POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informatica e Bioingegneria

Corso di Laurea Magistrale in Computer Science and Engineering

Estensione di Android Studio per la generazione automatica di Applicazioni Mobili secondo un approccio model-driven

Relatore: Prof. Luciano BARESI

Tesi di Laurea di:
Nicolas Locatelli Matr. 841601

Anno Accademico 2017-2018

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

I dispositivi mobili sono ormai diventati uno strumento indispensabile nella vita di tutti i giorni e il loro mercato è costantemente in crescita. I colossi di mercato Google ed Apple mettono oggi a disposizione strumenti, chiamati rispettivamente Android Studio ed Xcode, che permettono di creare applicazioni mobili native per le piattaforme Android ed iOS, tuttavia i tempi e i costi di progettazione sono spesso il motivo per cui si rinuncia alla potenza dello sviluppo in codice nativo, in favore di altre soluzioni ibride. Per ovviare a queste carenze della programmazione nativa Apple ha realizzato Storyboard, uno strumento integrato in Xcode che permette di generare un'applicazione mediante un tool grafico, fornendo all'utente una visione d'insieme dell'applicazione e la possibilità di gestire le interazioni fra le varie schermate. Il lavoro svolto in questa tesi, consiste nel creare un'estensione delle funzionalità di Android Studio che permetta di generare applicazioni mobili per la piattaforma Android, traducendo in codice nativo un unico modello astratto dell'applicazione, realizzato semplicemente attraverso un editor grafico: questo approccio, chiamato model-driven, permetterà di generare un'applicazione in linguaggio nativo a partire da un modello grafico iniziale, abbattendo quindi i tempi e i costi di sviluppo del software almeno nelle sue fasi iniziali. Lo strumento realizzato permetterà all'utente di avere una visione d'insieme dell'applicazione sviluppata e di poterne realizzare l'intero flusso di navigazione, andando così a colmare il gap con la relativa controparte del mondo iOS rappresentata da Storyboard. Viene dunque descritto il funzionamento del software realizzato e condotta un'analisi quantitativa su di esso, in un vero processo di sviluppo, per dimostrare quanto questo approccio possa essere decisivo nel ridurre i tempi e i costi della realizzazione di applicazioni mobili native Android.

Abstract

Mobile devices have now become an essential tool in everyday life, and their market is constantly growing. The giants of the market Google and Apple now allow you to create native mobile applications for Android and iOS platforms through tools called respectively Android Studio and Xcode. However, the high development times and costs are often the reasons for missing out on the power of the native code development in favour of other hybrid solutions. To overcome these lacks of native programming Apple developed Storyboard, an Xcode's integrated tool that allows generating an application through a graphic tool that provides the user an overview of the entire application and the possibility to manage the interactions between the various screens. The work done in this thesis is to create an extension of Android Studio functionalities that allows to generate mobile applications for Android platform, translating into native code a single abstract model of the application achieved simply through a graphic editor. This approach, called model-driven, will allow to generate an application in a native language starting from an initial graphic model, thus halving software development times and costs at least in its preliminary stages. The developed tool will allow the user to have an overview of the entire developed application and to be able to realize the entire navigation flow. This will fill the gap with the relative counterpart of the iOS world represented by Storyboard. Is then described the behaviour of the developed software and, finally, it is conducted a quantitative analysis on this in order to demonstrate how much this approach can be decisive in reducing the time and cost of the realization of Android native mobile applications.

Capitolo 1

Introduzione

I dispositivi mobili sono ormai diventati uno strumento indispensabile nella vita di tutti i giorni, riuscendo a compensare i bisogni dei consumatori sotto tutti i punti di vista, dallo svago alla produttività. Il loro strettissimo legame con i social media li ha portati ad essere lo strumento per eccellenza della comunicazione e della condivisione di informazioni a livello mondiale. Sin dai primi anni del nuovo millennio il mercato dei dispositivi mobili è costantemente in crescita. Questo settore di mercato è tutt'oggi dominato dai due principali produttori di smartphone, l'americana *Apple* [1] e il colosso coreano *Samsung* [2], con la crescente imposizione del marchio cinese *Huawei* [3] [4] che negli ultimi anni ha iniziato ad imporsi sul mercato. Per quanto riguarda il lato software, la situazione dei sistemi operativi con cui questi dispositivi sono equipaggiati rimane invariata praticamente dal 2008, con *iOS* [5] di Apple e *Android* [6] di *Google* [7] che dominano incontrastati la scena, lasciando ai competitor *Windows Phone* [8] e *BlackBerry OS* [9] meno del 1% del mercato [10].

In particolare, è proprio Android ad imporsi al primo posto delle classifiche di vendita in tutto il mondo grazie anche agli oltre mille dispositivi equipaggiati. Questo dominio viene attestato anche dal recente (Aprile 2017) sorpasso del sistema operativo di Google ai danni di *Windows* [11] come sistema operativo più utilizzato per la navigazione web, segnando la fine di un dominio che durava dagli anni '80. [12] Questo dato va inoltre a confermare il sempre maggiore predominio dei dispositivi mobili nei confronti dei PC.

Fatte queste premesse, risulta evidente che lo sviluppo di applicazioni mobili sia diventato un settore software all'apice della richiesta di mercato.

Come evidenziato dai dati esposti, le soluzioni più promettenti in termini di ricavi e diffusione sul mercato sono senza dubbio quella di Google ed Apple.

La vera questione in termini di sviluppo sta nello scegliere se sviluppare un'applicazione in linguaggio nativo (Java per Android o Swift per iOS) o adottare un approccio ibrido. Quest'ultima soluzione permette di creare applicazioni mobili mediante tool grafici o simulando la struttura di un'applicazione mobile mediante comuni linguaggi di programmazione rivolti alla programmazione web, come Javascript. Questo approccio riduce notevolmente i costi e il tempo di produzione ma va a incidere negativamente sotto l'aspetto qualitativo garantito da un'applicazione nativa, in quanto il codice generato sarà soltanto un'emulazione di quello nativo e non consentirà di raggiungere lo stesso livello di dettaglio, complessità ed user experience.

