

Capitolo 1

Progettazione e Implementazione

Per la fase di progettazione e implementazione 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 (visibili in figura 1.1). 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.

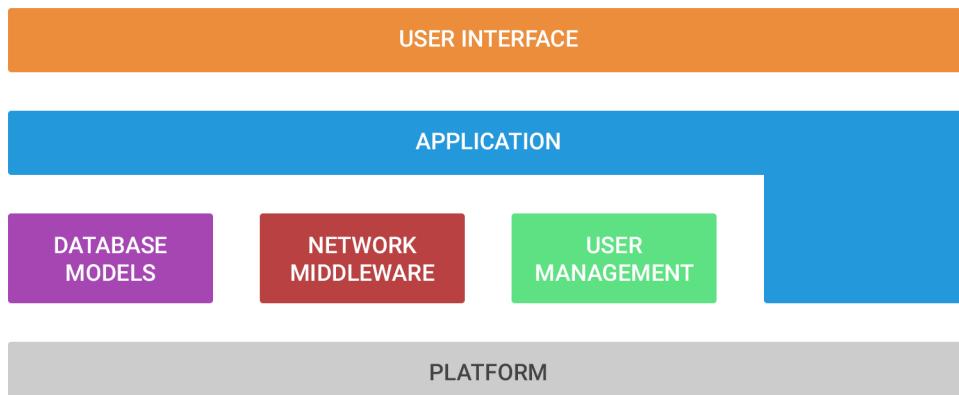


Figura 1.1: Componenti Applicazione

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.

canismi 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*. [?] 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.

Si sono poi realizzati alcuni modelli dei dati presenti nel database, ad esempio ogni gelateria inserita all'interno del circuito MyGelato è stata formalizzata concettualmente come un oggetto *Shop*, sia all'interno del frontend sia sul backend, che incapsula tutte le informazioni disponibili; 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. In questo caso il salvataggio in locale è stato inoltre necessario poichè in molti casi i dati devono poter essere accessibili anche offline per quanto riguarda le operazioni che non richiedono un collegamento istantaneo con il backend.

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.

In figura 1.2 sono visualizzati nel dettaglio i flussi logici dell'applicativo, mostrando da sinistra verso destra quali siano i passaggi necessari per effettuare ogni azione. Si tratta quindi di uno schema che descrive pienamente i componenti su cui si è lavorato durante la fase di progettazione e implementazione.



Figura 1.2: Flussi Logici

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 affetti da basse prestazioni, come ad esempio la mappa di ricerca.

1.1 Interfaccia Grafica

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

In figura 1.3 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 encapsulare tutte le funzioni riguardanti i temi all'interno di un unico oggetto si è creata la classe *Flavors* che rappresenta il tema attuale, ne carica le specifiche 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.

Listato 1.1: Classe Flavors

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

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.

Listato 1.2: FlavorObject

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

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 modo 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.

Listato 1.3: CustomTextView

```
<com.carpigiani.mygelato.custom.CustomTextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...
    app:typefaceAsset="fonts/Pacifico.ttf"/>
```

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.

Listato 1.4: FlavorActivity

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

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 1.3. 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. Si sono rese dinamiche la maggior parte delle immagini in modo che si aggiornassero a partire dal Flavor selezionato durante lo scorrimento.



Figura 1.3: Temi

1.2 Network Middleware

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 semplificata 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* [?] 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 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.

Listato 1.5: Result

```
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;  
    }  
}
```

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*) [?] 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.

1.3 Ricerca

Lo sviluppo della sezione di Ricerca è stato il fulcro del lavoro svolto per questo elaborato, sia perché come componente si pone a nodo centrale di tutta l'applicazione, sia perché in termini di prestazioni si sono dovuti valutare molti aspetti per poter mantenere 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 1.4), 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.

