Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Progettazione e implementazione di un'app iOS per la gestione di uno stabilimento balneare

Design and implementation of an iOS app for the management of a beach property

Relatore

Prof. Domenico Ursino

Candidato

Ettore Zamponi

Anno Accademico 2019-2020

Indice

Elenco delle figure

Elenco dei listati

Introduzione

Al giorno d'oggi gli smartphone si stanno diffondendo a macchia d'olio; basti pensare che, nel mondo, 5,9 miliardi di persone hanno un telefonino, di cui almeno l'80% è uno smartphone. Nell'ultimo decennio si è vista un'evoluzione tecnologica impressionante, e questa non sembra destinata a rallentare. Gli smartphone sono, oramai, in grado di eseguire qualsiasi operazione, dalla più semplice, come inviare un SMS, alla più complessa, come riprodurre oggetti con la realtà aumentata.

Esistono applicazioni inerenti a qualsiasi ambito, e capaci di qualunque operazione; è molto difficile trovare un settore in cui non ci sia un'applicazione di riferimento da utilizzare. Nella sfera degli stabilimenti balneari non è ancora presente un'app universalmente utilizzata per la gestione di un lido. Inoltre, la situazione pandemica attuale richiede massima attenzione a tutte quelle situazioni di contatto che, prima, sarebbero state la normalità, ma che, purtroppo, al momento possono rappresentare fonte di contagio.

In questa cornice si impegna Umbrella, un'applicazione per smartphone che si occupa della gestione di uno stabilimento balneare in un periodo storico in cui ogni esercizio commerciale, o attività, ha dovuto modernizzarsi e reinventarsi per poter far fronte al COVID-19. Umbrella vuole aiutare a rendere l'esperienza estiva di un lido più facile dal punto di vista gestionale, in termini, soprattutto, di rispetto delle normative vigenti.

L'applicazione è divisa in quattro macro-aree, ciascuna delle quali offre informazioni o servizi differenti. Essa fornisce, nella home page, una panoramica generale sulle informazioni inerenti allo stabilimento balneare, tra cui il meteo, la posizione GPS o il menù giornaliero. Tuttavia, il cuore dell'applicazione sarà rappresentato dalla sezione che permette la prenotazione giornaliera di un ombrellone da parte di un utente registrato. Questo passaggio della registrazione, e gestione dell'account, sarà possibile in un'altra sezione dedicata. Al fine di condividere l'esperienza avuta, l'applicazione mette a disposizione la possibilità di lasciare una recensione agli utenti che hanno sfruttato tali servizi.

Il punto di forza che caratterizza Umbrella è la semplicità d'uso; essa risulta, infatti, anche ad un occhio meno esperto, davvero intuitiva e pronta all'uso a chiunque ne abbia bisogno.

In questo caso, però, l'aspetto inganna, perchè una grafica semplice e pulita non rispecchia a dovere la complessità delle operazioni che è in grado di gestire, e la

conseguente difficoltà durante le fasi di progettazione ed implementazione.

La presente tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 viene data un'infarinatura storica del sistema operativo Apple, se ne analizzano le capacità per poi passare ad una descrizione dell'ambiente di sviluppo che ha permesso la realizzazione di tale applicazione.
- Nel Capitolo 2 vengono analizzati i requisiti, sia funzionali che qualitativi che vincolanti, i quali saranno rispettati all'interno dell'applicazione. Successivamente vengono descritti i principali software che sono stati utilizzati durante la progettazione e implementazione.
- Nel Capitolo 3 viene trattata l'intera fase di progettazione. Viene presentata una mappa che descrive visivamente l'intera applicazione, per poi addentrarsi nella progettazione vera e propria; sia del database che delle funzionalità da implementare. Infine, sono raffigurati i mockup delle interfacce offerte dall'applicazione.
- Nel Capitolo 4 viene esposta tutta la fase di sviluppo delle funzioni offerte dall'app. Per mezzo di listati che ripropongono il codice esecutivo, vengono descritti i processi che regolano il funzionamento dell'app. Infine si passa al manuale utente, dove viene proposta una guida al suo utilizzo.
- Nel Capitolo 5 viene analizzato a fondo il progetto finale, evidenziando le conoscenze acquisite durante il lavoro svolto. Così facendo, vengono proposti nuovi aspetti dell'applicazione che potrebbero esser migliorati in futuro.
- Nel Capitolo 6 vengono tratte le conclusioni riguardo l'applicazione finale, sottolineandone le potenzialità, specialmente in un periodo di pandemia mondiale.

1

L'ambiente iOS

In questo capitolo si introducono l'ambiente iOS e gli strumenti di sviluppo relativi al sistema operativo mobile sviluppato da Apple per iPhone ed iPad. iOS rappresenta il secondo sistema mobile più diffuso al mondo dopo Android, e di gran lunga il più affidabile e sicuro.

1.1 Storia

La prima versione del sistema operativo viene presentata da Steve Jobs nel Gennaio del 2007 a San Francisco e avviene con l'uscita dell'iPhone di prima generazione, ovvero l' iPhone 2 G, il 29 Giugno dello stesso anno.

Il sistema iOS, durante la sua evoluzione, non ha semplicemente introdotto nuove tecnologie, ma bensì un modo completamente nuovo di pensare ed approcciarsi ad uno smartphone. iOS è stato il capostipite della nuova generazione di sistemi operativi *Touch-based* e *App-based*; infatti ha portato con sé il concetto di Market (App Store) dal quale scaricare e installare miliardi di differenti applicazioni aggiuntive che potenzialmente possono rendere lo smartphone in grado di fare qualsiasi cosa. Proprio questa peculiarità apre le porte a un nuovo modo di concepire i dispositivi mobili.

Inizialmente il nome del sistema operativo era un generico *iPhone OS* per poi divenire *iOS* nel 2010, insieme all'uscita dell'iPhone 3GS. La prima versione era denominata OS 1 e prevedeva soltanto applicazioni native, con la possibilità di eseguire web app sviluppate da terze parti. Già da OS 2 inizia a farsi strada lo store con vere e proprie applicazioni di terze parti; questo contava già 500 app disponibili appena rilasciato nel 2008. Inoltre, si introducevano nuove funzionalità, come la compatibilità con i server Microsoft Exchange o aggiornamenti in Mail.

Nel 2009 viene presentato OS 3 con la celeberrima funzione *copia e incolla*, le notifiche push, il supporto agli MMS e ai messaggi vocali. L'anno dopo, con l'avvento del primo tablet di casa Apple, il sistema operativo raggiunge la Versione *3.2* e cambia nome da *OS* ad *iOS*.

Successivamente viene rilasciato *iOS 4*, che rappresenta un enorme passo avanti rispetto i suoi predecessori. Questo porta con sé importantissime novità che utilizziamo quotidianamente tutti noi ancora oggi, come il multitasking, la possibilità di

cambiare sfondo, di creare sottocartelle di applicazioni nella home o la funzionalità di videochat FaceTime.

Il 6 Giugno 2011, durante la WWDC (Apple Worldwide Developers Conference) venne presentato *iOS 5* che incorporava oltre 200 novità tra cui *iCloud*, per lo storage online dei dati, *Siri* come uno dei più avanzati assistenti vocali, il centro notifiche, o, ancora, gli aggiornamenti OTA (Over-The-Air).

L'anno successivo vengono introdotte applicazioni proprietarie per sostituire servizi di terze parti preinstallati, come accade con l'app Google Maps, che lascia spazio all'app Mappe di Apple (la quale, inizialmente, lasciava parecchio a desiderare). Inoltre *iOS 6* portò l'integrazione con Facebook, miglioramenti a Siri e la possibilità di scattare foto panoramiche.

Il sistema operativo successivo sconvolge totalmente l'aspetto grafico apportando cambiamenti significativi in ogni piccolo angolo di iOS. L'intero restyling grafico viene presentato dal *Chief Design Officer* di Apple in persona, Jony Ive, che introduce *iOS 7* e tutte le nuove grafiche incentrate sull'ideale di semplicità e minimalismo. Inoltre, finalmente, si aveva il tanto richiesto centro di controllo.

Nel 2014 *iOS 8* viene presentato più come un importante aggiornamento per gli sviluppatori piuttosto che come una novità per gli utenti finali, introducendo novità funzionali come *Metal*, *HomeKit* o le funzionalità di *Hand Off*. Nonostante questo, vennero aggiunte le caratteristiche di Apple Pay, il supporto allo smartwatch di casa Apple, o anche l'editor fotografico integrato nell'applicazione Foto.

La versione successiva portò piccole implementazioni, come la modalità di risparmio energetico, suggerimenti proattivi su Siri o la funzione *Night Shift* (modalità che rende i toni dello schermo più caldi durante le ore notturne). Più importanti, invece, furono le migliorie introdotte su iPad, tra le quali spiccano nuove e interessanti funzioni di multitasking, come *Slide Over* o *Split View*.

iOS 10 portò con sé un'app Mappe ridisegnata, l'apertura a Siri per l'utilizzo dell'assistente vocale anche con applicazioni di terze parti, e non solo con app native, e l'app di messaggistica ridisegnata e potenziata notevolmente.

L'importanza di *iOS 11* sta nel fatto che fu il primo sistema operativo Apple interamente a 64 bit, escludendo, di fatto, tutti i precedenti dispositivi a 32 bit. Inoltre, si rinnova leggermente l'aspetto grafico; in primis rinnovando la grafica del centro di controllo e, poi, favorendo un font più scuro per ottenere un aspetto complessivamente più pulito.

Nel recente 2018 viene rilasciato *iOS 12*, che non rivoluziona, ma apporta comunque importanti novità, oltre che una migliore fluidità e stabilità generale. Vengono introdotte, ad esempio, la funzionalità *Tempo di utilizzo* per tener sotto traccia il tempo passato davanti allo smartphone ed, eventualmente, cercare di limitarlo, o le *Memoji*, ovvero delle emoji personalizzate in grado di ricreare un'immagine 3D del proprio volto. La Figura 1.1 fornisce un overview della timeline di iOS dalla prima versione fino alla dodicesima.

Nell'autunno successivo viene proposto *iOS 13* che introduce novità non percepibili a primo impatto, ma di gran peso specifico. Le applicazioni vengono ridotte di dimensioni fino al 101% ma vantano una velocità di apertura quasi raddoppiata, viene introdotta la *modalità scura* che trasforma i colori chiari in scuri per una maggiore visibilità in contesti di scarsa luminosità. Da citare l'app File che, finalmente, si apre ad accogliere file e dati da dispositivi di archiviazione esterni, come schede

Figura 1.1. iOS Timeline

SD o unità flash USB. Questo dato è molto importante perchè segnala il fatto che il sistema operativo *iOS*, che è sempre stato molto chiuso sin dagli inizi nei confronti di aziende che non siano Apple stessa, inizia ad aprire i propri orizzonti sia in termini software che hardware .

L'ultimo sistema operativo è *iOS 14* rilasciato al pubblico a Settembre 2020. Questo ha introdotto i *widget* nella schermata home, una nuova libreria app che organizza tutte le applicazioni presenti nello smartphone in un'unica vista, la riproduzione di video o videochiamate FaceTime nella modalità *picture-in-picture*, un'applicazione Messaggi piena di nuove funzioni, un browser molto più stabile e sicuro, la modalità *App Clip*, per poter eseguire vere e proprie app ma in versione semplificata, per svolgere rapide azioni solamente nel momento in cui servono, o, ancora, ulteriori migliorie per quanto riguarda la sicurezza e la correzione di bug (Figura 1.2).

Figura 1.2. Nuove funzioni introdotte con iOS 14

1.2 Funzionalità e punti forti

La caratteristica che di sicuro colpisce a prima vista è la semplicità e il minimalismo che *iOS* incorpora in ogni suo piccolo dettaglio.

Tralasciando gli aspetti grafici e come essi si manifestano, ci chiediamo quali siano i punti cardine che differenziano il sistema operativo mobile di Apple rispetto al suo maggior competitor, nonché sistema operativo mobile più diffuso al mondo, ovvero Android (Figura 1.3).

A differenza di Android, *iOS* non è un sistema *Open Source*; questo significa che non ha il concetto di apertura e personalizzazione alla base. Anzi, si è sempre distinto per essere un sistema fortemente chiuso e limitato, non aperto a estensioni o app di terze parti. Idealmente agli antipodi, Android distribuisce il proprio codice sorgente rendendosi modificabile a una miriade di programmatori e sviluppatori. Attualmente l'open source tende ad assumere rilievo filosofico, consistendo in una diversa concezione di quello che sta alla base di ogni oggetto tech, ed è proprio questo il punto cardine che divide gli appassionati Apple da quelli Android. Continuando su questo filone, Android permette l'accesso diretto al file system e al cuore del sistema operativo, cosa che Apple non permette in nessun modo.

D'altro canto, però, uno degli aspetti di maggiore rilevanza è la sicurezza e la capacità di resistere a virus o malware che in qualunque momento possono prendere di mira lo smartphone. Negli anni sono stati davvero pochi i casi in cui alcuni bug del sistema Apple hanno messo in pericolo la sicurezza degli utenti; a motivare questa affermazione è proprio la disponibilità di un unico canale ufficiale per ottenere le applicazioni che vengono rese disponibili solo dopo un lungo processo di attivazione e controllo della qualità. L'uso di store alternativi o l'installazione di pacchetti non autorizzati, grazie a un file system accessibile e a un sistema open source, aumenta drasticamente il rischio di subire un attacco. Questo non vuole assolutamente dire che *iOS* sia un sistema impenetrabile; anzi, ci sono stati casi in cui anche gli smartphone della mela morsicata hanno dovuto arrendersi e subire assalti.

