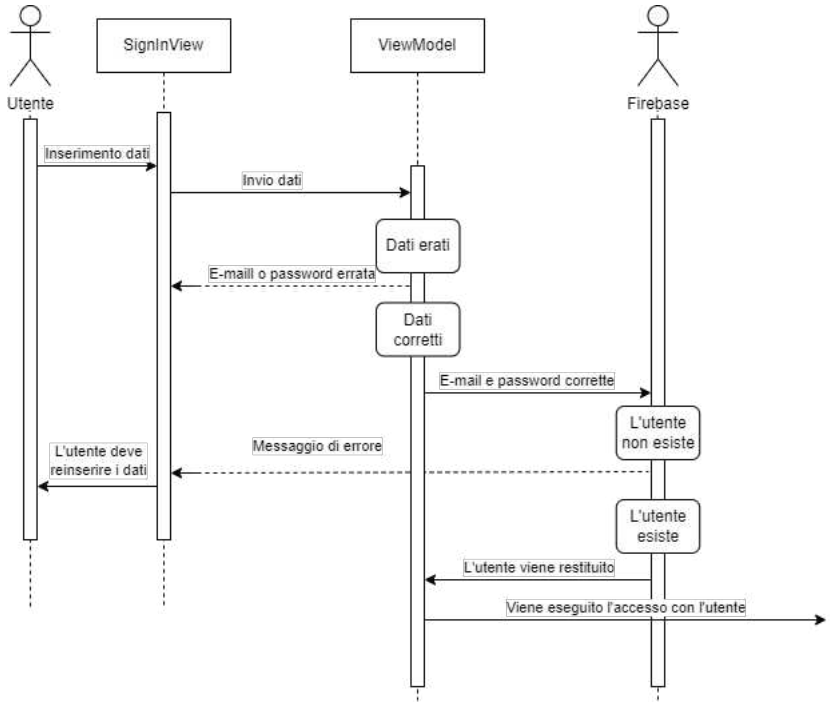

Figura 3.12: Diagramma di sequenza Login

- *Mockup di aggiornamento dell'e-mail:* riportato in Figura 3.30
- *Mockup di aggiornamento delle informazioni dell'utente:* riportato in Figura 3.31
- *Mockup di per la visualizzazione del profilo:* riportato in Figura 3.32
- *Mockup della lista dei preferiti:* riportato in Figura 3.33
- *Mockup della lista delle abitazioni:* riportato in Figura 3.34