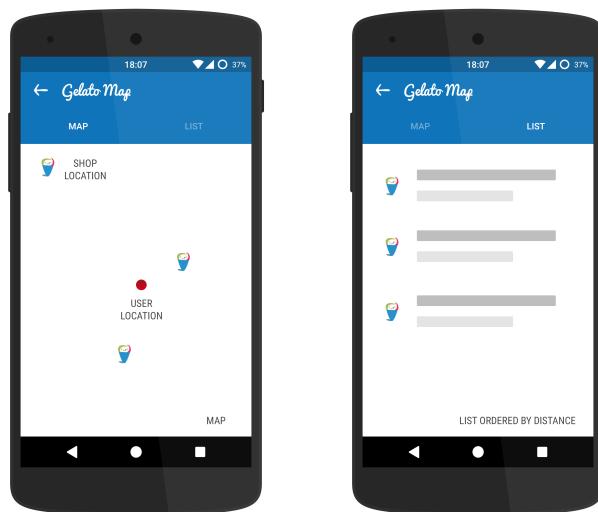


Figura 1.4: Struttura Mappa

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 permettendo di ottenere la posizione precisa 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 1.7.

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. Il sistema di processi è descritto in figura 1.5.

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.

Listato 1.6: *LoadShopAtAsyncTask*

```

private class LoadShopAtAsyncTask
    extends AsyncTask<String , String , Result<> {
        @Override
        protected Result<> doInBackground( String ... params ) {
            Result<> result = Network.loadShopsAt(Map.this , data );
            Realm.copyToRealmOrUpdate(result.result);
            return new Result<>(shops , result.error );
        }
        @Override
        protected void onPostExecute( final Result<> response ) {
            EventBus.getDefault().post(new Event());
        }
    }
}

```

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, portando 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.

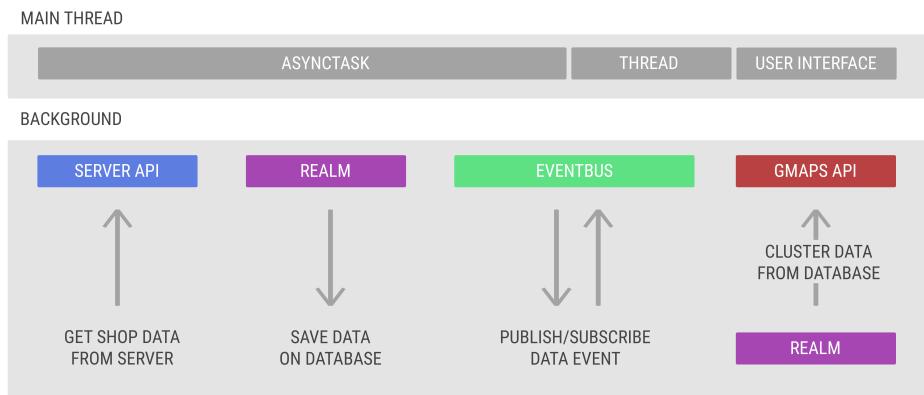


Figura 1.5: Processi

Conclusa la realizzazione 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 e la soluzione è visibile in figura 1.6.

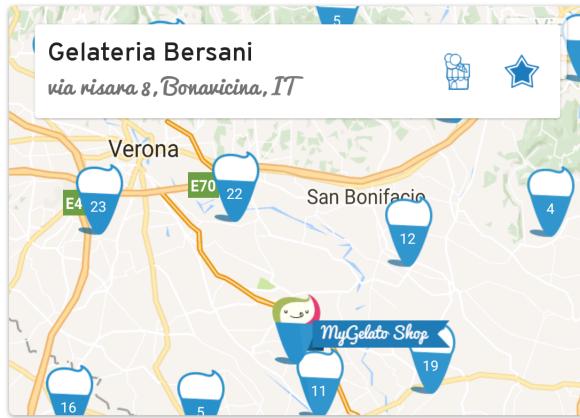


Figura 1.6: Info Window

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 creare 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 siano 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.