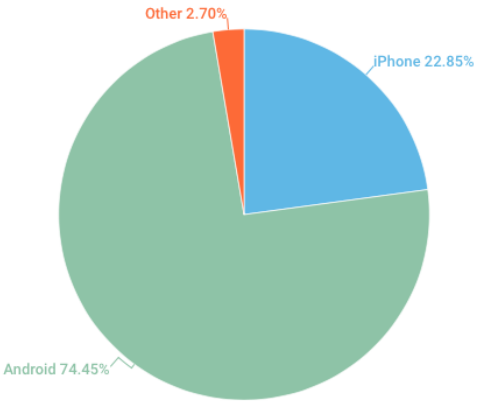

Figura 1.3. La diffusione dei sistemi operativi mobile

Un gran punto a favore dei sistemi operativi Apple è quello di continui e costanti aggiornamenti anche a distanza di anni. Infatti Apple fornisce aggiornamenti software ai propri iPad e iPhone per 5 o 6 anni dopo la data del loro rilascio. Così facendo si ottengono e diffondono nuove funzionalità e app, supporti per nuovi formati immagini e video, protocolli di sicurezza e nuove patch per proteggere il proprio smartphone. É pur vero che i dispositivi più datati potrebbero non supportare ogni singola nuova funzionalità, ma continuano, comunque, a poter utilizzare la maggior parte delle nuove funzioni e tutti gli update di sicurezza, tanto a lungo quanto l'hardware dura nel tempo. Tra i cellulari Android, solo ai telefoni Google Pixel e i dispositivi dotati del programma Android One sono concessi aggiornamenti garantiti, e anche in questo caso, solo per due o tre anni dalla data di rilascio del dispositivo. Proprio per questo la maggior parte dei dispositivi Apple in circolazione esegue l'ultimo sistema rilasciato, o al massimo il precedente; proprio per la filosofia di aggiornare tutta la linea di prodotti menzionata in precedenza. Al contrario, gli smartphone Android in circolazione hanno installate le più disparate versioni di Android lasciando in circolazione poca uniformità, telefoni magari potenti ma non più supportati e sicuri al giorno d'oggi, creando, tra l'altro, confusione all'utente finale ma, soprattutto, ai programmatori che devono far fronte a una grandissima differenziazione quando vanno a sviluppare codice per le più disparate versioni del sistema operativo.

Ad esempio, appena dopo un mese dal rilascio di *iOS 13*, già il 50% dei dispositivi Apple in circolazione lo aveva installato, il 40% aveva la precedente versione e soltanto una minima parte di questi non era ancora stata aggiornata. D'altro canto, vediamo come i dispositivi Android in circolazione nello stesso anno eseguivano molteplici versioni, tra le quali le più diffuse erano Android Lollipop e Marshmallow, rispettivamente del 2014 e del 2015 (Figura 1.4).

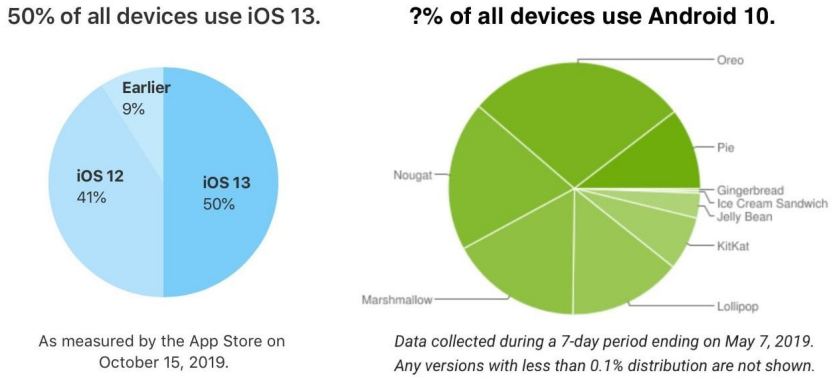

Figura 1.4. Versioni di Android ed iOS in circolazione

Come già accennato, il design e la semplicità d'uso non possono non esser menzionati se si parla di Apple. La cura impiegata in ogni dettaglio è sbalorditiva: a partire dal sistema operativo per arrivare al design fisico di ogni prodotto, passando, persino, per i manuali utente. Questo non si nota soltanto nel prodotto, una volta

arrivato tra le mani del consumatore, ma, soprattutto, nella semplicità d'utilizzo che ogni utente sperimenta sin dal primo avvio.

Il sistema operativo è davvero completo in ogni suo piccolo particolare; è una macchina potentissima pronta all'uso sin dalla prima accensione senza bisogno di particolari aggiunte di app, o configurazioni, o, ancora, di download aggiuntivi. Appena avviato si hanno migliaia di strumenti a disposizione per effettuare le più disparate operazioni, il tutto in maniera così semplice che sembra magico. Dalle operazioni più semplici, come effettuare chiamate o inviare messaggi, fino a poter scansionare documenti, trasformarli in PDF e firmarli per inviarli via mail al proprio datore di lavoro, oppure registrare un video in altissima risoluzione, modificarlo e ritagliarlo, aggiungere testi o emoticon per poi condividerlo su un qualsivoglia social network (Figura 1.5).

Figura 1.5. Applicazioni disponibili di default su iOS

La facilità con cui tutto è a portata d'uso in iOS è davvero difficile da ritrovare in un diverso sistema operativo, che sia esso Android o altro. Ogni piccolo aspetto che fa parte del dispositivo, che sia esso hardware o software, è UCD (User-Centered Design), ovvero basato sulle esigenze dell'utente finale. Lo UCD è più della classica usabilità: è l'applicazione di una filosofia tutta incentrata e rivolta a identificare i bisogni dell'utente finale (Figura 1.6), e questa filosofia viene applicata dalla Apple in qualsiasi cosa l'azienda realizza.

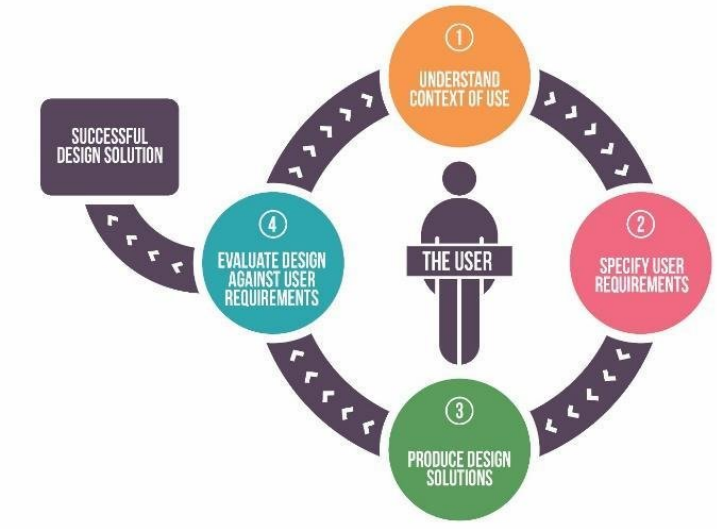

Figura 1.6. Ciclo User-Centered Design (UCD)

1.3 Ambiente di sviluppo

L'ambiente di sviluppo proprietario, noto come IDE (Integrated Development Environment), totalmente sviluppato e mantenuto da Apple, si chiama *Xcode* e permette di scrivere e sviluppare software. Inizialmente Xcode era fornito in bundle con il sistema operativo presente nei Mac; più recentemente, esso è stato eliminato dal sistema operativo ma rimane pur sempre reperibile a tutti gratuitamente dallo store.

Tra le funzionalità più avanzate e potenti troviamo la distribuzione in rete del lavoro di compilazione di un progetto. Sfruttando due tecnologie proprietarie, ossia *Bonjour* e *Xgrid*, Xcode è in grado di suddividere il lavoro da compiere e distribuirlo su più computer contemporaneamente riducendo notevolmente il tempo di compilazione, specialmente se si tratta di progetti molto pesanti e complessi.

Altra caratteristica notevole è la compilazione incrementale; in altre parole, Xcode è in grado di compilare il codice man mano che viene scritto, così da ridurre notevolmente i tempi di compilazione e, soprattutto, permettere di trovare un errore poco dopo averlo scritto, piuttosto che aspettare di finire il lavoro e doverlo andare a cercare tra migliaia di righe di codice.

Con il nome Xcode spesso si fa riferimento all'intero set di strumenti di sviluppo per *iOS*, *OS X*, *watchOS* e *tvOS*, l'IDE stesso, il simulatore per far girare tutte le applicazioni e gli *SDK* (Software Development Kit) aggiornati. Tuttavia, è da specificare che, in linea di principio, Xcode comprenderebbe soltanto l'IDE, ovvero l'ambiente di sviluppo vero e proprio (Figura 1.7).

Il compilatore presente nella suite è *LLVM* (in precedenza acronimo di Low Level Virtual Machine, macchina virtuale di basso livello), introdotto da Xcode 4.2 nel 2010 e reso l'unico compilatore dell'intera suite tre anni più tardi. Questa è un'infrastruttura di compilazione scritta in C++ che ottimizza la fase di compila-

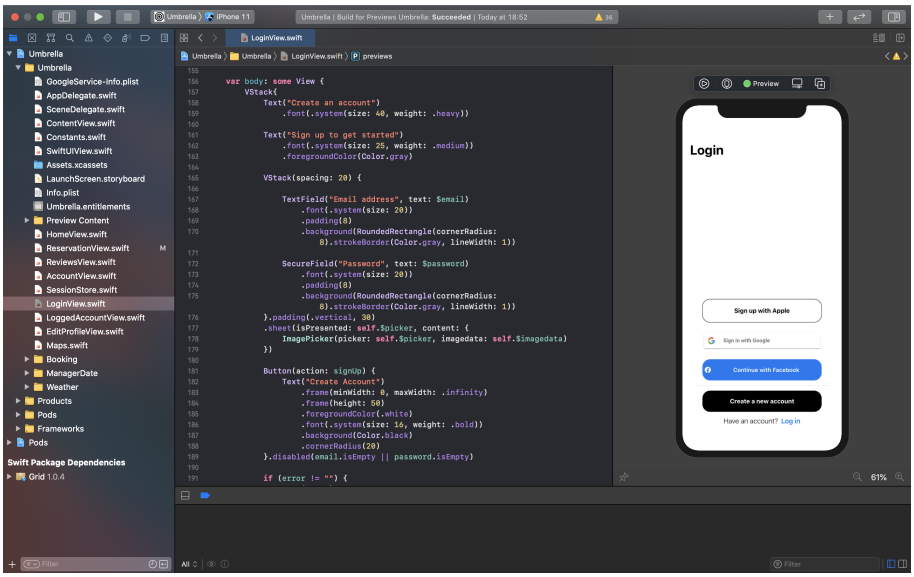

Figura 1.7. Layout Xcode e linguaggio dichiarativo SwiftUI

zione, di linking, di esecuzione e di non utilizzo dei vari programmi che gestisce. I linguaggi attualmente supportati da Xcode sono *C*, *C++*, *Objective C*, *Java*, *Python* e, soprattutto, *Swift*.

Dal 2010 Apple, insieme all'autore originale del progetto LLVM, ha sviluppato il proprio linguaggio di programmazione denominato, appunto, *Swift*. Questo è un linguaggio *object-oriented* per sistemi operativi Apple (anche se, recentemente, è stata rilasciata una versione anche per Windows) ideato per coesistere al fianco del più diffuso *Object-C*, tipico linguaggio di programmazione per sistemi operativi della mela morsicata. *Swift* sfrutta il già citato LLVM e il run time di *Objective-C* così da permettere la presenza nello stesso programma di linguaggi differenti, come *Objective C*, *C++* e *Swift*. Nel 2015 la stessa Apple rilascia una nuova versione del suo linguaggio di programmazione rendendolo open source su licenza Apache 2.0 e su un repository GitHub; inoltre rilascia anche una versione del compilatore per sistemi operativi Linux (creato su misura per Ubuntu).

Va fatta una menzione speciale per un potentissimo e rivoluzionario framework, nonché il principale linguaggio su cui si basa l'applicazione oggetto della presente tesi, sviluppato da Apple e interamente scritto in *Swift*, ossia *SwiftUI*. Questo framework permette di costruire complesse interfacce utente tramite una sintassi dichiarativa, e un unico set di tool e API, coinvolgendo tutto l'ecosistema Apple e senza usare distintamente, come avveniva in precedenza, *AppKit* per macOS, *UIKit* per iOS oppure *WatchKit* per watchOS. Tra tutte queste novità, quella fondamentale è il Live Rendering (come accade già con React Native), cioè quel processo che permette di vedere istantaneamente il risultato di ciò che si sta scrivendo iniettandolo nell'app in esecuzione (Figura 1.7).

A partire dalla prima versione rilasciata, *Swift* viene dichiarato come 8,4 volte più veloce di *Python* e 2,6 volte più veloce di *Objective-C* in alcuni tipi di algoritmi.

1.4 Perchè iOS

L'ecosistema Apple permette un'esperienza d'uso davvero appagante, e questo è innegabile, con una cura al dettaglio maniacale che difficilmente si può ritrovare altrove. La facilità d'uso lo rende molto intuitivo e semplice nell'utilizzo quotidiano. Il sistema operativo garantisce un'affidabilità senza pari. Tutto questo viene supportato da un hardware davvero performante. Tali caratteristiche lo rendono uno strumento dalle potenzialità infinite da scoprire, sfruttare e completare, sviluppando applicativi sempre più utili.

2

Analisi dei requisiti

Questo capitolo si propone di trattare e descrivere i requisiti definiti con la collaborazione dallo studente e degli altri attori del contesto di riferimento per l'applicazione.

2.1 Requisiti funzionali

I requisiti funzionali sono elenchi di funzionalità, o servizi, che l'applicazione si propone di fornire. Essi, inoltre, definiscono il comportamento del sistema a fronte di particolari input, o come il sistema debba reagire in determinate situazioni (Figura 2.1). Due importanti caratteristiche da rispettare sono la completezza e la coerenza. Questo vuol dire che tutti i requisiti devono essere ben definiti e non ci devono essere, invece, requisiti in conflitto tra loro.

