



UNIVERSITÀ DEGLI STUDI DI FERRARA
DIPARTIMENTO DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria
Informatica e dell'Automazione

**Progettazione e Sviluppo di un'Applicazione
Mobile di Marketing ed E-Commerce**

Tesi di laurea di:

Tommaso Berlose

Relatore

Chiar.mo Prof. Ing.

Cesare Stefanelli

Correlatore

Dott. Ing. Andrea Mazzini

Anno Accademico 2015-2016

Indice

1	Contesto e Finalità	1
1.1	Application Economy	2
1.2	Ecosistema MyGelato	4
2	Specifiche Progettuali	7
2.1	Design	8
2.2	Ricerca	10
2.3	Utente	11
2.4	Marketing Digitale	12
2.4.1	Carte Promozionali	13
2.4.2	Preferiti	14
2.5	E-Commerce	15
2.5.1	Coupon	16
2.5.2	Acquisto	17
2.5.3	Condivisione e Riscatto	18
2.5.4	Utilizzo e Validazione	19
3	Progettazione	21
3.1	Sviluppo Android	22
3.1.1	Programmazione Nativa	23
3.1.2	Android Studio	25

3.1.3	Git	26
3.2	Database	27
3.2.1	Realm	27
3.3	Network Connection	29
3.3.1	OkHttp	30
3.4	Gestore di Eventi	31
3.4.1	EventBus	32
3.5	E-Commerce	34
3.5.1	Stripe	34
4	Implementazione	37
4.1	Design	38
4.2	Network	41
4.3	Utente	43
4.4	Shop	45
4.5	Ricerca	47
4.6	Marketing Digitale	51
4.6.1	Carte Promozionali	53
4.6.2	Preferiti	55
4.7	E-Commerce	58
4.7.1	Coupon	59
4.7.2	Acquisto	60
4.7.3	Condivisione e Riscatto	63
4.7.4	Utilizzo e Validazione	65

Introduzione

Questo momento storico è caratterizzato se non forse pienamente descritto dall'esplosione dello sviluppo tecnologico che ha drasticamente cambiato i metodi di comunicazione e il modo con cui le persone accedono alle informazioni. Ed è proprio l'*Informazione* che è diventata il nodo fondamentale di ogni sistema economico moderno: tracciabilità, trasparenza e accesso ai dati relativi al singolo prodotto sono solo alcuni dei concetti che si sono diffusi grazie all'entrata di dispositivi, come gli smartphone, nella quotidianità del pubblico consumer.

La larga diffusione di queste tecnologie standardizzate e disponibili sul mercato con un target di acquirenti molto ampio, ha permesso all'utente medio di poter accedere in ogni istante e in mobilità a internet, realtà da cui può ottenere informazioni riguardanti le attività commerciali a cui è maggiormente interessato e acquistare moltissimi prodotti grazie alla diffusione di sistemi di reselling online.

Tutte le fasi di commercializzazione di un prodotto o di un servizio, sono quindi da rivalutare in considerazione di questo cambio di paradigma che ha spostato online sia la pubblicizzazione che la vendita. Queste nuove interazioni tra il produttore e il consumatore hanno però comportato la nascita di un sistema di scambio di informazioni bidirezionale molto più stretto: i sistemi di *interest-based advertising* permettono all'utente di avere consigli e pubblicità personalizzate mentre le aziende ottengono la possibilità di creare sistemi di commercializzazione efficaci e diversificati grazie alla presenza di moderne intelligenze artificiali.

Vi è quindi stato un completo rinnovamento dei meccanismi alla base del nostro sistema di mercato, che vede nel capitalismo la sua forma più evoluta ma che proprio a causa della forte ingerenza della tecnologia inizia a mostrare i primi segni di uno spostamento degli equilibri, portando l'intero sistema a una deriva difficilmente prevedibile e che si potrà assestarsi solo nel momento in cui le aziende prenderanno consapevolezza di come molti settori stiano cambiando. Sarà sempre più importante quindi tenere in considerazione l'inserimento di una componente tecnologica in ogni ambito della produzione di un'azienda, ma specialmente durante la fase di commercializzazione.

In questa tesi si tratterà la soluzione adottata dall'azienda Carpigiani, leader mondiale nel settore della vendita di macchine per gelato artigianale, nell'intraprendere lo sviluppo dell'ecosistema denominato *MyGelato*, piattaforma di marketing digitale e di e-commerce fruibile tramite applicazione mobile con l'obiettivo di dare nuovi strumenti ai propri clienti. Si presenta il design e l'implementazione dell'applicazione mobile legata a questo progetto e distribuita per il sistema operativo Google Android. Le tecnologie scelte e la generalità della soluzione consentono di impiegare gli stessi principi per altri progetti con architetture analoghe.

La stesura di questo elaborato segue logicamente i passaggi fondamentali che si presentano durante lo sviluppo di questo tipo di sistemi, partendo dall'introduzione del contesto in cui si inserisce il lavoro in oggetto, insieme alla valutazione delle finalità implicite ed esplicite che hanno portato l'azienda Carpigiani a dare vita a questo ecosistema.

Nel primo capitolo si passa alla presentazione e alla valutazione delle specifiche richieste ponendo maggiore attenzione ai sistemi di marketing digitale ed e-commerce che rappresentano il fulcro dell'applicativo, uniti ovviamente ai concetti principali che formano i modelli della piattaforma e che devono essere considerati nel dettaglio per capire alcune scelte strutturali effettuate durante lo sviluppo.

Sono poi introdotte le principali tecnologie, librerie e metodi di sviluppo utiliz-

zati durante l'implementazione di ogni singolo componente dando ampio spazio alle funzionalità rese disponibili dall'utilizzo di ottime librerie open source, descrivendo come hanno potuto semplificare e velocizzare il processo lavorativo fornendo un forte imprinting modulare all'architettura di tutta l'applicazione.

Si prosegue con la descrizione del lavoro svolto, discernendo ogni singolo componente dell'applicativo così da poter spiegare nel dettaglio le scelte implementative che hanno portato al prodotto finale di questa tesi che, dopo una serie di test funzionali e prestazionali, potrà essere pubblicato e reso disponibile sui principali mezzi di distribuzione di applicazioni Android. Alcune considerazioni di carattere generale e possibili sviluppi futuri concludono, infine, la tesi.

Capitolo 1

Contesto e Finalità

Per comprendere nella sua complessità il progetto elaborato in questa tesi è necessario tenere conto, prima di tutto, del contesto in cui viene sviluppato e delle finalità implicite ed esplicite che si vogliono raggiungere. Si tratta di descrivere lo sviluppo di un applicativo mobile fortemente legato ai sistemi di marketing e alle nuove strategie di business di aziende leader all'interno dei propri mercati, in molti casi saturi o fortemente instabili, scossi dalle innovazioni tecnologiche di questi decenni.

Oltre alle ingerenze esterne sul progetto, come il cambiamento repentino delle tecnologie o la presenza di eventuali competitor, deve essere valutata la posizione del singolo componente all'interno dello sviluppo dell'interno progetto, la quale va tenuta in forte considerazione specialmente durante la prima fase di sviluppo in cui si determinano le prime linee guida da seguire durante tutto il lavoro.

Questo elaborato rientra nel concetto di *Application Economy*, che descrive perfettamente il trend degli ultimi anni. Il cambiamento dei metodi con cui le masse ottengono informazioni e si lasciano influenzare hanno portato a un sostanziale sconvolgimento del sistema di marketing e delle strategie commerciali anche di aziende multinazionali, spostando l'interesse sulla ricerca nel campo delle applicazioni mobile.

Proprio in questo contesto è necessario inquadrare le motivazioni che hanno por-

tato Carpigiani, azienda leader nel settore della produzione di macchine per il gelato, a dare vita all'ecosistema MyGelato, di cui questa tesi sviluppa solo una componente, valutando sia le finalità in ambito di marketing sia in ambito commerciale e produttivo.

1.1 Application Economy



Figura 1.1: Application Economy

Non vi è forse modo di descrivere la società attuale e questo periodo storico senza valutare l'importanza dello sviluppo tecnologico che ci ha portato in quella che viene definita l'*Era dell'Informazione*. La rivoluzione tecnologica che sta avvenendo in questi anni, specialmente a partire dagli anni Novanta, ha portato la connessione globale a Internet ad assumere un ruolo essenziale in ogni aspetto della società moderna e della nostra vita.

È cambiato drasticamente il modo con cui le persone accedono alle informazioni, che siano queste di tipo personale o di tipo commerciale. Allo stesso modo si sono dovute adeguare le strategie di tutte quelle aziende che hanno visto cambiare in

maniera drastica il proprio mercato, invaso molte volte da tecnologie sempre più diversificate e innovative.

Secondo il giornalista Paul Mason, è stata la dottrina economica degli ultimi decenni della nostra epoca che da una parte ha avuto il merito di promuovere la più grande ondata di sviluppo economico che il mondo abbia mai visto, ma dall'altra ha portato a mercati incontrollati e a vorticosi cambiamenti sociali innescati dalla tecnologia. Si sono diffusi, infatti, concetti come i progetti open-source, la sharing economy e le licenze creative commons che hanno messo in crisi le fondamenta del capitalismo odierno. Questo fenomeno unito alla saturazione di molti mercati ha portando tante aziende a dover rivedere la propria business strategy obbligandole a dirigere i propri investimenti sulla diversificazione e sulle nuove strategie di vendita.

[12]

C'è stato un cambio di paradigma per tutti i meccanismi di commercializzazione di un prodotto: sono diventati di fondamentale importanza i concetti di *e-commerce* e *marketing digitale*, fortemente spinti dalle tecnologie e dalla nuova possibilità di accedere a una risorsa comune (Internet) da parte della maggior parte delle persone anche di cultura, età e ambienti sociali diversi. L'utente medio, per esempio, viene ormai maggiormente raggiunto e coinvolto tramite email pubblicitarie o advertising online rispetto a meccanismi tradizionali ormai meno incisivi di marketing non digitale.

Legato alla diffusione sempre crescente di smartphone, che sono di fatto diventati l'accesso principale degli utenti a Internet, nasce quindi l'*Application Economy*: lo sviluppo e l'utilizzo di applicazioni mobile per raggiungere e dialogare tramite un collegamento bidirezionale con i consumatori. Tramite un applicativo mobile è quindi possibile svolgere una serie di azioni online in mobilità e con semplicità; potendo confidare, nella maggior parte dei casi, in un'alta sicurezza della propria connessione e nello scambio dei dati.

Se questa strategia può essere applicata, per esempio, in ambito marketing per pubblicizzare un proprio prodotto e fidelizzare il consumatore alla propria azienda, è forse più incisivo osservare come l'e-commerce abbia ormai soppiantato la commercializzazione tradizionale in molti ambiti, specialmente tramite applicativi mobile (una statistica del BI Intelligence riporta che entro il 2020 il commercio mobile supererà il 45% degli acquisti online complessivi). L'abbattimento dei costi di un rivenditore non fisico e la nascita di servizi di pagamento online come PayPal hanno portato questa rivoluzione in ogni ambito commerciale permettendo inoltre all'utente di pagare direttamente tramite sistemi bancari online in sicurezza e con semplicità.

Tutti questi concetti sono ormai più che affermati in ambiti come la vendita di prodotti di abbigliamento o tecnologici, ma l'azienda Carpigiani ha scelto di portare questi nuovi meccanismi anche all'interno della vendita di gelati, in modo innovativo e fortemente avanti nei tempi grazie al progetto *MyGelato*.

1.2 Ecosistema MyGelato

L'ecosistema MyGelato si pone all'interno del contesto descritto nel capitolo precedente, a unire le strategie di marketing dell'azienda produttrice Carpigiani e delle singole gelaterie convenzionate: una piattaforma web e mobile che si rivolge ai gelatieri e ai consumatori di gelato. Gli obiettivi commerciali sono diversi, primi tra i quali quello di incentivare la compravendita di gelati tramite un sistema di coupon digitali e quello di promuovere il consumo di gelato tramite offerte, fornendo anche ai gelatieri un sistema semplice ed efficace per pubblicizzare il proprio negozio.

Attraverso l'applicativo, l'utente può informarsi sulle gelaterie presenti in zona verificando anche quali facciano parte del circuito MyGelato: insieme delle gelaterie convenzionate alla vendita e alla validazione dei coupon online. È possibile ottenere

alcune informazioni utili, come indirizzo e numero di telefono, salvarle nei preferiti e verificare se vi sono promozioni attive. Sono strategie ovviamente di carattere pubblicitario e permettono anche a gelaterie minori, facenti però parte del circuito, di essere pubblicizzate agli utenti che utilizzano la piattaforma.

Questo sistema permette all'azienda Carpigiani di avere un controllo maggiore su un sistema centralizzato di advertasing per le gelaterie di cui è principale fornitore, ottenendo quindi la possibilità di intervenire su realtà locali standardizzandone alcuni meccanismi.

Seconda feature fondamentale dell'applicazione, che riguarda in questo caso solo gli utenti iscritti al sistema, è la possibilità di comprare online dei coupon digitali che permettono l'acquisto di un gelato presso una determinata gelateria. Questi coupon sono visualizzabili in ogni momento e possono essere regalati anche ad altri utenti tramite un semplice metodo di condivisione che permette al destinatario di usufruire dell'oggetto comprato. Ogni gelateria che supporta questo sistema sarà provvista della stessa applicazione con un account specifico per il gelatiere che permetterà di validare i coupon utilizzati e ricevere il proprio compenso tramite il sistema di pagamento online.

Questo sistema di e-commerce sposta la vendita di un prodotto molto semplice come i gelati, online. È infatti possibile comprarli tramite carta di credito e l'azienda Carpigiani, nel frattempo, si inserisce in un meccanismo legato a delle realtà locali sul quale altrimenti non riuscirebbe ad avere informazioni. Questo sistema, per quanto diffuso in altri mercati come il reselling online di oggettistica è fortemente innovativo in questo campo e rende l'ecosistema MyGelato uno dei primi nel suo settore.

Capitolo 2

Specifiche Progettuali

L’ecosistema MyGelato è un progetto ampio che si basa sulla cooperazione tra più elementi fondamentali. Alla base troviamo un backend basato sul framework Ruby on Rails che implementa le funzionalità di creazione, gestione e fruizione di contenuti multimediali affiancate da un sistema e-commerce per la compravendita di coupon digitali. Per poter usufruire di tali servizi è necessario l’utilizzo di un’applicazione mobile, chiamata *MyGelato*, disponibile per i sistemi operativi mobile Apple iOS e Google Android; quest’ultima argomento di questa tesi.