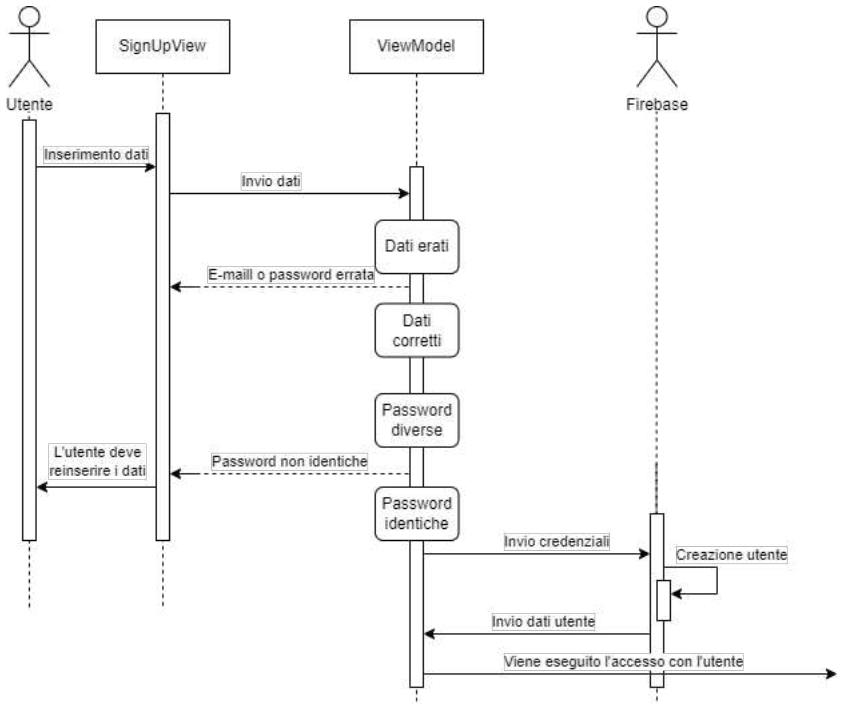

Figura 3.13: Diagramma di sequenza Registrazione

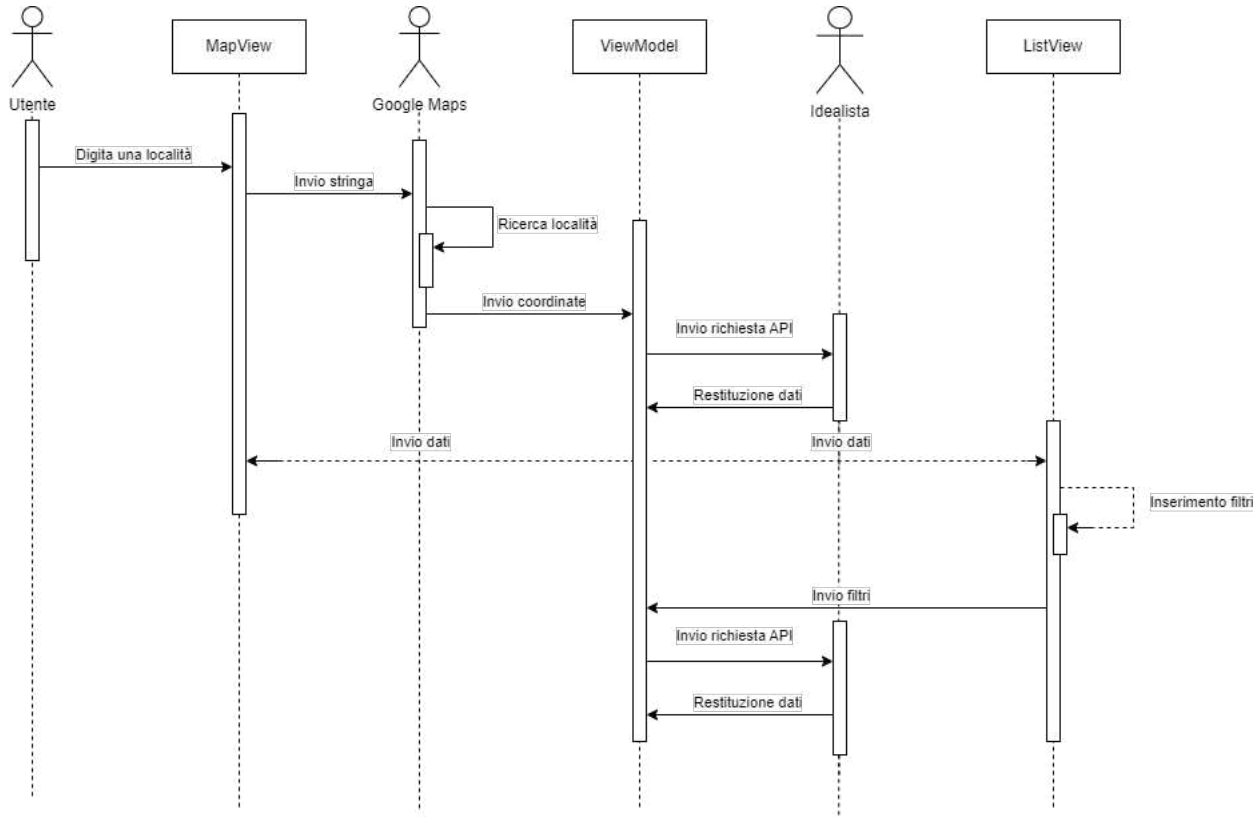

Figura 3.14: Diagramma di sequenza ricerca filtrata

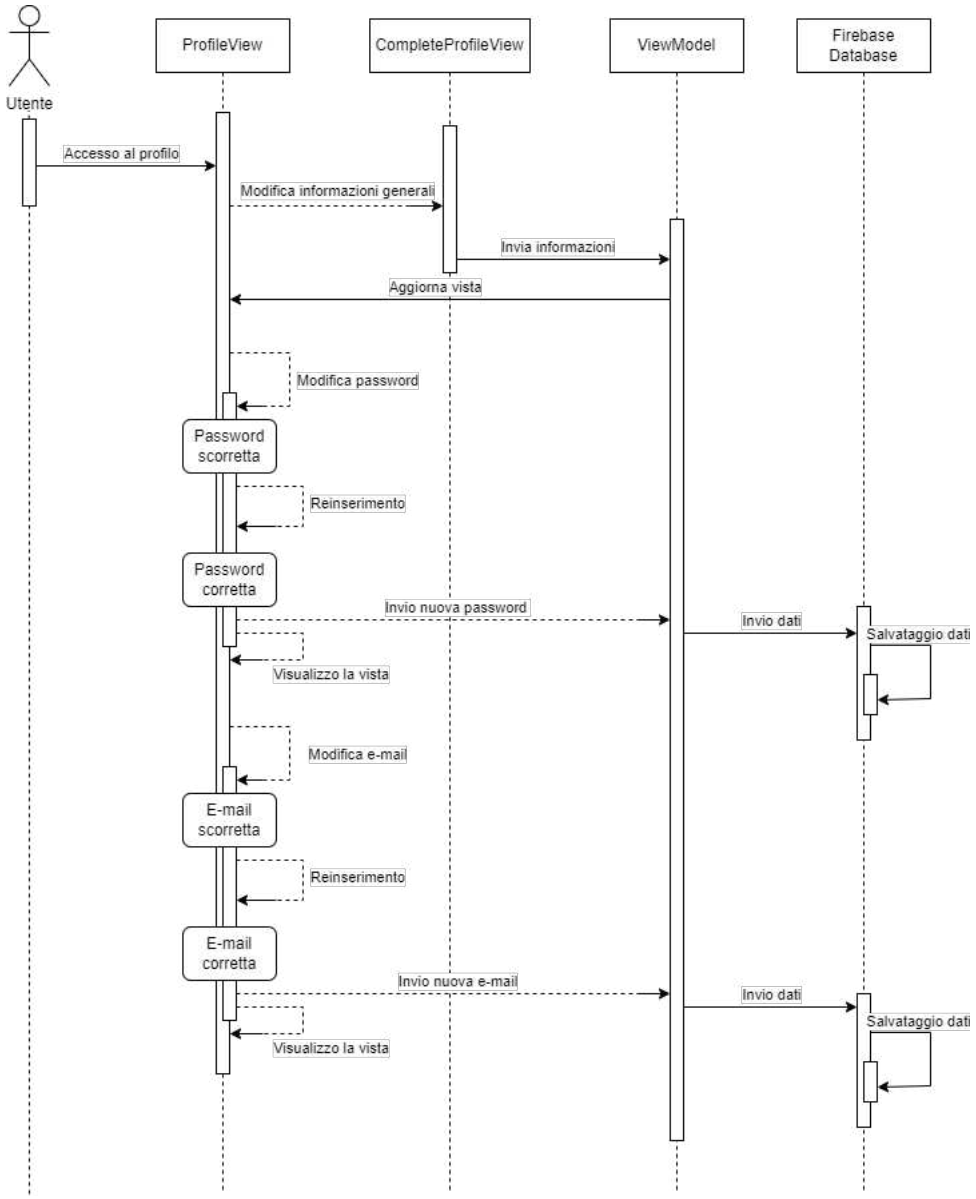

Figura 3.15: Diagramma di sequenza modifica profilo

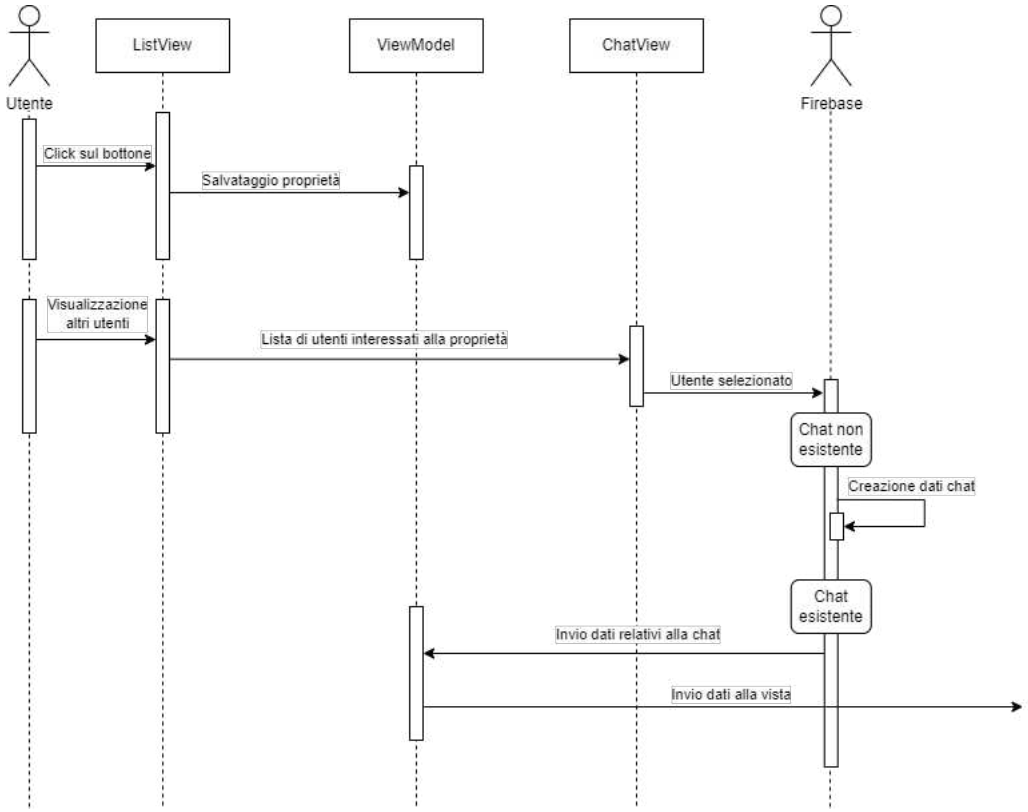

Figura 3.16: Diagramma di sequenza Chat

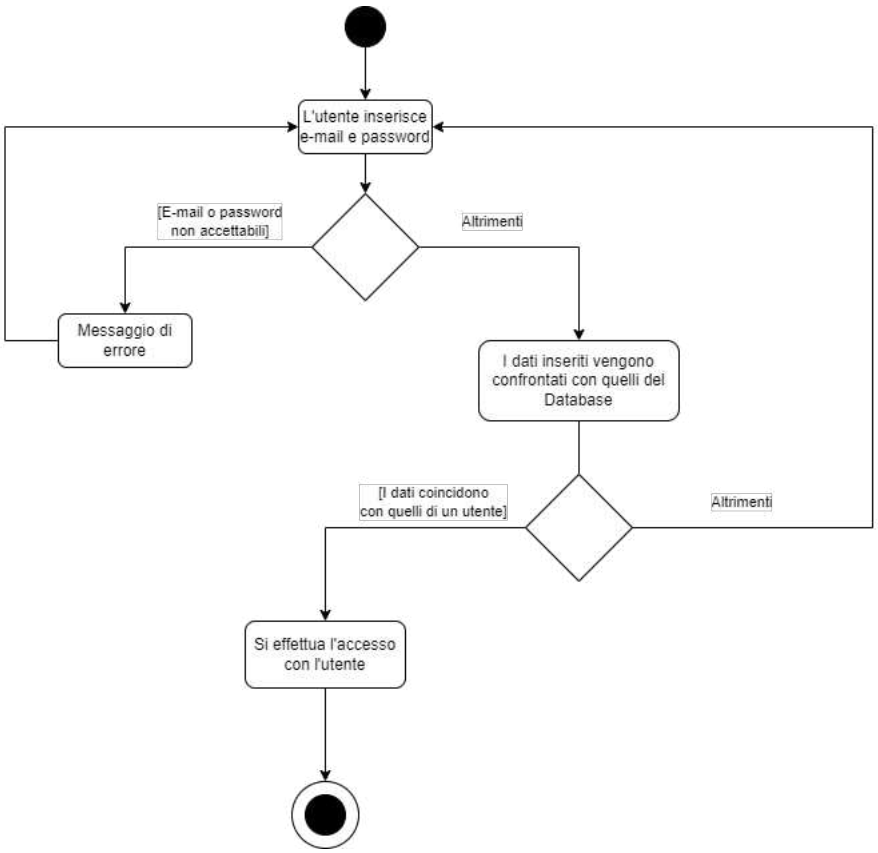

Figura 3.17: Diagramma di attività Login

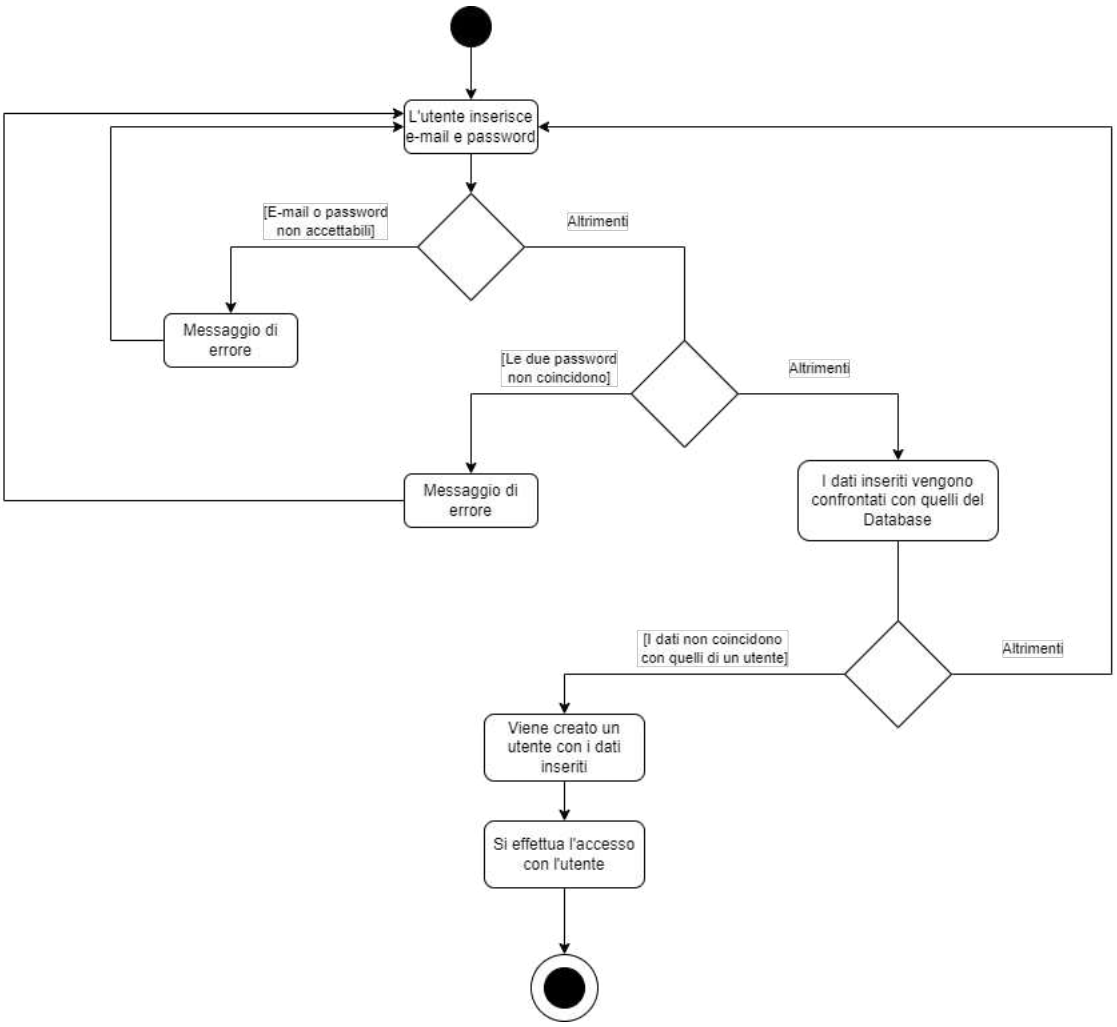

Figura 3.18: Diagramma di attività Registrazione

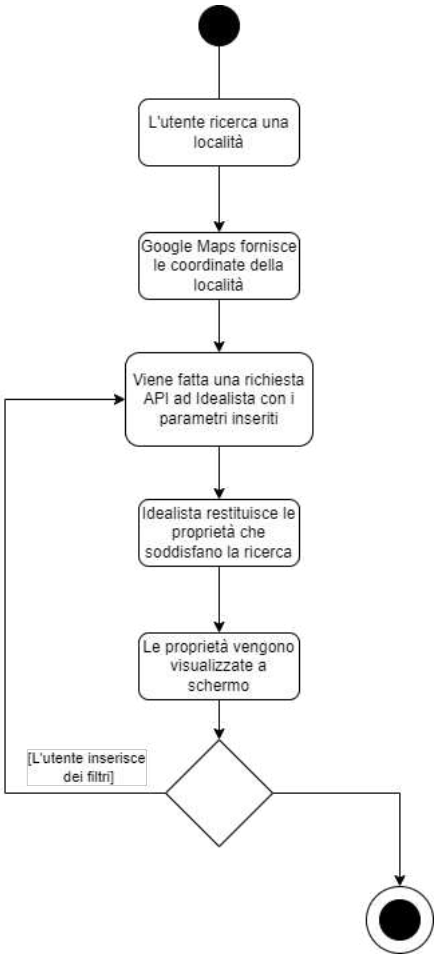

Figura 3.19: Diagramma di attività ricerca filtrata

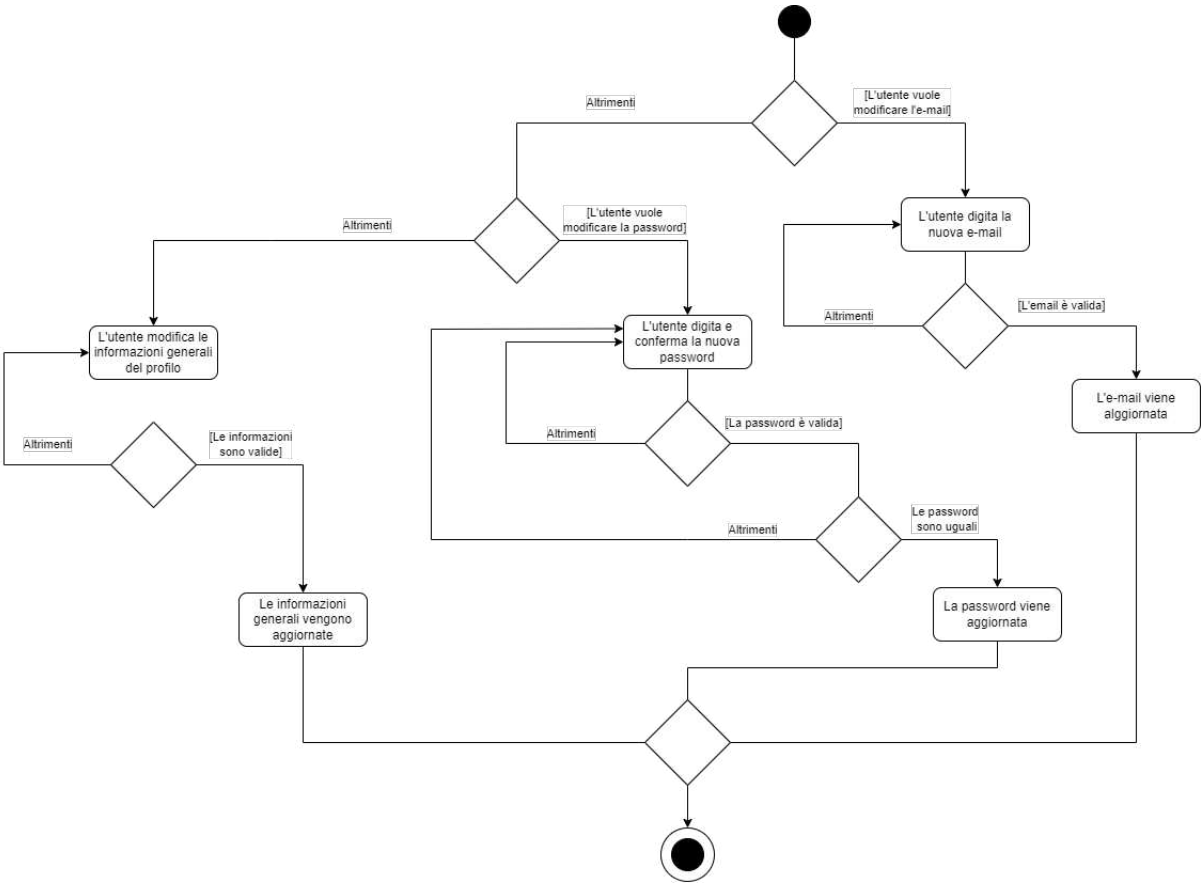

Figura 3.20: Diagramma di attività modifica profilo

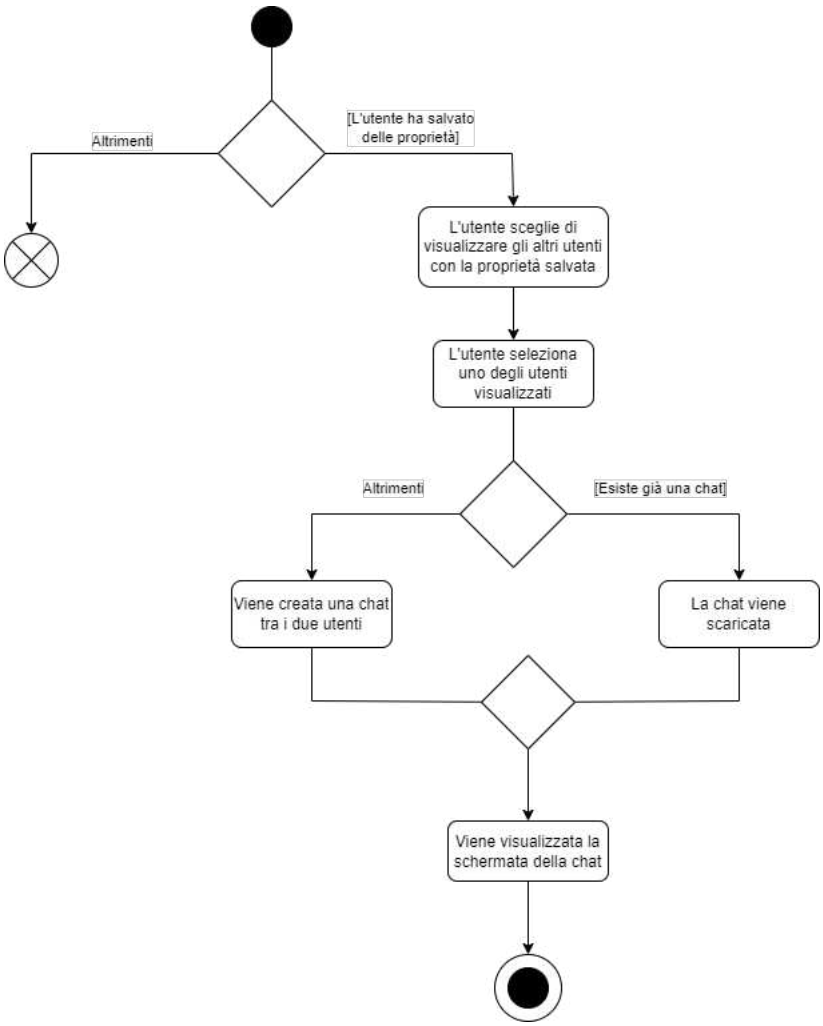

Figura 3.21: Diagramma di attività Chat

Figura 3.22: Mockup: dettagli abitazione

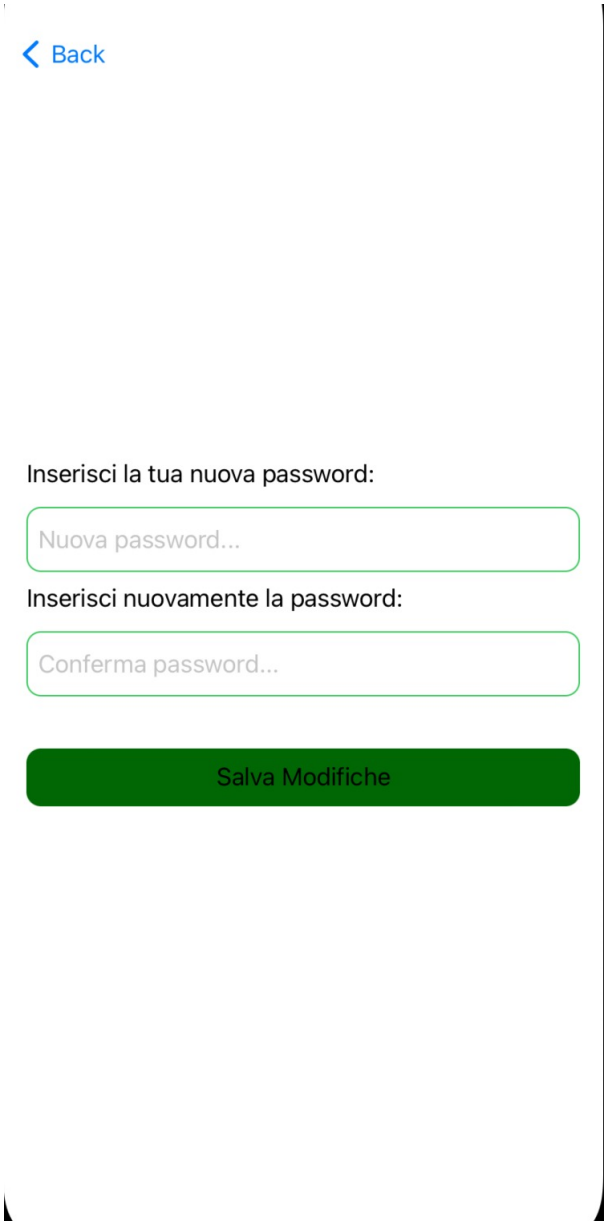

Figura 3.23: Mockup: modifica della password

Figura 3.24: Mockup: mappa

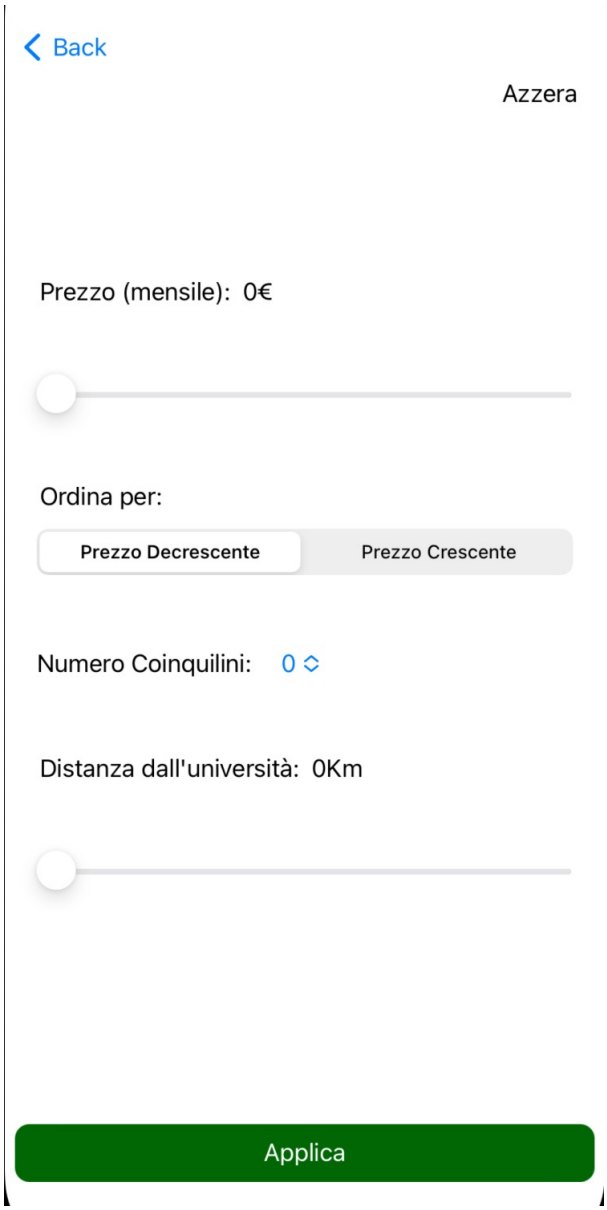

Figura 3.25: Mockup: filtri

Figura 3.26: Mockup: login

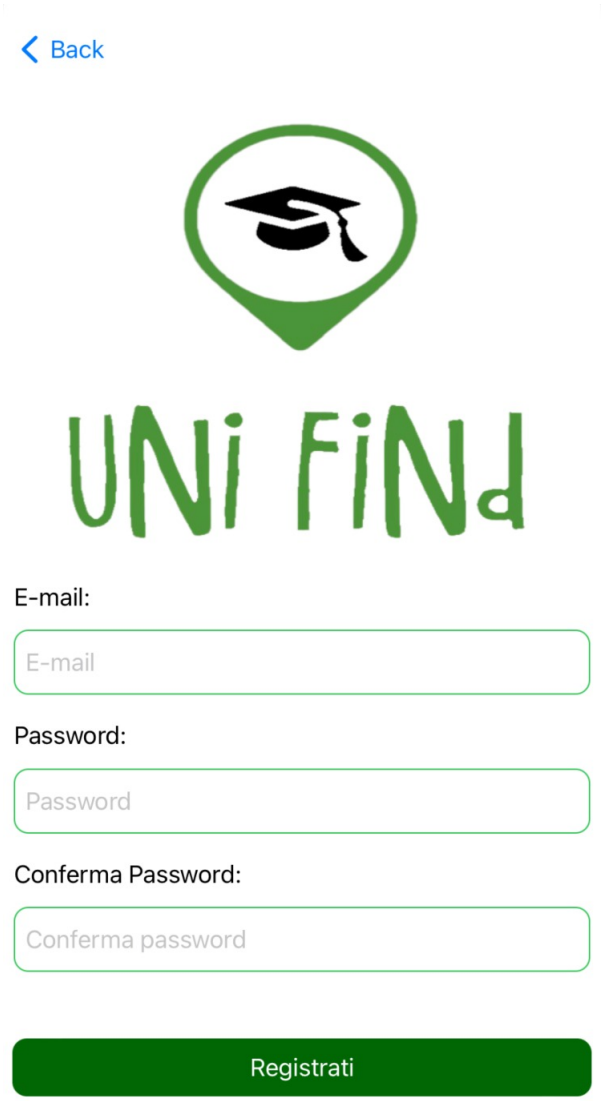

Figura 3.27: Mockup: registazione

Figura 3.28: Mockup: Chat

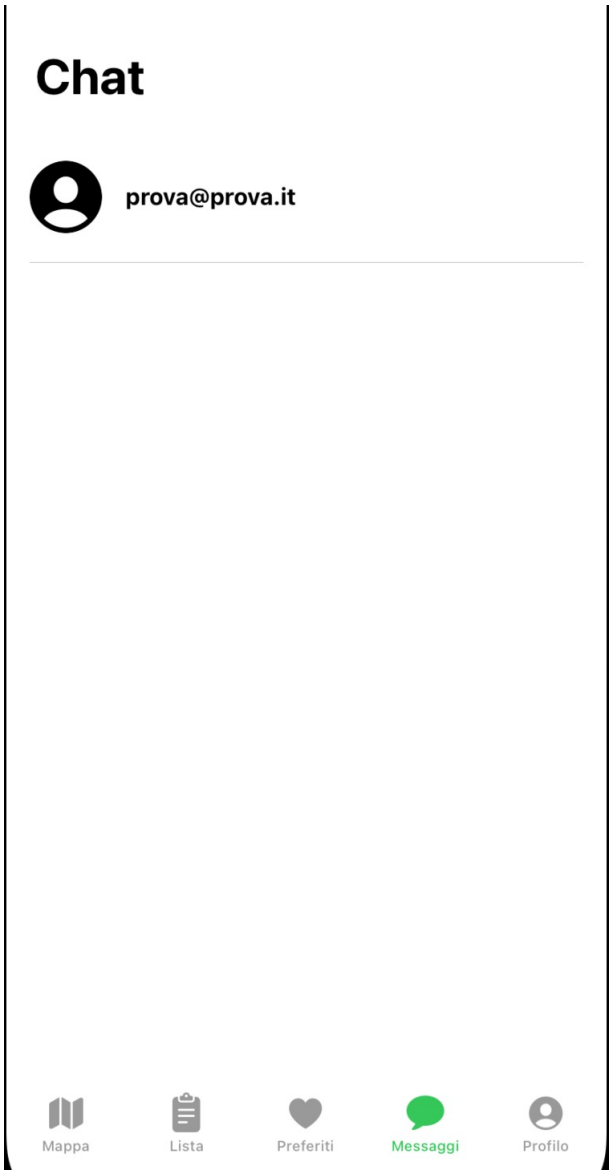

Figura 3.29: Mockup: selezione dell'utente

Figura 3.30: Mockup: aggiornamento dell'e-mail

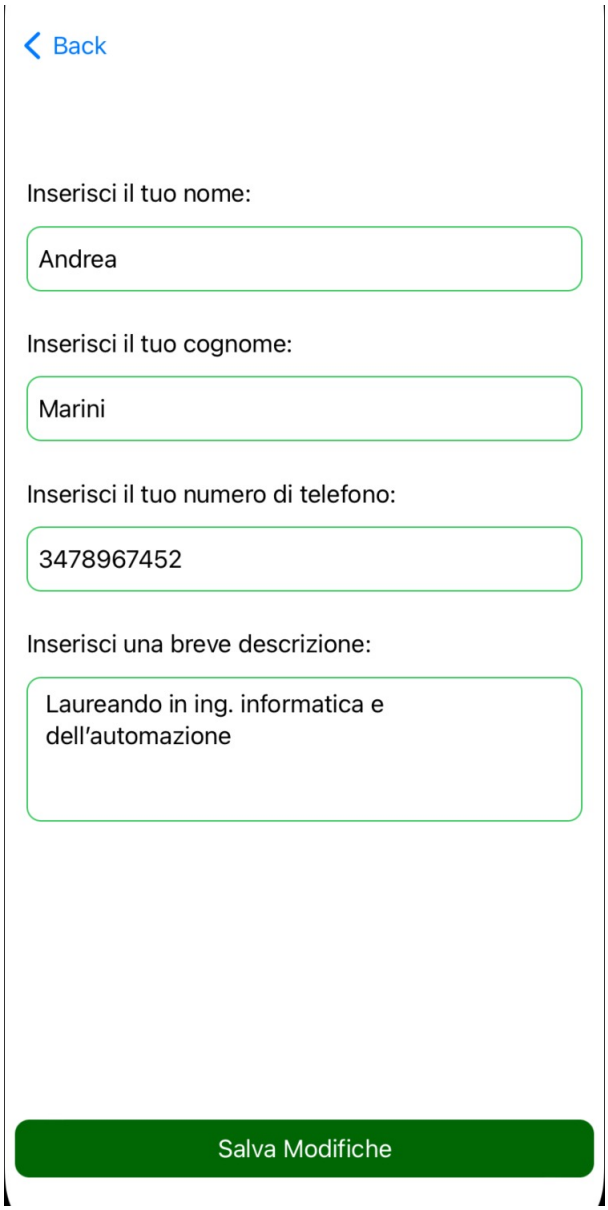

Figura 3.31: Mockup: aggiornamento delle informazioni utente

Figura 3.32: Mockup: visualizzazione del profilo

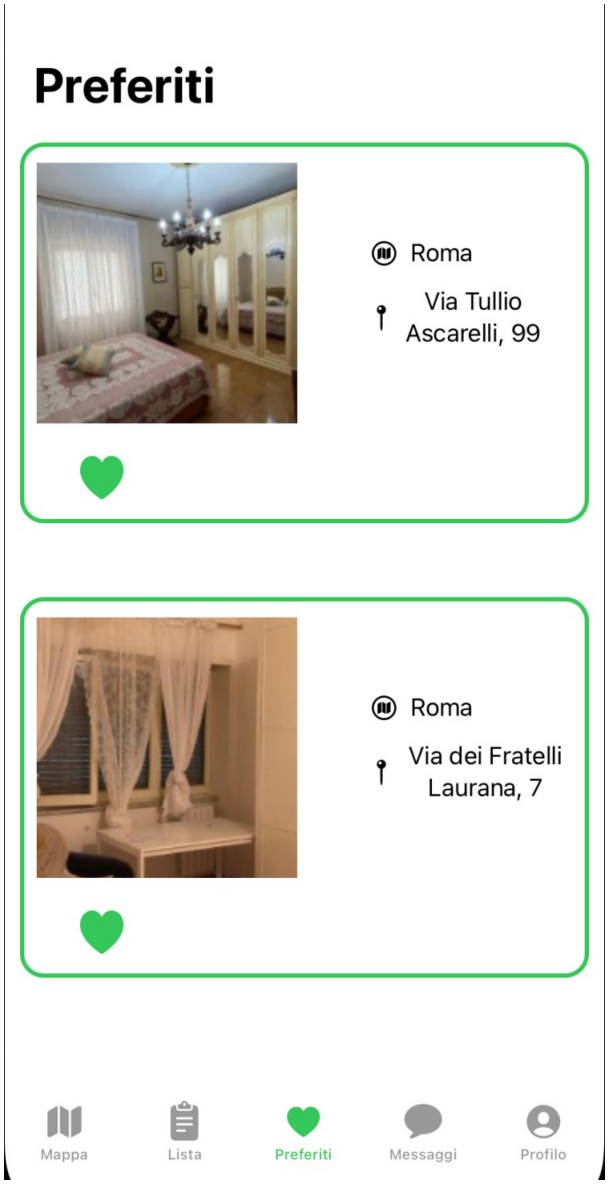

Figura 3.33: Mockup: lista dei preferiti

Figura 3.34: Mockup: lista delle abitazioni

CAPITOLO 4

Implementazione