I requisiti funzionali che il progetto di riferimento per la presente tesi deve implementare sono i seguenti:

- effettuare il login;
- registrare un nuovo utente;
- visualizzare i contenuti informativi;
- prenotare un ombrellone;
- recensire lo stabilimento balneare;
- modificare il profilo utente.

2.2 Requisiti di qualità

Il progetto si propone di rispettare, inoltre, i seguenti requisiti di qualità al fine di ottenere un applicazione qualitativamente eccellente:

- la home page deve essere dinamica;
- deve essere presente una Bottom Navigation Bar per la navigazione tra le sezioni;
- la disposizione degli ombrelloni deve rispecchiare quella reale della spiaggia;
- i pulsanti e la grafica devono essere coerenti con le linee guida per gli sviluppatori distribuite da Apple.

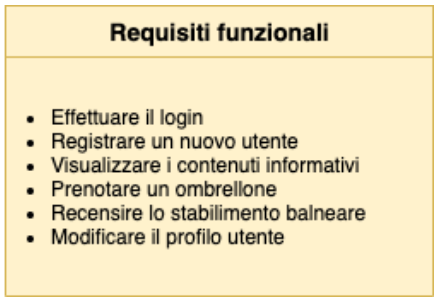

Figura 2.1. Elenco dei requisiti funzionali

2.3 Requisiti di vincolo

L'applicazione finale deve rispettare i seguenti vincoli:

- le informazioni di ogni sezione devono essere aggiornate automaticamente all'apertura della stessa recuperando le informazioni dal database;
- la registrazione dell'utente avviene tramite la propria mail e la propria password;
- un utente che non ha effettuato l'accesso non può prenotare ombrelloni;
- un utente può prenotare solo ombrelloni per il giorno stesso;
- un utente che non ha effettuato l'accesso non può scrivere recensioni.

2.4 Requisiti software

I requisiti software specificano i principali software necessari allo studio e all'analisi dell'applicazione, nonché alla progettazione e realizzazione della stessa.

Nel nostro caso, i requisiti software prevedono l'utilizzo di tre tool, ovvero Balsamiq Mockup 3, Firebase e CocoaPods. Essi saranno illustrati in dettaglio nelle prossime sottosezioni.

2.4.1 Balsamiq Mockups 3

Balsamiq Mockups 3 è un semplice, ma valido, software che permette di creare mockup (ovvero, rappresentazioni grafiche delle interfacce) di applicazioni o siti web in fase di progettazione.

Il software, tramite un efficace sistema di drag&drop, permette di realizzare wireframe (schermate), o interfacce utente, per sviluppatori, designer o progettisti.

2.4.2 Firebase

Firebase è un potentissimo servizio, sviluppato da Google, per la creazione e gestione di dati elaborati da applicazioni web e mobile. Tali funzionalità rientrano nella categoria dei servizi online, ormai sempre più diffusi, noti come *backend*. Dispositivi molto potenti, insieme a connessioni ormai velocissime, tendono a rendere

le applicazioni odierne sempre meno chiuse e sempre più capaci di interagire con servizi remoti. Per il funzionamento di tale meccanismo, è necessaria una controparte server in grado di gestire, tramite API (Application Programming Interface), servizi quali autenticazione, storage dei dati o notifiche push. Ecco che Firebase ci fornisce tutti i tool necessari per gestire le esigenze di un'applicazione mobile di questo genere (Figura 2.2).

Tra i servizi offerti troviamo *Cloud Firestore*, il quale permette di archiviare e sincronizzare dati tra utenti e dispositivi sfruttando un database NoSQL; esso inoltre, offre una sincronizzazione offline e consente la gestione di query molto efficienti, garantendo, così, un servizio davvero efficace per ogni tipo di applicazione. Il tool di *Autenticazione*, offerto da Firebase per gestire il login degli utenti, assicura la creazione di interfacce utenti completamente personalizzabili effettuando l'accesso non solo con account Google, ma anche di terze parti, come Facebook, Apple o, semplicemente, tramite mail e password. Altro pilastro fondamentale è il servizio di *Cloud Storage* per archiviare e gestire contenuti generati dagli utenti, come immagini, audio o video. Indipendentemente dalla qualità della rete, gli SDK Firebase garantiscono sicurezza e affidabilità a tutti gli upload e download.

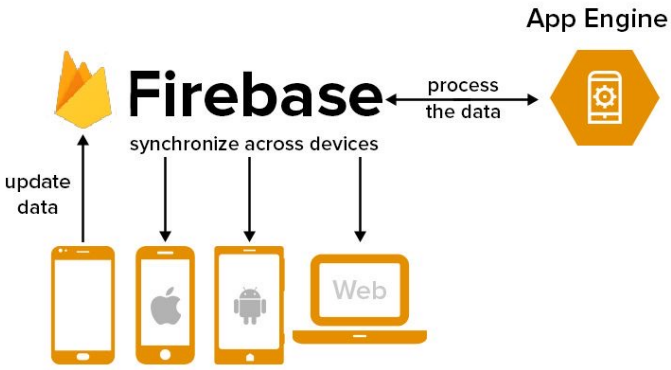

Figura 2.2. Funzionamento Firebase

2.4.3 CocoaPods

CocoaPods è un applicazione per la gestione delle dipendenze di un progetto per dispositivi Apple. In particolare, permette di installare ed aggiornare automaticamente i framework di altri sviluppatori. Esso può essere visto come un insieme di risorse e funzionalità, già sviluppate da altri programmatori, messe a disposizione per essere implementate direttamente in nuove app (Figura 2.3).

I *Pods* sono i singoli framework che, a loro volta, rappresentano un pacchetto di classi, strutture e metodi per la risoluzione di un problema o per la realizzazione di una determinata funzione. Essi possono spaziare dal campo grafico alle funzionalità più stravaganti ed utili. In definitiva, CocoaPods permette di aggiungere funzioni automaticamente ad un progetto senza doverle programmare da zero.

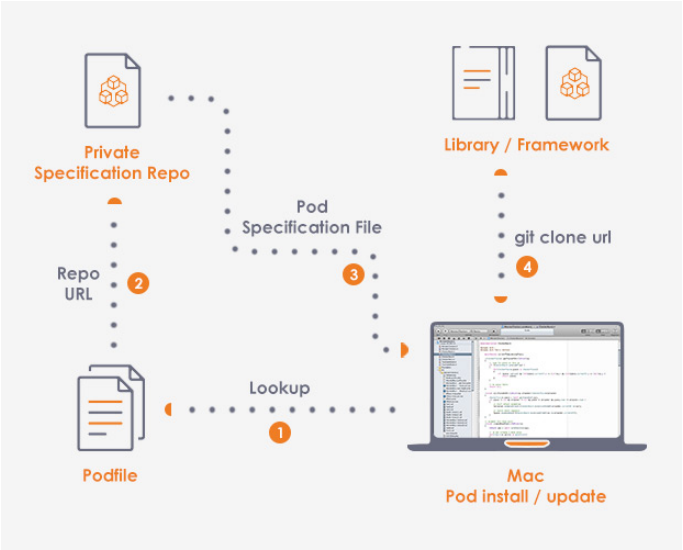

Figura 2.3. Installazione ed uso di CocoaPods

2.5 Diagramma dei casi d'uso

Una volta chiariti i requisiti, attraverso l'uso di diagrammi UML, si analizza il sistema in esame astraendone le funzionalità e creando un template (uno schema) che specifichi tanto la struttura quanto la dinamica del sistema da realizzare.

Si sfrutta, allora, il diagramma dei casi d'uso (Use Case Diagram), il quale visualizza, mediante uno schema, come gli utenti finali (attori) interagiranno con le diverse funzioni (casi d'uso) che l'applicazione finale propone.

Nel caso del progetto sono presenti due tipologie di attori: l'utente che utilizza l'applicazione senza essersi registrato (utente semplice) e l'utente registrato. I casi d'uso, d'altra parte, sono differenti in base a quale utente si sta interfacciando all'app.

Nella Figura 2.4 si mostrano i casi d'uso dell'applicazione che si intende realizzare.

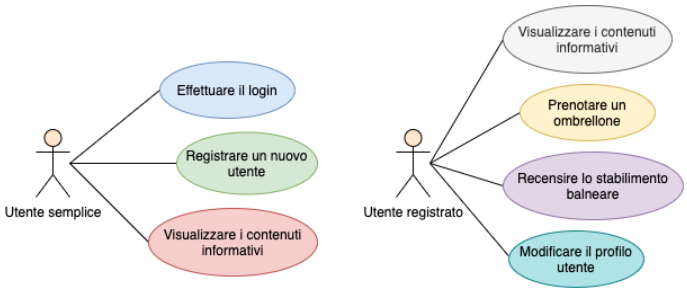

Figura 2.4. Diagramma dei casi d'uso

3

Progettazione

In questa sezione vengono definite le funzionalità che l'applicazione dovrà essere in grado di eseguire e l'aspetto che essa assumerà. In questo modo si può già avere un'idea embrionale di quello che sarà il risultato del progetto.

3.1 Mappa dell'applicazione

La mappa dell'applicazione è uno schema, intuitivo ed efficace, che raffigura il percorso, e, quindi, i passaggi da compiere per eseguire una determinata funzione.

Nella Figura 3.1 si rappresenta la mappa dell'applicazione, la quale illustra le varie sezioni dell'app e le relative azioni disponibili; in base alla tipologia di utente che si interfaccia, si avranno alcune funzionalità disponibili, piuttosto che altre.

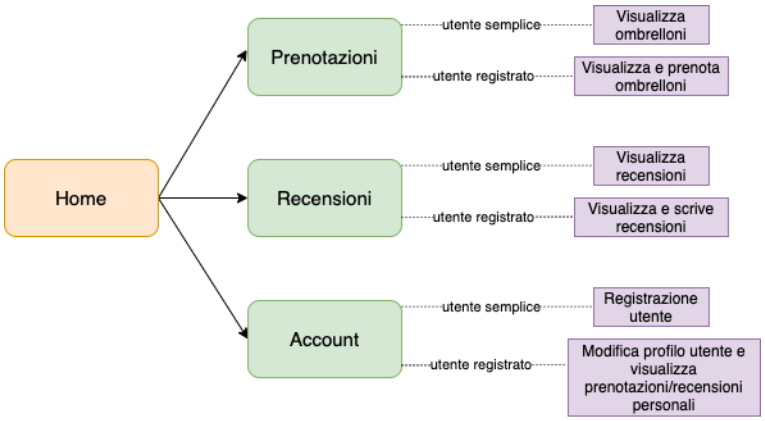

Figura 3.1. Mappa dell'applicazione

3.2 Progettazione della componente dati

I dati sono il cuore di ogni applicazione, sia essa una pagina web, un programma per personal computer o un'applicazione per smartphone. Le informazioni hanno un valore sempre più grande e, di conseguenza, vanno gestite nel migliore dei modi.

La progettazione della componente dati è il processo che porta alla definizione di un modello logico dettagliato del database. Quest'ultimo modello descriverà le scelte progettuali e l'effettiva strutturazione del database da utilizzare.

Prima di tutto, è fondamentale definire i dati da memorizzare rispettando i requisiti dell'applicazione. Nel caso in esame le informazioni più importanti da gestire sono quelle che rappresentano gli utenti registrati e gli ombrelloni previsti dallo stabilimento balneare. In secondo piano, invece, troviamo altre informazioni, come quelle riguardanti i servizi offerti dallo stabilimento o, ancora, le recensioni rilasciate dagli utenti. Nella Figura 3.2 è facile capire come sia stato organizzato il database e, soprattutto, come sono collegate le varie informazioni tra loro.

I titoli delle colonne vengono chiamati *attributi* e sono i campi che si trovano nel database, mentre ogni riga che compone la tabella è una cosiddetta *tupla*, cioè un insieme di dati. I metadatati sottolineati nelle tabelle rappresentano le *chiavi primarie*; queste sono informazioni univoche che non si ripetono mai all'interno dell'intero database e che ci permettono di identificare una tupla precisa. Ogni tabella, in questo caso, raffigura un'entità, cioè una classe di oggetti. Inoltre, le varie tabelle sono state associate (tramite chiavi primarie) per raffigurare le dinamiche che portano al collegamento tra i dati.

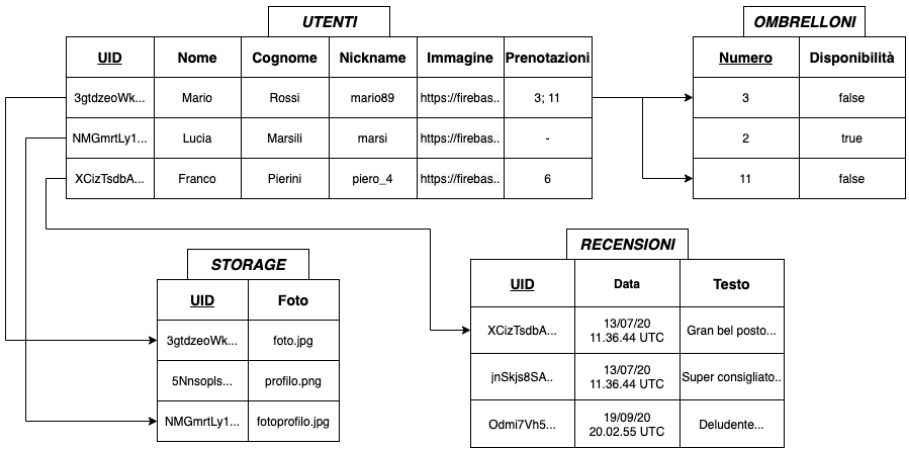

Figura 3.2. Progettazione del database alla base dell'applicazione

3.3 Progettazione della componente applicativa

La progettazione della componente applicativa è la fase in cui vengono definite le dinamiche che regoleranno le funzioni dell'applicazione.