In questo elaborato di tesi si è scelto di progettare un sistema in grado di supportare lo sviluppo di applicazioni mobili compatibili con Android, che coniughi i vantaggi dei due approcci nativo ed ibrido. In particolare si è scelto di espandere *Android Studio* [13], lo strumento ufficiale per la programmazione nativa Android, aggiungendo la possibilità di poter generare buona parte del codice sorgente dell'applicazione a partire da un modello realizzato mediante un tool grafico. Questo permette sostanzialmente di andare a guadagnare un'enorme quantità di tempo nelle prime fasi di progetto, andando a colmare il gap temporale rispetto alle soluzioni ibride di cui si è parlato, pur mantenendo un approccio nativo e tutti i vantaggi che esso comporta.

Per quanto riguarda la programmazione di applicazioni iOS esiste già uno strumento del genere integrato in *Xcode* [14], l'IDE ufficiale per lo sviluppo di applicazioni sviluppato da Apple. Questo strumento, chiamato *Storyboard* [15] offre all'utente una visione d'insieme dell'intera applicazione ed ha lo scopo di fornire un modo semplice, grafico e per quanto possibile privo di codice per realizzare le varie schermate dell'applicazione e gestire la navigazione fra di esse. In Android Studio è presente un editor grafico per la realizzazione del layout di una singola schermata ma non esiste nulla che permetta di gestire il flusso di navigazione dell'applicazione o di generare del codice per via grafica.

Prendendo ispirazione da Storyboard questo progetto di tesi si pone l'obiettivo di portare questo tipo di programmazione anche in ambiente Android, consentendo all'utente non solo di poter generare buona parte di codice attraverso un tool grafico, ma anche di avere una visione d'insieme dell'applicazione e di poterne definire graficamente il flusso di navigazione e le interazioni fra le varie schermate.

Di seguito è riportata una breve descrizione del contenuto di ciascun capitolo della tesi:

- Il *secondo capitolo* descrive il contesto attuale dello sviluppo di applicazioni mobili per il sistema operativo Android, a partire dai tool di prototipizzazione interattiva, fino agli strumenti per sviluppare applicazioni in codice nativo o secondo un approccio ibrido.

- Il *terzo capitolo* illustra l'architettura software del plugin realizzato, in grado di generare un modello grafico di un'applicazione mobile e di tradurlo in codice nativo.

- Il *quarto capitolo* presenta un'analisi quantitativa dello strumento che è stato sviluppato, descrivendone l'utilizzo nello sviluppo di una reale applicazione mobile e mettendo in evidenza quanto codice di quella applicazione è stato possibile generare automaticamente.

- Il *quinto capitolo* infine, riporta le considerazioni conclusive sul lavoro svolto e illustra i possibili sviluppi futuri.

Capitolo 2

Contesto generale

In questo capitolo viene presentato il contesto generale in cui si trovano ad operare gli sviluppatori di applicazioni per dispositivi mobili. Verranno brevemente introdotti gli strumenti di supporto alla fase di prototipizzazione, concentrandosi in modo particolare sugli strumenti di effettiva utilità per la successiva fase di sviluppo. Dopodiché verranno prese in analisi alcune delle migliori possibilità presenti sul mercato per lo sviluppo vero e proprio. Si analizzerà la realizzazione di applicazioni mediante codice nativo, mediante applicazioni di tipo ibrido e la realizzazione di app mobili tramite tool grafici, senza generazione di codice da parte dello sviluppatore, tenendo conto sia della capacità di garantire uno sviluppo rapido ed ottimizzato per la piattaforma Android sia della qualità e delle prestazioni del risultato finale.

2.1 Stato dell'arte

Dopo aver determinato i requisiti del software ed averne definito accuratamente l'architettura e l'interazione delle varie componenti nella fase di design di un'applicazione, incomincia la fase di sviluppo vero e proprio. Generalmente la stesura del codice sorgente viene preceduta da una fase intermedia detta di prototipizzazione. Questo processo consiste nella realizzazione di una versione dimostrativa dell'applicazione in grado di simularne i principali aspetti, dal comportamento dell'interfaccia grafica alla navigazione dei contenuti. In questa sezione viene mostrato lo stato dell'arte del processo di sviluppo.

2.1.1 Applicazioni di prototipizzazione

Le applicazioni di prototipizzazione sono uno strumento fondamentale nella progettazione di un'applicazione di successo. Seguendo la scia di Apple, che fa da sempre dell'interfaccia grafica un suo valore fondante, anche Google ha ormai da anni assunto una linea guida per stabilire i pattern grafici che un applicazione Android dovrebbe sempre rispettare, il cosiddetto *Material Design* [16].

Di seguito verranno riportati i più famosi tool di prototipizzazione per applicazioni mobili presenti sul mercato.

FluidUI

FluidUI è un'applicazione web che permette lo sviluppo di un prototipo di applicazione mobile attraverso editor grafico. Tramite l'uso di widgets l'utente può facilmente creare un certo numero di schermate e decorarle con i controlli messi a disposizione dall'applicazione. Inoltre è possibile gestire la navigazione fra le varie schermate tramite bottoni o gestures. Una volta realizzato il prototipo, FluidUI permette di simulare il comportamento finale dell'applicazione mediante un web player oppure di esportare le varie immagini delle relative schermate.