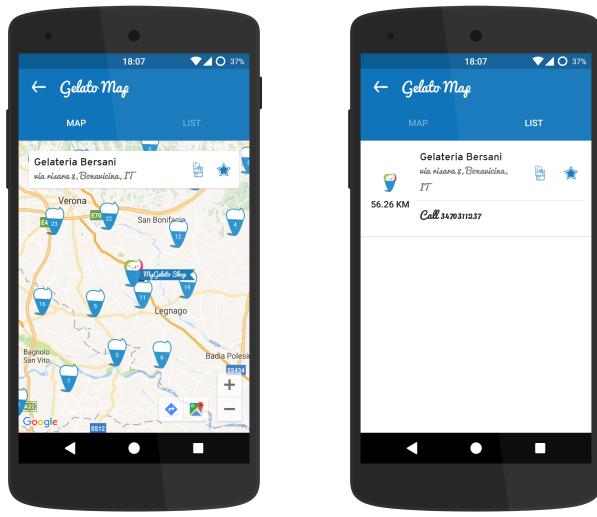


Figura 1.7: Ricerca

1.4 Autenticazione

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 1.7 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.

Listato 1.7: User Model

```
@PrimaryKey
public String uuid;

public String firstName;
public String lastName;
public String email;
public String avatar;
public String client;
public String token;
```

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 restituisca 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 sì 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 1.8 insieme agli elementi presentati in questo capitolo.



Figura 1.8: Sezione Utente

1.5 Marketing Digitale

Il primo flusso logico principale a essere stato implementato è stato quello del marketing digitale poichè legato solo in parte alla presenza di un utente registrato e autenticato 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 visualizzazione 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 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 funzionalità sono spiegate in dettaglio nel capitolo 4.6.2.

1.5.1 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:

Listato 1.8: Modello Card

```
@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;
```

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 perché, 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, questa funzionalità è spiegata in dettaglio nel capitolo successivo.

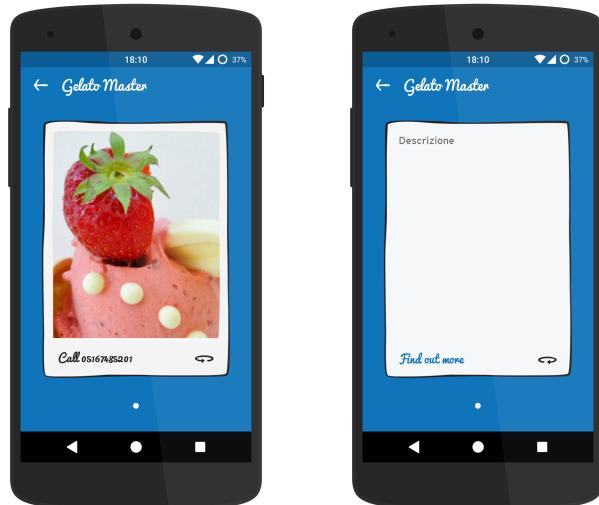


Figura 1.9: Card

1.5.2 Preferiti

Il primo sviluppo legato alla sezione dei preferiti è stato 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 sia 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.

Listato 1.9: Firebase

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

Ogni volta che l'utente aggiungerà o 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. In figura 1.10 infine è visibile l'implementazione della lista degli Shop preferiti insieme ad un esempio di notifica push inviata da MyGalato.

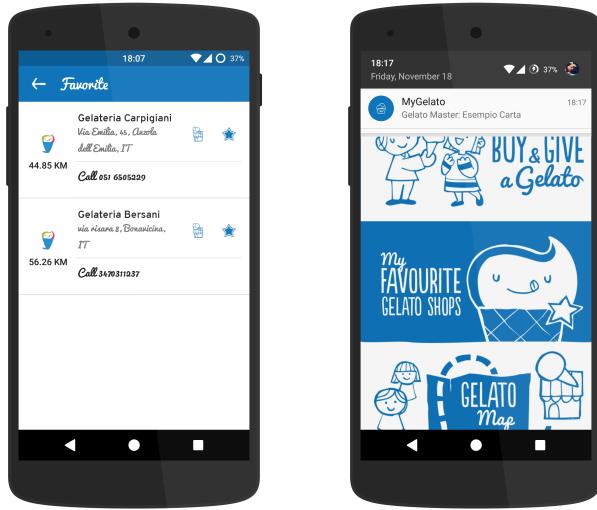


Figura 1.10: Preferiti

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 chilometri 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.