Nel capitolo corrente parleremo della parte implementativa dell'applicazione e, quindi, di tutte le sue componenti da noi programmate in Swift. In particolare, parleremo dell'architettura da noi adottata e della suddivisione in package svolta. In seguito, passeremo ad analizzare le singole classi una ad una descrivendone componenti e funzionamento. Partiremo dalla parte di frontend dell'applicazione, descrivendo le interfacce grafiche, per poi passare alla parte di backend. Qui, prenderemo in esame i Model ed i ViewModels spiegando i motivi che ci hanno spinto a compiere determinate scelte.

4.1 Architettura dell'applicazione

La nostra applicazione utilizza l'architettura *MVVM* (Model-View-ViewModel). Questo tipo di architettura è molto usato per la creazione di applicazioni mobile, proprio grazie alla sua componente ViewModel, che costituisce anche la differenza principale tra l'architettura *MVVM* e quella *MVC* (Model-View-Controller). L'architettura *MVC* è l'architettura classica utilizzata dalle applicazioni web, ma non molto utilizzata per quelle mobile. Questo perchè il controller è attivo in contemporanea alla vista con la quale è legato, e ciò causa problemi con le applicazioni mobile, in quanto queste ultime richiedono spesso che le viste vengano sostituite o messe in background durante l'esecuzione di un'applicazione. Così facendo, il controller smette di funzionare quando non dovrebbe, non consentendo di effettuare determinate operazioni. Nel caso del ViewModel, invece, il problema non si manifesta, poichè, essendo definito separatamente dalla vista, questo rimane attivo anche in caso di distruzione o sparizione di quest'ultima. Per tale motivo, abbiamo optato per l'architettura *MVVM* all'interno dell'applicazione Unifind.