Il piano gratuito prevede un limite di un progetto di 10 schermate con alcune limitazioni come ad esempio la non abilitazione all'esportazione delle immagini delle schermate. Il piano più economico invece comporta un abbonamento di 15$/mese e consente la creazione di 3 progetti con numero illimitato di schermate e tutte le funzionalità abilitate. [17]

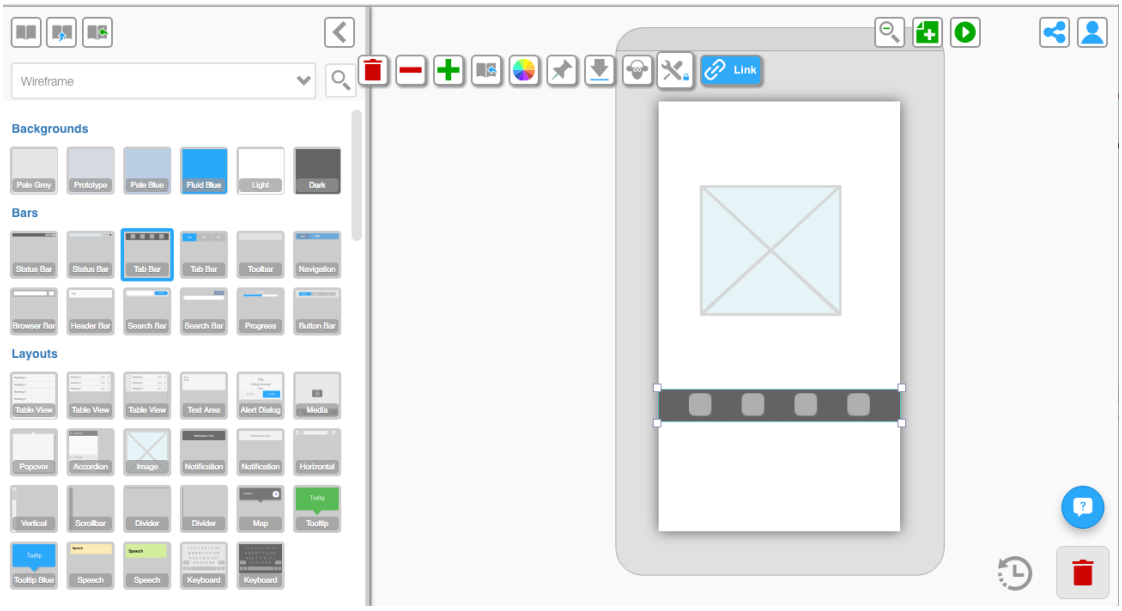

Figura 2.1: Interfaccia grafica di FluidUI

Proto.io

Anche Proto.io è realizzato sotto forma di web app ed ha un funzionamento simile al tool precedentemente descritto. Infatti, anche in questo caso, l'utente può creare un certo numero di schermate del prototipo mediante widgets e decorarle con i controlli messi a disposizione dall'editor, mentre la gestione della navigazione può essere gestita tramite bottoni e gestures. L'elemento caratteristico di questo tool di prototipizzazione risiede nella possibilità di simulare il comportamento del prototipo direttamente sul proprio dispositivo mobile, grazie all'app mobile di Proto.io che una volta installata su smartphone è in grado di riprodurre il prototipo generato via web. Rimane possibile scaricare le immagini delle varie schermate.

È disponibile un piano di prova gratuita di 15 giorni, mentre l'abbonamento più economico ha un costo di 29$/mese e permette di gestire fino a 5 progetti differenti. [18]

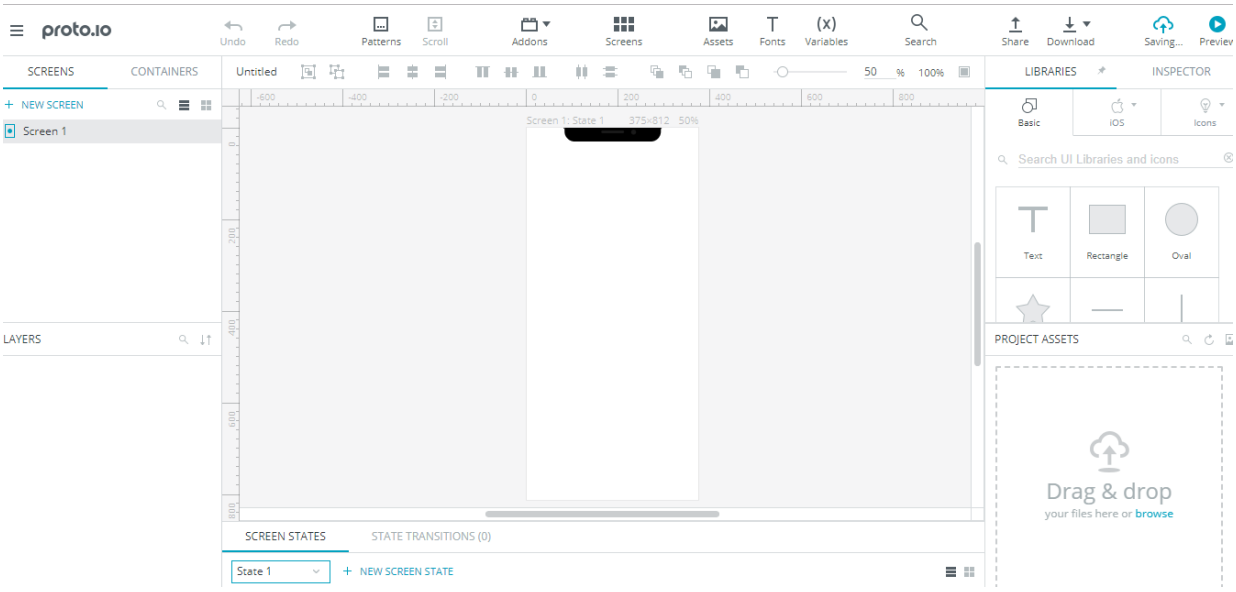

Figura 2.2: Interfaccia grafica di Proto.io

InVision App

Anche in questo caso ci troviamo di fronte ad un'applicazione di prototipizzazione disponibile via web. A differenza delle precedenti, InVision App adotta un approccio completamente differente: l'utente può solamente caricare delle immagini precedentemente realizzate tramite apposita applicazione di manipolazione grafica e posizionarle all'interno delle varie schermate.