Nel caso in esame, avendo analizzato precedentemente i casi d'uso (Capitolo 2.5), si procederà alla progettazione dei flussi di ciascuno di essi per capirne il funzionamento e, di conseguenza, poterli implementare al meglio.

3.3.1 Registrazione utente

La creazione di un profilo, da parte di un utente non registrato, rappresenta una delle implementazioni primarie perchè ciò permetterà la registrazione dell'utente e la successiva possibilità a quest'ultimo di prenotare un ombrellone oppure di rilasciare una recensione, sbloccando, di fatto, tutte quelle operazioni che altrimenti non potrebbe svolgere.

Un *Activity Diagram* (o diagramma di attività) è una rappresentazione grafica di un preciso flusso di operazioni; nel caso in esame, verrà utilizzato per modellare l'insieme di azioni che compongono i casi d'uso dell'applicazione. Tramite la Figura 3.3 è possibile capire immediatamente quali saranno gli step da seguire e rispettare per concludere l'operazione di registrazione di un utente.

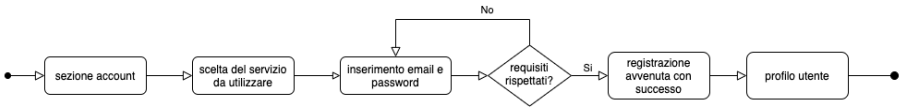

Figura 3.3. Activity Diagram: Registrazione utente

3.3.2 Login

Il caso di login è molto simile alla registrazione di un nuovo utente; esso è mostrato nella Figura 3.4. L'unica differenza sta nel fatto che, una volta inseriti email e password, il sistema verifica la presenza di un profilo con tali informazioni, piuttosto che crearne uno nuovo con le informazioni inserite, come avviene durante la registrazione.

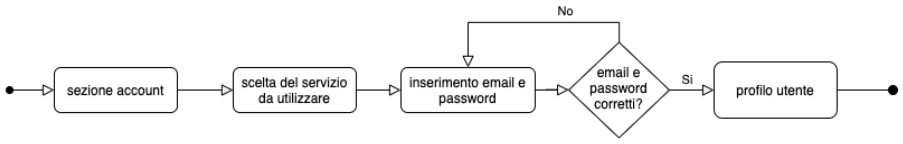

Figura 3.4. Activity Diagram: Login

3.3.3 Visualizzare i contenuti informativi

All'apertura dell'applicazione avremo il caso della visualizzazione dei contenuti informativi che la home page ci offre. La maggior parte degli elementi di questa pagina

si popolano dinamicamente, recuperando le varie informazioni necessarie per tutti i tipi di utenti. Esempi di informazioni visualizzate sono quelle relative al meteo o al menù giornaliero. La particolarità, che si nota chiaramente nella Figura 3.5, è nel comportamento della zona più alta della home page che, in base a quale tipologia di utente si sta approcciando, mostra l'area personale, nel caso di un utente registrato, oppure, l'invito a registrarsi per poter usufruire di tutte le funzionalità dell'applicazione, in caso contrario.

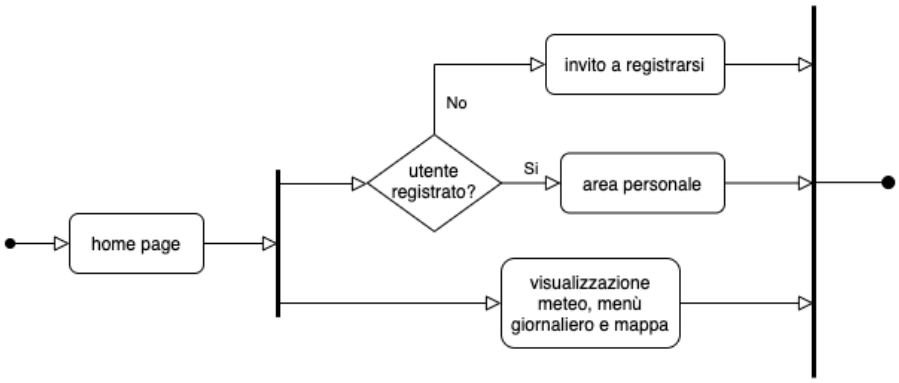

Figura 3.5. Activity Diagram: Visualizzazione contenuti informativi

3.3.4 Prenotare un ombrellone

La prenotazione di un ombrellone è un altro caso da analizzare e rappresenta una di quelle funzionalità che l'utente non può completare se non ha precedentemente effettuato il login. Nella Figura 3.6 viene raffigurato il diagramma di attività del caso d'uso appena citato.

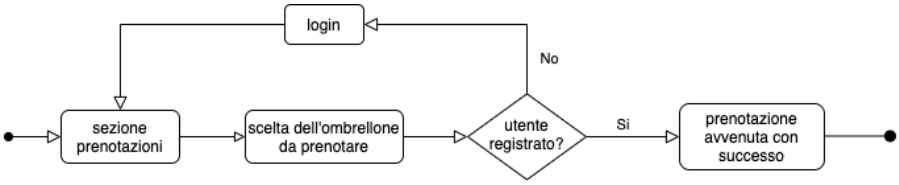

Figura 3.6. Activity Diagram: Prenotazione di un ombrellone

3.3.5 Recensire lo stabilimento balneare

Come accade nel caso precedente, anche in questo, l'operazione non può essere portata a termine se l'utente che sta tentando di recensire lo stabilimento balneare non ha registrato il proprio profilo utente (Figura 3.7).

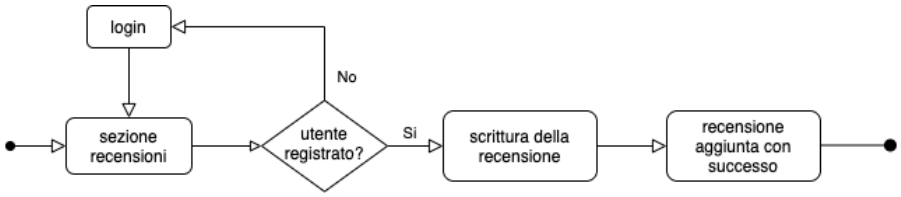

Figura 3.7. Activity Diagram: Recensione dello stabilimento balneare

3.3.6 Modificare il proprio profilo utente

Anche questo caso d'uso è riservato a chi possiede un account registrato. Infatti, in questo caso d'uso si va proprio a modificare il profilo utente, aggiungendo ulteriori informazioni a quelle già salvate durante la fase di registrazione (Figura 3.8).

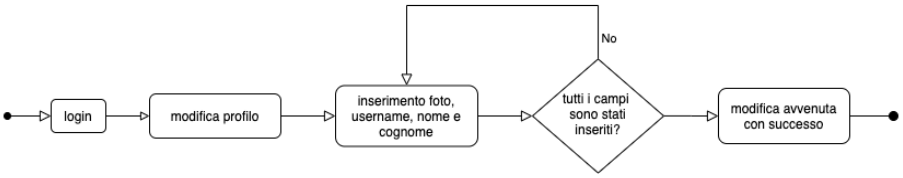

Figura 3.8. Activity Diagram: Modifica del proprio profilo utente

3.4 Progettazione dei mockup

I mockup sono rappresentazioni grafiche che ci permettono di visualizzare le interfacce di siti web o applicazioni ancor prima del loro stesso effettivo sviluppo. Essi possono essere di Livello 0, se poveri di dettagli, di Livello 1 se, invece, sono più curati, ed, infine, di Livello 2 se riproducono fedelmente il progetto finale.

Di seguito sono presentati i mockup dell'applicazione in esame.

3.4.1 Home Page

Nella Figura 3.9 viene raffigurata la prima pagina visualizzata ogni volta che si apre l'applicazione. Tale schermata ha il compito di proporre una panoramica dei servizi offerti dallo stabilimento balneare, insieme ad una piccola sezione che riguarda le informazioni personali dell'utente registrato, se esso ha effettuato l'accesso.

3.4.2 Prenotazioni

Nella seconda pagina dell'applicazione si ha una disposizione degli ombrelloni intuitiva e stilizzata, essa rispecchia la disposizione effettiva degli ombrelloni sulla spiaggia in modo tale che l'utente, in procinto di prenotare un ombrellone, ne conosca già la posizione esatta (Figura 3.10).

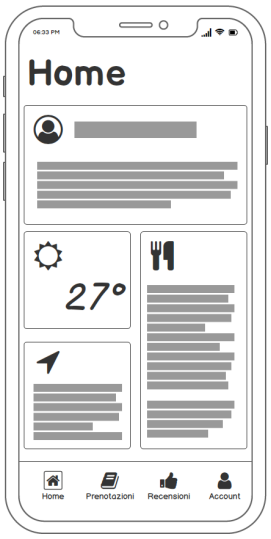

Figura 3.9. Mockup riguardante la Home Page

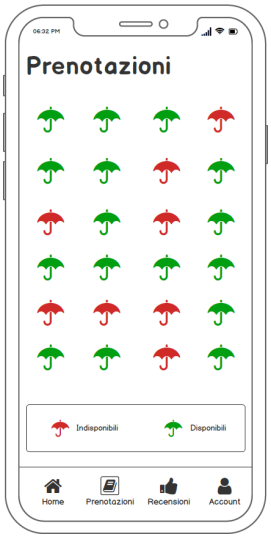

Figura 3.10. Mockup riguardante la disposizione degli ombrelloni

3.4.3 Recensioni

La terza pagina che l'applicazione offre è quella delle recensioni (Figura 3.11), ordinate cronologicamente dalla meno recente alla più recente. Tramite l'apposito pulsante di aggiunta, è possibile scrivere una nuova recensione (Figura 3.12).

Figura 3.11. Mockup riguardante la sezione delle recensioni

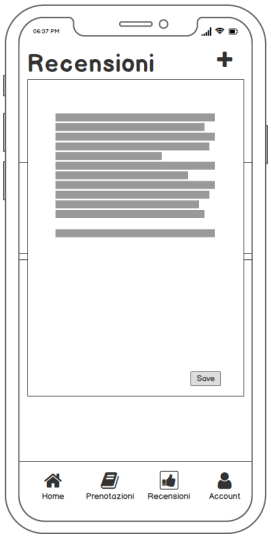

Figura 3.12. Mockup riguardante l'aggiunta di una recensione

3.4.4 Account

La sezione account rappresenta l'ultima sezione che compone lo scheletro dell'applicazione. Inizialmente è possibile scegliere quale servizio utilizzare per effettuare il login o oppure se effettuare la registrazione (Figura 3.13). Nella stessa schermata è anche possibile decidere se registrarsi tramite email e password (Figura 3.14), oppure se effettuare il login con email e password (Figura 3.15).

Dalla sezione account, una volta effettuato l'accesso, è possibile vedere le prenotazioni effettuate e le recensioni rilasciate (Figura 3.16).

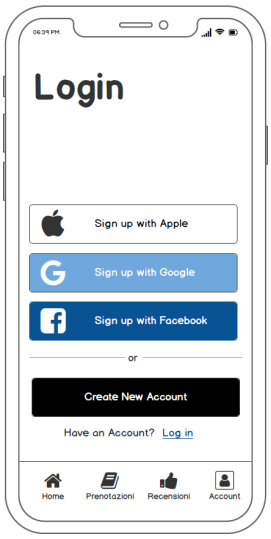

Figura 3.13. Mockup riguardante il login e la registrazione

Figura 3.14. Mockup per la registrazione con email e password

Infine, c'è anche la possibilità di modificare o aggiungere ulteriori informazioni al proprio profilo utente (Figura 3.17).

Figura 3.15. Mockup per il login con email e password

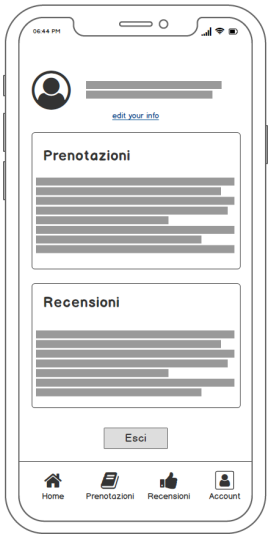

Figura 3.16. Mockup riguardante l'area personale

Figura 3.17. Mockup per la modifica del proprio profilo

4

Implementazione e manuale utente

Il presente capitolo tratta l'implementazione e lo sviluppo di tutte le funzionalità studiate e analizzate nel capitolo precedente. Inoltre, offre un manuale utente, affinché si abbia una guida all'uso dell'applicazione. Inizialmente, sarà descritta la struttura alla base del progetto, per poi esaminare come sono state implementate le varie funzioni mostrandone degli script, infine, verrà illustrato come utilizzare l'app presentando gli screen-shot dell'applicazione finale.

4.1 Struttura del progetto

La struttura dell'applicazione, sviluppata per mezzo dell'IDE Xcode, è mostrata nella Figura 4.1.

Gli elementi di fondamentale importanza, dal punto di vista implementativo, sono i seguenti:

- `AppDelegate.swift`: rappresenta il punto d'ingresso dell'applicazione ogni volta che viene lanciata; esso inizializza tutti i pacchetti necessari ed attiva, effettivamente, tutti i framework indispensabili al funzionamento della stessa.
- `ContentView.swift`: contiene l'interfaccia iniziale; è la prima schermata che viene visualizzata appena aperta l'applicazione.
- `Assets.xcassets`: racchiude tutte le icone, gli sticker o le immagini necessarie all'applicazione.
- `Info.plist`: è un elenco di informazioni circa il funzionamento dell'app; gestisce le informazioni relative, ad esempio, alla versione, all'orientamento dello smartphone, ecc.
- `HomeView.swift, ReservationView.swift, ReviewsView.swift, AccountView.swift, LoginView.swift, LoggedAccountView.swift, EditProfileView.swift`: dichiarano le interfacce utenti e, quindi, definiscono cosa visualizzare sullo schermo dello smartphone. Inoltre, in essi, si trova la maggior parte dei metodi utilizzati in quelle stesse interfacce.
- `SessionStore.swift`: contiene tutte le funzioni necessarie per la gestione degli account.