Essendo l’applicazione Android solo una componente di un più ampio progetto è essenziale fin dalla prima fase di sviluppo valutare correttamente e nella loro completezza tutte le specifiche date dall’azienda che ha commissionato il lavoro e tutte quelle date dalla necessità di far cooperare l’applicazione con le altre parti del sistema. Si valutano inizialmente le specifiche strutturali e logiche che determinano le interazioni principali dell’utente con l’applicativo: navigazione, design e flussi logici. Questi macro argomenti sono presenti in ogni singolo componente del sistema e per questo devono essere valutati prima di iniziare qualsiasi tipo di sviluppo: serve considerare le interazioni tra i flussi logici e valutare come mantenere coerente il design all’interno di tutta l’interfaccia utente.

Fatte queste considerazioni si valuta come implementare una navigazione che permetta all’utente la scelta delle principali funzioni disponibili all’interno dell’applicazione: *Acquisto/Utilizzo Coupon*, *Lista Gelaterie Preferite*, *Mappa per la Ricerca*, *Mastro Gelatiere* e *Cambio Tema*. Ogni sezione fa parte dei due principali flussi logici presenti all’interno dell’applicativo: quello di marketing digitale che permette l’esplorazione da parte dell’utente delle gelaterie in una determinata zona geografica e delle relative promozioni; e quello di e-commerce che permette l’acquisto online, la gestione, il regalo e l’utilizzo dei coupon gelato resi disponibili dalle gelaterie del circuito MyGelato.

Si valutano infine le specifiche tecnologiche richieste sia dall’azienda che dalla necessaria cooperazione con gli altri elementi del progetto. Vi deve essere la possibilità di avere un’applicazione multilingua, con autenticazione tramite social network e compatibile con la maggior parte dei device Android attualmente sul mercato. Sono di forte impatto anche alcune scelte a livello di comunicazione con il backend che indirizza lo sviluppo in una comunicazione bidirezionale tra applicazione e server tramite API REST e notifiche push. Essenziale poi mantenere la coerenza tra le due applicazioni mobile sviluppate per le piattaforme iOS e Android, rispettando in ognuna i propri pattern tipici.

2.1 Design

Il design dell’applicazione è uno degli elementi di maggior risalto: sviluppato fin nel dettaglio permette di personalizzare anche i componenti più basilari della *User Interface (UI)* come icone, testi e bottoni. Sono presenti tre font custom che devono essere utilizzati in situazioni e contesti differenti in modo che possa capire fin dal primo colpo d’occhio quali siano i messaggi informativi e quali siano quelli di contorno.

Altro elemento fondamentale presente in quasi ogni schermata è il tema dell'applicazione. I temi sono principalmente delle colorazioni differenti che devono caratterizzare e modificare il *main color* della UI cambiando anche i testi e le icone in base al fatto di essere temi chiari o scuri fornendo rispettivamente testi e icone neri o bianchi. Ogni tema, che deve essere selezionabile a piacimento dall'utente direttamente da una funzione raggiungibile dalla navigazione principale, ha come nome un gusto differente di gelato e un proprio set di risorse.

Le risorse grafiche rese disponibili all'interno delle specifiche richieste servono a rendere coerente l'interfaccia utente in ogni suo elemento: fin dall'icona per chiudere una sezione a concludere con ogni elemento della navigazione principale. Il fatto di avere un'insieme di risorse ben organizzate è necessario, così da includere di volta in volta in maniera dinamica immagini in base alla lingua dell'applicazione e al tema chiaro/scuro.

In figura 2.1 è possibile vedere alcune delle immagini presentate per la navigazione principale e alcune icone in tutte le versioni disponibili rispettivamente per un tema chiaro ed uno scuro.



Figura 2.1: Risorse Design

2.2 Ricerca

L'elemento centrale di tutta l'applicazione è la possibilità di effettuare una ricerca delle gelaterie, iscritte al circuito MyGelato, presenti in una determinata zona geografica così da ottenere alcune informazioni essenziali sull'esercizio e le relative promozioni disponibili. Questa funzionalità è infatti l'elemento cardine che lega ogni possibile azione dell'utente all'interno dell'applicazione: è essenziale sia per il sistema di marketing che per il sistema di e-commerce.

Ogni utente, anche se non registrato, deve poter accedere a una mappa che visualizzi tutti gli shop disponibili, sia sotto forma di marker visibili in base al luogo di ricerca sia sotto forma di lista ordinata in base alla distanza dall'utente. Ogni elemento, scaricato e aggiornato in base alle ultime informazioni presenti sul server, permette di ottenere il nome della gelateria, l'indirizzo e un eventuale recapito telefonico.

Questo tipo di ricerca permette all'utente di esplorare la zona attorno al luogo in cui trova, o anche ad un luogo di suo interesse, così da ottenere informazioni, sia di carattere commerciale sia pubblicitario, riguardo agli esercizi presenti in maniera molto diretta; permettendo anche di salvare lo shop all'interno della lista preferiti. Nel frattempo ogni gelateria presente all'interno del sistema ottiene la possibilità di essere trovata anche da utenti nuovi che non conoscono la zona e di pubblicizzarsi tramite una strategia di marketing unificata e standardizzata: le *carte promozionali*.

Oltre al sistema di marketing la ricerca delle gelaterie è essenziale per il sistema di e-commerce poichè la mappa viene utilizzata per la scelta da parte dell'utente della gelateria per la quale vuole acquistare un determinato coupon. La ricerca è quindi presente sia nel sistema di navigazione principale dell'applicazione sia raggiungibile all'interno di ogni flusso logico scelto dall'utente e come si può vedere in figura il mockup della mappa permette di capire la necessità di avere un'implementazione semplice e facilmente utilizzabile da qualsiasi utente, dove sia possibile

passare dalla visualizzazione a mappa a quella a lista e dove siano già visibili le principali informazioni riguardo a ogni gelateria.

È possibile osservare in figura 2.2 un mockup esplicativo per la schermata della ricerca tramite mappa con dettaglio sulla selezione di un marker informativo.

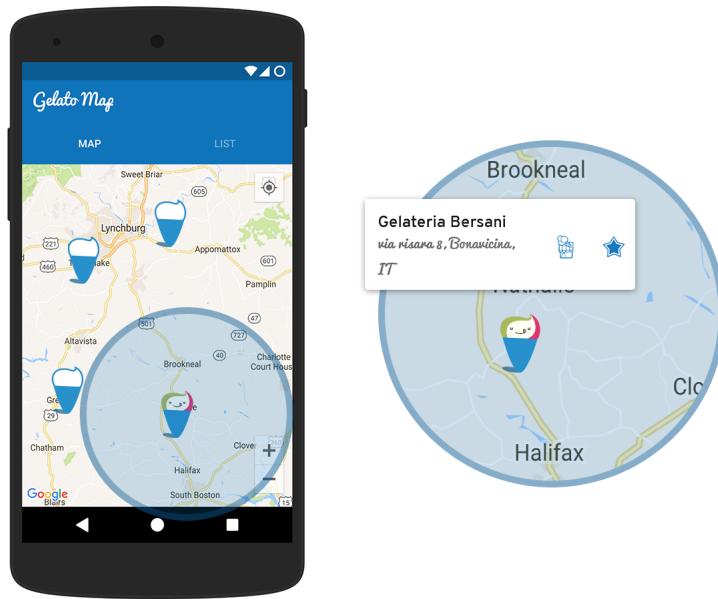


Figura 2.2: Mockup Mappa

2.3 Utente

La piattaforma MyGelato per poter offrire alcune funzionalità personalizzate al singolo utente implementa un sistema di registrazione e di autenticazione. Questa scelta progettuale è ovviamente fondamentale da considerare durante lo sviluppo dell'applicazione mobile poichè si deve considerare un'intera sezione che dia la possibilità all'utente di registrarsi, modificare parte delle proprie informazioni salvate sul server ed eseguire il login sulla piattaforma.

Sarà necessario poter accedere direttamente dalla navigazione principale alle informazioni riguardanti il proprio stato di accesso alla piattaforma, verificando se si è già autenticati o meno. Le funzionalità che devono essere più facilmente accessibili sono quindi quelle di login e di logout, in modo che l'utente in pochi passaggi possa accedere o rimuovere i propri dati dal dispositivo utilizzato.

I nuovi utenti potranno utilizzare un form per iscriversi alla piattaforma MyGelato inserendo solo alcune informazioni essenziali come nome, cognome, email e password. Le funzionalità di registrazione si basano principalmente sull'inserimento manuale di un indirizzo di posta elettronica come metodo identificativo dell'utente, ma deve essere possibile anche eseguire la registrazione al servizio tramite social network. Il sistema prenderà in automatico i dati di cui necessita per inserire l'utente all'interno del proprio database e ogni volta l'accesso all'applicazione sarà legato al login sui social network.

All'interno della schermata di login, nel caso vi sia stata una registrazione manuale, deve essere possibile accedere ad una sezione che permetta di ricevere nuovamente la password legata all'account sulla email inserita durante la registrazione. Infine i dati inseriti che non vengono utilizzati per identificare l'utente potranno essere modificati una volta effettuato l'accesso direttamente dalla sezione utente.

2.4 Marketing Digitale

Uno dei due *workflow* principali presenti all'interno dell'applicazione riguarda il marketing digitale che viene svolto per le gelaterie legate all'ecosistema MyGelato. Ha lo scopo di rendere più facile la ricerca delle gelaterie, sponsorizzarne eventuali promozioni e permettere a ogni utente di salvare gli esercizi preferiti così da rimanere aggiornato sulle nuove promozioni disponibili.

Queste funzionalità, disponibili in buona parte per qualsiasi utente anche non registrato, seguono un flusso logico che parte dalla ricerca degli shop e trova effetto principale nella scoperta e nell'utilizzo delle carte promozionali, della singola gelateria e del Mastro Gelatiere.

2.4.1 Carte Promozionali

Le carte promozionali sono il principale mezzo di advertising all'interno dell'applicazione: sono dei volantini digitali formati da un'immagine o un video, un titolo, un testo descrittivo, un eventuale recapito telefonico e un link per ottenere maggiori informazioni riguardo alla promozione in oggetto. Ci sono due tipologie di carte: le carte specifiche di ogni esercizio, ovvero i volantini informativi della singola gelateria, e le carte del mastro gelatiere che invece si rifanno alle promozioni generiche proposte per l'intero sistema.

Le carte della singola gelateria altro non sono che le ultime promozioni legate all'attività: sconti, novità, messaggi informativi. L'utente in questo modo può facilmente reperire le informazioni pubblicitarie di una determinata gelateria online e in mobilità; considerando che difficilmente in questo campo vi è già una diffusa pubblicizzazione sui social network o sui sistemi di advertising. Ogni shop iscritto al circuito MyGelato ha quindi la possibilità di ottenere una maggior copertura pubblicitaria tramite l'utilizzo di un sistema centralizzato e standardizzato.

Le carte di uno stesso esercizio devono essere raggruppate in modo che l'utente possa scorrerle e visualizzare tutte le promozioni o informazioni disponibili insieme. La schermata per questa visualizzazione deve essere raggiungibile ogni volta che sono mostrate le informazioni della gelateria: sia internamente alla ricerca che nella lista dei preferiti dell'utente.

Oltre alle singole gelaterie l'ecosistema MyGelato prevede anche le carte del Mastro Gelaterie che comprendono promozioni, informazioni e novità pubblicate

genericamente dagli amministratori del sistema e che sono valide per tutti gli esercizi facenti parte del circuito. Essendo un insieme di carte totalmente generiche, quindi slegate da qualsiasi flusso logico, e di fondamentale importanza per il sistema di advertising creato, questa sezione deve essere inserita all'interno della navigazione principale del sistema.

Infine, come avverrà poi per le gelaterie preferite, ogni utente registrato dovrà avere la possibilità di essere notificato di eventuali nuove promozioni nel momento stesso in cui diventeranno disponibili.

2.4.2 Preferiti

La gestione delle proprie gelaterie preferite è uno degli elementi presenti all'interno della navigazione principale dell'applicazione, accessibile da qualsiasi utente anche non registrato: permette di salvare sul proprio dispositivo gli esercizi di maggiore interesse.

Questo dà modo all'utente di poter accedere in ogni momento, specialmente in mobilità, alle informazioni che più gli interessano di ogni gelateria: indirizzo, recapito telefonico ed eventuali promozioni. La ricerca in questo caso è molto semplificata poichè viene presentata una sezione a parte con una lista degli shops preferiti, senza dover obbligare l'utente a effettuare una nuova ricerca all'interno della mappa. Il salvataggio all'interno dei preferiti avviene direttamente all'interno della ricerca, sia tramite la mappa che tramite la lista grazie a un'icona esemplificativa.

Questa funzionalità è pensata principalmente per fidelizzare il consumatore: ogni utente avendo la possibilità di salvare una gelateria ha anche la possibilità di ottenere velocemente informazioni su nuove promozioni, trovare velocemente i contatti dell'esercizio come se li avesse salvati in rubrica e valutare ogni volta la distanza tra sé e lo shop.

Il passo successivo in questo senso è dato dal rendere bidirezionale questo collegamento, rendendo in alcuni casi non necessaria la ricerca da parte dell'utente di nuove informazioni fornendogliele invece a ogni aggiornamento. Per fare questo, solo nel caso di utenti registrati, l'aggiunta di uno shop ai preferiti include le funzionalità di geofencing e notifiche push, quest'ultima presente anche per le carte promozionali del mastro gelatiere.

Il geofencing permette di ricevere una notifica ogni qualvolta l'utente si trovi a meno di 5 km da una delle proprie gelaterie, così da essere informato di essere vicino in termini di localizzazione. Le notifiche push invece sono attivate per avvertire un utente che una delle proprie gelaterie ha pubblicato una nuova promozione tramite l'aggiunta una carta sul sistema MyGelato. L'utente così rimane informato costantemente delle ultime promozioni disponibili e il proprietario di una gelateria ha la certezza di effettuare pubblicità diretta tra se e i suoi clienti più affezionati.

2.5 E-Commerce

Il secondo flusso logico presente all'interno dell'applicazione riguarda l'acquisto e l'utilizzo di coupon gelato online e in completa mobilità. Questo sistema già più che diffuso in tantissimi altri ambiti della vendita online è uno dei cardini su cui maggiormente si vuole puntare sia per funzionalità sia per innovazione.