Successivamente sarà possibile selezionare alcune aree all'interno di ogni specifica schermata e trasformarle in collegamenti che permettono di navigare tra i contenuti e rendere così interattivo il prototipo. La simulazione del comportamento dell'applicazione di cui si è costruito il prototipo può avvenire sia tramite web player sia mediante l'applicazione di InVision App direttamente scaricabile sul proprio smartphone.

La qualità finale del prototipo realizzato con questo tipo di applicazioni dipende molto dall'abilità dell'utente nell'utilizzo di software indipendenti di elaborazione delle immagini, in quanto questi tool di prototipizzazione non comprendono strumenti di manipolazione grafica ma offrono soltanto un supporto nel dimensionamento delle immagini che vengono inserite nelle varie schermate.

Per questo tool è disponibile un periodo gratuito di prova di 7 giorni mentre il piano più economico costa 15$/mese e permette di gestire al massimo 3 differenti progetti. [19]

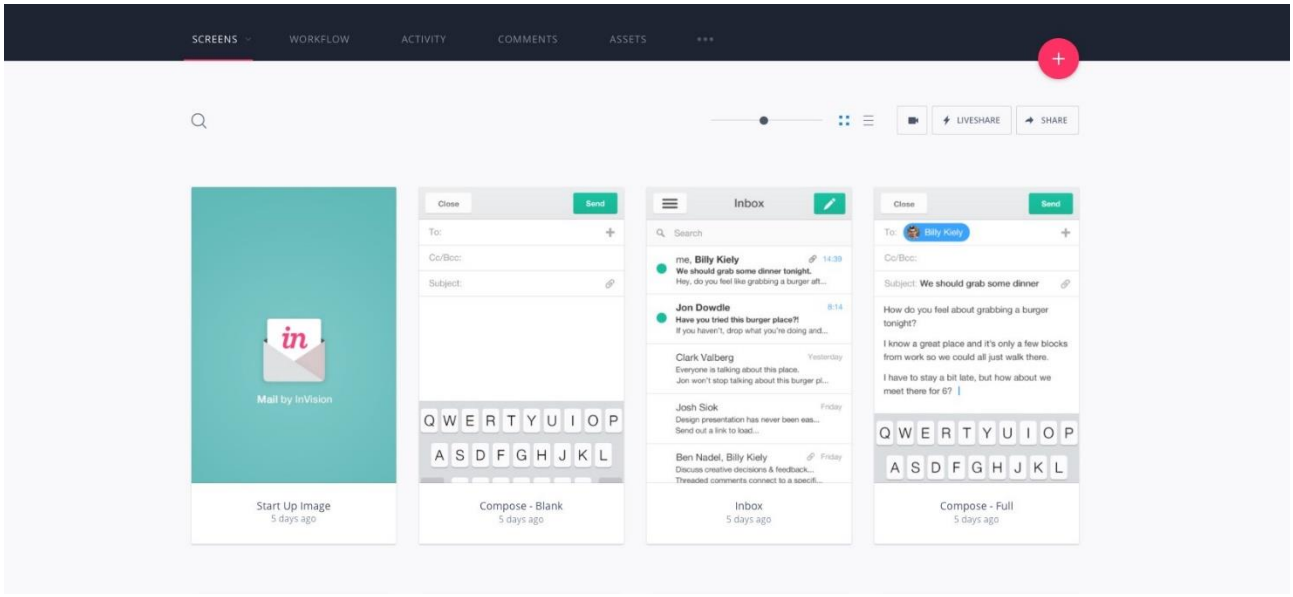

Figura 2.3: Interfaccia grafica di InVision App

Marvel App

Un'altra applicazione di prototipizzazione disponibile in forma di web app è Marvel App. Quest'ultima permette all'utente sia di decorare le varie schermate per mezzo dei vari widget e controlli messi a disposizione, sia di creare le varie schermate servendosi di immagini precedentemente elaborate con altri strumenti.

Anche questo tool offre la possibilità di configurare le gestures e i click dei vari componenti del prototipo per simularne il flusso di navigazione. Marvel App è disponibile su tutti i tipi di dispositivi mobili e permette quindi non solo di poter simulare il proprio prototipo su smartphone ma addirittura di sviluppare tutta la fase di prototipizzazione direttamente da dispositivo mobile.

Marvel App offre un periodo di prova che consente la realizzazione di due progetti, mentre il piano più economico è di 16$/mese e permette la realizzazione di un numero illimitato di progetti. [20]