- `Booking, ManagerDate, Weather`: racchiudono file più piccoli, i quali permettono la corretta esecuzione di determinate funzionalità dell'app.

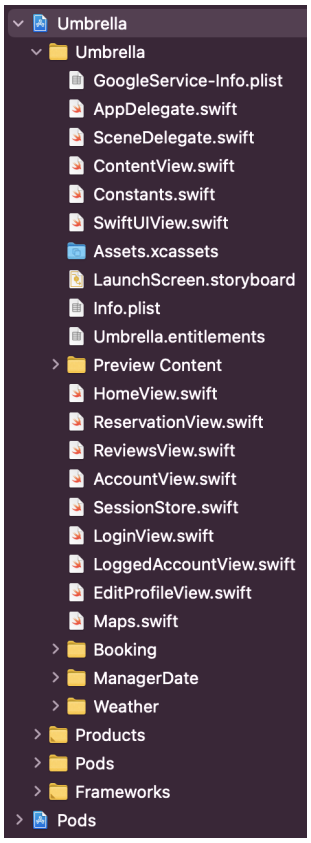

Figura 4.1. Struttura dell'applicazione

4.2 Implementazione

In questa sezione si analizzeranno in dettaglio i codici che realizzano tutte le funzionalità e le caratteristiche fin qui studiate nei precedenti capitoli.

4.2.1 AppDelegate.swift

Da questo file ha inizio l'esecuzione dell'applicazione; come detto in precedenza, esso rappresenta il punto d'avvio di tutto il codice.

Come prima istruzione è presente la dicitura `import UIKit` la quale serve per implementare il framework che contiene le classi necessarie per la gestione, la manipolazione e la visualizzazione di tutti quegli elementi che compongono un'interfaccia

utente. Questi framework nascono per dare allo sviluppatore gli strumenti di base per interagire tanto con il software quanto con l'hardware.

Successivamente si trova `@UIApplicationMain`; esso rappresenta il marcatore dal quale l'applicazione inizierà ad eseguire codice e, quindi, il punto di partenza vero e proprio.

Procedendo con il Listato 4.1, si arriva al primo metodo, `application`, il quale viene lanciato appena l'applicazione finisce di inizializzare tutti i componenti di cui ha bisogno. Le funzioni all'interno del metodo `application` servono per gestire il login di un utente e mantenerne la sessione di login attiva nel caso in cui egli avesse effettuato l'accesso precedentemente.

I restanti metodi, infine, serviranno all'applicazione per gestire eventuali cambi di stato della stessa, come, ad esempio, il passaggio dallo stato di esecuzione a quello di background.

```
import UIKit
import Firebase
import GoogleSignIn

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, GIDSignInDelegate {



    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.
    LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        FirebaseApp.configure()
        GIDSignIn.sharedInstance().clientID = FirebaseApp.app()?.options.clientID
        GIDSignIn.sharedInstance()?.delegate = self
        return true
    }

    func application(_ application: UIApplication, open url: URL, sourceApplication: String?, annotation: Any)
    -> Bool {
        return GIDSignIn.sharedInstance().handle(url)
    }

    func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!, withError error: Error?) {
      // ...
      if let error = error {
        // ...
        print(error.localizedDescription)
        return
      }

      guard let authentication = user.authentication else { return }
      let credential = GoogleAuthProvider.credential(withIDToken: authentication.idToken,
                                                     accessToken: authentication.accessToken)
      // ...
        Auth.auth().signIn(with: credential) {(res, err) in
            if err != nil {
                print((err?.localizedDescription)!)
                return
            }

            print ("user=" + (res?.user.email)!)
        }
    }

    func sign(_ signIn: GIDSignIn!, didDisconnectWith user: GIDGoogleUser!, withError error: Error!) {
        // Perform any operations when the user disconnects from app here.
        // ...
    }

    // MARK: UISceneSession Lifecycle

    func application(_ application: UIApplication, configurationForConnecting connectingSceneSession:
    UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {
        // Called when a new scene session is being created.
        // Use this method to select a configuration to create the new scene with.
        return UISceneConfiguration(name: "Default␣Configuration", sessionRole: connectingSceneSession.role)
    }

    func application(_ application: UIApplication, didDiscardSceneSessions sceneSessions: Set<UISceneSession>)
     {
        // Called when the user discards a scene session.
        // If any sessions were discarded while the application was not running, this will be called shortly after
         application:didFinishLaunchingWithOptions.
        // Use this method to release any resources that were specific to the discarded scenes, as they will not
        return.
```

```
    }
}
```

Listato 4.1. Il codice relativo ad `AppDelegate.swift`

4.2.2 ContentView.swift

Questo codice rappresenta la prima pagina che viene visualizzata all'avvio dell'applicazione. I file che contengono le direttive per realizzare le varie interfacce vengono denominati `View Page`, e, infatti, i nomi di tutta questa tipologia di file sono caratterizzati dalla presenza della parola `View` nella loro nomenclatura.

Nel Listato 4.2 relativo al progetto, si nota subito come il corpo principale del codice, `ContentView` (stesso nome del file), sia un oggetto di tipo `View`; questo vuol dire che il codice, al suo interno, conterrà indicazioni per un'interfaccia grafica.

Per prima cosa viene inizializzata una *Bottom Navigation Bar* nel metodo `init()`; esso sarà l'elemento principale dell'app con cui l'utente navigherà all'interno delle quattro sezioni disponibili. Ogni elemento della Navigation Bar corrisponde ad una View differente; inoltre, a ciascuna View, è associato un numero per far capire all'applicazione quale sezione stiamo consultando. Per far ciò si crea, quindi, una variabile `selected`, inizializzata a 0, che viene passata alla `TabView` in modo da avviare l'applicazione con il primo elemento della Navigation Bar selezionato.

```
import SwiftUI

struct ContentView: View {
    @State private var selectedSeats: [Seat] = []

    init() {
        UITabBar.appearance().backgroundColor = UIColor.white
    }

    @State var selected = 0

    var body: some View {

        TabView(selection: $selected) {
            HomeView().tabItem({
                Image(systemName: Constants.TabBarImage.tabBar0)
                Text("\(Constants.TabBarText.tabBar0)")
            }).tag(0)

            ReservationView().tabItem({
                Image(systemName: Constants.TabBarImage.tabBar1)
                Text("\(Constants.TabBarText.tabBar1)")
            }).tag(1)

            ReviewsView().tabItem({
                Image(systemName: Constants.TabBarImage.tabBar2)
                Text("\(Constants.TabBarText.tabBar2)")
            }).tag(2)

            AccountView().tabItem({
                Image(systemName: Constants.TabBarImage.tabBar3)
                Text("\(Constants.TabBarText.tabBar3)")
            }).tag(3)
        }.accentColor(Color.black)
    }
}
```

Listato 4.2. Il codice relativo a `ContentView.swift`

4.2.3 HomeView.swift

La HomeView è la prima View che viene iniettata tramite la Navigation Bar all'avvio dell'app. In essa si definisce l'interfaccia per tutte le varie informazioni che vengono rappresentate nella home page dell'applicazione.

Nel codice (Listato 4.3) si dichiarano, inizialmente, delle variabili `var` che serviranno al corretto funzionamento dei metodi successivi; inoltre, nel codice stesso, si fa uso della funzione `getUser`, che controlla se un utente ha già effettuato il login e ne mantiene la sessione attiva senza richiedere nuovamente l'accesso. A questo punto si dichiara il primo dei contenuti informativi che l'applicazione è in grado di mostrare, ovvero la sezione utente.

Tramite la struttura di controllo `if` viene controllato se è presente o meno un utente che ha effettuato il login. In caso negativo, `session.session = nil`, si rappresenta un'immagine generica con l'invito ad effettuare il login o a registrarsi; altrimenti (`url != ""`), se l'utente ha effettuato il login e sono disponibili informazioni personali nel database, nella home page vengono iniettati lo username e la foto profilo dell'utente. Infine, se c'è una sessione attiva da parte di un utente registrato, il quale, però, non ha ancora aggiunto informazioni personali (foto profilo, username, nome e cognome), questo viene invitato a modificare il proprio account nella sezione dedicata e viene raffigurata un'immagine di profilo generica.

Gli oggetti che si utilizzano con maggior frequenza nel linguaggio SwiftUI per creare interfacce complesse sono `VStack` (Stack Verticale), `HStack` (Stack Orizzontale) o anche `ZStack` (Stack di Profondità), i quali permettono di comporre e creare disposizioni di più elementi di tipo View insieme, creando, quindi, una composizione di oggetti e lasciando completa libertà alla fantasia dello sviluppatore.

```
import SwiftUI
import SDWebImageSwiftUI
import Firebase

struct HomeView: View {

    @EnvironmentObject var session: SessionStore
    @State var url = ""
    @State var username = ""
    @State var receipe = ""
    @State private var selected = 0
    @ObservedObject var weather = CurrentWeatherViewModel()

    func getUser() {
        session.listen()
    }

    var body: some View {
        NavigationView {

            VStack {

                if (session.session == nil) {
                    HStack(alignment: .center) {

                        Image(systemName: Constants.accountImageAbsent)
                            .padding(.leading)
                            .frame(width: 60, height: 120)
                            .font(.system(size: 50))
                            .foregroundColor(.white)

                        Text ("Log␣in␣or␣sign␣in␣to␣use␣all␣functionalities␣of␣this␣app")
                            .font(.system(size: 25))
                            .fontWeight(.bold)
                            .frame(width:285, height: 140)
                            .foregroundColor(.white)

                    }
                     .background(Color(red: 0.90, green: 0.18, blue: 0.15))
                     .cornerRadius(30)
                     .frame(width: 60)

                } else if (url != "") {
```

```
                HStack(alignment: .center) {
                    AnimatedImage(url: URL(string: url)).resizable()
                        .frame(width: 90, height: 120)
                        .clipShape(Circle())
                        .padding(.leading)

                    Text ("Welcome␣back,␣\(username)")
                        .font(.system(size: 22))
                        .fontWeight(.bold)
                        .frame(width:245, height: 140)
                        .foregroundColor(.white)


                }.background(Color(red: 0.63, green: 0.81, blue: 0.96))
                 .cornerRadius(30)
                 .frame(width: 50)

            } else {
                HStack(alignment: .center) {

                    Image(systemName: Constants.accountImageAbsent)
                        .padding(.leading)
                        .frame(width: 65, height: 50)
                        .font(.system(size: 50))
                        .foregroundColor(.white)

                    Text ("Tell␣us␣more␣about␣you␣in␣the␣account␣section")
                        .font(.system(size: 25))
                        .fontWeight(.bold)
                        .frame(width:285, height: 140)
                        .foregroundColor(.white)

                }.background(Color(red: 0.63, green: 0.81, blue: 0.96))
                 .cornerRadius(30)
                 .frame(width: 50)
            }
```

Listato 4.3. Sezione account della pagina `HomeView.swift`

Il Listato 4.4 contiene, invece, l'interfaccia per rappresentare le restanti informazioni riguardanti il meteo, la mappa per raggiungere lo stabilimento balneare, ed infine, il menù giornaliero.

Per quanto riguarda il meteo, si fa uso del `GeometryReader`, il quale è un metodo predefinito di SwiftUI che permette di ridimensionare determinate porzioni dell'interfaccia per creare effetti visivi particolari; nel caso del progetto viene utilizzato per visualizzare sullo schermo la struttura `CurrentWeather`. Quest'ultima non è nient'altro che un oggetto di tipo View, definito in un altro file per mantenere il codice ordinato, che, semplicemente, riproduce sullo schermo dello smartphone le previsioni meteorologiche della località in cui si trova il balneare.

La mappa, come avviene per il meteo, viene visualizzata iniettando nella home page un'altra struttura di tipo View, chiamata `Maps`, che permette di realizzare, all'interno del proprio riquadro dedicato, la mappa satellitare con la posizione GPS esatta dello stabilimento balneare.

Per concludere, il menù viene riprodotto dopo aver soddisfatto la struttura di controllo `if`: se la variabile `receipe` è una parola con più di 5 lettere (ovvero una frase), il testo che contiene viene visualizzato accanto ad un'icona stilizzata raffigurante la sezione di ristorazione; in caso contrario, viene stampata una frase che avvisa di non aver ancora aggiornato il menù giornaliero.