I *Coupon* gelato sono buoni acquisto che si possono acquistare direttamente tramite l'applicazione e permettono a chiunque ne sia virtualmente in possesso di utilizzarli nelle gelaterie che li hanno rilasciati tramite validazione. L'acquisto dei buoni deve poter essere fatto in qualsiasi momento utilizzando le ultime tecnologie disponibili per i pagamenti online; essenziale mantenere un occhio di riguardo alla sicurezza di questo tipo di transizioni.

Tra le specifiche risulta essere presente anche il concetto di condivisione dei cou-

pon tramite un sistema di *share/redeem*: un utente che ha acquistato un coupon ha la possibilità di condividerlo con un altro utente registrato alla piattaforma che può riscuotere poi il buono. La condivisione deve avvenire tramite applicazioni e canali di comunicazioni facilmente utilizzabili all'interno di uno smartphone.

Infine vi deve essere ovviamente la possibilità di validare un coupon una volta che si decide di utilizzarlo all'interno della gelateria che lo ha rilasciato online tramite il circuito MyGelato. Questo procedimento prevede due entità in gioco, colui che possiede il coupon e il gelataio che deve poterlo validare; in entrambi i casi le funzionalità da implementare dovranno essere presenti all'interno dell'applicazione in base al tipo di account utilizzato così da mantenere coerenza negli strumenti utilizzati per collegarsi alla piattaforma.

2.5.1 Coupon

I Coupon sono buoni per l'acquisto di un bene materiale, normalmente gelati, che si può ritirare in qualsiasi momento in sede all'esercizio che ha effettuato la vendita. Hanno un nome e un valore, il prezzo, scelti durante l'inserimento tramite il sistema amministrativo gestionale che può inserire anche informazioni aggiuntive come la scadenza e la valuta. Sono generici per tutto l'ecosistema e sono quindi disponibili per ogni gelateria facente parte del circuito MyGelato anche se all'acquisto sarà necessario specificare l'esercizio nel quale si desidereranno poi utilizzare.

All'interno della navigazione principale dell'applicazione sarà necessario avere una sezione apposita per poter gestire tutti i propri coupon: una lista di quelli utilizzati, quelli regalati ad altri utenti e quelli che si possono ancora utilizzare personalmente. Questa sezione sarà disponibile offline ma dovrà ogni volta essere sincronizzata con il server in modo da avere coerenza anche utilizzando la stessa applicazione con lo stesso account da dispositivi differenti.

Oltre alla possibilità di acquistare coupon si dovrà poter accedere direttamente alle funzioni di condivisione, riscatto e utilizzo così da avere una gestione centralizzata di tutto il flusso logico legato all'e-commerce.

2.5.2 Acquisto

L'acquisto di un coupon è permesso a qualsiasi utente che si sia registrato, tramite mail o social network, al circuito MyGelato. Questa funzionalità è raggiungibile direttamente parte della navigazione principale dell'applicazione, anche se deve essere disponibile solo nel caso in cui l'utente abbia eseguito il login, altrimenti dovranno essere richieste le credenziali di accesso.

Le operazioni richieste per l'acquisto di un coupon dovranno essere tutte racchiuse all'interno di una stessa schermata in cui l'utente verrà guidato attraverso le varie fasi fino all'avvenuta transizione. Il primo passo da svolgere è la scelta della gelateria su cui eseguire l'acquisto, come spiegato precedentemente si andrà a sfruttare la mappa e la lista per la ricerca degli shop già utilizzata all'interno di altre funzionalità. In questo caso però verranno visualizzati solo le gelaterie che permettono l'acquisto di coupon e quando verranno selezionati i marker non si otterranno tutte le informazioni sull'esercizio ma sarà presente un'icona per selezionare lo shop.

Una volta selezionata la gelateria dovranno essere scaricati dal server i coupon disponibili e nel frattempo verrà resa disponibile la scelta del metodo di pagamento che si appoggerà ad un sistema di pagamento online esterno. Si potrà scegliere tramite una lista di metodi di pagamento quello che si preferirà usare che sarà successivamente impostato come predefinito.

Scelto il metodo di pagamento, scorrendo la lista dei coupon disponibili ed decidendo il valore che si vuole acquistare si potrà infine scegliere di completare l'acquisto che in caso di successo dovrà poi riportare l'utente sulla sezione di gestione dei coupon e altrimenti presentare un messaggio di errore.

2.5.3 Condivisione e Riscatto

Le funzionalità di condivisione e riscatto che si vogliono implementare all'interno dell'applicazione rendono importante capire come far dialogare due dispositivi differenti in modo che abbiano informazioni coerenti uno con l'altro e in modo che sia semplice l'utilizzo da parte di utenti non esperti. Per rendere questa funzionalità accessibile da chiunque si dovrà legare ogni coupon grazie ad un codice alfanumerico di sei cifre detto PNR (*Product Nr.*) che potrà quindi essere condiviso tramite qualsiasi mezzo disponibile.

L'uso di un codice inviabile facilmente come teso permette di sfruttare altri applicativi presenti sul dispositivo dell'utente per la condivisione tra persone differenti. Scegliendo il coupon da regalare si dovrà quindi chiedere all'utente tramite quale canale di comunicazione preferisce inviare il codice PNR per poi inserire il codice in un testo promozionale legato ad un link di reindirizzamento a una pagina web della piattaforma MyGelato.

L'inserimento del link a una pagina web della piattaforma accessibile online da qualsiasi dispositivo è una funzionalità di grande potenza: nel caso in cui la pagina web sia raggiunta da un dispositivo che ha installato sopra l'applicazione MyGelato aprirà direttamente l'applicativo stesso gestendo internamente il codice presente; altrimenti verrà presentata una pagina web che suggerisce all'utente di scaricare l'applicazione dai principali store mobile.

Valutato come scegliere di condividere un coupon è necessario inserire una funzione all'interno dell'applicazione che ne permetta il riscatto tramite il solo inserimento del codice PNR. Raggiungibile dalla navigazione principale vi sarà una sezione che permetta di eseguire questa operazione in maniera molto semplice e che, nel caso venga avviata l'applicazione tramite la gestione del link web di cui si è parlato precedentemente sarà compilata in automatico.

Tramite un semplice controllo sul server verrà quindi controllato se l'utente può riscattare il codice e le informazioni saranno aggiornate per entrambi gli utenti: chi ha regalato il coupon e chi invece può utilizzarlo.

2.5.4 Utilizzo e Validazione

L'ultimo passo all'interno del flusso logico di questo sistema di e-commerce è ovviamente l'utilizzo e la validazione di un coupon per acquistare un bene materiale direttamente in sede all'esercizio che ha venduto il buono. Per semplificare l'utilizzo della piattaforma MyGelato ci si è rifatti nuovamente al fatto che gli smartphone siano diventati dei beni di uso comune e che quindi anche i proprietari delle gelaterie possano utilizzare l'applicazione MyGelato come mezzo per la validazione dei coupon.

Allo stesso modo che per le funzionalità di condivisione e riscatto si è scelto di utilizzare i codici PNR dei coupon come mezzo per far dialogare i due dispositivi di chi vuole utilizzare un buono e di chi deve verificare che sia valido. Rispetto ad un sistema di condivisione che prevede la presenza di un canale di comunicazione personale tra i due utenti si dovrà valutare l'utilizzo di mezzi più semplici e slegati.

Si è quindi proposto di generare un QR Code a partire dal codice PNR di ogni coupon, raggiungibile direttamente cliccando sul buono presente nella lista di gestione interna all'applicazione. Il consumatore dovrà quindi mostrare il QR Code al proprietario dell'esercizio che, utilizzando la stessa applicazione MyGelato con un account di tipo *gelatiere*, utilizzerà una funzione di validazione tramite lettura del codice mostrato. Sarà infine necessaria una connessione al server da parte di entrambi i dispositivi per verificare l'avvenuta validazione del coupon così da procedere all'acquisto materiale del bene rappresentato dal buono.

Capitolo 3

Progettazione

Definiti gli obiettivi progettuali dell'applicazione si sono valutati gli strumenti di sviluppo da utilizzare durante lo svolgimento della tesi: come parametri si è tenuto conto di tempistiche di aggiornamento, adeguamento alle linee guida del sistema operativo in oggetto, documentazione disponibile e modernità delle tecnologie utilizzate.

Durante la prima fase di sviluppo strutturale si sono valutate le tecnologie da utilizzare in modo da tenere il passo con le ultime specifiche di Android e tenendo conto che, anche secondo gli ultimi report pubblicati da Google, rimane una forte frammentazione della distribuzione del sistema operativo, dovuta ai molteplici vendor. [8] È quindi da tenere in considerazione la retrocompatibilità delle librerie utilizzate, avendo scelto di supportare fino alla versione 16 del SDK di Android (versione 4.1 Jelly Bean), e la possibilità di utilizzare l'applicazione anche su dispositivi con schermi e hardware differenti.

Considerato infine che alcuni nodi centrali dell'elaborato necessitano di specifiche minime in termini di affidabilità e sicurezza si è scelto di utilizzare librerie esterne anche per le funzioni di utilizzo del database e di connessione al server, rispettiva-

mente Realm e OkHttp, che oltre a semplificare lo sviluppo garantiscono un'alta gestione degli errori.

3.1 Sviluppo Android

Questa tesi si inserisce all'interno del lavoro di creazione della piattaforma MyGelato, il quale include il supporto sia alla piattaforma Android sia alla piattaforma iOS. Essendosi però creati due differenti progetti durante la prima fase di progettazione, si è potuto scegliere come metodo implementativo l'utilizzo della programmazione nativa Android che si basa su Java per la programmazione e su XML per la creazione delle risorse.

Sicuramente lavorare in questo modo ha i propri pro e contro: mentre da un lato il nativo offre la possibilità di una gestione totale del dispositivo senza la paura di trovare limiti, d'altra parte richiede spesso una programmazione molto professionale e si concentra esclusivamente su una piattaforma impedendo un'agile riciclo dei propri sforzi su altri mercati del mobile. Il non-nativo, invece, offre diversi vantaggi ascrivibili alla necessità di creare applicazioni cross-platform distribuibili su sistemi operativi diversi. [4] Non avendo quindi necessità di mantenere alta la portabilità del codice su altre piattaforme, si è preferito sviluppare tramite programmazione nativa; potendo quindi sfruttare pienamente l'architettura del sistema operativo sottostante e gli strumenti di sviluppo forniti ufficialmente.

Si è scelto quindi di utilizzare come principale strumento *Android Studio*, Integrated Development Environment (IDE) dedicato alla programmazione Android distribuito da JetBrains e Google che è ormai il miglior software per la creazione dei applicazioni su questa piattaforma. [9]

Durante lo sviluppo del progetto si è utilizzato come sistema di versioning il software **Git**, così da poter mantenere uno storico del lavoro svolto e ottenendo tutti

i vantaggi dell'utilizzo di un Sistema per il Controllo di Versione (*Version Control System - VCS*), mentre per il test delle chiamate al backend della piattaforma si è utilizzata la web application Postman [13] che permette di testare e sviluppare le API con cui due applicativi possono dialogare.

3.1.1 Programmazione Nativa

Le applicazioni Android sono sviluppate utilizzando il linguaggio Java. È un linguaggio di programmazione molto popolare sviluppato da Sun Microsystems (ora di proprietà di Oracle) orientato agli oggetti a tipizzazione statica, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. [21]

Per lo sviluppo di un applicativo si devono sfruttare le principali caratteristiche di Java andando a estendere, per ogni schermata che si vuole creare, la classe *Activity* implementata come standard all'interno delle librerie Android. Qualsiasi progetto sarà quindi formato da un insieme di classi che rappresenteranno logicamente ogni singola schermata unite ad altre classi che incapsuleranno invece le funzioni, i modelli e ogni altro componente standard di Android. All'interno dell'ultima distribuzione di Android sono state incluse alcune delle funzionalità di Java 8, ma l'alta frammentazione della piattaforma non ne permette correttamente l'utilizzo. Questo si ripercuote su alcune funzionalità come le chiamate http che non supportano alcune delle semplificazioni che sono state fatte ad esempio nelle connessioni sicure grazie a TLS.

Come si può vedere all'interno del listato la classe *Main* estende la classe *AppCompatActivity* e dopo aver richiamato il costruttore della propria superclasse esegue la creazione dell'interfaccia utente grazie al file di layout *activity_main* che ne specifica la struttura e le impostazioni, scritto in XML.

```

public class Main extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

Listato 3.1: AppCompatActivity

XML (sigla di eXtensible Markup Language) è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. All'interno dello sviluppo di applicazioni Android rappresenta il linguaggio per la creazione delle risorse: layout, manifest, ecc...

L'esempio più eclatante è l'utilizzo dell'XML per produrre il manifesto dell'applicazione, che ne rappresenta ed espone la struttura principale dichiarando ogni singola schermata o servizio presenti specificando anche il tema da utilizzare, il nome e alcune opzioni di avvio. Sono scritte in XML anche tutte le risorse interne all'applicazione come si può notare nell'esempio di codice sottostante che descrive una schermata con una singola immagine.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        src="drawable/icon" />
</LinearLayout>

```

Listato 3.2: Layout XML

3.1.2 Android Studio

Android Studio [18] è un ambiente di sviluppo integrato (IDE) per lo sviluppo per la piattaforma Android. È stato annunciato il 16 maggio 2013 in occasione della conferenza Google I/O e la prima build stabile fu rilasciata nel dicembre del 2014. Basato sul software della JetBrains, Android Studio è stato progettato specificamente per lo sviluppo di Android. Sostituisce gli Android Development Tools (ADT) di Eclipse, diventando l' IDE primario di Google per lo sviluppo nativo di applicazioni Android offre ancora più funzioni che migliorano la produttività durante la creazione di applicazioni Android, come ad esempio: un sistema di compilazione Gradle, un ambiente unificato dove è possibile sviluppare per tutti i dispositivi, esecuzione e costruzione di un file APK per la pubblicazione di una applicazione sui canali di distribuzione più importanti e l'integrazione GitHub.

Ogni progetto creato all'interno del software contiene alcuni moduli principali che riguardano l'applicazione vera e propria, le librerie incluse e i moduli proprietari di Google che permettono l'utilizzo dei suoi servizi principali come Firebase e le Google Maps API. [11]

A livello di build del sistema si trova uno script Gradle[2] che a partire da un manifesto, che dichiara la struttura dell'applicazione, i file Java contenenti il codice sorgente (inclusi i test) e l'insieme delle risorse e degli asset può generare un file firmato installabile su qualsiasi dispositivo.