4.2 Divisione in package

Prima di iniziare la parte implementativa del nostro progetto, abbiamo dovuto scegliere come suddividere in package le varie parti del codice. Riportiamo la suddivisione in package da noi effettuata sulla Figura 4.1. Abbiamo suddiviso il codice in queste 4 macrocategorie, poichè sono quelle che rappresentano le parti principali della nostra applicazione. All'interno del diagramma non figurano le viste presenti nell'applicazione, che sono raggruppate in un package a parte dedicato solo alle interfacce grafiche.

Ora analizzeremo le componenti di ogni package nel dettaglio, per rendere più chiari i compiti di ogni classe riportata nel diagramma.

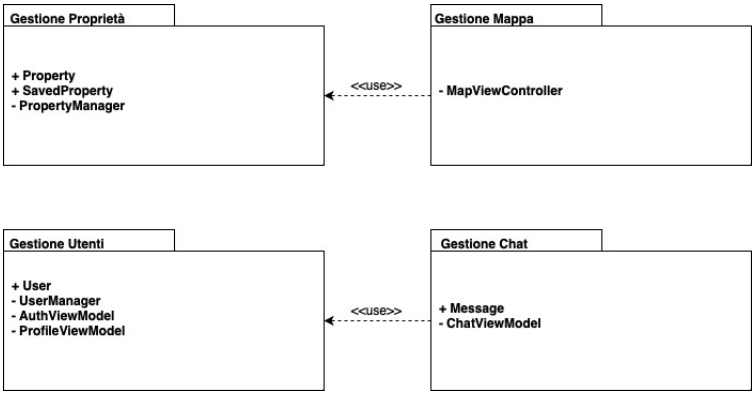

Figura 4.1: Divisione in package

Gestione Proprietà

- *Property:* la classe incaricata di contenere i dati delle abitazioni, ottenuti effettuando richieste API tramite Idealista.
- *SavedProperty:* la classe incaricata di contenere i dati delle abitazioni salvate dai singoli utenti. Questa si differenzia dalla precedente in quanto le informazioni memorizzate al suo interno sono limitate rispetto a tutte quelle fornite dalla richiesta API.
- *PropertyManager:* il ViewModel incaricato della gestione delle operazioni riguardanti le abitazioni e le viste per la loro visualizzazione.

Gestione mappa

- *MapViewController:* il componente incaricato della gestione delle operazioni riguardanti la vista della mappa. In questo caso non abbiamo adoperato un ViewModel poichè, per riuscire ad utilizzare le estensioni di GoogleMaps da noi richieste, era necessario un Controller.

Gestione User

- *User:* la classe incaricata di contenere tutte le informazioni relative ad un utente.
- *UserManager:* la classe utilizzata per gestire vari metodi riguardanti le operazioni da svolgere sugli utenti. Non è un ViewModel, poichè i metodi che contiene vengono richiamati da molte viste del programma ed, in questo modo, possono sempre essere utilizzati.
- *AuthViewModel:* il ViewModel per gestire l'autenticazione (quindi la parte di login e registrazione) dell'utente ed i metodi riguardanti l'utente acceduto all'applicazione (ad esempio il metodo per ottenere l'id dello user autenticato).
- *ProfileViewModel:* il ViewModel per gestire le operazioni svolte dall'utente sul proprio profilo (modifica e-mail, password, etc.).
- *Gestione Chat*
- *Message:* la classe incaricata di contenere i dati riguardanti le chat dell'utente.
- *ChatViewModel:* il ViewModel che gestisce le operazioni riguardanti la chat e tutte le sue componenti.

4.3 Implementazione frontend

In questa sezione, parleremo dell'implementazione della parte di frontend dell'applicazione, analizzando tutte le viste presenti all'interno dell'app.

4.3.1 ChatUserInfoView

Questa è vista dedicata alla visualizzazione delle informazioni di un utente, da parte di colui con cui sta chattando. Questa vista è molto simile a ProfileView, che vedremo in seguito. Alla creazione, essa accede allo User di cui si vogliono visualizzare le informazioni (che viene fornito dalla vista della chat al momento del click sul nome dell'utente) e, partendo dall'id dell'utente, riesce ad accedere all'immagine di profilo salvata su FirebaseStorage (nel caso in cui questa non sia presente viene fornita un'immagine di default) e a tutti gli altri attributi.

ChatView

Questa è la vista dedicata alla selezione di un utente con il quale chattare. Quando quest'interfaccia viene creata, essa utilizza il metodo di UserManager per poter accedere agli User con una determinata abitazione salvata ed, in questo modo, ottiene l'elenco degli utenti che deve mostrare. Grazie alla lista degli utenti, accede anche alle immagini ad essi relative, che verranno poi mostrate a fianco dell'apposito utente. Da questa vista è possibile accedere alla vista MessageView semplicemente cliccando su un utente.

ChatView2

Questa vista è identica alla precedente nella visualizzazione, ma differisce per funzionamento; questa è, infatti, la vista a cui si accede cliccando sulla sezione chat della barra di navigazione. Di conseguenza, alla creazione non vengono acceduti gli utenti con una determinata proprietà salvata, ma quelli che hanno una chat aperta con l'utente attualmente collegato. Anche da questa vista si accede alla chat al click su un utente.

CompleteProfileView

La vista che permette di inserire le informazioni facoltative all'interno del profilo utente. Alla creazione, si ottiene l'utente attuale grazie al ViewModel AuthViewModel. E, con il suo id, si accede al database con tutti i dati del suddetto utente. Nel caso in cui questi siano presenti, vengono salvati all'interno delle apposite TextField; altrimenti, queste appariranno vuote. L'utente, dopo aver inserito i dati negli spazi appositi, può salvare le modifiche oppure eliminarle, lasciando invariati i dati.

FavouritesView

La vista che permette di visualizzare tutte le abitazioni salvate all'interno di una lista. Quando viene creata, questa vista accede all'utente collegato tramite l'AuthViewModel ed in seguito scarica dal Database tutte le abitazioni salvate dall'utente. Da questa vista, l'utente può accedere ai dettagli delle abitazioni salvate tramite il click; inoltre, si possono rimuovere abitazioni dai preferiti semplicemente cliccando sull'apposito pulsante.

FilterView

La vista che contiene i filtri applicabili alle ricerche. Nel caso in cui i parametri dei filtri vengano modificati, verrà eseguita automaticamente una richiesta API con i nuovi parametri inseriti.

ListView

La vista che permette di visualizzare le abitazioni sotto forma di lista. Alla creazione di questa vista, l'id dell'utente collegato viene ottenuto tramite AuthViewModel e, grazie all'identificatore, si accede alle proprietà salvate dall'utente, in modo da identificarle, nel caso in cui siano presenti nella lista che l'utente sta per visualizzare. Inoltre, per ogni proprietà, vengono richiesti al database anche i rating; sia quello medio che quello attribuito dall'utente collegato. Infine, tutti i dati ottenuti vengono organizzati in pagine prima di essere mostrati all'utente per rendere più piacevole la loro consultazione. Anche da questa vista è possibile accedere ai dettagli di tutte le singole abitazioni nonchè alla vista dei filtri.

MapView

Questa è la vista contenente la mappa. Questa vista è legata ad un controller (l'unico presente nell'applicazione) che implementa tutte le funzionalità richieste alla vista. Questo accade poichè, per implementare tutte le funzionalità da noi desiderate, abbiamo dovuto coniugare in una sola vista l'interfaccia di GoogleMaps con le funzionalità di Mappe di Apple. L'unico modo per farlo era ricorrere ad un'interfaccia grafica che utilizzasse un controller per essere istanziata, motivo per il quale quest'interfaccia fa ricorso ad un controller e ad una vista implementata senza SwiftUI. Dalla MapView è possibile accedere ai dettagli di una proprietà cliccando sull'apposito marker.

MessagesView

Questa è la vista che contiene le chat tra due utenti. Nella sua creazione essa riceva un utente fornito da ChatView (o ChatView2); a questo punto essa scarica dal database i dati riguardanti l'utente con cui si sta "chattando" ed i dati della chat (quindi i messaggi inviati e ricevuti e la proprietà iniziale). Inoltre, alla sua creazione, la vista controlla il numero di proprietà salvate dall'utente e, nel caso in cui ve ne siano più di una, lo notifica nella parte superiore dello schermo. Da questa vista si può accedere alle informazioni dell'utente coinvolto nella chat e ottenere i dettagli di tutte le proprietà in comune tra i due utenti.

MoreInformationView

Questa vista permette di accedere ai dettagli dell'abitazione selezionata. Al momento della creazione, l'abitazione, che viene fornita dall'interfaccia grafica precedente, viene acceduta da questa vista che ne stampa tutti gli attributi.

NavigationBar

Questa vista implementa l'interfaccia grafica della barra di navigazione. Nella barra sono presenti 5 sezioni: mappa, lista, preferiti, chat e profilo. Cliccando su uno degli elementi grafici si apre la vista relativa all'icona selezionata.

ProfileView

Questa vista contiene le informazioni del profilo dell'utente collegato. Alla creazione da profileViewModel. essa ottiene l'utente che risulta al momento collegato; di questo utente, vengono scaricati tutti gli attributi che saranno visualizzati sullo schermo. Da questa vista, è possibile modificare tutte le informazioni che vengono visualizzate.

SignInView

Questa vista permette di accedere al proprio profilo. Inserendo le informazioni corrette ed accedendo si viene riportati alla vista MapView.

SignUpView

Questa vista permette di registrarsi all'applicazione. Una volta creato il proprio account, si viene riportati alla vista MapView.

4.4 Implementazione backend

In questa sezione parleremo dell'implementazione della parte di backend dell'applicazione, analizzando Model e ViewModel all'interno dell'app.

4.4.1 Model

All'interno dell'applicazione abbiamo istanziato 4 Model che rappresentano le entità principali; ovvero Message, Property, SavedProperty e User.

Message

La struttura dati Message serve per contenere i messaggi inviati e ricevuti dagli utenti. Al suo interno abbiamo:

- *idMessage*: una stringa per identificare univocamente i messaggi.
- *receiverId*: una stringa che contiene l'id dell'utente che riceve il messaggio.
- *senderId*: una stringa che contiene l'id di chi invia il messaggio.
- *text*: una stringa contenente il testo del messaggio.
- *received*: una variabile booleana che identifica il messaggio come ricevuto o inviato.

Property

Questa struttura dati è stata realizzata per comodità. Infatti, viene utilizzata soltanto per avere un tipo di variabile disponibile dove poter immagazzinare le abitazioni fornite da Idealista. Quindi, non descriveremo nel dettaglio tutti gli attributi poichè quelli che vengono utilizzati all'interno dell'applicazione saranno descritti all'interno del Model SavedProperty.

SavedProperty

Questa struttura dati contiene gli attributi delle abitazioni che vengono sfruttati all'interno dell'applicazione. Nel dettaglio vi sono i seguenti attributi:

- *propertyCode*:una stringa contenente il codice dell'abitazione che la identifica univocamente (fornito da Idealista).
- *thumbnail*:una stringa contenente il link all'immagine dell'abitazione.
- *price*: un double (tipo di variabile per rappresentare un numero decimale) contenente il prezzo mensile dell'abitazione.
- *size*: un intero contenente l'estensione in metri quadri dell'abitazione.
- *propertyType*: una stringa contenente il tipo dell'abitazione inteso come casa, appartamento, villa, etc.
- *rooms*: un intero contenente il numero di camere da letto dell'abitazione.
- *bathrooms*: un intero contenente il numero di bagni presenti nell'abitazione.
- *address*: una stringa contenente l'indirizzo dell'abitazione.
- *province*: una stringa contenente la città dell'abitazione.
- *latitude*: un double contenente la latitudine dell'abitazione.
- *longitude*: un double contenente la longitudine dell'abitazione.
- *itemize*: una stringa contenente la descrizione dell'applicazione (a volte non è presente).

User

Questa classe contiene i dati degli utenti. I suoi attributi, in particolare, sono:

- *userId*: una stringa contenente l'identificatore univoco dell'utente.
- *email*: una stringa contenente l'e-mail dell'utente.
- *nome*: una stringa contenente il nome dell'utente.
- *cognome*: una stringa contenente il cognome dell'utente.
- *telefono*: una stringa contenente il numero di telefono dell'utente.
- *descrizione*: una stringa contenente la descrizione dell'utente.
- *photoUrl*: una stringa contenente il link all'immagine del profilo dell'utente.
- *propertiesCodes* : un vettore di stringhe contenente tutte le proprietà salvate dall'utente.

4.4.2 ViewModel

In questa sezione, prenderemo in esame i ViewModel presenti nella nostra applicazione, spiegandone le funzionalità principali.

AuthViewModel

Questo è il ViewModel che contiene tutti i metodi relativi all'autenticazione degli utenti. Elenchiamo di seguito i metodi presenti, descrivendoli:

- *SignUp:* il metodo che permette ad un utente di registrarsi all'interno dell'applicazione inserendo delle credenziali valide.
- *SignInUser:* il metodo che permette ad un utente di accedere al proprio profilo inserendo le credenziali corrette.
- *GetAuthenthicatedUser:* il metodo che permette di ottenere l'utente che attualmente sta accedendo all'applicazione.
- *SignOut:* il metodo che permette ad un utente di uscire dal proprio profilo per effettuare nuovamente il login.
- *UpdateEmail:* il metodo per cambiare l'email del proprio profilo.
- *UpdatePassword:* il metodo per cambiare la password del proprio profilo.

ChatViewModel

Questo è il ViewModel che contiene tutti i metodi riguardanti l'interazione tra gli utenti tramite la chat. Esso possiede i seguenti metodi:

- *CreateNewMessage :* il metodo per memorizzare un messaggio nel database.
- *FetchMessagesFromSenderId :* il metodo per ottenere tutti i messaggi inviati da un determinato utente.

OpenPropertyViewModel

Questo ViewModel è stato implementato per semplicità: al suo interno è presente un solo metodo, che permette di passare una proprietà fornita da Idealista ad una del tipo salvabile nel nostro Database.

ProfileViewModel

Il ViewModel che contiene tutte i metodi che riguardano la gestione del profilo di un utente.

- *LoadCurrentUser:* il metodo che consente inserendo l'id dell'utente di accedere al suo intero profilo con tutte le informazioni relative.
- *SaveProfileImage:* il metodo che consente di inserire o modificare la propria immagine di profilo.