Notevole attenzione va prestata al metodo `onAppear` alla fine del codice, il quale, come suggerisce il nome, esegue le istruzioni al suo interno ogniqualvolta viene lanciata la schermata; in particolare esso inizializza le variabili per permettere ai metodi analizzati precedentemente di essere eseguiti istantaneamente. Tra le funzioni messe in atto da tale metodo si ha il caricamento dei dati dell'utente se la sessione di login è ancora attiva; ciò avviene tramite `getUser` e `Auth.auth().currentUser?.uid`; sempre tale metodo per mezzo delle varie strutture di controllo `if`, salva in differen-

ti variabili i dati provenienti dal database, come username, foto del profilo oppure menù.

```
                HStack {

                    VStack{

                        VStack{
                            GeometryReader { gr in
                                CurrentWeather(weather: self.weather.current, height: self.selected == 0 ?
                                gr.size.height: gr.size.height * 0.50).animation(.easeInOut(duration: 0.5))
                            }
                        }.frame(minWidth: 0, maxWidth: .infinity, minHeight: 0, maxHeight: .infinity).cornerRadius(30)

                        VStack{
                            Maps()
                        }.cornerRadius(30)
                    }.padding(.horizontal, 8)

                    VStack {
                        Image(systemName: Constants.forkImage)
                            .padding(.top, 60.0)
                            .frame(width: 60, height: 60)
                            .font(.system(size: 50))
                            .foregroundColor(.white)

                        if (receipe.count > 5) {
                            Text ("\(receipe)")
                            .font(.headline)
                            .font(.system(size: 45))
                            .frame(width:140, height:340)
                            .padding(.horizontal)
                            .foregroundColor(.white)

                        } else {
                            Text ("Menù non ancora aggiornato!")
                                .font(.headline)
                                .frame(width:170, height:340)
                                .foregroundColor(.white)
                        }
                    }
                     .background(Color(red: 0.45, green: 0.70, blue: 0.60))
                     .cornerRadius(30)
                     .padding(.horizontal, 10)
                }.padding(.bottom, 30)

            }
            .navigationBarTitle("Home")
            .onAppear(){
                let uid = Auth.auth().currentUser?.uid
                let storage = Storage.storage().reference()
                let db = Firestore.firestore()
                let date = Date()
                let dateFormatter = DateFormatter()
                dateFormatter.dateFormat = "EEEE"
                let dayInWeek = dateFormatter.string(from: date)

                if (uid != nil) {
                    storage.child("profilepics").child(uid!).downloadURL { (url, err) in
                        if err != nil {
                            print((err?.localizedDescription)!)
                            return
                        }
                        self.url = "\(url!)"
                    }
                    db.collection("users").document(uid!).getDocument { (document, error) in
                        if let document = document, document.exists {
                            let property = document.get("username")
                            self.username = property as! String
                        } else {
                            print("Document does not exist")
                        }
                    }
                    db.collection("restaurants").document(dayInWeek).getDocument { (document, error) in
                        if let document = document, document.exists {
                            let prop = document.get("receipe")
                            self.receipe = prop as! String
                        } else {
                            print("Document does not exist")
                        }
                    }
                } else {

                    db.collection("restaurants").document(dayInWeek).getDocument { (document, error) in
                        if let document = document, document.exists {
                            let prop = document.get("receipe")
                            self.receipe = prop as! String
                        } else {
                            print("Document does not exist")
                        }
                    }
```

```
                return
            }

            }

        }
        .onAppear(perform: getUser)
    }

}
```

Listato 4.4. Sezione meteo, mappa e menù nella `HomeView.swift`

4.2.4 ReservationView.swift

La ReservationView è la seconda sezione dell'applicazione ed è quella che offre la possibilità di prenotare un ombrellone agli utenti registrati.

L'interfaccia grafica, consultabile nel Listato 4.5, viene realizzata intrecciando e coordinando vari oggetti di tipo Stack. La realizzazione è affidata alla struttura `Reservation()` che, poi, viene iniettata nella pagina principale, ovvero la `ReservationView: View`.

La rappresentazione del mare (stilizzata) viene resa possibile grazie al metodo `ScreenShape`, che realizza l'arco di colore blu, e al metodo `Rectangle()` che, a sua volta, permette di visualizzare un effetto sfuocato sotto il precedente arco.

Gli ombrelloni, invece, raffigurati con dei cerchi di colore differente, sono ordinati per mezzo di una griglia con il metodo `Grid`; per ciascun ombrellone presente nel database viene creato un bottone `i` corrispondente, il quale incorpora il numero e l'id dell'ombrellone stesso, con i codici `i.id` e `i.number`; infine, ogni ombrellone creato viene ritratto nella griglia per mezzo del metodo `ChairViewTry`. Quest'ultimo gestisce l'effettiva prenotazione degli ombrelloni. Alla pressione sul singolo ombrellone, il metodo esegue una serie di controlli mediante il costrutto `if`, verificando, inizialmente, se l'utente ha effettuato o meno il login; a questo punto controlla nel database se l'ombrellone selezionato è effettivamente disponibile ed, infine, in caso di successo, conferma la prenotazione tramite un `AlertItem` stampato sullo schermo.

Per concludere, la legenda relativa al colore degli ombrelloni è permessa grazie al metodo `createSeatsLegend()`, e il rettangolo che ne determina i confini viene gestito tramite il metodo `RoundedRectangle`.

```
import SwiftUI
import Firebase
import Grid

struct ReservationView: View {
    var body: some View {
        Reservation()
    }
}

struct Reservation : View {
    @ObservedObject var Umbrell = getUmbrella()
    @State var docnumber = ""
    @State var docID = ""
    @State var docavailable = ""

    var body: some View {

           VStack{

               HStack {
                   Text("Booking")
                       .font(.system(size: 30))
                       .fontWeight(.heavy)
                       .padding(.leading, 15)
```

```
                    .padding(.top, 50)

                Spacer()
            }

            ZStack{

                Rectangle()
                    .fill(LinearGradient(gradient: Gradient(colors: [Color.blue.opacity(0.3), .clear]),
                    startPoint: .init(x: 0.5, y: 0.0), endPoint: .init(x: 0.5, y: 0.5)) )
                    .frame(height: 420)
                    .clipShape(ScreenShape(isClip: true))
                    .cornerRadius(20)

                ScreenShape()
                    .stroke(style: StrokeStyle(lineWidth: 5, lineCap: .square ))
                    .frame(height: 420)
                    .foregroundColor(Color.blue)

                if self.Umbrell.data.isEmpty {
                    Text("Database updating...")
                }

                else
                {
                    Grid(self.Umbrell.data){ i in
                        Button(action: {
                            self.docID = i.id
                            self.docnumber = i.number

                        }) {
                            HStack{
                                ChairViewTry(umbrella: i)
                            }
                        }
                    }.padding(.top, 100)
                    .gridStyle(StaggeredGridStyle(tracks: 5, spacing: 15))
                }
            }
            ZStack{
                RoundedRectangle(cornerRadius: 10)
                    .stroke(Color.blue, lineWidth: 2)
                    .frame(width: 330, height: 60)
                    .padding(.bottom, 70)
                    .padding(.horizontal, 20)
                createSeatsLegend()
            }
        }
    }

    fileprivate func createSeatsLegend() -> some View{

        HStack{
            ChairLegend(text: "Reserved", color: .red)
            ChairLegend(text: "Available", color: .green)
        }
         .padding(.horizontal, 20)
         .padding(.bottom, 70)
    }

}

struct ChairViewTry: View {
    @ObservedObject var umbrella: Umbrella
    let db = Firestore.firestore()
    @EnvironmentObject var session: SessionStore
    @State private var showingAlert = false
     @State private var alertItem: AlertItem?

    var body: some View {

       VStack{

            Button (action: {
                if (self.session.session == nil) {

                    self.alertItem = AlertItem(title: Text("Account missing"), message: Text("You need to
                    log in or sign in to book your umbrella"), dismissButton:.cancel(Text("Ok")))

                } else if (self.umbrella.available) {
                    self.db.collection("umbrellaXY").document(self.umbrella.id).updateData(["available": false])
                    self.alertItem = AlertItem(title: Text("Umbrella reserved"), message: Text("Successful
                    confirmation, enjoy your holiday"), dismissButton:.cancel(Text("Ok")))

                } else {
                    self.alertItem = AlertItem(title: Text("Umbrella not available"), message: Text("You can not
                    reserve this umbrella, try with another available"), dismissButton:.cancel(Text("Ok")))
                }

            }) {
                Circle()
                    .frame(width: 32, height: 32)
                    .foregroundColor(self.umbrella.available ? Color.green : Color.red)
            }
```

```
                .alert(item: $alertItem) {
                    alertItem in
                    Alert(title: alertItem.title, message: alertItem.message, dismissButton: alertItem.dismissButton)
                }
            }
        }
    }
```

Listato 4.5. Il codice relativo a `ReservationView.swift`

4.2.5 ReviewsView.swift

La penultima sezione dell'app è quella relativa alle recensioni; essa, infatti, deve permettere agli utenti che hanno effettuato il login di visualizzare tutte le recensioni e di inserirne di nuove.

Come nella `ReservationView`, anche in questo caso si crea l'interfaccia nella struttura `Home()`; e solo poi, viene inserita nel corpo principale, che è il `ReviewsView: View` (Listato 4.6). Dopo aver inizializzato le variabili di cui si serve il codice, viene subito controllata la presenza di recensioni nel database tramite la struttura di controllo `if self.Notes.data.isEmpty`; se non sono presenti recensioni verranno visualizzati messaggi di avviso o aggiornamento in corso, mentre, in caso contrario, una `ScrollView` organizzerà verticalmente le recensioni che sono state estrapolate dal database, analizzandole una alla volta per mezzo della struttura `ForEach(self.Notes.data)` e stampando sullo schermo il contenuto del campo corrispondente alla recensione vera e propria, `Text(i.note)`. Inoltre, essendo rappresentate, non solo come testo, ma anche e soprattutto come `Button`, l'utente registrato può interagire con esse, ad esempio per modificarle o eliminarle.

L'aggiunta di una nuova recensione, invece, viene effettuata mediante il tasto di aggiunta posizionato in alto a destra; esso è disabilitato nel caso in cui l'utente non sia registrato per mezzo del controllore `.disabled(session.session?.email == nil))`, usato in modo analogo anche in altre funzioni riservate agli utenti registrati. L'azionamento del pulsante mette in moto una *Sheet View*, ossia una pagina sovrastante a quella già esistente, dove, all'interno, viene iniettata, a sua volta, una nuova View, ovvero la `EditView`. Questa non è nient'altro che una pagina dove l'utente può scrivere la propria recensione e, successivamente, salvarla mediante il tasto apposito. Il metodo `.animation(.default)` si preoccupa della gestione della Sheet View visualizzandola o ritirandola al bisogno, in base alle interazioni dell'utente.

```
import SwiftUI
import Firebase

struct ReviewsView: View {
    var body: some View {
        Home()
    }
}

struct Home : View {

    @ObservedObject var Notes = getNotes()
    @State var show = false
    @State var txt = ""
    @State var docID = ""
    @State var remove = false
    @EnvironmentObject var session: SessionStore

    var body : some View{

        NavigationView {
```

```
                VStack(spacing: 0){

                    if self.Notes.data.isEmpty{

                        if self.Notes.noData{

                            Spacer()
                            Text("No Reviews !!!")
                            Spacer()
                        }
                        else{

                            Spacer()
                            Indicator()
                            Spacer()
                        }
                    }

                    else{

                        ScrollView(.vertical, showsIndicators: false) {

                            VStack{

                                ForEach(self.Notes.data){i in

                                    HStack(spacing: 15){

                                        Button(action: {

                                            self.docID = i.id
                                            self.txt = i.note
                                            self.show.toggle()

                                        }) {
                                            VStack(alignment: .leading, spacing: 12){

                                                Text(i.note).lineLimit(1)
                                                Divider()

                                            }.padding(10)
                                             .foregroundColor(.black)

                                        }.disabled(self.session.session?.email == nil)

                                        if self.remove{

                                            Button(action: {

                                                let db = Firestore.firestore()
                                                db.collection("notes").document(i.id).delete()

                                            }) {
                                                Image(systemName: "minus.circle.fill")
                                                    .resizable()
                                                    .frame(width: 20, height: 20)
                                                    .foregroundColor(.red)
                                            }
                                        }
                                    }.padding(.horizontal)
                                }
                            }
                        }
                    }
                }
                .navigationBarTitle("Reviews")
                .navigationBarItems(trailing:
                    Button(action: {

                    self.txt = ""
                    self.docID = ""
                    self.show.toggle()

                }) {
                    Image(systemName: "plus").resizable()

                }.disabled(session.session?.email == nil))
            }
            .sheet(isPresented: self.$show) {

                EditView(txt: self.$txt, docID: self.$docID, show: self.$show)
            }
            .animation(.default)
        }
    }
```

Listato 4.6. Il codice relativo a `ReviewsView.swift`

4.2.6 AccountView.swift

L'AccountView è la sezione che permette di registrarsi o effettuare il login, e, successivamente, di gestire il proprio profilo personale. Si nota dal Listato 4.7 che il codice di questo file è molto breve ma efficace: `if (session.session != nil)` controlla la presenza di una sessione di login ancora attiva; in caso positivo si inietta la pagina del profilo utente `LoggedAccountView()`, mentre, in caso contrario, si inietta la pagina per permettere all'utente di registrarsi o effettuare il login, ovvero `LoginView()`.

```
import SwiftUI

struct AccountView: View {
    @EnvironmentObject var session: SessionStore

    func getUser() {
        session.listen()
    }

    var body: some View {
        Group {
            if (session.session != nil) {
                LoggedAccountView()
            } else {
                LoginView()
            }
        }.onAppear(perform: getUser)

    }
}
```

Listato 4.7. Il codice relativo a `AccountView.swift`

A seguito di ciò, viene riscontrato un utente che ha effettuato il login precedentemente; quindi si inietta, nella quarta sezione relativa all'account personale, la `LoggedAccountView` (Listato 4.8).

Una struttura di controllo `if url.count` cerca nel database se è presente l'indirizzo URL (maggiore di 6 lettere) con la foto del profilo caricata dall'utente registrato; se viene riscontrato un indirizzo valido, la foto viene scaricata e visualizzata tramite la struttura predefinita in SwiftUI, `AnimatedImage`; invece, se non è presente nessuna foto per l'utente, verrà visualizzata un'immagine standard `Image(systemName: Constants.accountImageAbsent)`. Analogo meccanismo si ha per la visualizzazione dello username con il controllore `if username.count` (parola maggiore di 0 lettere).

Successivamente, si definisce il tasto per modificare il profilo utente `Button("Edit info"`; questo mette in moto la presentazione di una vista, sovrastante a quella già presente con il meccanismo di una Sheet View (analizzata nel capitolo precedente), la quale visualizza, a sua volta, la `EditProfileView.swift`. Tale vista permette di aggiungere una foto profilo, uno username, un nome e un cognome aggiornando in tempo reale il database; una volta salvati i dati, la Sheet View verrà chiusa automaticamente ritornando all'interfaccia principale, ovvero la LoggedAccountView.