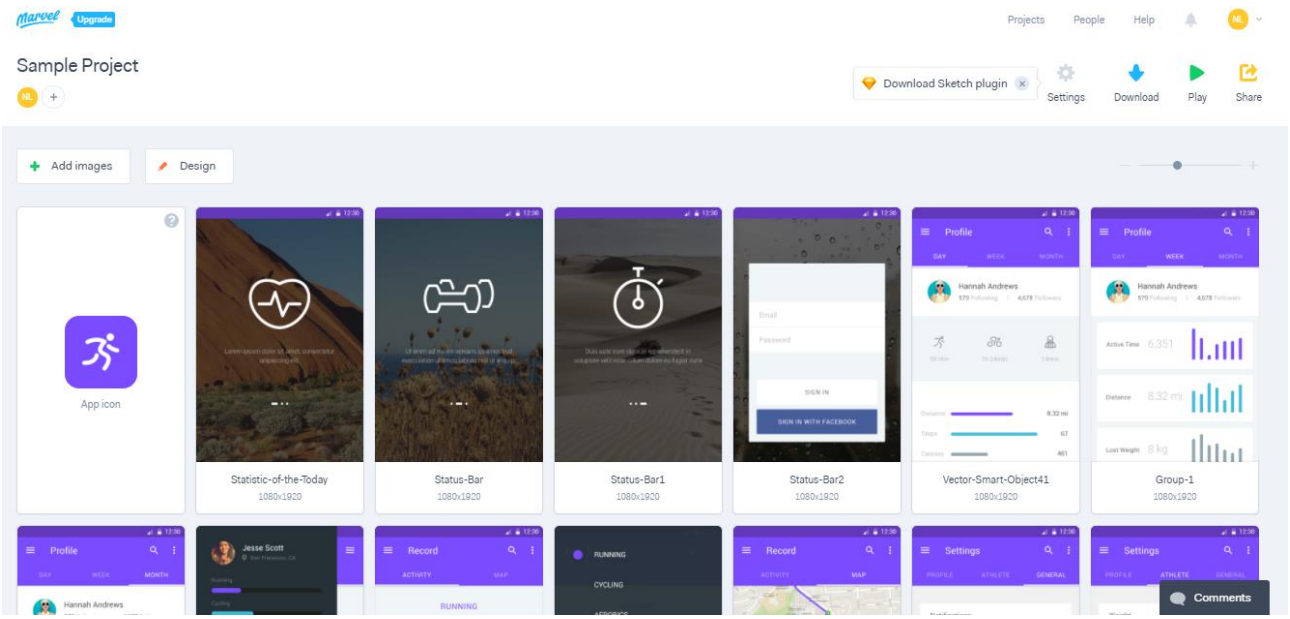

Figura 2.4: Interfaccia grafica di Marvel App

Allo stato attuale dell'arte questo genere di applicazioni, indipendentemente dall'approccio basato sulla composizione via widget o sul caricamento di immagini, rimane esclusivamente destinato alla fase di prototipizzazione, senza poter dare alcun contributo concreto nella stesura del codice. Infatti, nonostante un buon livello di interattività, non è possibile estrarre dai prototipi creati alcun tipo di informazione che possa essere riutilizzata automaticamente nella fase di sviluppo del software.

2.1.2 Applicazioni per sviluppatori

Le applicazioni per sviluppatori costituiscono il cuore della progettazione di un'applicazione mobile: indipendentemente dal fatto che sia stato sviluppato o meno un prototipo, esse rappresentano lo strumento tramite il quale avviene la stesura vera e propria del codice sorgente e sono solitamente in grado di supportare tutta la fase di sviluppo del software, dalla generazione del codice al building dell'applicazione finale. Di seguito verranno analizzate le varie soluzioni disponibili sul mercato, descrivendone la tecnologia utilizzata, il grado di competenze richieste all'utente e la qualità del prodotto finale. In particolare verranno descritte applicazioni che consentono la realizzazione di app mobili mediante stesura di codice nativo, applicazioni ibride e applicazioni che permettono di sviluppare applicazioni mobili senza scrittura di codice.

2.1.2.1 Applicazioni native

Vengono realizzate mediante software che consentono lo sviluppo di applicazioni mobili mediante stesura di codice nativo. Questi strumenti sono presenti sul mercato sotto forma di IDE (Integrated Development Environment). Gli IDE costituiscono il cosiddetto ambiente di sviluppo del software e sono in grado di fornire al programmatore tutto ciò che serve per la stesura di codice sorgente: un editor di testo, un compilatore, un tool di building automatico e solitamente un debugger. Gli IDE più adatti allo sviluppo di applicazioni mobili native sono sicuramente *Eclipse* [21] e Android Studio. In particolare Android Studio è l'IDE ufficiale realizzato da Google che nasce appositamente con l'obiettivo di diventare lo strumento per eccellenza nello sviluppo di applicazioni mobili Android. Esso permette di realizzare progetti per smartphone e tablet, nonché per dispositivi indossabili, Android Auto e Android TV. Oltre ad offrire tutti gli elementi caratteristici di un ambiente di sviluppo, mette a disposizione dell'utente alcune funzionalità integrate interamente dedicate a facilitare la realizzazione di applicazioni mobili. Una di queste è sicuramente AVD (Android Virtual Device) Manager, grazie al quale l'utente può creare e gestire simulatori di alcuni dei più famosi dispositivi reali in circolazione sul mercato. Questi dispositivi virtuali consentono di testare il funzionamento dell'applicazione creata proprio come se fosse eseguita su un dispositivo reale.

Per quanto riguarda le singole schermate dell'applicazione, Android Studio mette a disposizione dei template per la creazione automatica delle schermate più diffuse.

Con questa funzionalità è possibile generare automaticamente il codice base di questo tipo di schermate, che lo sviluppatore potrà poi andare ad implementare a seconda delle esigenze e dei contenuti desiderati. È inoltre presente un editor grafico che tramite funzionalità drag-and-drop supporta l'utente nella creazione del layout delle singole schermate.

La grande espansione dei dispositivi mobili ha contribuito allo sbarco degli IDE anche su questo tipo di piattaforme. Il più utilizzato e adatto allo sviluppo di applicazioni Android è sicuramente *AIDE* [22]. Quest'ultimo, scaricabile gratuitamente da Play Store, consente lo sviluppo di applicazioni mobili native in linguaggio Java e mette a disposizione del programmatore buona parte degli strumenti forniti dai comuni IDE per pc. Sono disponibili alcune funzionalità addizionali a pagamento che permettono di incrementare le potenzialità di sviluppo di questo strumento, come ad esempio funzionalità grafiche per la personalizzazione dell'interfaccia grafica dell'applicazione sviluppata.