L'applicazione si registra al servizio di Geofencing del sistema una volta che una gelateria viene inserita tra i preferiti, situazione che va ripetuta ogni volta che il dispositivo dell'utente viene riavviato.

1.6 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 per 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 accessibili all'utente le 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.

1.6.1 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 1.11, ogni coupon visualizza alcune informazioni di base e sono presenti i collegamenti diretti per l’utilizzo e lo share tramite altri canali di comunicazione. 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 1.11: Coupon

1.6.2 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 la 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 verrà 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 1.12 dopo l’helper iniziale viene richiesto all’utente di scegliere da quale gelateria egli voglia 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 nella quale richiede il codice identificativo dell’utente rispetto alla libreria di e-commerce Stripe, il quale 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 slider in modo che al centro della schermata vi sia quello scelto dall'utente.



Figura 1.12: Acquisto

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 transazione e aggiungere il coupon a quelli legati all'account utilizzato. Nel caso di errori verrà visualizzato un messaggio e si potrà rientrare nell'operazione, altrimenti se non vi dovessero essere problemi si verrà reindirizzati alla schermata di gestione dei coupon personali.

1.6.3 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 comunicazione 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, come inserito tra le 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 sarà 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 esposta 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 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, 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 nuovamente scaricati i coupon disponibili e l'utente che ha effettuato il riscatto verrà immediatamente reindirizzato sulla schermata di lista dei coupon, in caso di fallimento dell'operazione, verrà invece notificato con un messaggio di errore.

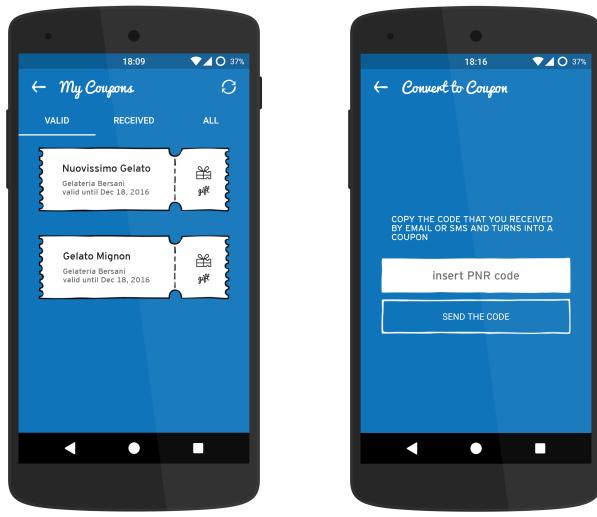


Figura 1.13: Condivisione e Riscatto

1.6.4 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 perché condivisi da altri utenti. In questo processo entrano 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 1.14 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.

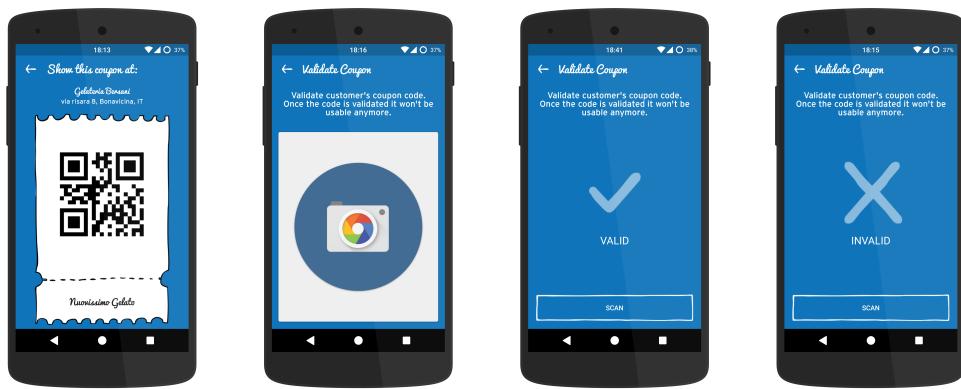


Figura 1.14: Utilizzo e Validazione