Anche in questo caso si sfrutta la funzione `.onAppear()` per estrapolare tutte le informazioni necessarie ogni volta che viene lanciata la schermata; così facendo, per visualizzare la foto del profilo o lo username appena aggiornati, sarà semplicemente necessario cambiare schermata per permettere all'interfaccia di aggiornarsi, e non dover uscire dall'applicazione; sarà, altresì, possibile chiudere tale interfaccia, per permettere ad essa di aggiornarsi con i nuovi dati.

Quindi, in una `VStack`, si presenta una sintesi delle recensioni e delle prenotazioni effettuate dall'utente. Infine, viene implementato il tasto `Button(...)Text("Sign Out")` per effettuare il logout; esso pone fine definitivamente alla sessione utente invocando il metodo `session.signOut`.

```
import SwiftUI
import Firebase
import FirebaseStorage
import SDWebImageSwiftUI

struct LoggedAccountView: View {
    @EnvironmentObject var session: SessionStore
    @State var isModal: Bool = false
    @State var url = ""
    @State var username = ""

    var body: some View {
        VStack{
            VStack(alignment: .center){
                HStack(alignment: .center) {

                    if url.count < 6 {
                        Image(systemName: Constants.accountImageAbsent)
                        .padding(.leading)
                        .frame(width: 60, height: 60)
                        .font(.system(size: 50))

                    } else {
                        AnimatedImage(url: URL(string: url)).resizable().frame(width: 100, height: 150)
                        .clipShape(Circle())

                    }

                    if username.count > 0 {

                        Text ("Welcome␣back␣\(username)")
                            .font(.system(size: 25))
                            .fontWeight(.bold)
                            .frame(width:250, height: 60)

                    } else {

                        Text ("Welcome,␣tell␣us␣more␣about␣you")
                        .font(.system(size: 25))
                        .fontWeight(.bold)
                        .frame(width:250, height: 60)
                    }
                }

                Button("Edit␣info") {
                    self.isModal = true
                    }.sheet(isPresented: $isModal, content: {
                        EditProfileView()
                    }).foregroundColor(Color.blue)
                }
            .padding(.top, 50)
            .onAppear(){...}

            Spacer()

            VStack{
                VStack{...}
                .frame(width:360, height: 80)
                .padding(.bottom, 25)
                .background(Color(red: 0.26, green: 0.47, blue: 0.59))
                .cornerRadius(20)

                Spacer()

                VStack{...}
                .frame(width:360, height: 280)
                .padding(.bottom, 60)
                .padding(.horizontal, 2)
                .background(Color(red: 0.94, green: 0.80, blue: 0.55))
                .cornerRadius(20)

            }

            Spacer()

            Button(action: session.signOut) {
                Text("Sign␣Out")
                    .frame(width: 310, height: 55)
                    .font(.system(size: 16, weight: .bold))
                    .background(Color.white)
                    .cornerRadius(20)
                    .overlay(
                        RoundedRectangle(cornerRadius: 20)
                            .stroke(Color.red, lineWidth: 1)
```

```
                        .opacity(0.7))

            }
            .padding(.bottom, 10.0)
        }
    }
}
```

Listato 4.8. Il codice relativo a `LoggedAccountView.swift`

Invece, nel caso in cui nessuna sessione utente sia attiva, verrà iniettata la `LoginView`, reperibile nel Listato 4.9.

All'interno di uno Stack verticale vengono raffigurati tutti i tasti relativi ai vari servizi offerti al fine di poter effettuare il login o la registrazione. Il primo servizio offerto è quello di Google con il metodo predefinito realizzato da Firebase, ovvero `google()`. Procedendo si incontra il servizio di accesso tramite Facebook, anch'esso offerto e incorporato da Firebase.

Quindi si propone la registrazione, o il login, classico, mediante mail e password. Questa opzione viene realizzata per mezzo di due View differenti che vengono iniettate sopra alla vista principale, momentaneamente, dal metodo predefinito `NavigationLink` (simile al metodo Sheet visto in precedenza). Una volta viene raffigurata la vista per la registrazione passando al metodo `NavigationLink` l'oggetto `destination: SignUpView()`; un'altra volta viene raffigurata la vista per il login, passando al metodo `NavigationLink` l'oggetto `destination: SignInView()`.

```
struct LoginView: View {
    var body: some View {
        NavigationView {

            VStack{
                Spacer()

                google()

                loginFB()
                    .frame(width: 310, height: 55)
                    .cornerRadius(20)
                    .padding(5)

                Divider()

                NavigationLink(destination: SignUpView()) {
                    HStack {
                        Text("Create␣a␣new␣account")
                            .frame(width: 310, height: 55)
                            .foregroundColor(.white)
                            .font(.system(size: 16, weight: .bold))
                            .background(Color.black)
                            .cornerRadius(20)
                    }
                }.padding(5)

                NavigationLink(destination: SignInView()) {
                    HStack {
                        Text("Have␣an␣account?")
                            .font(.system(size: 18, weight: .light))
                            .foregroundColor(.primary)

                        Text("Log␣in")
                            .font(.system(size: 18, weight: .semibold))
                            .foregroundColor(Color.blue)

                    }
                }
            }
            .padding([.leading, .bottom, .trailing], 32)
            .navigationBarTitle("Login")
        }
    }
}
```

Listato 4.9. Il codice relativo a `LoginView.swift`

Nella prima opzione (Listato 4.10), il codice inizializza le variabili necessarie, implementa funzioni per aggiornare il database, come `loadData` o `CreateUserDB`, e inietta l'interfaccia vera e propria. La View propone due campi da riempire con i dati richiesti, ovvero mail e password, mediante `TextField` per l'email e `SecureField` per la password. Per concludere, si ha il tasto per la creazione dell'account `Button(action: signUp)`, il quale lancia in esecuzione il metodo che permette di salvare nel database l'utente con i dati appena inseriti.

```
struct SignUpView: View {
    @State var email: String = ""
    @State var password: String = ""
    @State var error: String = ""
    @EnvironmentObject var session: SessionStore
    //per i dati inseriti nel DB
    @State var alert = false
    @State var picker = false
    @State var loading = false
    @State var imagedata : Data = .init(count: 0)
    @State var username: String = ""


    //Funzione per aggiungere i dati nel DB
    func loadData () {...}

    //Funzione per aggiungere un utente nel DB
    func CreateUserDB (username: String, imagedata: Data) {...}

    //struttura per gestire la scelta dell'immagine da caricare
    struct Indicator : UIViewRepresentable {...}

    //Funzione di login
    func signUp() {...}

    var body: some View {
        VStack{
            Text("Create␣an␣account")
                .font(.system(size: 40, weight: .heavy))

            Text("Sign␣up␣to␣get␣started")
                .font(.system(size: 25, weight: .medium))
                .foregroundColor(Color.gray)

            VStack(spacing: 20) {

                TextField("Email␣address", text: $email)
                    .font(.system(size: 20))
                    .padding(8)
                    .background(RoundedRectangle(cornerRadius: 8).strokeBorder(Color.gray, lineWidth: 1))

                SecureField("Password", text: $password)
                    .font(.system(size: 20))
                    .padding(8)
                    .background(RoundedRectangle(cornerRadius: 8).strokeBorder(Color.gray, lineWidth: 1))
            }.padding(.vertical, 30)
            .sheet(isPresented: self.$picker, content: {
                ImagePicker(picker: self.$picker, imagedata: self.$imagedata)
            })

            Button(action: signUp) {
                Text("Create␣Account")
                    .frame(minWidth: 0, maxWidth: .infinity)
                    .frame(height: 50)
                    .foregroundColor(.white)
                    .font(.system(size: 16, weight: .bold))
                    .background(Color.black)
                    .cornerRadius(20)
            }.disabled(email.isEmpty || password.isEmpty)

            if (error != "") {...}
            Spacer()
        }.padding(.horizontal, 32)
    }
}
```

Listato 4.10. Il codice relativo a `SignUpView.swift`

Nella seconda opzione, nel caso in cui l'utente abbia già un profilo e debba soltanto effettuare l'accesso mediante mail e password, viene iniettata l'interfaccia `SignInView` (Listato 4.11).

Questo codice inizializza, anch'esso, le variabili di cui ha necessità; successiva-

mente, esso incorpora il metodo per effettuare l'accesso, ovvero `signIn`. L'interfaccia viene realizzata tramite due campi da riempire, come accade nel caso della registrazione; questi sono `TextField` e `SecureField`. Da ultimo, viene mostrato il tasto per avviare la funzione di login, ovvero `Button(action: signIn)`.

```
struct SignInView: View {
    @State var email: String = ""
    @State var password: String = ""
    @State var error: String = ""
    @EnvironmentObject var session: SessionStore

    func signIn() {...}

    var body: some View {
        VStack{
            Text("Welcome back!")
                .font(.system(size: 40, weight: .heavy))

            Text("Sign in to continue")
                .font(.system(size: 25, weight: .medium))
                .foregroundColor(Color.gray)

            VStack(spacing: 20) {
                TextField("Email address", text: $email)
                    .font(.system(size: 20))
                    .padding(12)
                    .background(RoundedRectangle(cornerRadius: 8).strokeBorder(Color.gray, lineWidth: 1))

                SecureField("Password", text: $password)
                    .font(.system(size: 20))
                    .padding(12)
                    .background(RoundedRectangle(cornerRadius: 8).strokeBorder(Color.gray, lineWidth: 1))
            }.padding(.vertical, 65)

            Button(action: signIn) {
                Text("Sign In")
                    .frame(minWidth: 0, maxWidth: .infinity)
                    .frame(height: 50)
                    .foregroundColor(.white)
                    .font(.system(size: 16, weight: .bold))
                    .background(Color.black)
                    .cornerRadius(20)
            }

            if (error != "") {...}

            Spacer()

        }.padding(.horizontal, 32)
    }
}
```

Listato 4.11. Il codice relativo a `SignInView.swift`

4.3 Manuale utente

In questa sezione si illustrano le istruzioni necessarie al corretto utilizzo delle principali funzionalità che l'applicazione finale mette a disposizione dell'utente.

4.3.1 Creazione di un account

Una volta scaricata ed installata l'applicazione, ad ogni avvio verrà visualizzata la home page (Figura 4.2).

Tramite la barra di navigazione posta in basso, l'utente dovrà spostarsi nella sezione *Account*, ultima delle quattro, e da qui potrà scegliere quale servizio utilizzare per la creazione del proprio profilo utente (Figura 4.3).

Se in possesso di un account Google o Facebook, l'utente può scegliere di utilizzare tale account per effettuare l'accesso, sia nel caso in cui non abbia mai avuto

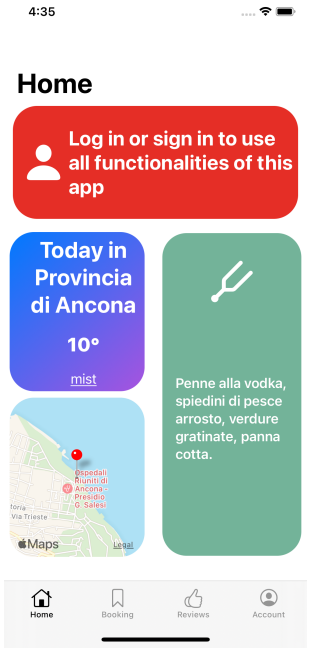

Figura 4.2. Home page dell'applicazione

a che fare con tale applicazione, sia nel caso in cui debba solamente effettuare di nuovo il login.

Per quanto riguarda la creazione di un account con mail e password, invece, l'utente dovrà procedere sull'apposito tasto *Create a new account*, e, successivamente, inserire la propria email e una password scelta adeguatamente (composta almeno da 6 caratteri).

Per terminare la sessione di login, basterà soltanto disconnettersi con l'apposito tasto in rosso *Sign Out*.

4.3.2 Prenotazione di un ombrellone

Per la prenotazione di un ombrellone, l'utente, prima di tutto, deve effettuare l'accesso al proprio account, come spiegato nella sezione precedente, altrimenti riceverà un messaggio di avviso per la mancanza di un account valido.

Effettuato il login, egli dovrà spostarsi nella sezione *Booking* tramite la barra di navigazione. Qui vedrà visualizzati tutti gli ombrelloni offerti dallo stabilimento balneare; tali ombrelloni saranno di colore verde se ancora liberi e disponibili alla prenotazione, e di colore rosso se non disponibili (Figura 4.4).

A questo punto, una volta scelto l'ombrellone da prenotare, all'utente basterà soltanto cliccarci sopra per ricevere un `AlertItem` (avviso rettangolare che compare in primo piano) con il messaggio di avvenuta prenotazione (Figura 4.5). Fatto ciò, l'ombrellone appena prenotato cambierà colore in rosso, essendo stato appena riservato all'utente che lo ha prenotato.