In questa sezione non è presente nessun ViewModel relativo alla gestione della mappa. Questo perchè, come spiegato in precedenza, la mappa viene gestita da un controller, che sfrutta vari metodi già presenti all'interno del kit per la gestione della mappa di Swift; sovrascrivendo questi metodi, otteniamo l'effetto da noi desiderato. Ad esempio, il metodo tapGesture, che si attiva quando avviene un click sulla mappa, è stato sovrascritto per inserire dei marker relativi alle abitazioni vicine al punto cliccato. Di conseguenza, non descriveremo nel dettaglio i singoli metodi del Controller, poichè non sono stati completamente implementati da noi.

CAPITOLO 5

Manuale dell'utente

Nel capitolo corrente forniremo delle indicazioni su come utilizzare l'applicazione. Queste non sono necessarie prima di utilizzarla, vista l'intuitività delle interfacce, però forniscono, comunque, all'utente una panoramica delle funzionalità di Unifind.

Nel capitolo inizialmente proponiamo una breve introduzione, che spiega i punti di forza di Unifind rispetto ad altre applicazioni per la ricerca di immobili. Di seguito troviamo il manuale per l'accesso, che spiega quali sono le condizioni da rispettare nell'immissione delle proprie credenziali. Successivamente, abbiamo la spiegazione dettagliata delle funzionalità dell'applicazione, divise per sezioni corrispondenti a quelle che si trovano nella barra di navigazione. In questo modo, l'utente può orientarsi meglio e sapere in quale sezione recarsi per soddisfare le sue esigenze.

5.1 Introduzione all'applicazione

In questa sezione effettueremo un'introduzione generale dell'applicazione, spiegandone i vantaggi rispetto a quelle più conosciute. Unifind è un'applicazione per l'affitto di abitazioni, realizzata specificatamente per studenti universitari: per questo motivo, se si intende affittare un'abitazione per un lungo periodo di tempo (almeno 9 mesi), la superiorità di Unifind rispetto ad altre applicazioni è schiacciante; infatti, su Unifind verranno filtrate le abitazioni in modo tale che l'utente visualizzi solo quelle che corrispondono ai criteri scritti sopra, senza che quest'ultimo inserisca alcun filtro.

Un'altra comodità implementata su Unifind per rendere migliore l'esperienza degli utenti è la chat. Una volta salvata un'abitazione, Unifind permette di chattare con tutti gli altri utenti interessati, in modo da rendere più semplice l'organizzazione e la creazione di gruppi di studenti per l'occupazione di abitazioni con molti posti letto.

Un'ulteriore funzionalità importante offerta da Unifind è la possibilità di lasciare recensioni sulle abitazioni, senza il bisogno di averle obbligatoriamente occupate. In questo modo, anche coloro che hanno visitato un'abitazione particolarmente brutta o bella possono farlo sapere agli altri utenti, risparmiando loro il tempo della visita.

5.2 Accesso all'applicazione

In questa sezione forniremo le istruzioni per gli utenti che desiderano accedere all'applicazione. Quando si accede ad Unifind, viene richiesto di autenticarsi. Per fare questo si hanno due diverse modalità di accesso: registrazione e login.

5.2.1 Registrazione

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.1. Il "pulsante indietro" consente di tornare alla schermata *Login*. Per registrarsi nell'applicazione occorre digitare l'e-mail nell'apposito "campo e-mail". Questa deve essere valida, ovvero seguire il formato X@X.X dove X è una qualsiasi stringa composta da caratteri che diversi dalla @. Nel "campo password" e nel campo"ripeti password" è necessario inserire, invece, la password in maniera identica. L'unica limitazione ad essa imposta è che la sua lunghezza superi gli 8 caratteri. Una volta riempiti correttamente i campi è possibile registrarsi nell'applicazione premendo il "pulsante di registrazione". Nel caso in cui i dati inseriti non fossero corretti viene visualizzato un messaggio di errore.

Figura 5.1: Analisi dell'interfaccia di registrazione

5.2.2 Login

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.2. Per accedere all'applicazione da questa schermata è necessario avere un account registrato. Inoltre, occorre inserire le credenziali di accesso in maniera corretta e negli appositi spazi. Bisogna, quindi, inserire la propria e-mail nel "campo e-mail" e la password nel "campo password". Una volta inserite entrambe le credenziali correttamente, premendo il "pulsante di login", se le credenziali sono corrette si avrà accesso all'applicazione, altrimenti verrà visualizzato un messaggio di errore. Premendo "registrati" si accede alla schermata di registrazione.

5.3 Istruzioni per l'uso delle sezioni principali

In questa sezione analizzeremo il funzionamento dell'applicazione, dividendola nelle sezioni della barra di navigazione.

Figura 5.2: Analisi dell'interfaccia di login

5.3.1 Mappa

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.3. La "barra di ricerca" in alto consente di ricercare una località (ad esempio, la sede di un'università); una volta ricercata, vi si posiziona il "centro della ricerca". Un altro modo per posizionare il centro della ricerca è tramite il click sulla mappa. Una volta che il centro della ricerca viene posizionato, appariranno i "marker di un'abitazione" sulla mappa nelle posizioni in cui si trovano le case che rispettano i nostri parametri di ricerca (impostati dai filtri, come spiegheremo in seguito nella sezione *Filtri*). Una volta cliccato il "marker di un'abitazione", verremo riportati alla schermata con i dettagli ad essa relativi (questa schermata verrà analizzata di seguito nella sezione *Più informazioni*). Tramite la "navigation bar" è possibile accedere a schermate diverse cliccando nei simboli corrispondenti.

5.3.2 Lista

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.4. Il "pulsante filtri" permette di visualizzare la schermata relativa ai filtri (che analizzeremo in seguito), una volta cliccato. Il "rating" permette di visualizzare il voto medio attribuito dagli utenti all'abitazione che si sta visualizzando. "Città", "indirizzo" e "prezzo" permettono, rispettivamente, di conoscere città, indirizzo e prezzo dell'abitazione visualizzata. L'"immagine abitazione" permette di visualizzare un'anteprima dell'abitazione. Il "pulsante preferiti pieno" consente di sapere che l'abitazione corrente è salvata all'interno della nostra lista dei preferiti; cliccandolo questa verrebbe rimossa e l'icona del pulsante verrebbe svuotata (come il "pulsante preferiti vuoto" sottostante). Tutte queste informazioni si trovano all'interno della

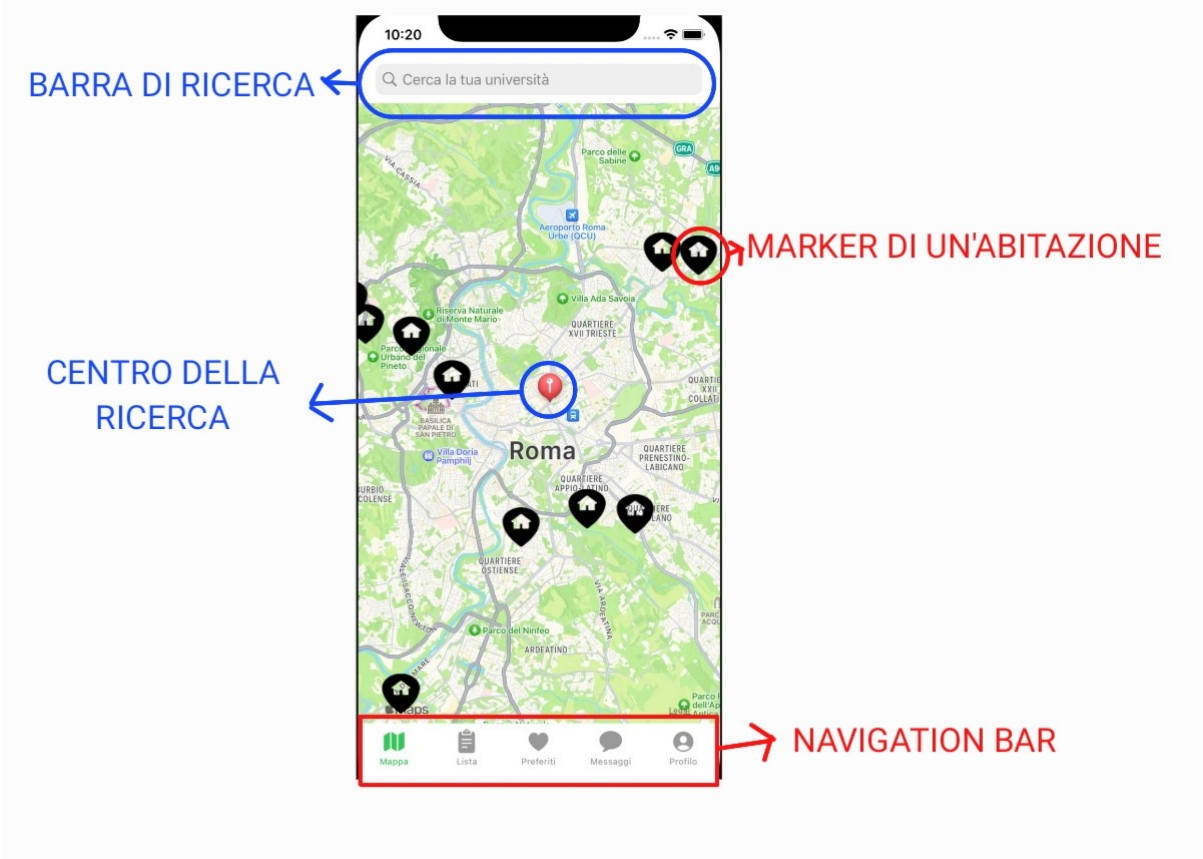

Figura 5.3: Analisi dell'interfaccia della mappa

"card abitazione", la quale, una volta cliccata, permette di accedere alla schermata contenente i dettagli dell'abitazione (che analizzeremo in seguito).

Filtri

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.5. Il "pulsante indietro" permette di tornare alla schermata della lista di abitazioni. Il "pulsante azzera" consente di rimuovere tutti i filtri inseriti finora riportandoli allo stato che avevano quando abbiamo aperto l'interfaccia. Lo "slider del prezzo" consente di modificare il prezzo massimo da pagare mensilmente; la "scelta ordinamento" consente di variare l'ordine di visualizzazione delle abitazioni nella sezione lista; il "menù coinquilini" una volta cliccato, consente di selezionare un numero da 1 a 10, che rappresenta il numero minimo di inquilini che le abitazioni visualizzate possono contenere. Infine, lo "slider distanza" consente di selezionare la massima distanza che le abitazioni possono avere dal centro selezionato. Il "pulsante applica" permette di salvare i filtri inseriti ed applicarli alle prossime ricerche che verranno effettuate.

Più informazioni

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.6. Il "pulsante indietro" permette di tornare alla schermata precedente. L'"immagine abitazione" permette di visualizzare un'immagine relativa all'abitazione corrente. Nella sezione "informazioni abitazione" sono contenute tutte le informazioni relative all'abitazione che stiamo visualizzando (l'indirizzo, la provincia, le dimensioni in metri quadri, il prezzo mensile,

Figura 5.4: Analisi dell'interfaccia della lista di abitazioni

il numero di camere da letto, il numero di bagni ed una breve descrizione). Il "pulsante di chat" è presente solo perchè l'abitazione in questione è tra i preferiti e, quando viene cliccato, consente di aprire il menù di selezione degli utenti con i quali è possibile iniziare una conversazione (le specifiche di questa schermata saranno spiegate nella sezione chat, poichè l'interfaccia grafica è identica).

5.3.3 Preferiti

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.7. Come possiamo vedere dalla figura ad essa relativa, questa interfaccia è quasi identica alla schermata della lista e, per questo, ometteremo le spiegazioni che riguardano gli stessi elementi. Anche a livello di funzionalità, le schermate sono molto simili, l'unica differenza sta nel fatto che, una volta premuto il "pulsante preferiti pieno", l'abitazione corrispondente verrà rimossa dai preferiti e, di conseguenza, scomparirà anche dalla schermata.

5.3.4 Chat

Per rendere chiare le spiegazioni che forniremo faremo riferimento alla Figura 5.8. Il "contatto disponibile" rappresenta coloro con cui è stata aperta una conversazione. Una volta cliccato, verrà aperta la chat relativa all'utente selezionato (analizzeremo la schermata della chat di seguito). Anche in questo caso, la "navigation bar" permette di cambiare schermata selezionando l'icona ad essa relativa.

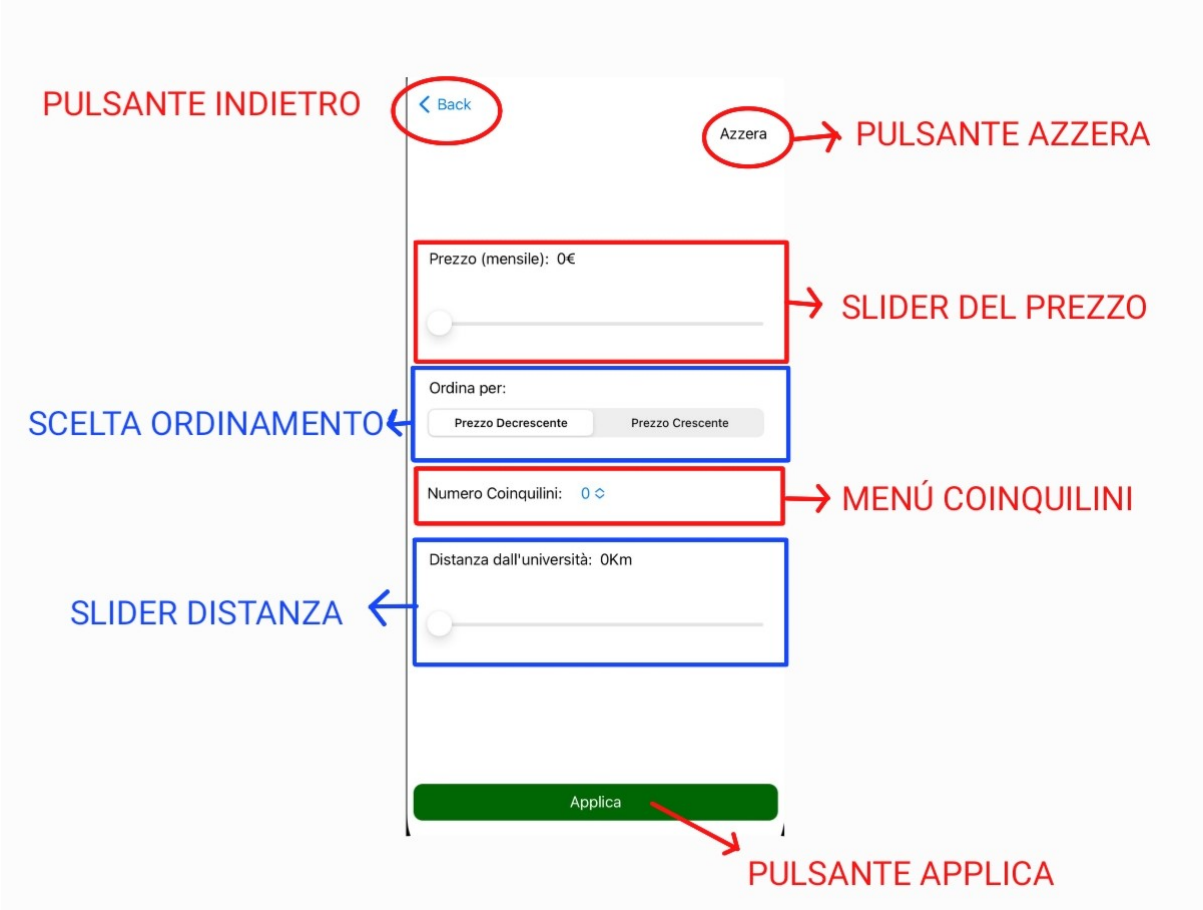

Figura 5.5: Analisi dell'interfaccia dei filtri

Messaggi

Per rendere chiare le spiegazioni che forniremo faremo riferimento alla Figura 5.9. Il "pulsante indietro" permette di tornare alla schermata di selezione della chat. La "card abitazione" permette di visualizzare l'abitazione oggetto della conversazione tra i due utenti ed indica, anche, quante altre abitazioni i due utenti abbiano in comune. Nel caso in cui venga cliccata la "card abitazione", si apre la schermata *Più informazioni* relativa alla proprietà in essa contenuta; nel caso in cui vi siano altre proprietà in comune, è possibile visualizzare anche la schermata *Più informazioni* ad esse relativa (l'analisi di questa schermata è riportata in precedenza nella sezione *Più informazioni*). Il "contatto utente" permette di visualizzare e-mail ed immagine di profilo dell'utente con cui si sta chattando. Una volta cliccato, si apre la schermata che contiene le informazioni del corrispettivo utente (l'analisi di questa schermata è riportata in seguito). Il "messaggio inviato" rappresenta un messaggio inviato dall'utente mentre il "messaggio ricevuto" rappresenta un messaggio ricevuto dall'utente. La "sezione scrittura messaggio" permette di scrivere il testo di un messaggio che verrà inviato dopo aver premuto il "pulsante invia messaggio".