Scegliere di sviluppare un'applicazione nativa consente al programmatore di poter sfruttare a pieno le potenzialità messe a disposizione da Android, configurando a suo piacimento ogni aspetto del prodotto finale come l'aspetto grafico, la struttura e funzionalità di ogni schermata, l'interazione con servizi esterni, l'intero flusso di navigazione e molto altro. In termini di risultato finale questa soluzione rappresenta quindi il massimo che si possa ottenere. D'altra parte, tutto ciò genererà tempi e costi di sviluppo più elevati rispetto alle altre soluzioni disponibili e richiederà delle maggiori competenze agli sviluppatori che dovranno avere una buona conoscenza del linguaggio nativo e dell'ambiente di sviluppo.

2.1.2.2 Applicazioni ibride

Un'altra soluzione disponibile sul mercato per la realizzazione di applicazioni mobili è quella delle applicazioni ibride. Anche in questo caso il software di sviluppo di riferimento sono gli IDE ma a differenza delle applicazioni native le applicazioni ibride non vengono sviluppate in codice nativo Java. Queste applicazioni sono spesso realizzate tramite tecnologie che ne permettono l'esecuzione su qualunque tipo di dispositivo mobile, sia esso Android, iOS o qualunque altro sistema in grado di riprodurre contenuti web. Un'applicazione ibrida nasce originariamente come una normale web application HTML5, che viene successivamente incapsulata in un generico involucro di contenuti web, la cosiddetta Web View, disponibile nativamente su tutti i dispositivi mobili.

In pratica è come se il dispositivo mostrasse una normale pagina web costruita tramite le tradizionale tecnologie HTML5, CSS3 e Javascript.

Il flusso di navigazione in questo caso viene soltanto simulato, infatti solitamente queste web app vengono progettate seguendo il paradigma SPA (Single Page Application), ovvero viene simulata la navigazione dei contenuti rimpiazzando gli elementi presenti nella pagina ogni volta che viene cliccato un link, dando la sensazione di navigare all'interno di una vera e propria applicazione piuttosto che su un normale sito web.

Tutto questo è possibile grazie a librerie Javascript come *AngularJS* [23]. Per quanto riguarda lo sviluppo software vero e proprio, per l'incapsulamento di una web application all'interno di una generica WebView, il componente di riferimento sono i framework. Il più celebre e diffuso dei framework per lo sviluppo di applicazioni mobili ibride è *Apache Cordova* [24]. Questo strumento non si limita ad impacchettare la web app in un contenitore differente dal semplice browser ma fornisce delle API (Application Programming Interface) che lo sviluppatore può usare per accedere ad una serie di funzionalità esclusive di un dispositivo mobile, come l'accesso in memoria, alla fotocamera o ai vari sensori presenti sullo smartphone.
Questo tipo di approccio riduce considerevolmente i costi e i tempi di sviluppo, grazie anche alla sua natura multi piattaforma che consente di creare una sola applicazione compatibile con più sistemi operativi. Tuttavia, essendo l'applicazione soltanto simulata all'interno di una web app, la qualità del risultato finale seppur buona sarà notevolmente inferiore rispetto a quella garantita dall'approccio nativo.

2.1.2.3 Applicazioni generate senza scrittura di codice

Questo genere di applicazioni è stato ideato per consentire anche agli utenti meno esperti di poter sviluppare una propria applicazione mobile. Solitamente sono realizzate sotto forma di web application e sono dotate di tool grafici in grado di guidare e supportare l'utente durante tutto il processo di creazione. Molti passaggi che nelle precedenti soluzioni erano a carico dell'utente vengono ora automatizzati, lasciando a quest'ultimo solo il compito di assemblare il layout e la composizione delle varie schermate della sua applicazione, il tutto senza che l'utente debba scrivere nemmeno una sola riga di codice.

Di seguito verranno descritte alcuni dei principali strumenti per lo sviluppo di applicazioni mobili senza scrittura di codice presenti sul mercato.

Andromo

Andromo è realizzato sotto forma di applicazione online e consente all'utente di svolgere l'intero processo di sviluppo senza dover scrivere del codice. Vengono messi a disposizione dell'utente diversi stili e strumenti per creare e personalizzare il layout dell'applicazione. Le schermate che possono essere inserite nel progetto sono di tipi preimpostati che comprendono per lo più schermate per la visualizzazione di contenuti multimediali o di pagine web. Non è possibile gestire il flusso di navigazione tra le varie schermate, esse sono accessibili o attraverso il click di icone poste nella schermata principale o attraverso un Navigation Drawer. Una volta terminato lo sviluppo Andromo procede con il building automatico e dopo pochi minuti invia all'utente via email il file dell'applicazione pronto per essere installato su smartphone.

È disponibile una prova gratuita che consente la realizzazione di una sola applicazione con l'aggiunta automatica di inserti pubblicitari di Andromo, mentre il piano più economico costa 8$/mese. [25]

Appy Pie

Si tratta di una delle più famose e quotate soluzioni per lo sviluppo di applicazioni mobili senza scrittura di codice ed è disponibile sotto forma di web app. L'utente può scegliere di realizzare la propria applicazione da zero oppure di avvalersi di un set iniziale di applicazioni preformate divise per categoria che possono essere usate come base di partenza per lo sviluppo. Vengono inoltre messi a disposizione diversi layout con cui personalizzare l'aspetto grafico dell'applicazione. Per quanto riguarda le schermate, Appy Pie fornisce un buon numero di contenuti. Come nel precedente tool, anche in questo caso sono configurabili schermate relative a contenuti multimediali, elenchi di contatti, pagine web o pagine di social network; per quanto riguarda i contenuti più avanzati invece Appy Pie si basa su servizi e tool esterni, che oltre a richiedere delle buone conoscenze tecniche, sono spesso a pagamento e privi degli adeguati sistemi di tutorial. Anche in questo caso non sono presenti strumenti per la definizione del flusso di navigazione.

Il funzionamento dell'applicazione creata può essere testato sia attraverso un simulatore online che consente anche di riposizionare gli elementi delle varie schermate tramite funzionalità drag-and-drop, sia scaricando direttamente il file dell'applicazione.