Per la generazione delle risorse Android Studio mette a disposizione alcuni tool, tra cui alcuni grafici, che permettono l'implementazione e il test dell'interfaccia utente senza dover di volta in volta avviare l'applicativo su un emulatore, messo a disposizione dal software stesso, o su un dispositivo fisico.

Ovviamente AS incapsula tutti i tool utili alla programmazione come la formattazione, lo stile e la correzione in realtime del codice scritto effettuando il controllo sintattico e l'analisi statica seguendo i pattern standard della piattaforma.

3.1.3 Git

Git è un software di controllo versione distribuito (*Distributed Version Control Systems - DVCS*) utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005. [20] Nacque per essere un semplice strumento per facilitare lo sviluppo del kernel Linux ed è diventato uno degli strumenti di controllo versione più diffusi.

Per quanto riguarda qualsiasi tipo di progetto di IT (*Information Technology*), specialmente se si tratta di un lavoro da dover svolgere in team, Git dà la possibilità di mantenere in memoria tutte le versioni del proprio lavoro (sia in locale che online). Questo permette a più persone di poter accedere alla cronologia del lavoro condiviso, avendo anche la possibilità di verificare la presenza di errori e di eliminarli tornando a una versione precedente del progetto. Git sfrutta alcuni algoritmi avanzati per calcolare le differenze riga per riga tra i file di diverse versioni così da verificare la presenza di errori o conflitti.

Git è fortemente direzionato verso uno sviluppo progettuale non lineare. Supporta diramazione e fusione (*branching and merging*) rapide e comode, e comprende strumenti specifici per visualizzare e navigare l'intero storico delle versioni anche se proposte da sistemi differenti. È veloce e scalabile nella gestione di grandi progetti ed è molto utile nello sviluppo in team, specialmente se affiancato dal modello GitFlow che permette di gestire rami specifici per il developing o per le release.

Infine grazie all'utilizzo di piccoli accorgimenti come il file `.gitignore` (che permette di non salvare anche i file temporanei e non importanti del progetto) e soluzioni online come GitHub o BitBucket (quest'ultimo utilizzato durante lo sviluppo di questa tesi) il processo di sviluppo software viene drasticamente avvantaggiato.

3.2 Database

La piattaforma Android fornisce diversi metodi e strumenti per salvare i dati delle applicazioni in modo persistente. Per quanto riguarda le preferenze relative alle impostazioni dell'applicazione si possono implementare le *Shared Preferences* per salvare le scelte dell'utente; mentre per quanto riguarda la memorizzazione persistente di una grossa mole di dati si possono utilizzare strumenti come l'*Internal Storage* o un *Database SQLite*. [5]

Esistono però anche alternative agli strumenti standard forniti ufficialmente per la piattaforma e uno di questi è la libreria *Realm* [14]: database object-oriented che non incapsula SQLite ma lo sostituisce con un sistema di memorizzazione scritto in C++ che permette l'accesso ai dati anche tramite l'utilizzo di linguaggi di programmazione differenti. Modifica inoltre il metodo di accesso ai dati salvati non richiedendo query esplicite integrando molte delle funzionalità dei sistemi basati su l'*Object-Relational Mapping (ORM)*, semplificando di gran lunga l'utilizzo del database durante lo sviluppo di qualsiasi applicazione.

3.2.1 Realm

La prima configurazione per l'utilizzo di Realm è rintracciabile nella creazione di una semplice dipendenza all'interno dei file *Gradle* (tool di Andriod Studio per il compile, il deploy e la gestione delle dipendenze) dove va inserita l'ultima versione disponibile della libreria. È inoltre necessario inserire una prima inizializzazione all'interno dell'avvio dell'applicazione, andando quindi a estendere la classe *Application* esposta dalla libreria dell'SDK di Android.

Una volta configurato il plugin si possono creare i modelli degli oggetti che saranno da salvare all'interno della base di dati, come per esempio un oggetto *Utente* rappresentato ad esempio dalla classe *User* come si può vedere nel listato 3.3 dove

sono descritte le principali variabili che lo compongono, tra cui la chiave primaria denotata dalla keyword *PrimaryKey*.

```
import io.realm.RealmObject;
import io.realm.annotations.PrimaryKey;
public class User extends RealmObject {
    @PrimaryKey
    public String uuid;
    public String firstName;
    public String lastName;
    public String email;
    public User() {
    }
}
```

Listato 3.3: Model Realm

L'utilizzo di questi modelli è molto semplificato poichè Realm permette di creare delle transizioni sui singoli oggetti creati così da poter aggiornare la loro versione contenuta all'interno del database, assicurando in ogni istante le proprietà ACID delle operazioni. [17] Si può vedere nel seguente listato un esempio di creazione o aggiornamento di un oggetto user in base al fatto che sia o non sia già presente all'interno della base di dati.

```
Realm realm = Realm.getDefaultInstance();
User user = new User(Params...);
realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        realm.copyToRealmOrUpdate(user);
    }
});
realm.close();
```

Listato 3.4: Realm Transaction

Allo stesso modo con cui vengono semplificati gli inserimenti di dati all'interno del DB è forse fondamentale capire come vengono gestite anche le query sulla base di dati; Un esempio molto semplice visibile nel listato 3.5 mostra come sia possibile richiedere un utente con email non nulla all'interno dei dati salvati ottenendoli in un *ArrayList* utilizzabile direttamente per scorrere ogni singolo oggetto ottenuto dalla richiesta.

```
RealmResults<User> results =
    Realm.getDefaultInstance()
        .where(User.class)
        .notNull("email")
    .findAll();
```

Listato 3.5: Realm Query

Ultimo elemento da presentare della libreria Realm sono i *listener* che si possono attivare sia sull'intero database sia su una singola porzione in modo che eseguano una determinata callback nel momento in cui avviene una modifica dei dati monitorati. Un breve esempio è visualizzato nel codice seguente.

```
Realm realm = Realm.getDefaultInstance();
realm.addChangeListener(new RealmChangeListener<Realm>() {
    @Override
    public void onChange(Realm element) {
        // do something...
    }
});
```

Listato 3.6: Realm Listener

3.3 Network Connection

Attualmente ogni tipo di piattaforma che voglia permettere al proprio utente di accedere da remoto, e quindi da qualsiasi dispositivo, alle proprie informazioni ha come

parte fondamentale lo sviluppo e il mantenimento di un server di backend (quindi nascosto nelle funzionalità alla vista dell’utente) che mantenga, gestisca e restituisca i dati necessari alle funzioni degli applicativi. Nello stesso modo la piattaforma MyGelato si basa su un server che mantiene ogni informazione riguardante utenti, shop e card fornendo delle API REST che vengono sfruttate dalle applicazioni mobile e web per poter fornire i propri servizi. [6]

Il passaggio di informazioni tra i due applicativi avviene tramite chiamate internet con protocollo Http protette grazie a una connessione criptata dal Transport Layer Security (TLS) o dal suo predecessore, Secure Sockets Layer (SSL). Grazie alla API REST è possibile seguire alcuni pattern delle tecnologie che permettono di valutare ogni possibile errore delle connessioni: errori nei dati, nella connessione, ecc... In Java le chiamate http sono difficilmente gestibili poichè non sono automatizzate e sono stati effettuati dei cambiamenti nelle ultime versioni che non permettono di lavorare correttamente su tutti i dispositivi Android. Per questo motivo è stata utilizzata la libreria open-source OkHttp sviluppata da Square, che permette di automatizzare le chiamate http sfruttando anche una notevole semplificazione del codice utilizzato per programmare. [15]

3.3.1 OkHttp

La libreria OkHttp nasce dal fatto che il protocollo http è l’attuale mezzo di comunicazione per ogni applicativo moderno. Per questo si pone l’obiettivo di semplificare l’utilizzo e di migliorare le prestazioni delle tecnologie che lo sfruttano fornendo allo sviluppatore alcuni automatismi comodi durante la programmazione e l’implementazione.

Supporta ogni tipo di comunicazione http, ne gestisce la latenza, il caching, la riconnessione, l’utilizzo di protocolli sicuri come TLS, l’utilizzo di indirizzi IP multipli ed è molto semplice da integrare e utilizzare all’interno del proprio software. Grazie

a un’interfaccia molto semplice permette di automatizzare le chiamate http utilizzando dei Builder che lasciano specificare solo i parametri necessari e restituiscono un unico oggetto da cui si ottiene la risposta, positiva o negativa, della richiesta.

```
OkHttpClient client = new OkHttpClient();
Request request = new Request.Builder()
    .url(endpoint)
    .addHeader("Accept", "application/json")
    .build();

Response response = client.newCall(request).execute();
```

Listato 3.7: Network Connection

Inizializzato il client OkHttp serve creare un oggetto *Request* a cui vanno passati i parametri necessari per poi eseguire la chiamata ottenendo infine un oggetto *Response* all’interno del quale sono presenti i dati scaricati, il codice di risposta della chiamata e anche eventuali errori.

3.4 Gestore di Eventi

Un passaggio fondamentale durante lo sviluppo di un software è l’aggiornamento delle view a seguito di un cambiamento nello stato dell’applicazione. In Android è necessario intervenire direttamente sugli elementi dell’interfaccia modificando ogni oggetto in base alle specifiche richieste senza poter sfruttare veri e propri automatismi. Una forte limitazione del sistema operativo è dovuta, per esempio, al fatto che l’unico thread che ha il permesso di accedere e di modificare l’interfaccia utente è il Main Thread, cioè il processo principale sul quale viene avviata l’applicazione; questo complica l’interazione con sistemi multithreading anche se, ovviamente, fa parte delle specifiche di sistema utili a limitare i problemi durante l’accesso alle risorse condivise (in questo caso le view).

Questo problema necessita di ampia valutazione durante lo sviluppo e anche in questo caso si è dovuto strutturare l'applicazione in modo che non incorresse in errori di inconsistenza tra le view a cui il programma voleva accedere e le view realmente visualizzate al momento. Nel caso si utilizzino script asincroni è molto facile tentare di accedere a elementi, o anche intere schermate, non più presenti a video; per questo motivo si è scelto di utilizzare la libreria EventBus sviluppata da GreenRobot [1] e pensata esattamente per gestire e risolvere questo tipo di problemi.

3.4.1 EventBus

EventBus è una libreria open-source per Android che utilizza il pattern publish/subscribe per far dialogare tutti i componenti di un'applicazione. Disaccoppia le classi che interagiscono sulla stessa interfaccia e le gestisce in maniera centralizzata monitorando le performance e gestendo gli errori che potrebbero essere sollevati da uno sviluppo multithread. [3]

I principali benefici che si riscontrano tramite l'utilizzo di questa libreria sono la semplificazione delle comunicazioni tra componenti, il disaccoppiamento tra mittenti e riceventi degli eventi, miglioramenti nelle perfomance, facile integrazione dei metodi da utilizzare ed infine anche caratteristiche avanzate come priorità e indirizzamento a thread specifici.

Durante lo svolgimento di questa tesi si è definita una struttura abbastanza comune formata da:

- AsyncTask, oggetto che permette di elaborare una serie di informazioni in background, utilizzato per eseguire il download delle informazioni dal server;
- View specifica dell'activity con determinati componenti da aggiornare in base allo stato dell'applicazione;

- Metodo `updateUI()` che presi in ingresso i dati scaricati doveva aggiornare le view rispetto allo stato.

Per far dialogare questi elementi si è utilizzato EventBus così da sfruttare il lavoro di thread in background senza limitare l’utente nell’attesa di un determinato evento. Si procedeva quindi creando un evento specifico per l’interazione da dover gestire di cui qui viene riportato un esempio:

```
public static class UpdateCouponUiEvent<T> {
    public T data;
    public UpdateCouponUiEvent(T data) {
        this.data = data;
    }
}
```

Listato 3.8: Event Class

EventBus permette di registrare dei metodi eseguendo una subscribe a un determinato evento: ogni qual volta un evento di quel tipo viene sollevato, il metodo viene eseguito. La subscribe deve essere definita nel codice nel seguente modo:

```
@Subscribe
public void updateUI(UpdateCouponUiEvent event) {
    // update UI with event.data
}
```

Listato 3.9: EventBus Subscribe

Grazie all’utilizzo di eventi custom è dunque possibile passare anche dei dati tramite l’evento che viene lanciato all’interno di un AsyncTask, a conclusione del download delle informazioni e dell’aggiornamento dello stato, grazie a una publish:

```
EventBus.getDefault().post(new UpdateCouponUiEvent());
```

Listato 3.10: EventBus Update

3.5 E-Commerce

Ogni sistema di e-commerce presente sugli applicativi moderni è normalmente basato su alcune delle librerie più diffuse nell'ambito dei pagamenti online. La sicurezza e l'affidabilità che questo tipo di sistemi devono assolutamente garantire rende necessario l'uso di tool testati e specifici per l'utilizzo. Questi sistemi ovviamente devono garantire l'utilizzo sia da parte dell'applicativo di frontend sia da parte del server di backend in modo che l'intero sistema sia consistente.

In questo specifico caso si è scelto di utilizzare il tool Stripe, [16] piattaforma per i pagamenti online disponibile anche come plugin da poter importare all'interno del proprio progetto che permette di avere una gestione dei metodi di pagamento di un singolo utente e di eseguire transizioni monetarie tra un account user e un account commerciante.

3.5.1 Stripe

L'implementazione di Stripe richiede un'inizializzazione da parte del sistema di backend della piattaforma, il quale espone poi per ogni account un token identificativo che sarà necessario per le richieste effettuate tramite la piattaforma Stripe. Ogni token viene inserito all'interno dei dati del singolo account in modo che grazie all'utilizzo del plugin importato come dipendenza all'interno del progetto Android sia possibile richiedere i metodi di pagamento già collegati all'account selezionato e nel caso aggiungerne di nuovi. Questo sistema non esporrà mai le informazioni sensibili e riservate date dall'utente all'aggiunta di una carta poichè saranno gestite internamente alla libreria e salvate solo sul sistema di e-commerce. Gli unici dati, utili all'identificazione e alla scelta di volta in volta del metodo di pagamento saranno solo le ultime quattro cifre della carta e la data di scadenza.

Nel momento in cui si vorrà eseguire una transizione monetaria tra due account sarà necessario avere il token identificativo dell'utente e quello dell'esercizio su cui si

vorrà acquistare, in questo caso le gelateria inserite all'interno del circuito MyGelato. Interverrà poi il sistema di pagamento interno alla piattaforma Stripe per effettuare il passaggio e concludere la transizione in sicurezza e con l'affidabilità di un alto controllo sugli errori.