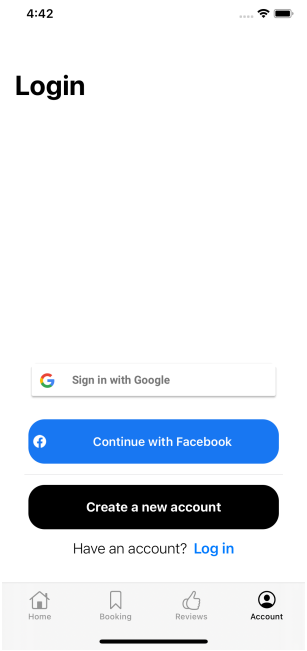

Figura 4.3. Sezione account per la registrazione o il login

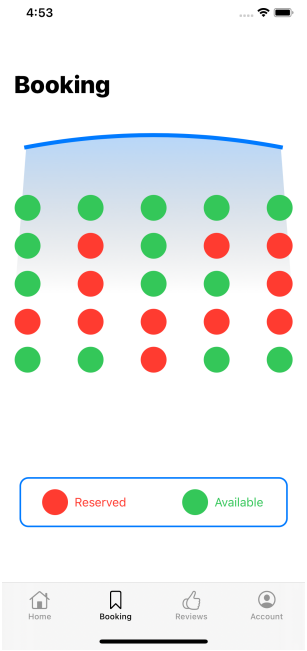

Figura 4.4. Sezione relativa alla prenotazione di un ombrellone

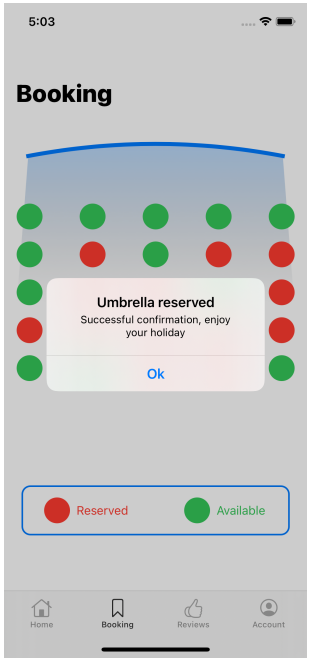

Figura 4.5. Conferma della prenotazione di un ombrellone

4.3.3 Modifica del profilo utente

Per la modifica del proprio profilo, l'utente deve effettuare l'accesso con le proprie credenziali. Fatto ciò, si troverà nella schermata *Account* relativa al proprio profilo (Figura 4.6).

Come indicato dal testo in alto, egli dovrà cliccare, a questo punto, sul tasto *Edit info*, che azionerà la Sheet View; quest'ultima, a sua volta, presenterà i campi da riempire per permettere all'utente di aggiungere o modificare i propri dati personali (Figura 4.7).

Conclusa tale operazione, verranno subito popolate le sezioni personali con lo username e la foto che l'utente ha scelto di caricare, sia nella *Home* che nella sezione *Account*.

Figura 4.6. Sezione account personale

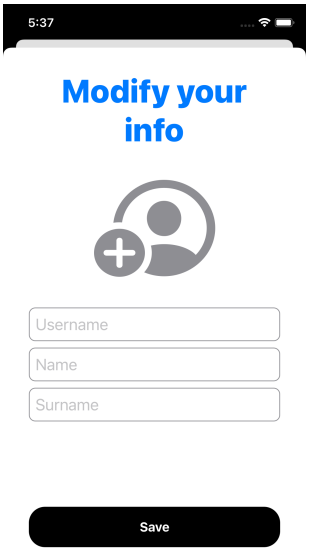

Figura 4.7. Modifica del profilo utente

5

Discussione in merito al lavoro svolto

In questo capitolo viene effettuata un'analisi conclusiva sui concetti appresi durante l'elaborazione e lo sviluppo del progetto. Successivamente, viene valutato il progetto finale per evidenziarne lati positivi e negativi. Per concludere, sono presentati alcuni possibili sviluppi futuri per migliorare alcuni aspetti dell'app non ancora ottimali.

5.1 Conoscenze acquisite sull'uso di un nuovo IDE e di Firebase

Sviluppare un'applicazione per dispositivi Apple implica l'utilizzo di un IDE particolare dedicato, denominato Xcode (Figura 5.1).

Durante la fase di sviluppo e implementazione delle varie funzionalità, si sono presentati non poche difficoltà e impedimenti; questi ostacoli, d'altro canto, hanno consentito di ottenere una maggiore conoscenza dell'ambiente di sviluppo in quanto, molto spesso, dover risolvere un errore di codice, o anche di natura tecnica, significa approfondire aspetti e particolarità che, in altri contesti o occasioni, non sarebbero stati affrontati e compresi al meglio.

Figura 5.1. Icona di Xcode

In aggiunta, l'approccio ad un nuovo linguaggio di programmazione, ovvero *SwiftUI*, ha sicuramente ampliato enormemente il bagaglio di conoscenze apprese grazie a questo progetto.

L'utilizzo di Firebase, inizialmente sconosciuto, si è rivelato davvero utile e performante. La qualità delle funzionalità messe a disposizione da esso è molto valida: sempre pronte all'uso, ben strutturate e intuitive dal punto di vista dell'implementazione.

Firebase, oltre ad aver permesso la realizzazione di moltissime funzionalità all'interno dell'applicazione, ha consentito di estendere le competenze in una branca oramai fondamentale al giorno d'oggi, ossia quella dei *backend*.

5.2 Punti di forza e di debolezza

Ogni progetto va analizzato oggettivamente per evidenziarne lati positivi, che ne costituiscono la peculiarità, e lati negativi, che, invece, vanno corretti e migliorati.

I punti di forza che caratterizzano l'applicazione e ne rappresentano un valore sono i seguenti:

- Le *funzioni* offerte sono precise e rapide, e, soprattutto, molto importanti, vista la situazione sanitaria COVID-19; poter effettuare una prenotazione da remoto aiuta a mantenere le distanze e preservare la salute rispettando le norme vigenti.
- La *grafica*; rende l'applicazione intuitiva e ben organizzata anche per un utente che si interfaccia ad essa per la prima volta.
- L'*ottimizzazione* dell'app è molto buona; l'app, infatti, richiede delle risorse hardware davvero limitate, ciò permette ad essa di esser eseguita su qualsiasi dispositivo Apple, anche i più datati.
- Il *risparmio* è notevole; questo può riguardare tanto il tempo quanto il denaro. Poter a colpo d'occhio, e senza muoversi da casa, controllare meteo e disponibilità di un ombrellone, piuttosto che percorrere molta strada per poi trovarsi difronte all'indisponibilità degli ombrelloni, oppure fronteggiare un temporale inatteso, diventa per il cliente un notevole valore aggiunto.

D'altro canto, vanno considerati i punti di debolezza riscontrati alla conclusione del progetto. Essi sono:

- La mancata possibilità di effettuare il pagamento online tramite l'app, così da eliminare del tutto qualsiasi (eventuale) contatto col personale dello stabilimento, specialmente in presenza del virus COVID-19.
- La connessione Internet richiesta per controllare e gestire i dati del database; quindi è necessaria una connessione costante per completare la maggior parte delle funzionalità.

5.3 Sviluppi futuri

L'app può avere diversi sviluppi futuri. In primis, si potrebbero aggiungere nuove funzionalità che la completerebbero. Inoltre, si potrebbe pensare di migliorare e aggiornare alcuni aspetti che sono stati trascurati nel presente progetto.

Dal punto di vista grafico, ad esempio, potrebbe essere aggiunta la *modalità notturna*, per adattare l'aspetto dell'applicazione in base a quello del dispositivo

durante le ore più buie; si potrebbe, altresì, definire un'icona dell'applicazione più dettagliata e curata di quella attuale (Figura 5.2).

Figura 5.2. Icona dell'applicazione

Una delle prime funzionalità da aggiungere, come riscontrato nei punti di debolezza, sarebbe quella del pagamento online. Questo permetterebbe un'esperienza completa senza il minimo intervento umano. Tale aggiunta assicurerebbe ancor di più il rispetto delle normative vigenti in un periodo di pandemia, nonché una rapidità maggiore nel pagamento rispetto a quello classico in presenza.

Altre ipotesi, sempre dal punto di vista funzionale, potrebbero essere l'implementazione di una chat di assistenza diretta col personale dello stabilimento, oppure, un servizio di informazione e avvisi in tempo reale tramite SMS, personalizzato per ogni utente, in merito alla prenotazione, così da risolvere, almeno in parte, il problema della continua connessione ad Internet attualmente richiesta.

6

Conclusioni

Nella presente tesi è stato descritto l'intero processo che ha portato allo sviluppo dell'applicazione Umbrella, app per dispositivi Apple per la gestione di uno stabilimento balneare.

Inizialmente è stato presentato il sistema operativo iOS, ripercorrendo tutta la sua storia ed evoluzione sin dalla prima versione. Successivamente sono stati illustrati gli strumenti necessari allo sviluppo di applicazioni per iOS, ossia l'IDE e il linguaggio di programmazione.

Nel Capitolo 2 è stata fatta un'attenta analisi dei requisiti per esaminare i comportamenti e le funzionalità che l'applicazione finale avrebbe dovuto essere in grado di garantire. Inoltre, sono stati presentati i software più rilevanti che hanno permesso lo sviluppo del progetto.

Nel Capitolo 3, invece, si è passati alla fase di progettazione, sia della componente dati che della componente applicativa. Inoltre, sono stati presentati la mappa dell'applicazione e i mockup per rendere un'idea del risultato finale atteso.

Il Capitolo 4 riguarda l'implementazione e, poi, il manuale utente. Nella prima parte viene descritta la struttura finale del progetto e, soprattutto, i listati che rappresentano il cuore delle funzionalità dell'app, adeguatamente commentati e spiegati. Nella seconda parte, invece, viene presentata una guida per permettere ad ogni tipologia di utente di sfruttare a pieno quello che l'app ha da offrire.

Nell'ultimo capitolo sono state tratte le conclusioni sottolineando, specialmente, le varie conoscenze acquisite durante tutte le fasi del progetto. Per di più, vengono analizzati punti di forza e debolezza dell'app, con lo scopo di riuscire a migliorarla colmando le carenze individuate.

Tale progetto si propone di fare da intermediario tra clienti e gestore di uno stabilimento balneare per rendere l'esperienza molto più smart e rapida, in un mondo dove ormai le app e la tecnologia stanno arrivando ovunque. Inoltre, esistono servizi analoghi per la prenotazione di ombrelloni in spiaggia, ma non c'è un'applicazione ancora ben strutturata ed affermata nel campo degli stabilimenti balneari.

Questa applicazione non è soltanto un mezzo tecnologico per rendere un lido più all'avanguardia, ma, piuttosto, è una grande opportunità per abbattere le barriere che il virus COVID-19 ha imposto. La possibilità di gestire una tale struttura da remoto è un grande aiuto tanto per il cliente, che si sente più sicuro e tutelato, quanto per il gestore, che offre un servizio adeguato alle norme anti contagio.

Riferimenti bibliografici

1. Apple's iOS through the years. https://www.gsmarena.com/a_decade_of_ios-news-30427.php, 2018.
2. Il framework SwiftUI. Cos'è? Il nuovo modo di dichiarare le interfacce in Xcode. https://www.xcoding.it/framework-swiftui-xcode/, 2019.
3. Understanding the basic structure of a SwiftUI app. https://www.hackingwithswift.com/books/ios-swiftui/understanding-the-basic-structure-of-a-swiftui-app, 2019.
4. Balsamiq Mockup. https://balsamiq.com/wireframes/desktop/, 2020.
5. Come installare e usare CocoaPods. https://www.xcoding.it/come-installare-e-usare-cocoapods/, 2020.
6. Creating and combining Views in SwiftUI. https://developer.apple.com/tutorials/swiftui/creating-and-combining-views, 2020.
7. Firebase. https://firebase.google.com, 2020.
8. Firebase: cos'è e come funziona il backend NoSQL. https://www.html.it/articoli/firebase/, 2020.
9. Human Interface Guideline for iOS. https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/, 2020.
10. iOS. https://it.wikipedia.org/wiki/IOS, 2020.
11. LLVM. https://it.wikipedia.org/wiki/LLVM, 2020.
12. SwiftUI Framework. https://developer.apple.com/documentation/swiftui/, 2020.
13. Xcode. https://it.wikipedia.org/wiki/Xcode, 2020.
14. E. Frontoni and A. Mancini. *Programmazione ad oggetti per l'Ingegneria Informatica*. McGraw-Hill Education, 2019.
15. IntroBooks. *History of Apple iOS*. Independently Published, 2020.
16. S. Gaburri L. Baresi and M. Fowler. *UML Distilled. Guida rapida al linguaggio di modellazione standard*. Pearson, 2018.
17. Matt Neuburg. *iOS 14 Programming Fundamentals with Swift: Swift, Xcode and Cocoa Basics*. O'Reilly media, 2020.
18. S. Paraboschi P. Atzeni, S. Ceri and R. Torlone. *Basi di dati. Modelli e linguaggi di interrogazione*. McGraw-Hill Companies, 2006.
19. Kragoselt Publications. *Deep Dive into Apple Xcode: Best Practices, Optimization Technique, Tips Tricks from real life Projects*. Independently Published (Kindle), 2020.
20. Yahiaoui. *Firebase Cookbook: Over 70 recipes to help you create real-time web and mobile applications with Firebase*. Packt Publishing, 2017.

Ringraziamenti

Finalmente sono giunto al termine della prima parte del mio percorso universitario. Nel corso di questi anni ho incontrato non pochi ostacoli che, per mia enorme fortuna, sono riuscito a superare grazie alla mia caparbietà, ai miei familiari che mi sono stati vicini e ai compagni di corso che hanno alleggerito tutto ciò. Ricordo molto bene le difficoltà iniziali: i primi esami e i primi scogli, l'ansia e il timore di fallire. Per mia indole non ho mai mollato la presa, i momenti di sconforto non sono mai stati abbastanza per farmi desistere, anzi rifarei tutto dal primo giorno, senza dubbi.

Vorrei, quindi, ringraziare immensamente il Professore Domenico Ursino per avermi sostenuto in questo progetto sin dal primo giorno. Trasmettere passione nel proprio insegnamento con tanta professionalità e destrezza, quanta simpatia e disponibilità, non è affatto da tutti.

Vorrei ringraziare il mio fedele compagno di studi e di corso Andrea, con il quale ho affrontato ogni sfida fin dal primo giorno di liceo.

Vorrei ringraziare davvero di cuore la mia ragazza Virginia per essermi stata sempre vicina in questo periodo impegnativo, confortandomi e spronandomi a dovere.

Ed, infine, vorrei fare un caloroso ringraziamento ai miei genitori che non mi hanno mai impedito nulla, anzi sono sempre stati al mio fianco in questi anni. Spero davvero di averli ripagati, anche un minimo, per tutto quello che mi hanno insegnato e trasmesso, rendendoli, da quest'oggi, un pizzico più fieri di me!