È disponibile un piano gratuito di 48 ore che consente di creare applicazioni eseguibili solo mediante browser proprietario, mentre per poter creare applicativi compatibili con Android è necessario un abbonamento di 15€/mese. [26]

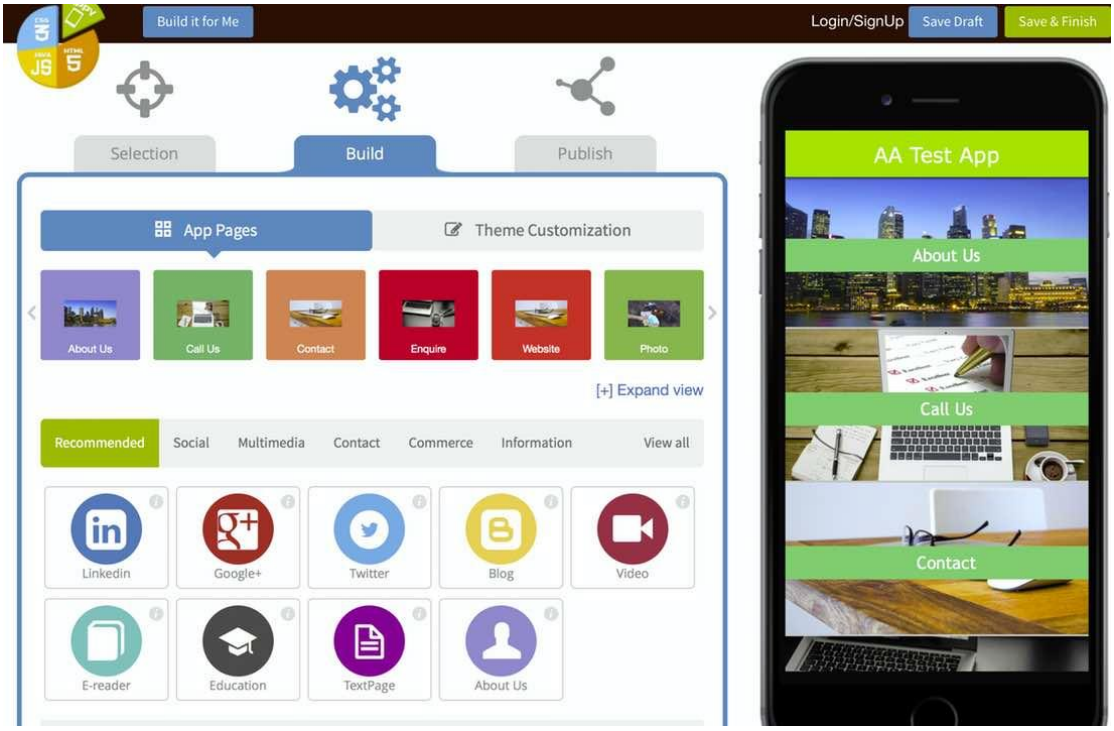

Figura 2.5: Interfaccia grafica di AppyPie

Buzztouch

Questa web application mette a disposizione un'interfaccia più povera e meno intuitiva rispetto agli strumenti precedentemente descritti. Le configurazioni di layout disponibili sono essenziali e per lo più i contenuti consistono in immagini elaborate e caricate dall'utente. Per quanto riguarda la decorazione delle schermate, vengono messi a disposizione contenuti base come liste e griglie di elementi testuali o bottoni, menu, schermate di login, email, sms, oltre alle solite schermate che rimandano a contenuti online (video di YouTube, pagine web, social network). È possibile raggruppare le schermate in una Tabbed Activity, anche se solo parte dei contenuti disponibili per la

decorazione delle schermate è compatibile con questo tipo di struttura. Non è invece presente la possibilità di configurare il flusso di navigazione.

La caratteristica che più di tutti distingue questo tool da quelli precedentemente descritti è rappresentata dal fatto che al termine dello sviluppo invece del file dell'applicazione pronto per essere installato ed eseguito viene restituito il codice nativo dell'applicazione. Questo dà modo agli utenti più esperti di poter andare a integrare e modificare il codice a loro piacimento, mentre per i meno esperti il fatto di dover eseguire in modo autonomo il building dell'applicazione può rappresentare un notevole ostacolo.

Buzztouch offre una versione di prova di un mese che consente lo sviluppo di tre progetti. Per avere a disposizione il prodotto completo occorre invece un abbonamento di 15$/mese a cui vanno aggiunte ulteriori spese per eventuali componenti avanzati per la personalizzazione delle schermate. [27]

Questo genere di applicazioni permette ad un più ampio numero di sviluppatori di poter creare applicazioni mobili, non richiedendo specifiche conoscenze di linguaggi di programmazione. Anche i tempi di sviluppo vengono notevolmente ridotti, a discapito di una qualità notevolmente inferiore sia rispetto all'approccio nativo sia a quello ibrido. Non è infatti possibile programmare nessun tipo di navigazione fra i contenuti e le funzionalità a disposizione sono ristrette a quelle predefinite dallo strumento di sviluppo e difficilmente personalizzabili. Il prodotto finale così realizzato sfrutterà solo una piccola parte delle potenzialità messe a disposizione dal sistema operativo Android.

2.2 Obiettivo della tesi

Stabilito che non sia possibile in alcun caso superare la qualità né tantomeno le prestazioni di un'applicazione nativa, ci si chiede se sia possibile ridurre il tempo necessario allo sviluppo di un applicativo Android, visto che proprio i tempi e i costi di sviluppo costituiscono i difetti di questo tipo di programmazione. L'obiettivo di questa ricerca è quello di creare uno strumento di supporto allo sviluppo di un'applicazione, che consenta di coniugare i vantaggi qualitativi e prestazionali dell'approccio nativo con la semplicità realizzativa e l'abbattimento dei tempi di sviluppo garantito da approcci meno performanti come i tool di generazione di applicazioni senza scrittura di codice.