Oltre alla semplicità di utilizzo di un tool di questo tipo che permette l'inserimento di pagamenti online all'interno dei propri progetti anche senza una specifica esperienza nel campo, è fondamentale la parte di test resa disponibile da Stripe che permette di aggiungere dei metodi di pagamento non reali e di eseguire delle transizioni senza che vi sia un reale spostamento monetario ai fine del testing dei proprio applicativi.

Capitolo 4

Implementazione

A seguito della fase di progettazione si è scelto di suddividere lo sviluppo dell'applicazione in alcune fasi principali che seguono logicamente i workflow presentati nel dettaglio all'interno del capitolo 2. Questo ha permesso di dedicare maggiore attenzione a ogni componente fino a un livello di dettaglio molto alto, in modo da poter anche ottenere una buona valutazione sulle prestazioni dei nodi più critici dell'applicativo.

Per poter ottenere uno sviluppo abbastanza lineare è stato necessario considerare che alcune specifiche richieste erano presenti in tutto il sistema e che quindi non potevano essere sviluppate a se stante dagli altri componenti. Prima di tutto si sono valutati i componenti standard dell'interfaccia grafica in relazione al design richiesto, portando alla creazione di alcune view custom che incapsulano i meccanismi di personalizzazione così da non doverli replicare singolarmente in ogni schermata.

Strutturato il metodo di sviluppo per la grafica si è passati a gestire la navigazione principale scegliendo di utilizzare alcuni dei principali pattern Android: il *NavigationDrawer* e le *RecyclerView*. [10] Il menù iniziale è servito a separare anche concettualmente le principali funzioni da dover sviluppare così da poter procedere successivamente con l'implementazione di ogni componente potendovi accedere anche se altre parti dell'applicazione non erano ancora disponibili.

Ragionando ulteriormente dal generale al dettaglio si è scelto di sviluppare la mappa di ricerca degli shop poichè presente in entrambi i flussi logici principali e quindi nodo cardine dell'applicazione, specialmente in termini di prestazioni. La fase successiva ha fornito l'intera gestione della registrazione e dell'autenticazione di un utente poichè facente parte sia del sistema di marketing sia di quello di e-commerce, ultimi componenti implementati durante la fase di sviluppo.

A conclusione dell'implementazione si sono effettuati dei test per valutare le prestazioni dell'applicazione, in particolar modo sui nodi centrali che avrebbero potuto inficiare l'esperienza utente se con basse prestazioni, come ad esempio la mappa di ricerca.

4.1 Design

Il Design è stata la prima specifica considerata durante lo sviluppo poichè avrebbe rappresentato una costante durante l'implementazione di qualsiasi schermata dell'applicazione.

In figura 4.1 sono visualizzati rispettivamente la scelta di due temi differenti, uno chiaro e uno scuro con a fianco mostrate le differenze dell'interfaccia di navigazione in base al tema selezionato. Tutti i componenti sono personalizzati implementando i tre font differenti, utilizzando le risorse messe a disposizione sia per le icone che per le azioni di navigazione.

Per incapsulare tutte le funzioni riguardanti i temi all'interno di un unico oggetto si è creata la classe *Flavors* che rappresenta il tema attuale, recuperandolo dalle preferenze dell'applicazione, e contiene tutti i dati utili a personalizzare i componenti dell'interfaccia come ad esempio il *main* e il *secondary color*.

Questo oggetto viene automaticamente generato unicamente a partire da un contesto di una schermata senza dover eseguire nessuna operazione di inizializzazione.

```

public class Flavors {
    Context mContext;
    public int getPrimaryColor();
    public int getStatusBarColor();
    public int getPrimaryTextColor();
    [...]
}

```

Listato 4.1: Classe Flavors

A ogni cambio di tema vi sono elementi dell'UI da aggiornare, in special modo tutte le ImageView e le TextView che rappresentano rispettivamente ogni immagine e ogni testo presenti sulla schermata. Per automatizzare la gestione di questi componenti si è scelto di implementare un'interfaccia comune ai due oggetti chiamata *FlavorObject* che dichiara un nuovo metodo chiamato *updateFlavor()* da richiamare a ogni aggiornamento e che modifica le proprietà necessarie come sfondo, colore e immagini visualizzate.

```

public interface FlavorObject {
    public void updateFlavor(Flavors flavor);
    public void updateFlavor(Context context, String taste);
}

```

Listato 4.2: FlavorObject

Da quest'interfaccia si sono ereditate le tre view principali (*CustomImageView*, *CustomTextView* e *CustomEditText*) che sono state utilizzate durante tutto lo sviluppo di questa tesi e che hanno permesso di richiamare in metodo ricorsivo su tutti i componenti lo stesso metodo - *updateFlavor()* - in modo da aggiornare la schermata senza dover conoscere esattamente come ogni singolo elemento andava modificato. Ogni custom view ha quindi implementato il metodo dell'interfaccia così da gestire in maniera automatica i temi chiari/scuri e il cambio a ogni aggiornamento.

Per la gestione dei font dei testi si è sfruttato il fatto di aver implementato una versione personalizzata della TextView così da implementare un nuovo elemento all'interno del componente scritto in xml. Così si è potuto rendere il codice Java molto più pulito indicando il font per ogni testo direttamente nella definizione della view all'interno della risorsa di tipo *layout* senza dover gestire ogni elemento alla creazione della schermata.

```
<com.carpigiani.mygelato.custom.CustomTextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    ...  
    app:typefaceAsset="fonts/Pacifico.ttf"/>
```

Listato 4.3: CustomTextView

Per eseguire l'aggiornamento automatico di tutte le view di una schermata si è scelto di creare una custom Activity, che rappresenta il componente di una schermata dell'interfaccia utente, così da implementare anche in questo caso un metodo che venisse richiamato ogni volta che vi erano elementi da modificare. Questa *FlavorActivity* incapsula la gestione stessa del tema attuale, grazie a una variabile *Flavors*, aggiornando in maniera automatica oltre agli elementi dichiarati anche la toolbar e la statusbar di sistema.

```
public class FlavorActivity extends AppCompatActivity {  
    private Flavors flavors;  
    @Override  
    public void onResume() {  
        this.flavors = new Flavors(this);  
        updateBar();  
    }  
    [...]  
}
```

Listato 4.4: FlavorActivity

Concluso lo sviluppo di ogni componente si è inserita all'interno della navigazione principale la possibilità di accedere alla sezione per la scelta del tema da parte dell'utente, schermata che elenca ogni tema identificandolo con un nome di gelato e alcune immagini personalizzate come si può notare in figura 4.1. Questa schermata è formata da un *ViewPager* che permette lo scorrimento laterale da un tema al successivo, sia tramite slide sullo schermo sia tramite le icone laterali mostrate anche in figura. Per permettere la visualizzazione di elementi personalizzati in ogni dettaglio si sono rese dinamiche la maggior parte delle immagini, come quella in background e quella a rappresentazione del gelato in modo che si aggiornassero a partire dal Flavor selezionato durante lo scorrimento.



Figura 4.1: Temi

4.2 Network

Tutte le funzionalità di collegamento al backend tramite API Rest sono state raccolte e incapsulate all'interno di un unico componente *Network* utilizzato a ogni chiamata server in maniera standardizzata. Sfruttando la libreria OkHttp si è potuto semplificare la gestione delle chiamate con protocollo Http potendo incapsulare il risultato in un classe *Result* contenente i dati sia in caso di successo sia in caso di errore.

Le API Rest rese disponibili dal backend hanno permesso di standardizzare le chiamate richiedendo a ogni chiamata l'inserimento solo del metodo, dei parametri da inviare e dell'eventuale payload. I codici identificativi Http hanno reso la gestione di tutti gli errori maggiormente standardizzata così da poter visualizzare un messaggio di errore il più specifico possibile e localizzato in base alla lingua utilizzata sul dispositivo.

L'utilizzo di una libreria esterna molto performante ha permesso di rendere il codice maggiormente leggibile e manutenibile evitando l'utilizzo delle librerie Apache che, anche se elementi fondamentali della programmazione Java in generale, nello specifico dello sviluppo Android risultano essere troppo complesse.

Per la gestione della risposta di una richiesta al server la possibilità di utilizzare i *Generics* [19] durante la dichiarazione del tipo delle variabili ha permesso di utilizzare lo stesso oggetto Result anche con risultati di tipo differente, come per esempio i dati relativi a un utente o a una carta promozionale. In questo modo tutte le chiamate server che venivano effettuate all'interno delle sezioni dell'applicazione potevano essere replicate e adeguate facilmente al contesto senza dover riscrivere ogni volta il codice necessario al funzionamento di una particolare sezione.

```
public class Result<T, E> {  
    public T result = null;  
    public E error = null;  
    public Result(T result, E error) {  
        this.result = result;  
        this.error = error;  
    }  
}
```

Listato 4.5: Result

L'utilizzo di un middleware così sviluppato per ogni chiamata al backend ha permesso anche di inserire alcuni controlli su ogni richiesta inviata dall'applicativo,

per esempio la possibilità di verificare che le chiamate autenticate non dessero come risultato il codice 401 (*Unauthorized*) [7] in caso di utente non autorizzato; situazione che invoca il logout dell’attuale utente dall’applicazione.

L’utilizzo dell’oggetto Network, che espone i metodi statici necessari all’utilizzo delle sue funzionalità in ogni punto dell’applicazione, è stato ogni volta inserito all’interno di un componente *AsyncTask* che permette l’esecuzione di un blocco di codice in maniera asincrona dal processo principale, migliorando di gran lunga le performance dell’applicativo.

4.3 Utente

Lo sviluppo della sezione legata all’utente è iniziata con la realizzazione di un modello, definito all’interno della classe *User*, che racchiude tutte le informazioni legate ad un account (mostrate nel codice 4.6 per come sono implementate all’interno del modello). Utilizzando Realm è stato possibile definire una tabella nel database che descrive gli account collegati, mantiene in locale sul dispositivo tutti i dati inseriti e mette a disposizione un insieme di funzioni utili per la gestione delle richieste.

```
@PrimaryKey  
public String uuid;  
  
public String firstName;  
public String lastName;  
public String email;  
public String avatar;  
public String client;  
public String token;
```

Listato 4.6: User Model

La sezione per la gestione del proprio account e delle funzioni di autenticazione sono inserite in una parte del menù laterale (*NavigationDrawer*) della pagina principale

dell'applicazione, permettendo di verificare se si è effettuato o meno l'accesso rapidamente. Si è inserito un header in cui sono visibili un'immagine circolare, un messaggio di benvenuto e un bottone per eseguire l'accesso e una volta che l'utente sarà autenticato saranno invece visibili l'avatar, il nome, un'icona per accedere alla schermata di modifica dell'account e il bottone per eseguire il logout.

Si è poi implementata un'unica schermata dalla quale si può accedere al login tramite credenziali personali (*EmailLogin*), al login tramite social network (*FacebookLogin*), alla registrazione di un nuovo account (*SignUp*) e si verrà reindirizzati a questa sezione ogni volta che l'utente tenterà di eseguire delle azioni in cui è necessario aver eseguito l'accesso senza essere autenticato.

Ogni azione che si vuole compiere, come per esempio l'accesso tramite credenziali, deve essere gestito all'interno dell'applicazione in modo molto accurato: ogni campo di input è stato controllato in modo da attuare una validazione anche lato client informando in maniera diretta l'utente nel caso in cui i dati inseriti non siano corretti. A seguito dell'invio di ogni form viene eseguita una chiamata alle API del backend tramite l'interfaccia di supporto *Network* implementata in modo che ritorni eventuali messaggi di errore in maniera standardizzata.

Tutte le schermate facenti parte della sezione del Login sono state create con un tema differente da quelli disponibili volendo creare una separazione anche visiva tra i sistemi di marketing ed e-commerce e le funzionalità legate all'utente che sono concettualmente parallele ai flussi logici dell'applicazione e a un livello maggiore di generalità.

Nel momento in cui l'utente esegue il login vengono aggiornate sul dispositivo le informazioni legate all'account, creando un oggetto Realm partendo dal modello, incapsulando ogni informazione scaricata dal server così che ogni sezione dell'applicativo possa accedervi localmente. L'autenticazione e il logout sono funzioni gestite in maniera centralizzata grazie a un *helper* che incapsula tutte le operazioni per far

si che nessun dato sensibile rimanga salvato in locale a seguito della disconnessione dell'account e non vi siano casi di inconsistenza tra le informazioni presenti in locale e sul server.

L'accesso permette inoltre di accedere ad alcune sezioni dell'applicazione che altrimenti non si potrebbero utilizzare poiché ogni chiamata alle API deve essere corredata delle informazioni legate all'utente che esegue una determinata azione. Per questo motivo nel menù laterale della navigazione principale dell'applicazione alcuni elementi sono disabilitati finché l'utente non esegue l'autenticazione, come mostrato anche in figura 4.2 insieme agli elementi presentati in questo capitolo.

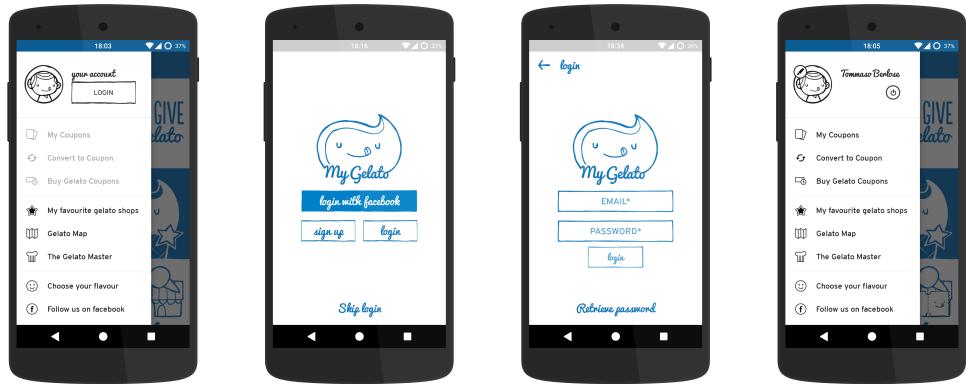


Figura 4.2: Sezione Utente

4.4 Shop

Ogni gelateria inserita all'interno del circuito MyGelato è formalizzata concettualmente come un oggetto *Shop*, sia all'interno del frontend sia sul backend, che incapsula tutte le informazioni disponibili al download di volta in volta, anche qui presentati sotto forma di codice per rendere più completa la descrizione delle variabili utilizzate.

```

@PrimaryKey
public String uuid;

public double latitude;
public double longitude;
public String name;
public String address;
public String phone;
public String status;
public boolean canBurnCoupons = false;
public boolean myGelatoShop = false;

```

Listato 4.7: Shop Model

Le informazioni riguardanti le gelaterie sono essenziali per la fruizione dei contenuti all'interno del sistema di marketing digitale e per la selezione dell'esercizio all'interno di quello di e-commerce. La gestione e il salvataggio in locale degli Shop all'interno dell'applicazione è stato necessario poichè in molti casi dati devono poter essere accessibili anche offline per quanto riguarda le operazioni che non richiedono un collegamento istantaneo con il backend, come per esempio la visualizzazione degli esercizi preferiti.