Profilo dell'utente con cui si sta chattando

Per rendere chiare le spiegazioni che forniremo faremo riferimento alla Figura 5.10. Il "pulsante indietro", quando premuto, consente di tornare alla schermata della conversazione tra i due utenti. L'"immagine utente" riporta l'immagine che l'utente visualizzato ha scelto

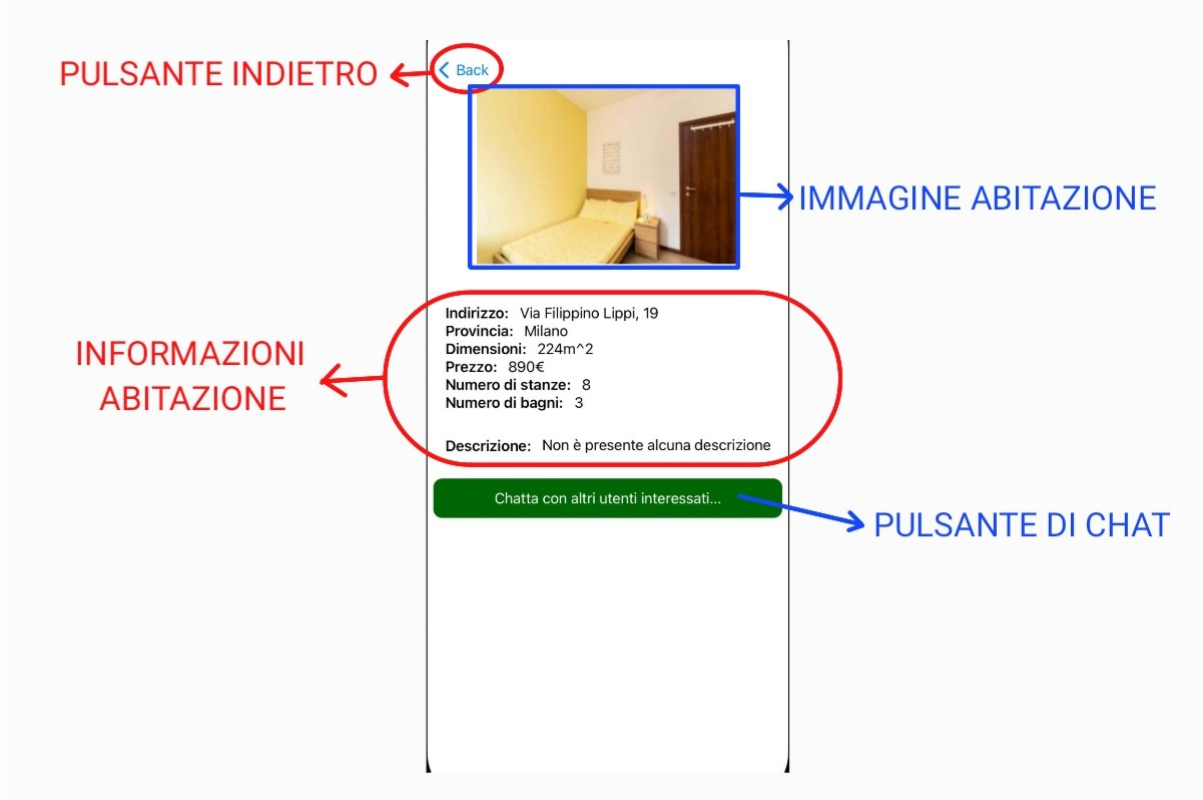

Figura 5.6: Analisi dell'interfaccia più informazioni relative ad un'abitazione

come sua immagine di profilo; le "informazioni utente" riportano nell'ordine nome, cognome, e-mail, numero di telefono e descrizione del profilo che si sta visionando.

5.3.5 Profilo

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.11. La sezione "immagine di profilo" è la porzione dell'interfaccia dedicata all'immagine del profilo dell'utente. Da questa è possibile visualizzare l'immagine attualmente salvata dall'utente e cliccandoci, è possibile modificarla e salvarne un'altra. Nella sezione "informazioni utente" sono visualizzate tutte le informazioni dell'utente, eccetto per la password. Queste saranno modificabili dai pulsanti sottostanti. Il "pulsante modifica e-mail" è il pulsante che consente di modificare l'informazione relativa all'e-mail. Una volta cliccato, verrà visualizzata l'interfaccia relativa all'e-mail, che analizzeremo in seguito. Il "pulsante modifica password" permette, una volta cliccato, di visualizzare l'interfaccia per modificare la password, che analizzeremo in seguito. Il "pulsante completa profilo" permette di modificare tutte le informazioni relative all'utente diverse da e-mail e password da un'apposita interfaccia, che analizzeremo in seguito. Il "pulsante logout" permette di uscire dal proprio profilo tornando alla schermata di *Login*. Anche in questa interfaccia la " *Navigation Bar* permette di variare l'interfaccia visualizzata cliccando sulle icone presenti al suo interno.

Modifica e-mail

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.12. Il "pulsante indietro" permette di tornare alla schermata di visualizzazione del profilo, una volta cliccato. La sezione "nuova e-mail" permette di inserire una nuova e-mail per il proprio profilo

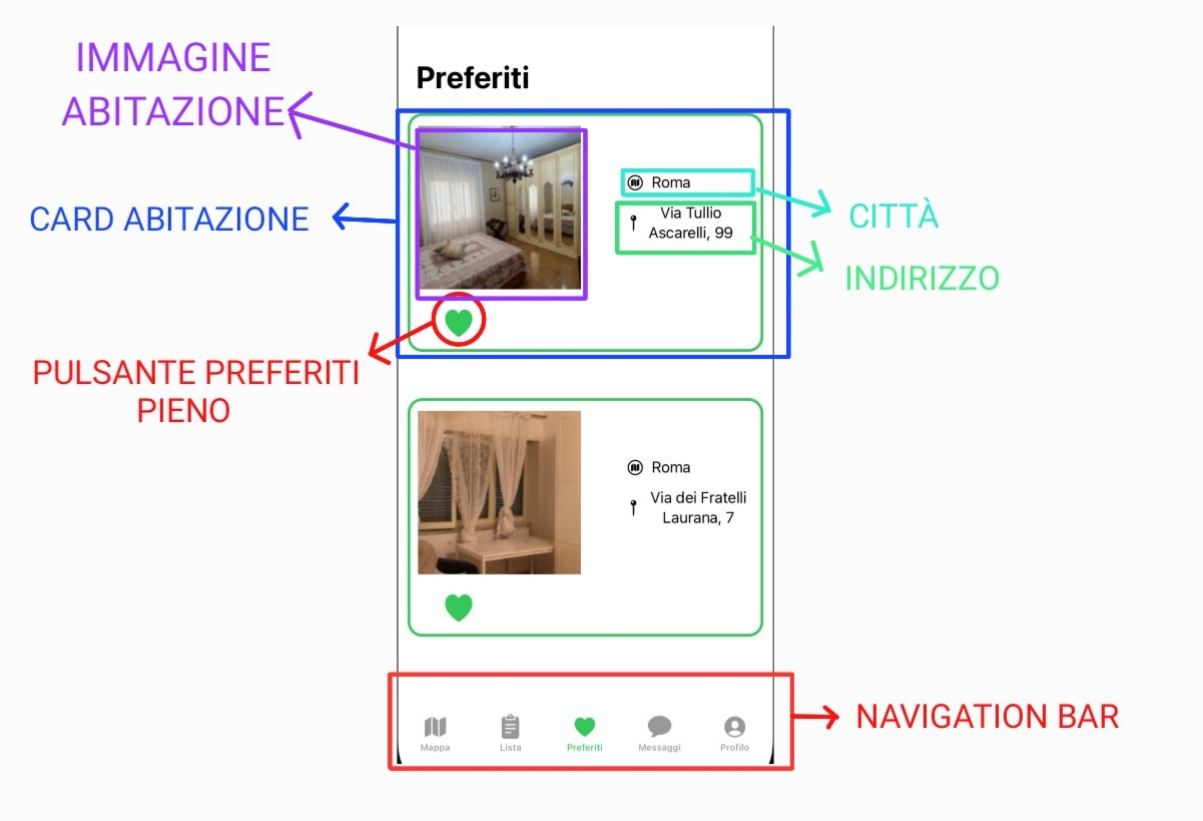

Figura 5.7: Analisi dell'interfaccia dei preferiti

utente. Il "pulsante salvataggio modifiche" permette di salvare la mail inserita sostituendola a quella precedente.

Modifica password

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.13. Il "pulsante indietro", quando viene cliccato, visualizza la schermata precedente (quindi quella di profilo). La "sezione nuova password" è la parte della schermata che permette di inserire una nuova password, che andrà confermata nella "sezione password 2". Il "pulsante salvataggio modifiche", una volta cliccato, permetterà di sostituire l'attuale password con quella appena inserita.

Modifica Informazioni profilo

Per rendere chiare le spiegazioni che forniremo, faremo riferimento alla Figura 5.14. Il "pulsante indietro" permette di visualizzare nuovamente l'interfaccia di profilo dell'utente. Il "campo nome" ed il "campo cognome" permettono di inserire al loro interno un nuovo nome e cognome da inserire nel proprio profilo. Il "campo telefono" permette di inserire il proprio numero di telefono, mentre il "campo descrizione" permette di modificare la descrizione precedente. Per salvare tutte le modifiche apportate, è necessario premere il "pulsante salva modifiche" che, inoltre, ci riporterà alla schermata di profilo con le informazioni aggiornate.