È stato quindi creato un modello per l'oggetto Shop sempre derivante da un oggetto Realm così da sfruttare tutta la potenza della libreria per il salvataggio e la gestione dei dati in database. Il download delle informazioni sul dispositivo avviene all'interno della schermata della Ricerca ed è eseguita in background poichè avviata in maniera asincrona, a causa della grossa mole di dati da dover scaricare. A ogni ricerca infatti le gelaterie vengono nuovamente scaricate così da avere sempre tutte le informazioni aggiornate, creando però un collo di bottiglia in termini di prestazioni che verrà spiegato e gestito nel dettaglio nel capitolo successivo.

Figura 4.3: Struttura Mappa

4.5 Ricerca

Lo sviluppo della sezione di Ricerca è stato il fulcro del lavoro svolto per questo elaborato, sia perchè come componente si pone a nodo centrale di tutta l'applicazione, sia perchè in termini di prestazioni si sono dovuti valutare molti aspetti per poter mantere un'alta fluidità e reattività anche su dispositivi meno prestanti.

Sono stati svolti molti passaggi successivi volti a migliorare di volta in volta l'algoritmo presente dietro la semplice interfaccia utente composta da una mappa e una lista che devono visualizzare gli shop disponibili nell'area sotto forma di marker. Di seguito verrà quindi descritta soltanto la struttura finale dell'activity *Map* che risulta essere implementata all'interno della versione in produzione dell'applicazione.

Come primo punto è essenziale valutare l'inizializzazione della schermata (visibile in 4.3), considerando che in caso di qualsiasi problema di rete si deve comunque presentare all'utente la struttura cardine anche senza nessun dato visualizzato. Il layout è quindi formato da un header all'interno dei quali vengono innestati i due tab per poter eseguire lo switch tra la visualizzazione a mappa e quella a lista, mentre il corpo è composto da una view che mostra rispettivamente una mappa, inserita grazie all'utilizzo delle API di Google Maps, e una lista, dove è stata nuovamente riutilizzata una RecyclerView unita a un custom Adapter.

Nel primo caso, che è inoltre l'opzione di default, verrà visualizzata una mappa vuota e si tenterà di posizionare il centro della mappa sull'attuale posizione dell'utente. Per eseguire questa operazione verrà richiesto all'utente il permesso di accedere ai dati relativi alla localizzazione e, nel caso venga concessa, si salverà in un oggetto locale l'attuale posizione, altrimenti si utilizzerà la lingua impostata sul dispositivo per centrare la mappa sullo stato corrispondente.

Per effettuare queste operazioni si sono utilizzati i metodi esposti dalle API di Google Maps e dalla libreria dei Google Play Services che permetto di inizializzare la mappa modificando alcune impostazioni di base come lo zoom, i controlli visibili e permettono di ottenere la posizione il più precisa possibile del dispositivo dell’utente.

L’inizializzazione della mappa è eseguita in maniera asincrona e solo una volta che il caricamento è completato vengono attivati i componenti custom di gestione della visualizzazione dei marker come ad esempio il sistema di Clustering che si pone a middleware tra la mappa e l’insieme dei marker aggiunti, in modo da aggregarli ad ogni movimento della view mostrando solo il numero di elementi presenti in ogni cluster. Un esempio è visibile nella figura FIG.

Come la mappa, anche la lista viene inizializzata con una base di dati inizialmente vuota mostrando un messaggio di notifica all’utente che segnala il fatto che non vi siano presenti degli Shop in zona. Nel frattempo viene caricato e inizializzato l’adapter che, dato l’insieme degli shop, mostrerà l’elenco delle gelaterie in ordine di distanza dall’attuale posizione dell’utente.

Una volta che entrambi i sistemi saranno stati inizializzati verranno avviati in background tutti i meccanismi di richiesta di download degli shop da mostrare a video. Per fare questo un forte componente di questa schermata risultano quindi essere i processi asincroni che vengono avviati tramite l’utilizzo di componenti AsyncTask o grazie all’uso di Thread.

L’oggetto *LoadShopAtAsyncTask*, funzione principale per il download dei dati, richiede in ingresso un insieme di parametri tra cui la posizione e il raggio all’interno del quale devono ricadere gli Shop, in questo modo la richiesta al backend non otterrà in risposta l’intera base di dati presente sul server ma solo le informazioni utili da essere visualizzate a video in quella porzione di mappa.

In background i dati scaricati verranno salvati direttamente sul database grazie all’utilizzo di Realm e una volta completato il processo senza errori, grazie all’utilizzo

di EventBus verrà richiesto l'aggiornamento dell'interfaccia che dovrà richiedere i dati dal database, gestirli come marker e aggiungerli al sistema di Clustering della mappa.

Questo è stato il primo punto critico in termini di prestazioni che si è presentato durante lo sviluppo. L'aggiunta dei marker è vincolata all'elaborazione dei dati del singolo shop, poichè saranno mostrati a video solo gli shop veramente presenti all'interno della vista della mappa, dovranno essere salvati in RAM tutti i dati che verranno richiesti una volta selezionato il singolo marker per essere mostrati all'utente e in ultimo dovrà essere anche calcolata la distanza del singolo esercizio dall'utente, così da poter ordinare la lista rispetto a questo parametro.

Dopo alcune rielaborazioni del codice si è spostato totalmente in background ogni singolo passaggio che non intervenisse direttamente sull'UI, poichè l'unico thread che vi può accedere è quello principale, così da non rendere bloccante l'operazione di aggiornamento della mappa, processo che rendeva poco fluido il sistema di clustering.

In questo modo si è risolto il problema del download delle informazioni, che spesso rappresentano comunque una grossa mole di dati, la loro rielaborazione ai fini delle operazioni rese disponibili sulla schermata per l'utente e l'aggiornamento delle view di conseguenza al cambio della base di dati.

Il meccanismo appena descritto però gestisce il fatto di visualizzare solo gli Shop presenti attorno all'utente all'apertura della schermata, mentre si vuole poter permettere all'utilizzatore anche di esplorare le zone su altri punti della mappa. Per questo motivo ad ogni conclusione di spostamento della view GoogleMap viene risollevato un evento grazie ad EventBus che porta ad eseguire nuovamente tutti i processi sopra descritti. Se logicamente il sistema ha risolto immediatamente il problema che si era creato ci si è trovati di fronte al secondo punto critico in termini di prestazioni.

Il movimento della mappa e il listener che solleva la callback a ogni fine spostamento della mappa verrebbe sollevato troppe volte, portanto a un rallentamento significativo dell'applicazione. Per risolvere questo problema si è utilizzato uno scheduler, definito dall'oggetto *ScheduledExecutorService*, che avvia l'esecuzione del processo di aggiornamento solo dopo 500 millisecondi che l'utente ha smesso di muoversi sulla mappa. Si è valutato questo intervallo di tempo per non dover gestire istruzioni che verrebbero lanciate anche ogni dieci millisecondi mantenendo però una forte reattività agli occhi dell'utente.

Conclusa l'implementazione della visualizzazione degli shop all'interno della mappa si è implementato, all'interno del sistema di elaborazione dati, il meccanismo di aggiornamento dell'adapter della lista. Questo adapter è stato poi utilizzato in maniera identica per la gestione della lista all'interno della sezione dei Preferiti poichè sono state notate una serie di similitudini che ne hanno permesso il riutilizzo.

Il successivo passaggio ha portato all'implementazione della InfoWindow del singolo shop, all'interno del quale vengono visualizzate le informazioni essenziali utili all'utente, come nome e indirizzo, unite alle icone per raggiungere la sezione delle Carte Promozionali e per la gestione delle funzionalità dell'aggiunta e rimozione dai Preferiti, spiegate dettagliatamente nei capitoli successivi.

Si è dovuto creare una view custom esterna alla mappa per questo utilizzo poichè le info window standard proposte dalle API di Google Maps non permettevano di gestire correttamente il click in punti differenti della finestra visualizzata.

Concluse le funzionalità relative al sistema di marketing si sono dovute implementare le funzionalità del flusso logico dell'e-commerce, poichè si è scelto di riutilizzare l'intera schermata senza crearne una prettamente identica. Per fare questo passaggio si è gestito l'Intent di avvio della sezione valutando l'*Action* inserita all'interno che demarca quale delle funzionalità sono richieste.

Nel caso in cui si arrivi a questa sezione durante l'acquisto di un coupon, le icone presenti nella info window verranno sostituite da un'icona per la selezione della gelateria e nel momento in cui l'utente farà una scelta questa activity verrà chiusa inserendo all'interno del risultato della chiamata l'identificativo dello shop. In questo ci si rifà all'architettura modulare di Android che permette l'utilizzo di activity differenti per ottenere oggetti come risultato della chiamata.

Concluso quest'ultimo passaggio, come già detto, si sono susseguiti alcuni test prestazionali sull'interfaccia che hanno portato a ottenere alta fluidità e reattività anche su dispositivi meno prestanti. Si è infine aggiunto il collegamento differenziato all'interno della navigazione principale dell'applicazione.

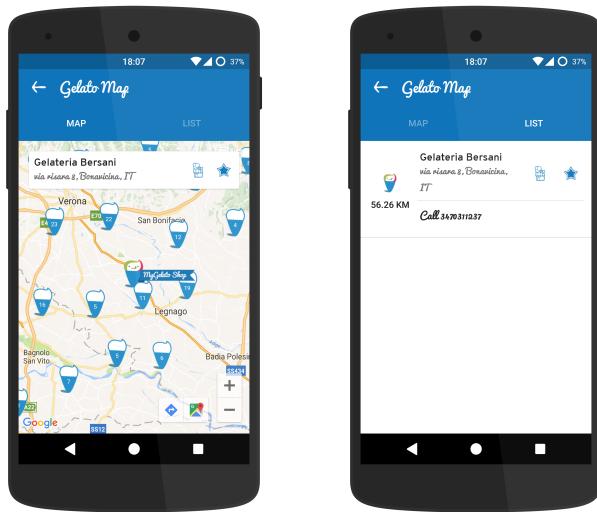


Figura 4.4: Ricerca

4.6 Marketing Digitale

Il primo flusso logico principale a essere stato implementato è stato quello del marketing digitale poiché legato solo in parte alla presenza di un utente registrato e auten-

ticato all'applicazione; sono quindi presenti alcune operazioni comunque disponibili anche senza aver eseguito il login.

La prima sezione a essere implementata è stata quella legata alla visualizzazioni delle carte del Mastro Gelatiere poichè il numero delle promozioni disponibili era da visualizzare anche all'interno della navigazione principale e quindi la richiesta per scaricarle doveva essere presente anche all'interno della schermata iniziale dell'applicazione.

Per ottenere un maggiore riutilizzo del codice si è scelto di creare una stessa sezione per visualizzare sia le carte del mastro gelatiere sia quelle delle singole gelaterie ottenendo una visualizzazione differente in base al collegamento (*Intent*) che avrebbe portato l'utente all'interno della sezione. Allo stesso modo si è utilizzato uno stesso modello per entrambe le carte definendo un elemento Realm di nome *Card* che incapsula i dati necessari all'utilizzo delle carte nel sistema differenziandole grazie a un flag presente anche nel modello fornito dal backend.

Avendo quindi implementato la visualizzazione delle carte scaricate tramite chiamata alle API, nella schermata appena descritta, è stato semplice eseguire il passaggio successivo che dalla visualizzazione della singola gelateria portava alle proprie carte promozionali: all'interno della finestra informativa con i dati dello shop nella ricerca si sono inserite un'icona che funge da collegamento alle carte e un'icona per aggiungere ai preferiti il singolo esercizio.

Per il salvataggio offline dei preferiti si è però dovuto creare un secondo modello all'interno del database *Favorite* che semplicemente identifica gli esercizi salvati tra i preferiti in modo che queste informazioni rimangano salvate in locale sul dispositivo anche senza che un utente che si sia loggato. L'inserimento di un flag all'interno del modello degli shop non avrebbe funzionato poichè sarebbe stato sovrascritto ad ogni aggiornamento degli shop durante il download e la modifica con le informazioni sul server.

Una volta che una gelateria viene aggiunta ai preferiti, nel caso l’utente si sia autenticato verrà eseguita una chiamata al server che attiverà la ricezione di notifiche push e in automatico verrà anche aggiunto un controllo per il geofencing sul dispositivo, entrambe le funzioni sono spiegate nel dettaglio nel capitolo 4.6.2.

4.6.1 Carte Promozionali

L’implementazione delle carte promozionali ha seguito uno sviluppo volto a utilizzare lo stesso codice e le stesse visualizzazioni sia per la presentazione delle carte legate al singolo esercizio sia per quelle del mastro gelatiere. Prima di tutto si è quindi realizzato il modello in Realm chiamato *Card* contenente le principali informazioni legate alla carta:

```
@PrimaryKey  
public String uuid;  
  
public String name;  
public String title;  
public String imageUrl;  
public String shopUUID;  
public String cardDescription;  
public String link;  
public String phone;  
public Long start;  
public Long end;
```

Listato 4.8: Modello Card

È stata creata una schermata unica per la visualizzazione delle carte affiancate, grazie all’utilizzo di un *ViewPager*, con un layout di ogni pagina elaborato dove ogni carta mostra la propria immagine e al click su un’icona in basso a destra ruota su se stessa per permettere di leggere la descrizione ed eventualmente ottenere nuove informazioni tramite il link proposto. Nel caso vi sia la presenza di un video il

tap sull'immagine porterà alla visualizzazione online della risorsa resa disponibile tramite URL.

Il download delle informazioni deve avvenire all'apertura della schermata grazie a una richiesta alle API in background e tutti i dati scaricati vengono salvati in locale così da essere disponibili anche offline e nel caso di modifiche saranno aggiornati eliminando le promozioni non più disponibili perchè, per esempio, scadute.

Nel caso delle carte del mastro gelatiere il procedimento di richiesta dei dati deve essere eseguito, come già detto, anche all'interno della schermata principale poichè il numero delle carte è da visualizzare all'interno dell'immagine del menù che rimanda alla sezione del mastro gelatiere. Inoltre nel caso di un utente con autenticazione effettuata verrà registrato il dispositivo sul server per la ricezione di una notifica push ogni volta che saranno pubblicate nuove promozioni, funzionalità spiegata nel dettaglio nel capitolo successivo.

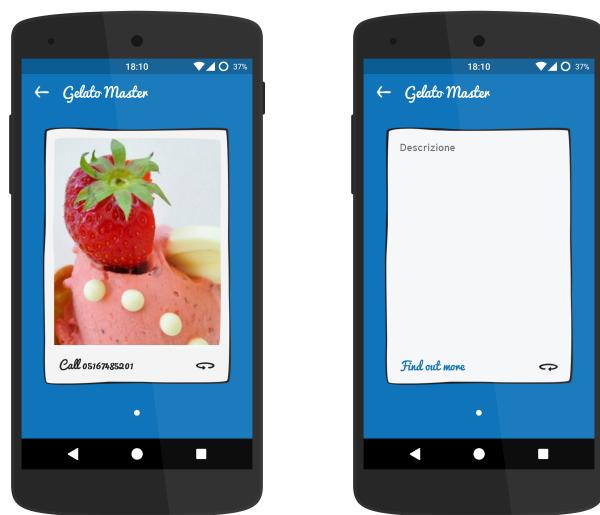


Figura 4.5: Card

4.6.2 Preferiti

Il primo sviluppo legato alla sezione dei preferiti è stata quella della gestione dell'aggiunta e della rimozione di una gelateria come esercizio d'interesse tramite un'icona all'interno della visualizzazione della ricerca con cui l'utente può salvare localmente le sue preferenze. Successivamente si è implementata la sezione dell'applicazione che permette la visualizzazione e la gestione della lista degli esercizi preferiti, direttamente raggiungibile dalla navigazione principale.

L'utente può aggiungere una gelateria ai preferiti cliccando sull'icona della stella presente nella finestra di informazioni visualizzata all'interno della ricerca degli shop una volta selezionato uno. L'aggiunta, come anche la rimozione, aggiorna immediatamente sia la mappa modificando il marker mostrato a video e anche la lista dei preferiti.

Grazie all'utilizzo del modello *Favorite* è possibile accedere ai codici identificativi degli Shop preferiti così da richiamarli dal database in locale visualizzando tutte le principali informazioni legate al sistema di marketing e dando la possibilità di accedere alle carte promozionali o direttamente al *dialer* selezionando il recapito telefonico, se presente.

Avendo gestito l'aggiunta e la rimozione dei preferiti grazie a un sistema centralizzato che racchiude tutte le operazioni necessarie è stato abbastanza intuitivo poter inserire un middleware che gestisse l'aggiunta e la rimozione delle funzionalità riguardanti l'iscrizione alle *Notifiche Push* ricevute del server e del *Geofencing* gestito dal dispositivo.

Per l'implementazione della funzionalità di notifiche push si è scelto di utilizzare i servizi resi disponibili da Google stessa utilizzando inizialmente *Google Cloud Messaging* per poi passare a utilizzare il servizio *Firebase*. Per far dialogare in maniera bidirezionale l'applicativo e il server nel momento in cui un utente esegue il

login all'applicazione viene attivato Firebase richiedendo al server un token che verrà poi passato al sistema di gestione delle notifiche, incluso all'interno del codice come libreria, che si metterà in ascolto di eventuali messaggi. Viene presentato per semplificazione solo l'inizializzazione parziale del sistema di ascolto di notifiche push e si rimanda alla documentazione completa per ottenere le informazioni necessarie alla descrizione dell'implementazione.

```
firebaseApp.initializeApp(  
    this,  
    firebaseOptions.fromResource(this)  
)
```

Listato 4.9: Firebase

Ogni volta che l'utente aggiungerà e rimuoverà un preferito verrà quindi eseguita una chiamata al backend con lo scopo di informarlo della modifica attivando o disattivando rispettivamente la ricezione delle notifiche push. In questo modo nel momento in cui sarà disponibile una nuova carta promozionale per quello *Shop* verrà visualizzata una notifica sul dispositivo che permetterà all'utente di accedere direttamente alla sezione delle carte promozionali specifiche per quell'esercizio.

Allo stesso tempo sul dispositivo verrà inoltre sottoscritto al sistema di geofencing che produrrà un *Intent* ogni qual volta l'utente rimarrà entro l'area di 5 kilometri da una gelateria presente tra i preferiti, il quale verrà gestito per mostrare anche in questo caso una notifica che permetterà di accedere invece alle informazioni della gelateria all'interno della lista dei preferiti. Ovviamente per i dispositivi con le ultime versioni di Android sarà necessario richiedere anche in questo caso la possibilità di accedere alla localizzazione come fatto all'interno della ricerca.

Nel listato 4.10 si può vedere come l'applicazione si regista al servizio di Geofencing del sistema una volta che una gelateria *shop* viene inserito tra i preferiti,

situazione che va oltretutto ripetuta ogni volta che il dispositivo dell’utente viene riavviato.

```
mGeofenceList.add(new
    Geofence.Builder()
        .setCircularRegion(
            shop.latitude,
            shop.longitude,
            GEOFENCE_RADIUS_IN_METERS
        )
        .setLoiteringDelay(2 * 60 * 1000)
        .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_DWELL)
        .build()
);

};
```

Listato 4.10: Geofencing

In figura 4.6 infine è visibile l’implementazione della lista degli Shop preferiti, di cui sono presentate le informazioni essenziali e la distanza, insieme ad un esempio di notifica push inviata da MyGalato per segnalare l’aggiunta di una nuova carta promozionale.

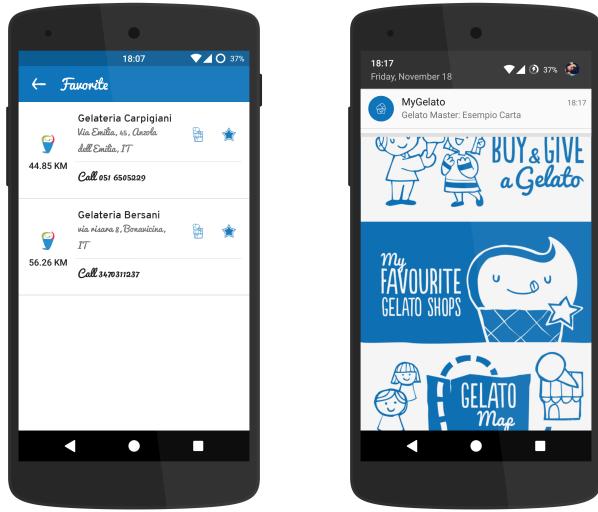


Figura 4.6: Preferiti

4.7 E-Commerce

Il sistema di e-commerce è stato l'ultimo tassello dell'applicazione che ha concluso il lavoro di sviluppo di questo elaborato. Si è trattato di unificare parte delle componenti già implementate creando un workflow che permetta all'utente di comprare e utilizzare i coupon digitali in sicurezza e con facilità.

La prima fase è stata legata alla rappresentazione formale dei buoni acquistati da un determinato account e di quelli invece disponibili all'acquisto legati ad una determinata gelateria, in particolare è stato importante valutare quali informazioni mantenere sempre in locale per non inficiare la consistenza dei dati presenti sul dispositivo rispetto a quelli online.

Definita la struttura degli elementi che si sarebbero andati ad utilizzare si è creata la sezione dedicata alla gestione dei coupon personali disponibili a ogni utente, ovviamente legando ogni funzione all'avvenuta autenticazione tramite account

MyGelato. Questa sezione permette di avere una visione completa delle proprie informazioni aggiornate costantemente rispetto ai dati presenti sul server online.

Le funzioni legate al sistema di e-commerce comprendono l'utilizzo di account e dispositivi diversi che interagiscono tra di loro e che non devono assolutamente creare casi di inconsistenza sul database sia online sia offline in locale sui dispositivi utilizzati.

Per rendere accessibile all'utente l'accesso alle funzioni di acquisto e di riscatto di un buono in maniera diretta si sono inseriti all'interno della navigazione principale dell'applicazione i collegamenti a queste sezioni insieme a un bottone flottante che riporta alla gestione dei coupon; nel caso si sia effettuato l'accesso con un account di tipo gelatiere si verrà riportati invece alla sezione per la validazione dei buoni.

4.7.1 Coupon

Per formalizzare il concetto di buono acquistato da ogni utente si è scelto di creare il modello *Coupon*, estensione di un oggetto Realm, che incapsula tutte le informazioni disponibili e assicura una corretta gestione delle transizioni insieme a un alto livello di sicurezza.

Ogni coupon è unico grazie a un codice identificativo, possiede alcune informazioni importanti come l'acquirente e l'attuale proprietario, la data di vendita e anche alcune informazioni non direttamente utili all'utilizzo all'interno dell'applicativo frontend che però vengono trasmesse per completezza e nel caso possano essere necessarie per futuri sviluppi.

Nel momento in cui si accede alla sezione per la gestione dei coupon vengono scaricati tutti i dati in background così da aggiornare il database locale e grazie a una funzionalità di Realm, una volta eseguito un update sulla base dei dati vengono ricaricate alcune view dell'interfaccia andando a riempire con tutte le informazioni le tre liste presenti: Coupon Validi, Coupon Ricevuti e Lista Completa dei Coupon.

Come si può vedere in figura 4.7, ogni coupon visualizza alcune informazioni di base e sono presenti i collegamenti diretti per l'utilizzo e lo share tramite altri canali di comunicazioni. Selezionando la parte sinistra del buono si verrà reindirizzati all'interfaccia di utilizzo presentata nel dettaglio al capitolo 4.7.4 mentre cliccando sulla parte destra si verrà invitati a chiedere il metodo di condivisione da utilizzare.

Nel caso in cui un coupon sia stato già utilizzato non sarà possibile eseguire nessuna delle due operazioni e la parte destra dell'immagine verrà nascosta per dare l'impressione di un buono strappato dopo averne usufruito. Per ottenere coerenza da questo punto di vista a ogni passaggio di schermata vengono ricercate eventuali modifiche sul server sempre per mantenere la coerenza dei dati interni al sistema MyGelato.



Figura 4.7: Coupon

4.7.2 Acquisto

Il sistema di acquisto di un coupon è stato quello più complesso poichè doveva operare con più di un componente esterno all'applicazione mantenendo però a ogni

passaggio un alto livello di sicurezza senza inficiare alla consistenza dei dati della piattaforma MyGelato: ad esempio una volta scelto lo shop, per modificare la gelateria si perdono le scelte successive riportando l'utente al passaggio iniziale per non rischiare inconsistenze nel sistema. Si è poi analizzato ogni singolo passaggio per poter guidare l'utente durante ogni step mostrando a video di volta in volta solo ciò che era richiesto.

La schermata di acquisto di un coupon è stata creata in modo che sia raggiungibile direttamente dal menù di navigazione principale, ma nel caso in cui l'utente non abbia eseguito l'autenticazione viene rimandato alla schermata di login dove vengono richieste le credenziali per poter continuare. In questo modo solo gli utenti registrati alla piattaforma hanno la possibilità di effettuare un acquisto che richiede un collegamento con il backend dove sono necessarie le credenziali di autenticazione.

Come si può vedere in figura 4.8 dopo l'helper iniziale viene richiesto all'utente di scegliere da quale gelateria vuole acquistare un coupon e, come già specificato, si verrà reindirizzati alla mappa di ricerca con un Intent d'avvio contenente alcune opzioni che disabiliteranno la visibilità degli shop in cui non è abilitato il sistema di e-commerce e sostituiranno alle icone presenti nella finestra di dettaglio un'icona per la selezione dell'esercizio. Una volta che l'utente avrà effettuato la selezione verrà riportato sulla schermata di acquisto dove saranno mostrati i dettagli della gelateria e verrà invece visualizzata la possibilità di scegliere il metodo di pagamento.

Per effettuare questa operazione l'applicazione esegue due chiamate differenti al server, una delle quali richiede il codice identificativo dell'utente rispetto alla libreria di e-commerce Stripe, che servirà per richiedere i metodi di pagamento disponibili dell'utente, e scaricherà inoltre anche i coupon disponibili per l'esercizio scelto. In questo caso i coupon sono salvati all'interno di un modello *AvailableCoupon* poichè le informazioni necessarie sono ridotte rispetto al singolo coupon dell'utente; i dati

scaricati inoltre non vengono salvati sul database del dispositivo poichè una volta usciti dalla schermata non sono più utili.

Lo step successivo con cui l'utente può procedere è la scelta del metodo di pagamento che verrà effettuato in una schermata separata dove sarà visualizzata la lista dei metodi precedentemente inseriti dall'utente e scaricati tramite chiamata al backend, che esporrà però solo gli ultimi numeri della carta inserita e la data di scadenza così da non trasmettere dati sensibili non necessari. Oltre alla possibilità di scegliere una delle opzioni già presenti l'utente potrà aggiungere una nuova carta grazie a una schermata separata, differenziata anche nella grafica, dove i dati non verranno memorizzati ma inviati direttamente alla libreria di Stripe che si occuperà di validare i dati e aggiornare i dati presenti sul backend. Si verrà infine reindirizzati alla schermata precedente dove le opzioni saranno aggiornate.

Scelto il metodo di pagamento si verrà riportati sulla schermata di acquisto dove saranno visualizzati i dati essenziali dell'opzione scelta dando, come nel caso dello shop, la possibilità di modificare la decisione presa. Infine l'ultima decisione sarà quella riguardante il coupon da acquistare grazie alla visualizzazione di quelli disponibili.

Sotto il collegamento alla scelta del metodo di pagamento è quindi presente un *ViewPager* che visualizza i coupon disponibili specifici per la gelateria scelta mostrando i dettagli principali: nome, descrizione, prezzo e valuta. In questo caso la scelta verrà effettuata scorrendo sullo slide in modo che al centro della schermata vi sia quello scelto dall'utente.

Concluse tutte le scelte l'utente potrà procedere con l'acquisto che si tratterà di una nuova chiamata alle API del server in modo che grazie anche all'utilizzo del codice Stripe sia dell'utente che dello shop potrà effettuare la transizione e aggiungere il coupon a quelli legati all'account utilizzato. Nel caso di errori verrà visualizzato

un messaggio e si potrà ritentare l'operazione, altrimenti se non vi dovessero essere problemi si verrà reindirizzati alla schermata di gestione dei coupon personali.