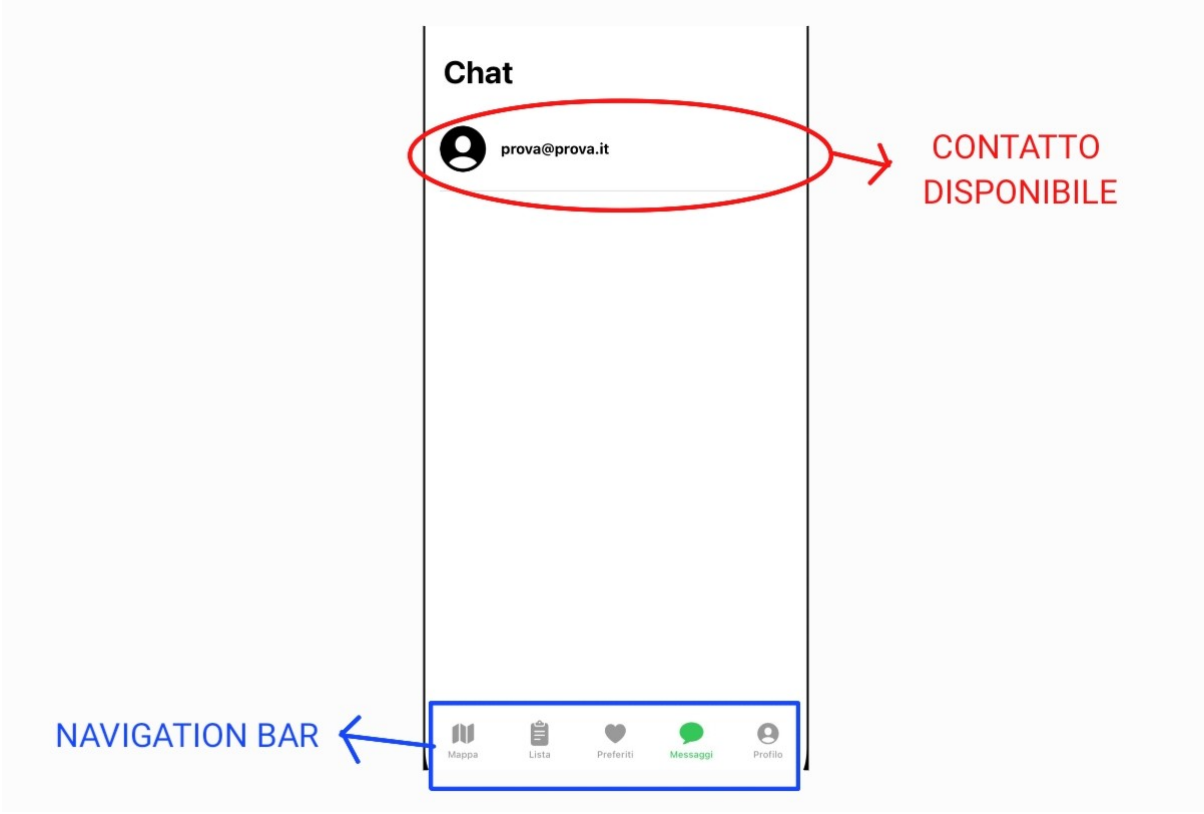

Figura 5.8: Analisi dell'interfaccia della chat

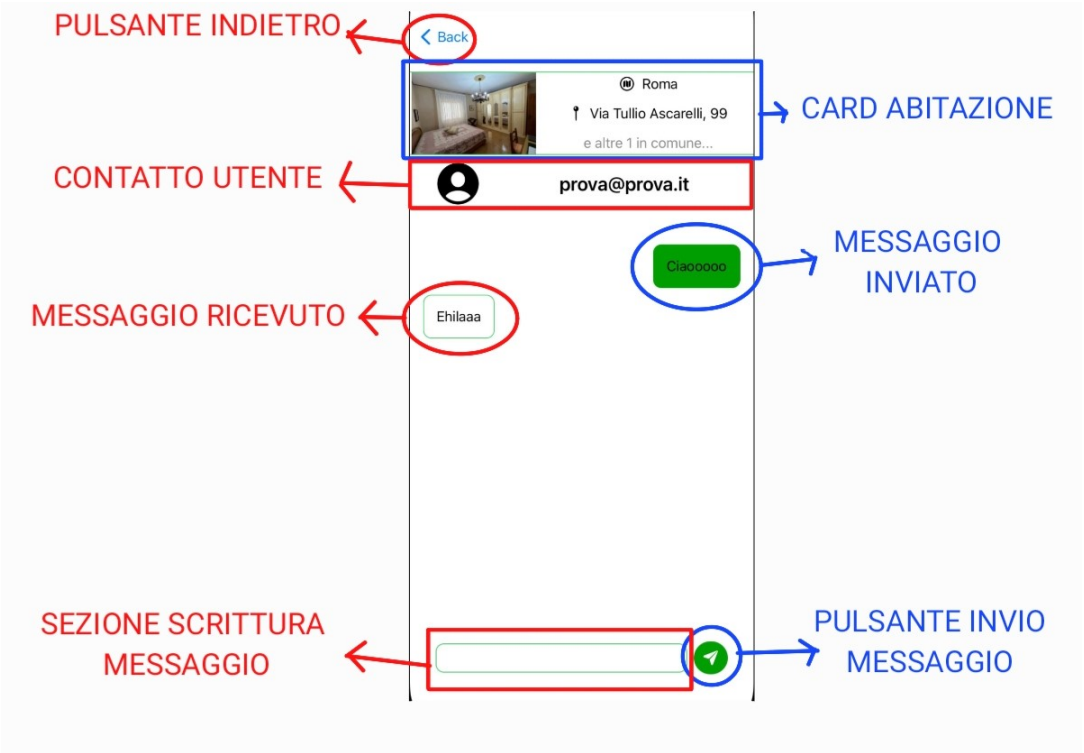

Figura 5.9: Analisi dell'interfaccia dei messaggi

Figura 5.10: Analisi dell'interfaccia del profilo dell'utente con cui si sta chattando

Figura 5.11: Analisi dell'interfaccia del profilo

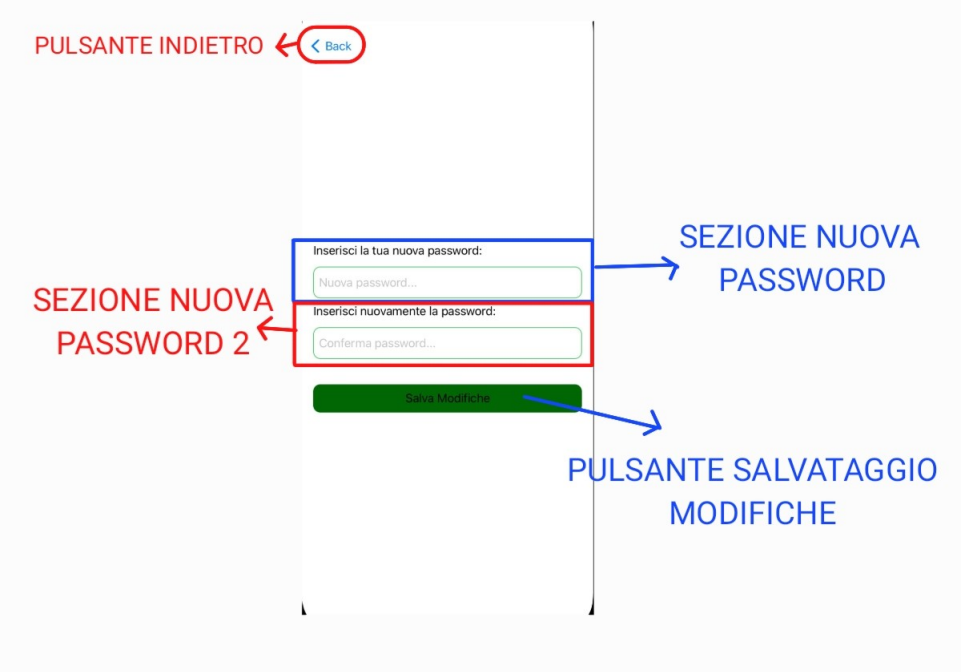

Figura 5.12: Analisi dell'interfaccia di modifica dell'e-mail

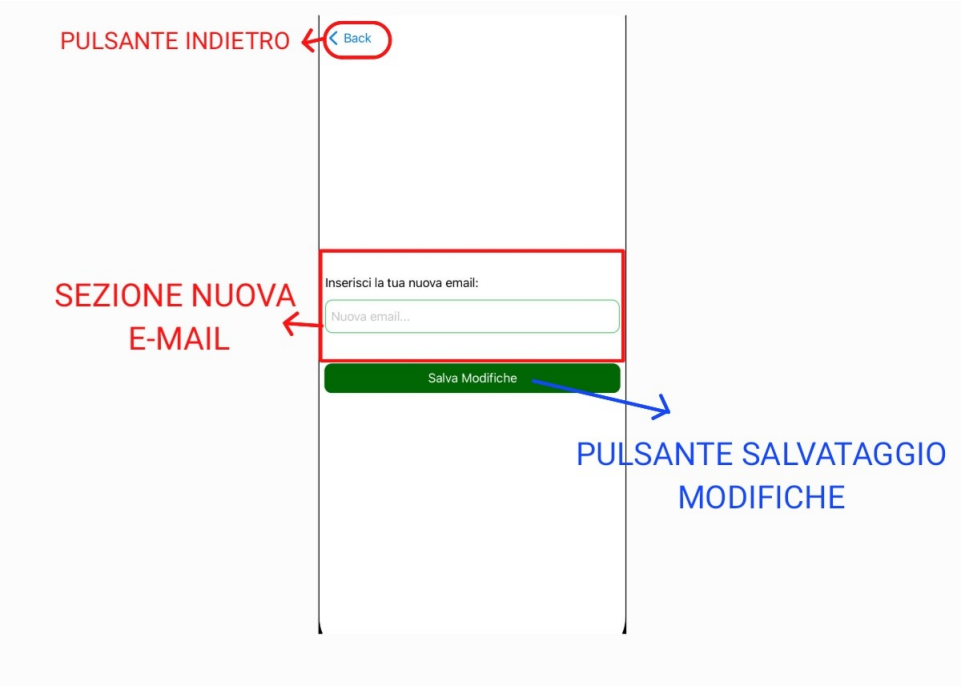

Figura 5.13: Analisi dell'interfaccia di modifica della password

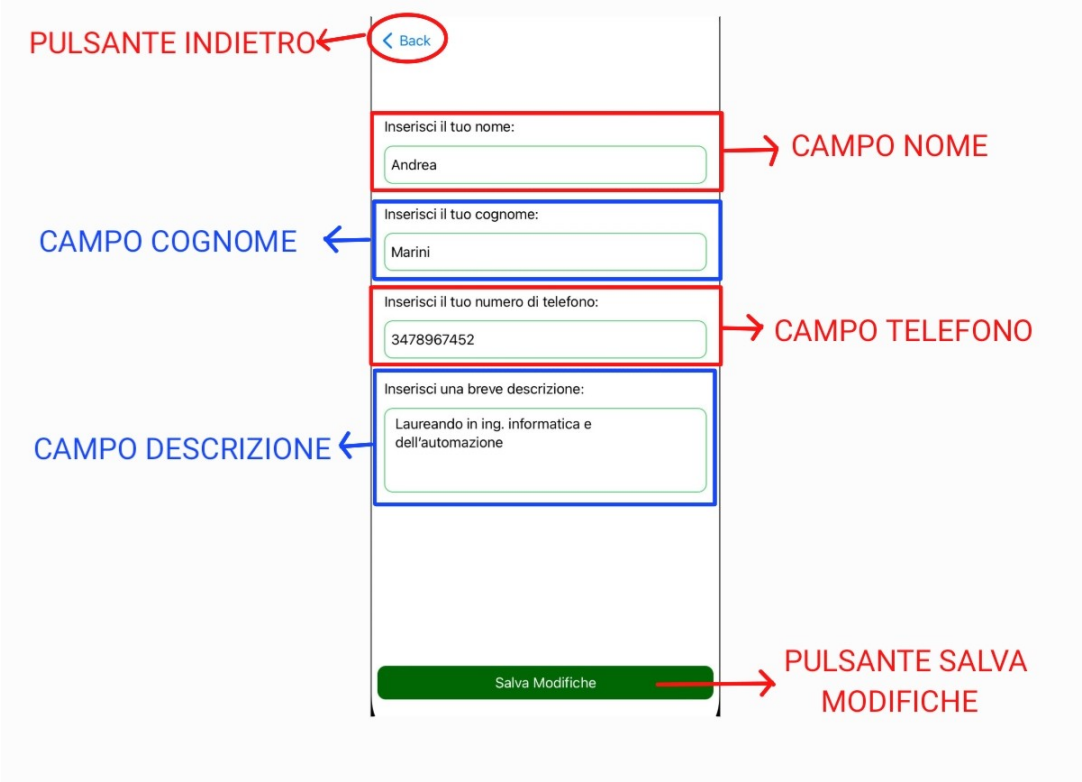

Figura 5.14: Analisi dell'interfaccia di completamento del profilo

CAPITOLO 6

Discussione

Nel capitolo corrente verranno prese in esame le caratteristiche dell'applicazione Unfind in un contesto più ampio rispetto a quelli analizzati finora. Il capitolo si apre con l'analisi SWOT (Strengths, Weaknesses, Opportunities, Threats) di Unifind, che ne elenca le caratteristiche divise in sezioni.

Di seguito verranno confrontate le funzionalità di Unifind con i maggiori siti immobiliari, in modo da capire quali potrebbero essere i maggiori punti di forza di quest'applicazione, nel caso in cui venisse pubblicata. I siti presi in esame sono nell'ordine: Idealista, Immobiliare, Booking, Casa e Trovacasa. Il capitolo sarà diviso in sezioni dedicate ai singoli confronti, dove vengono elencate le caratteristiche presenti nelle applicazioni, confrontandole con l'altra. Così facendo il lettore, una volta letti i confronti, otterà gli elementi per scegliere quale delle applicazioni sia superiore.

6.1 Analisi SWOT

L'analisi SWOT (Strengths, Weaknesses, Opportunities, Threats) è un'analisi utilizzata per spiegare le caratteristiche del software preso in esame. Divideremo questa sezione in quattro parti, corrispondenti alle quattro voci principali dell'analisi che compieremo.

6.1.1 Punti di forza

I principali punti di forza di Unifind sono i seguenti:

- *Specificità:* essendo pensata per studenti universitari, l'applicazione Unifind esprime il suo massimo potenziale quando utilizzata per ricercare abitazioni nei dintorni di luoghi specifici. Nel caso in cui si vogliano cercare immobili vicino ad un'università, sarà sufficiente ricercarne il nome o la sede e verranno mostrate sulla mappa tutte le abitazioni disponibili, senza il bisogno di dover filtrare i risultati, come accade sugli altri siti immobiliari.
- *Utilizzo API di GoogleMaps:* all'interno del codice dell'applicazione Unifind vengono effettuate delle richieste API a GoogleMaps. In questo modo l'utente è sicuro di avere a sua disposizione l'elenco di località più aggiornato che esista e, quindi, che le sue ricerche producano il risultato con le coordinate più precise possibili.
- *Alto coinvolgimento degli utenti:* nella nostra applicazione l'utenza è al centro del progetto. Infatti, è grazie agli utenti che le abitazioni vengono recensite ed è possibile farsi un'idea di come possano essere, anche prima di visitarle. Inoltre, è anche possibile un'interazione diretta con gli altri utenti tramite la chat. Grazie alle conversazioni è,

infatti, possibile, sia organizzarsi per l'affitto di determinate abitazioni, ma anche la richiesta di chiarimenti: per esempio, nel caso in cui un'utente abbia dei dubbi su una determinata abitazione, può rivolgersi ad un utente che l'abbia già visitata, o che vi abbia già soggiornato, senza il bisogno di recarvisi di persona.

6.1.2 Punti di debolezza

I principali punti di debolezza sono i seguenti:

- *Non pubblica:* non essendo ancora stata rilasciata pubblicamente, Unifind al momento non dispone di una grande utenza. Per questo motivo, tutte le funzionalità basate sull'utenza (chat, recensioni, etc.) sono al momento molto limitate. Questo problema verrebbe comunque risolto una volta pubblicata l'applicazione.
- *Piano API gratuito:* le abitazioni ricercabili all'interno di Unifind sono quelle fornite dalle API gratuite di Idealista. Per questo motivo, non sono presenti molte case ed il numero di richieste effettuabili è limitato. Nel caso in cui l'applicazione dovesse essere pubblicata a scopo di lucro, anche questo problema potrebbe essere risolto tramite la collaborazione con vari siti immobiliari, o il pagamento di servzi API migliori rispetto a quelle attuali, oppure ancora, con lo sviluppo di una sezione dell'applicazione dedicata ai proprietari delle abitazioni.
- *Impossibilità di interazione con i proprietari delle abitazioni:* non essendo stata pensata per la pubblicazione, Unifind sfrutta delle API esterne per fornire delle abitazioni ai propri utenti. Questo, però, non rende possibile l'affitto effettivo delle case, poichè non è presente un utente dedicato a coloro che mettono abitazioni in affitto. Pensando ad una futura pubblicazione, questo problema andrebbe risolto implementando una parte di gestione delle abitazioni da parte di utenti creati appositamente per i proprietari di immobili, oppure tramite la collaborazione con altri siti immobiliari che permettano di usufruire delle loro abitazioni e anche di interagire con i loro utenti con abitazioni in affitto.

6.1.3 Opportunità

La principale opportunità di Unifind consiste nell'assenza di un prodotto simile: l'idea per questa applicazione nasce dalla necessità di utilizzare un tool per ricercare soltanto abitazioni per studenti universitari. Al momento, questo tool non esiste sul mercato o ha un'utenza molto limitata ed è così che abbiamo deciso di svilupparne uno nostro. Per questo motivo riteniamo che, dopo un'eventuale pubblicazione, Unifind avrebbe un grande successo.