Figura 4.8: Acquisto

4.7.3 Condivisione e Riscatto

Il meccanismo di condivisione e riscatto ha richiesto lo sviluppo di funzioni complementari eseguite su dispositivi differenti dove l'interazione si basa fortemente sull'utilizzo di diversi componenti di sistema. Per permettere il dialogo tra due dispositivi differenti si è scelto per esempio di sfruttare i canali di comunicazioni più comuni e diffusi anche tra gli utenti medi in modo che il sistema di share sia il più facile possibile e del tutto conforme al sistema di condivisione standard della piattaforma. In genere questi metodi presuppongo una conoscenza da parte dei due utenti che interagiscono, ma ci si è basati sul fatto che da specifiche questo fosse una parametro valutato durante lo sviluppo del sistema MyGelato.

Dalla lista di coupon disponibili l'utente ha quindi la possibilità di cliccare sull'icona per la condivisione e verrà inviata una richiesta - *Intent* - a livello di sistema in broadcast per poter selezionare il metodo che si preferisce per inviare il contenuto da scambiare: un testo precompilato insieme ad un link che riporta ad una pagina web

posta online dal backend. Il testo presenta brevemente l'oggetto della condivisione e invita ad utilizzare il link per riscattare il coupon specificato.

La pagina web a cui si viene reindirizzati riconosce il sistema operativo dal quale ci si sta collegando per effettuare un redirect ad un link particolare (*mygelato://*) che verrà gestito in automatico dal dispositivo con cui si sta visualizzando il sito. Nel caso in cui vi sia installata sul dispositivo l'applicazione MyGelato allora verrà avviata la schermata di riscatto precompilata con i dati necessari, altrimenti si verrà reindirizzati agli store principali per consigliare il download dell'applicazione.

Il mezzo di identificazione del coupon condiviso è un codice alfanumerico a sei cifre, il PNR, che viene condiviso in automatico insieme al resto delle informazioni. Si è quindi creata la schermata per riscattare un buono, sezione raggiungibile direttamente dalla navigazione principale dell'applicazione, dove è presente una view di input dove si deve inserire il codice PNR; nel caso si arrivi sulla schermata tramite il link descritto sopra allora il campo di input del codice sarà già compilato.

Una volta inserito il codice sarà effettuata una richiesta al server tramite chiamata API che attuerà lo spostamento del buono da un utente all'altro. Per entrambi gli account vi sarà quindi l'aggiornamento delle informazioni sul proprio dispositivo la prima volta che verranno richiesti i coupon disponibili e l'utente che ha effettuato il riscatto verrà immediatamente reindirizzato sulla schermata di lista dei coupon altrimenti, in caso di fallimento dell'operazione, verrà notificato con un messaggio di errore.

La figura 4.9 mostra tutti i passaggi principali per la condivisione e il riscatto di un buono seguendo il flusso logico proposto all'utente a ogni step, molti dei quali anticipati da un'immagine informativa che esplica brevemente come utilizzare ogni singola schermata.

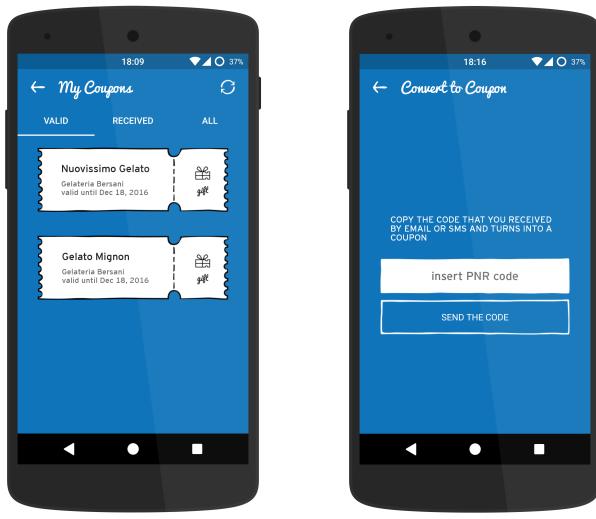


Figura 4.9: Condivisione e Riscatto

4.7.4 Utilizzo e Validazione

L’ultimo passaggio del workflow di e-commerce è l’utilizzo pratico dei coupon che si sono acquistati o che si sono riscattati perchè condivisi da altri utenti. In questo processo entrato in gioco due entità che sono utenti che spesso non entrano in contatto se non solo per i pochi momenti che servono all’acquisto materiale del valore del buono utilizzato. Questa situazione pone alcune limitazioni negli strumenti utilizzati per la comunicazione tra i due dispositivi che devono collaborare mantenendo uno stato costantemente consistente sia sui dispositivi fisici che sul server.

Per superare questo ostacolo si è scelto uno strumento molto diffuso che è l’utilizzo di un QR code che permette di generare un’immagine codificata a partire da un testo, in questo caso il PNR code legato al coupon che si vuole utilizzare, leggibile e interpretabile da un qualsiasi cellulare con un fotocamera. Chiunque abbia effettuato l’accesso all’applicazione con account di tipo gelatiere potrà validare il

coupon direttamente tramite l'applicazione MyGelato grazie a una chiamata API al backend che confermerà o meno l'avvenuta conclusione del processo senza problemi.

La sezione per generare e visualizzare il codice QR da mostrare in gelateria è raggiungibile cliccando sul singolo coupon all'interno della gestione dei propri buoni. Questa schermata mostra una versione del coupon a tutto schermo e, grazie ad una libreria esterna, inserisce il codice QR all'interno del coupon in modo che sia ben visibile e riconoscibile da qualsiasi software. Si deve quindi solamente mostrare il dispositivo utilizzato a chi dovrà poi validare il buono. Non essendo presente in questo caso una connessione bidirezionale con il backend, per aggiornare la schermata una volta concluso il processo di utilizzo, l'applicazione effettua un polling al server ogni cinque secondi per verificare se il coupon sia stato utilizzato o meno.

Chiunque possieda un account di tipo gelatiere avrà nella navigazione principale dell'applicativo il collegamento alla sezione per la convalida. Verrà avviata immediatamente la fotocamera che, tramite una libreria esterna, verificherà la presenza o meno di codici QR leggendoli. Dopo una validazione semplice si effettuerà la chiamata al server dando poi la possibilità di convalidare altri coupon senza dover ritornare ogni volta alla schermata iniziale. Nel caso in cui la fotocamera sia già occupata da altri processi attivi sul dispositivo o nel caso in cui l'utente abbia rimosso alcuni permessi all'applicativo verrà visualizzato un messaggio di errore che comunicherà al proprietario del dispositivo cosa fare per risolvere il problema.

A conclusione del processo di validazione di un coupon entrambi i dispositivi verranno aggiornati per mantenere coerenza tra i dati presenti sul sistema risolvendo così in maniera semplice il problema principale del dialogo tra due dispositivi mentre verranno nel frattempo applicati i passaggi commerciali totalmente all'interno del backend avendo passato il codice identificativo del commerciante.

La figura 4.10 permette di visualizzare i passaggi fondamentali per l'utilizzo e la validazione di un buono, anche in questo caso vengono visualizzati al primo avvio

alcune immagini informative per aiutare gli utenti nell'utilizzo delle funzionalità appena descritte.

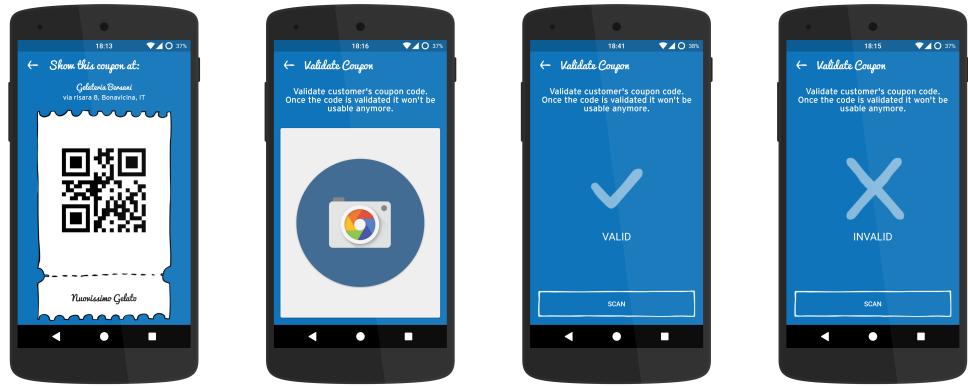


Figura 4.10: Utilizzo e Validazione

Conclusioni

Il lavoro esposto all'interno di questa tesi ha portato al completamento e alla pubblicazione della applicazione mobile per piattaforma Android che si inserisce all'interno della creazione dell'ecosistema denominato MyGelato, progetto dell'azienda Carpigiani leader nel settore della vendita di macchine per gelato. È stato necessario valutare attentamente ogni passaggio dello sviluppo partendo dalla fase di progettazione e studio delle specifiche richieste fino all'implementazione vera e propria dell'applicativo. Si è svolto un lavoro in team dovendo aggregare le informazioni derivanti dagli altri componenti della piattaforma sia testando ad ogni step le funzionalità implementate che includessero il collegamento con il sistema di backend sia studiando i pattern strutturali utilizzati nello sviluppo della rispettiva applicazione per sistema iOS.

Insieme al lavoro prettamente legato all'implementazione si sono anche proposte alcune valutazioni sul contesto all'interno del quale si inserisce la piattaforma che si è andati a creare, così da presentare sia le funzionalità maggiormente legate al progetto MyGelato sia le funzionalità che formano invece un pattern più generale adottabile anche da altre aziende che desiderino avviare lo sviluppo di un sistema con finalità simili.

Si è descritto quindi la situazione tipo in cui si sono trovate o si verranno a trovare tutte le aziende leader nei propri settori che saranno costrette a diversificare il proprio prodotto integrando un sistema di marketing e di vendita che sfrutti le

nuove tecnologie disponibili sul mercato. Il pubblico consumer diventa ogni giorno più esigente per quanto riguarda i servizi resi disponibili dalle aziende parallelamente ai loro core business, specialmente se si tratta come in questo caso di un mercato in cui non vi è ancora un'ingerenza tecnologica così profonda.

È stato quindi essenziale valutare e studiare ogni dettaglio sin dalla prima fase di progettazione così da poter fornire agli utenti un prodotto finale efficiente, curato nel design e al passo con gli ultimi sviluppi in termini di programmazione mobile. Per poter ottenere un risultato ottimo in questo senso si sono studiate tutte le molteplici possibilità disponibili in ambito open source sia per quanto riguarda i pattern strutturali sia per le librerie utilizzate.

Tutte le specifiche progettuali richieste sia in ambito di marketing digitale sia in termini di funzionalità e-commerce sono state pienamente soddisfatte rendendo già disponibile per l'utente finale un prodotto performante e in perfetta sinergia con le altre componenti della piattaforma. Sarà ora di fondamentale importanza considerare una buona campagna di marketing affiancata alla corretta comunicazione con i gelatieri in modo da garantire al progetto una diffusione capillare nel territorio e aprire la strada per un'espansione di livello mondiale.

L'elaborato, grazie all'architettura modulare con cui è stato sviluppato, permette l'implementazione di nuove funzionalità con il minimo sforzo, senza dover riprogettare l'attuale struttura dell'applicazione e si pone ad esempio per progetti simili che abbiano specifiche progettuali simili anche solo parzialmente potendo estrapolare da ogni flusso logico le scelte implementative fatte.

Bibliografia

- [1] GitHub. EventBus. <https://goo.gl/Q0MlhV>. 3.4
- [2] Gradle. Gradle build tool. <https://goo.gl/ssMyng>. 3.1.2
- [3] GreenRobot. EventBus. <https://goo.gl/KxdfvL>. 3.4.1
- [4] HTML.it. Come sviluppare app android, ibrido o nativo? <https://goo.gl/PA9SyC>. 3.1
- [5] Html.it. Gestione dei database in android. <https://goo.gl/303zu8>. 3.2
- [6] Html.it. I principi dell'architettura restful. <https://goo.gl/eFzOxw>. 3.3
- [7] HttpStatus. 401 unauthorized. <https://goo.gl/hY0XNd>. 4.2
- [8] Google Inc. Android developers. <https://goo.gl/pnFAUr>. 3
- [9] Google Inc. Android developers. <https://goo.gl/l3rI4f>. 3.1
- [10] Google Inc. Android navigation pattern. <https://goo.gl/h91v4x>. 4
- [11] Google Inc. Google maps apis. 3.1.2
- [12] Paul Mason. *Postcapitalismo*. 2016. 1.1
- [13] Inc. Postdot Technologies. Postman. <https://goo.gl/Vx9SBe>. 3.1
- [14] Realm. Realm mobile database. <https://goo.gl/4usiSu>. 3.2

- [15] Square. Okhttp. <https://goo.gl/KuodbM>. 3.3
- [16] Stripe. The new standard in online payments. <https://goo.gl/tu3sv3>. 3.5
- [17] Wikipedia. Acid operations. <https://goo.gl/53NXJw>. 3.2.1
- [18] Wikipedia. Android studio. <https://goo.gl/m6m4ae>. 3.1.2
- [19] Wikipedia. Generics. <https://goo.gl/42i9QX>. 4.2
- [20] Wikipedia. Git (software). <https://goo.gl/v4YsP3>. 3.1.3
- [21] Wikipedia. Java. <https://goo.gl/KUEWir>. 3.1.1

Elenco delle figure

1.1	Application Economy	2
2.1	Risorse Design	9
2.2	Mockup Mappa	11
4.1	Temi	41
4.2	Sezione Utente	45
4.3	Struttura Mappa	47
4.4	Ricerca	51
4.5	Card	54
4.6	Preferiti	58
4.7	Coupon	60
4.8	Acquisto	63
4.9	Condivisione e Riscatto	65
4.10	Utilizzo e Validazione	67

Elenco dei listati

3.1	AppCompatActivity	24
3.2	Layout XML	24
3.3	Model Realm	28
3.4	Realm Transaction	28
3.5	Realm Query	29
3.6	Realm Listener	29
3.7	Network Connection	31
3.8	Event Class	33
3.9	EventBus Subscribe	33
3.10	EventBus Update	33
4.1	Classe Flavors	39
4.2	FlavorObject	39
4.3	CustomTextView	40
4.4	FlavorActivity	40
4.5	Result	42
4.6	User Model	43
4.7	Shop Model	45
4.8	Modello Card	53
4.9	Firebase	56
4.10	Geofencing	57

Ringraziamenti