6.1.4 Minacce

Le principali minacce di Unifind sono le seguenti:

- *Sviluppo indipendente:* essendo stata sviluppata per scopi accademici, Unifind anche se pubblicata, riceverebbe grandi investimenti a scopo pubblicitario. Questo potrebbe portare al fatto che degli sviluppatori appartenenti a grandi aziende, una volta notate le grandi potenzialità dell'applicazione, ne copino i contenuti pubblicando un'applicazione molto simile, però pubblicizzandola di più.
- *Scarsa sicurezza;* essendo stata creata per scopi accademici, l'applicazione Unifind si appoggia su un database gratuito di Firebase, che ,quindi, può contenere una limitata

quantità di dati. Questi inoltre, essendo su un sito pubblico e molto conosciuto, sarebbero esposti a varie minacce. Occorerebbe, di conseguenza, riprogrammare la parte relativa alla gestione dei dati prima di rendere pubblica l'applicazione.

6.2 Confronto

In questa sezione confronteremo il nostro software con applicazioni simili del settore immobiliare. Prima di iniziare specifichiamo delle ipotesi alla base di questo confronto, che, altrimenti, risulterebbe impari. Come già accennato in precedenza, Unifind è stata sviluppata per scopi accademici. Al momento, risulta, quindi, incompleta e non pronta per la pubblicazione. Per questo motivo, durante il confronto supporremo invece che:

- All'interno di Unifind sia presente un numero di utenti, e quindi di abitazioni, simile a quello del software con cui la siamo confontando.
- All'interno di Unifind sia presente la possibilità di prenotare abitazioni interagendo direttamente con i proprietari.
- Il database di Unifind sia sicuro e possa ospitare un numero di informazioni adeguato alla sua utenza.
- Le abitazioni presenti nell'applicazione vengano scaricate da un proprio database e non tramite una ricerca API su un'altro sito immobiliare (e, quindi si supporrà che siano accessibili anche più informazioni, ad esempio più immagini della stessa casa, recapiti del proprietario, etc.)
- Gli utenti siano tutti interessati soltanto all'affitto di abitazioni per periodi superiori ai 9 mesi.
- L'utente che accede venga memorizzato e non sia, quindi, necessario accedere tutte le volte che si avvia l'app.

Poste queste ipotesi effettueremo i confronti con i software più utilizzati e famosi in Italia in questo settore, ovvero: Idealista, Immobiliare, Booking, Casa e Trovacasa.

6.2.1 Confronto con Idealista

Idealista, rispetto ad Unifind, offre una gamma di servizi più ampia. Questo è un vantaggio per tutta l'utenza interessata ad affitti a breve termine, brevi soggiorni o acquisti di immobili; ma, supponendo come detto sopra che tutti gli utenti siano interessati al solo affitto per lunghi periodi, queste funzionalità risulterebbero superflue. Di conseguenza, su Idealista un utente si troverebbe a dover filtrare per il periodo di affitto desiderato, prima di poter visualizzare delle abitazioni adatte alle sue esigenze. Una funzionalità presente su Idealista che Unifind non offre è la possibilità di disegnare la propria area di interesse all'interno di una cartina geografica, così da ottenere risultati solo all'interno dell'area stabilita. Unifind permette, invece, di prendere in esame soltanto un'area dalla forma circolare, che non può, quindi, adattarsi sempre perfettamente alle esigenze dell'utente. Inoltre, Idealista offre una gamma di filtri superiore a quella presente in Unifind. Un punto a favore di Unifind è la possibilità di comunicare con altri utenti tramite abitazioni in comune, funzionalità non presente su Idealista. Unifind permette, inoltre, di memorizzare le ricerche effettuate, così da non doverle ripetere una volta riaperta l'app.

6.2.2 Confronto con Immobiliare

Come in precedenza, ricordiamo che Immobiliare offre una gamma di servizi molto più ampia rispetto a quella di Unifind, ma noi analizzeremo solo il caso in cui l'utente debba usufruire di quelli che permettono di affittare abitazioni per lunghi periodi. Immobiliare offre la possibilità di effettuare ricerche specifiche e memorizzarle come Unifind; inoltre, una volta memorizzata una ricerca, offre la possibilità di ricevere una notifica nel caso in cui venga inserita da un'utente una nuova abitazione che soddisfa i parametri della ricerca salvata, funzionalità non presente in Unifind. Anche all'interno di Immobiliare è possibile tracciare un'area di forma irregolare sulla mappa per ottenere come risultato delle proprie ricerche solo le abitazioni all'interno di quell'area. Anche in questo caso, i filtri presenti su Immobilare risultano maggiori, ma si è costretti ad utilizzarli prima di effettuare una ricerca, nel caso che stiamo analizzando in questo confronto (quindi l'affitto di abitazioni per lunghi periodi). Infine su Unifind è possibile comunicare direttamente con altri utenti, funzionalità che non risulta presente su Immobiliare.

6.2.3 Confronto con Booking

Anche per Booking è doveroso ricordare l'ampissima gamma di servizi che quest'applicazione offre; però, come nei casi precedenti, prenderemo in esame solo il caso in cui l'utenza sia interessata ad affittare abitazioni per lunghi periodi. Booking offre la possibilità di prenotare trasporti, come treni aerei o traghetti, cosa che potrebbe risultare utile ad uno studente universitario fuori sede, in determinate occasioni. Inoltre, Booking offre una gamma di filtri da applicare alle abitazioni molto superiore rispetto a quello presente su Unifind. Dal punto di vista della mappa, le due applicazioni risultano simili, in quanto su entrambe è possibile visualizzare tutte le abitazioni in un'area prestabilita direttamente dalla cartina geografica. Anche in questo caso, Unifind risulta superiore dal punto di vista dell'interazione tra gli utenti grazie alla funzionalità della chat. Inoltre, come accade in tutti gli altri casi, anche su Booking, prima di poter effettuare una ricerca per abitazioni in affitto su lunghi periodi, è necessario inserire gli appositi filtri. Un'altra nota a favore di Unifind è il fatto che su Booking ogni volta che l'applicazione viene chiusa si perde memoria delle ultime abitazioni visualizzate.

6.2.4 Confronto con Casa

Diversamente dai precedenti siti presi in analisi, Casa permette ai suoi utenti soltanto di affittare mensilmente o di acquistare un'abitazione. Noi prenderemo in esame il caso dove l'utente voglia affittare un'immobile per un periodo superiore ai 9 mesi. In questo caso, su Casa i filtri presenti sono molto più vari di quelli disposti da Unifind. Inoltre, su Casa risulta possibile disegnare un'area all'interno della quale si stanno cercando delle abitazioni, come in alcuni dei siti visti in precedenza. Un'altro metodo di ricerca presente su Casa è la ricerca per località equivalente a quella implementata su Unifind. Come nei casi precedenti, su Casa non risulta possibile comunicare con altri utenti interessati ad un'abitazione. Per effettuare una ricerca mirata ad affitti su lunghi periodi non occorre inserire alcun filtro né su casa, né su Unifind; quindi, da questo punto di vista, le due applicazioni si equivalgono. Anche su Casa è possibile memorizzare le proprie ricerche una volta effettuate, così da non doverle ripetere ogni volta che si apre l'applicazione. A nostro avviso, Casa risulta il sito immobiliare migliore per l'affitto di immobili tra quelli presi in esame fino ad ora.

6.2.5 Confronto con Trovacasa

Trovacasa offre numerosi servizi che non verranno citati in questo confronto, poichè si prenderà in esame soltanto il caso in cui l'utente sia interessato all'affitto di abitazioni su lunghi periodi. Per effettuare una ricerca di immobili disponibili all'affitto su Trovacasa non occorre inserire alcun filtro, ma è sufficiente entrare nella sezione dedeicata all'affitto. Inoltre, i filtri che possono essere inseriti sono maggiori di quelli disponibili su Unifind. Anche su Trovacasa risulta possibile salvare una propria ricerca, in modo da non doverla effettuare nuovamente ogni volta che si utilizza l'app. Su Trovacasa non è possibile effettuare ricerche direttamente dalla mappa e neanche definire un'area delimitata all'interno della quale si vogliono ottenere i risultati delle proprie ricerche. Infine, non è possibile interagire con altri utenti interessati all'affitto di abitazioni. Per questo motivo, a nostro parere, Trovacasa risulta il sito immobiliare meno valido tra quelli presi in esame.

CAPITOLO 7

Conclusioni

L'idea per il nostro progetto nasce dalle difficoltà che gli studenti universitari possono incontrare nella ricerca di abitazioni per trascorrere il periodo accademico fuori sede. Confrontandoci con altri studenti, ci siamo accorti della mancanza di un software dedicato esclusivamente a questa utenza ristretta e abbiamo pensato di realizzarne uno.

Di conseguenza, abbiamo iniziato la realizzazione di questo progetto occupandoci, innanzitutto, di trovare una fonte di abitazioni che potesse soddisfare le ricerche degli utenti. Una volta trovata, ci siamo occupati della realizzazione dei diagrammi relativi al database della nostra applicazione, in modo da essere sicuri di realizzare entità adatte ad ospitare i dati fornitici da Idealista.

In seguito, abbiamo realizzato l'analisi dei requisiti e la descrizione degli obiettivi, per avere un'idea generale di quelli che sarebbero stati i punti focali di Unifind. Per soddisfare tutti i requisiti e gli obiettivi da noi posti, ci siamo figurati alcuni dei casi d'uso che l'applicazione avrebbe dovuto affrontare, in modo da rendere più piacevole (a nostro parere) l'esperienza degli utenti. Infine, ci siamo occupati di creare i diagrammi delle classi, quelli di sequenza e di deployment in modo da terminare la fase di progettazione.

Per la realizzazione dell'applicazione, abbiamo dato la priorità alla creazione delle interfacce grafiche, così da sapere quali fossero le informazioni che ognuna di esse avrebbe dovuto contenere. In seguito, abbiamo implementato le funzionalità di ogni interfaccia, iniziando da quelle di autenticazione per poi proseguire con quelle riguardanti il profilo dell'utente. Subito dopo, abbiamo usufruito delle API di GoogleMaps per poi implementare le funzionalità della mappa; così facendo, è stato possibile in seguito implementare anche quelle riguardanti le abitazioni (lista delle abitazioni, preferiti, etc.). Infine, abbiamo realizzato le interazioni tra gli utenti, comprendenti la chat e la visualizzazione del profilo altrui.

Per quanto riguarda il futuro di Unifind, se questa dovesse essere pubblicata, sarebbero richieste alcune migliorie, che ne rendano possibile un utilizzo su una scala più ampia di utenza (come accennato nel capitolo precedente). In particolare, sarebbe necessario che l'applicazione usufruisse di servizi API diversi da quelli attualmente presenti, per esempio pagando quelli offerti da Casa o da Immobiliare; l'alternativa sarebbe che Unifind implementasse delle funzionalità dedicate a coloro che mettono in affitto le proprie abitazioni, così da acquisire un'utenza composta dai proprietari immobiliari.

Un'altra miglioria necessaria per rendere l'applicazione pronta al rilascio è una differente gestione del database; sarebbe, infatti, necessario usufruire di un database diverso da quello offerto gratuitamente da Firebase. Anche le entità all'interno del database dovrebbero subire

delle modifiche, in modo da poter ospitare le informazioni relative ai proprietari delle abitazioni.

Inoltre, andrebbe migliorata la parte riguardante la chat, implementando, per esempio, le funzionalità di invio di immagini, video o documenti. Occorrerebbe infine, implementare la funzionalità di ricezione di notifiche da parte dell'utente che riceve messaggi, così da rendere migliori le comunicazioni tra due utenti.

A seguito di queste modifiche, riteniamo che l'applicazione possa essere pubblicata e svolgere adeguatamente il compito di semplificare la ricerca di immobili da parte di studenti fuori sede.

Bibliografia

BARTLETT, D. (2019), *Swift Programming in easy steps*, In Easy Steps.

BUTTFIELD-ADDISON, P. e MANNING, J. (2021), *Head First Swift*, "O'Reilly Media, Inc.".

HOFFMAN, J. (2019), *Mastering Swift 5*, Packt Publishing Ltd.

HOLLEMANS, M. (2014), *IOS Apprentice*, Matthijs Hollemans, Fahim Farook.

KEUR, C. e HILLEGASS, A. (2020), *iOS programming: the Big Nerd Ranch guide*, Big Nerd Ranch.

MANNING, J., BUTTFIELD-ADDISON, P. e NUGENT, T. (2018), *Learning Swift : building apps for macOS, iOS, and beyond*, O'reilly Media Inc.

MATHIAS, M. e GALLAGHER, J. (2016), *Swift Programming*, Pearson Technology Group.

MOON, K. e BARKER, C. (2021), *Swift Cookbook*, Packt Publishing Ltd.

SIMON e SCHUSTER (2018), *Swift in Depth*, Simon and Schuster.

TEAM, K., BELLO, A., MOREFIELD, B., REICHELT, S. e TAM, A. (2023), *SwiftUI by Tutorials (Fifth Edition)*, Bill Morefield, Antonio Bello, Audrey Tam, Sarah Reichelt.

Siti web consultati:

- The Good and the Bad of Swift Programming Language – `https://betterprogramming.pub/swiftui-2021-the-good-the-bad-and-the-ugly`
- Swift Introduction - Javatpoint – `https://www.javatpoint.com/swift-introduction`
- Swift Documentation – `https://docs.swift.org/swift-book/documentation/the-swift-programming-language/guidedtour/`
- Swift - Apple Developer – `https://developer.apple.com/swift/`
- Apple Developer Documentation – `https://developer.apple.com/tutorials/swiftui`
- SwiftUI by Example - free quick start tutorials for Swift developers – `https://www.hackingwithswift.com/quick-start/swiftui`

- Introduction to Swift | Swift Overview – `https://www.knowledgehut.com/tutorials/swift-tutorial/swift-overview`
- Stack Overflow - Where Developers Learn, Share, Build Careers – `https://stackoverflow.com/`
- GitHub – `https://github.com/`

Ringraziamenti

Il primo ringraziamento è per la mia famiglia, che mi ha permesso di intraprendere e concludere il percorso universitario.

Ringrazio il Prof. Ursino, che mi ha accompagnato nella stesura della tesi, con estrema disponibilità e rigore. Ringrazio anche il dott. Enrico Corradini, che mi ha aiutato durante l'attività di tirocinio.

Infine ringrazio il mio fidato compagno di studi Andrea, che ha condiviso con me tutto il percorso universitario, aiutandomi e rendendolo più piacevole.