Per realizzare uno strumento del genere verrà utilizzato un approccio basato su tecniche di modellazione, o Model Driven Development. Si tratta di un'alternativa alla programmazione tradizionale che non prevede la scrittura di codice sorgente da compilare o eseguire, bensì consiste nella realizzazione di un modello del sistema software che si vuole sviluppare. Il modello viene realizzato tramite appositi strumenti visuali e, partendo da questi, viene automaticamente generato il codice sorgente in linguaggio nativo.

Sul mercato esistono già soluzioni di questo tipo, la più nota è *Neonto Studio* [28]: questa applicazione disponibile solamente su OSX è in grado di generare automaticamente un progetto Android Studio (e/o un progetto Xcode) a partire da un semplice prototipo realizzato graficamente. Esso supporta una buona quantità di widgets con cui decorare le varie schermate dell'applicazione. Tuttavia lo sviluppo mediante questo strumento è focalizzato sulle singole schermate dell'applicazione e non fornisce una visione d'insieme delle varie schermate dell'app e delle interazioni fra esse. Visione d'insieme che invece viene garantita dal già citato Storyboard. Questo strumento integrato con Xcode permette di assemblare le varie schermate dell'applicazione in maniera grafica, consentendo di avere una visione d'insieme dell'applicazione e il controllo dell'intero flusso di navigazione potendo configurare ogni interazione fra le varie schermate. Una volta generato il modello grafico dell'applicazione, questo verrà automaticamente tradotto in codice nativo. Essendo questo strumento integrato nell'IDE, una volta ricavato il codice dal modello, l'utente può direttamente modificarlo, simularne le funzionalità attraverso il simulatore oppure procedere direttamente con il building.

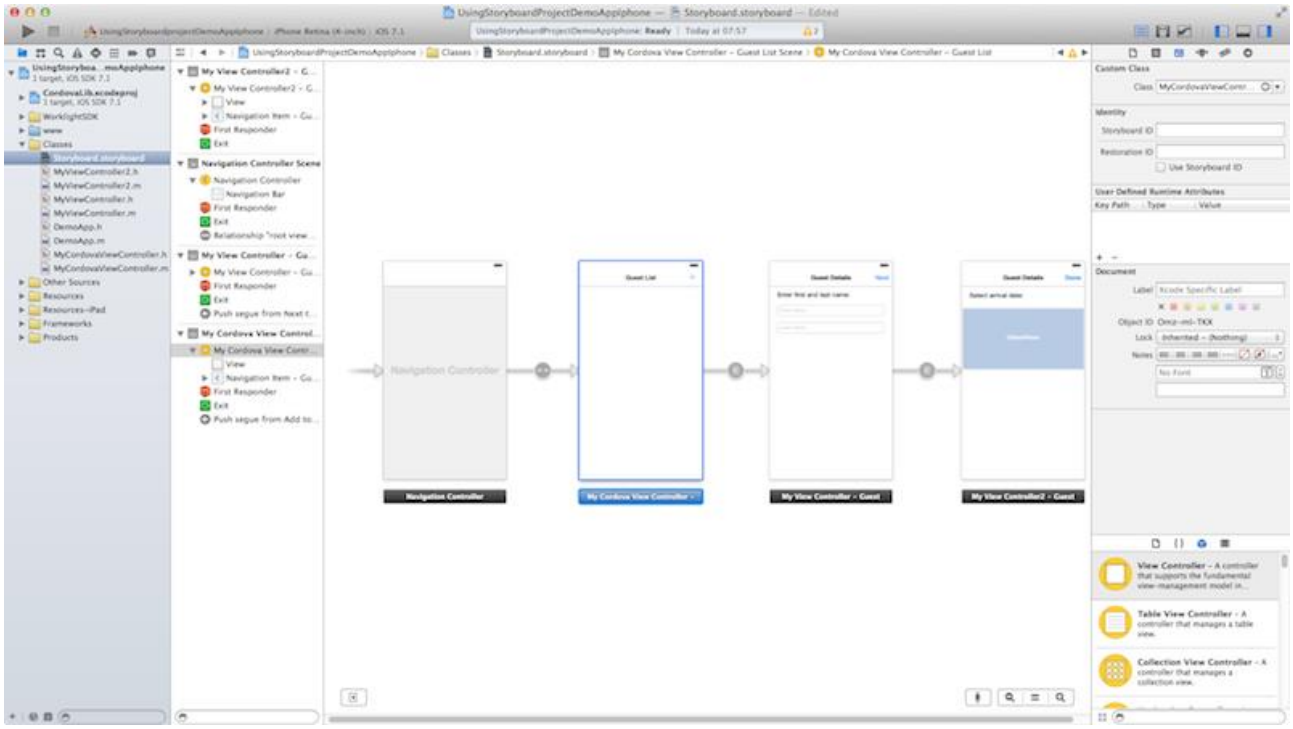

Figura 2.6: Interfaccia grafica di Storyboard

Prendendo come punto di partenza le novità di mercato introdotte in iOS da Storyboard, in questo progetto di tesi si andranno a estendere le funzionalità dell'ambiente di sviluppo Android Studio ottenendo così un prodotto che integri la semplicità della composizione grafica di un'applicazione mobile e la riduzione dei tempi di sviluppo, tipiche dei software di prototipizzazione e di sviluppo di applicazioni senza utilizzo di codice, con la qualità di performance garantita dall'approccio nativo; introducendo come elemento di novità assoluta per quanto concerne gli strumenti di sviluppo, la possibilità di avere una visione d'insieme del prodotto finale e di poterne gestire interamente il flusso di navigazione, andando così a colmare quanto più possibile il gap di Android Studio con la controparte del mondo iOS.

2.3 Soluzione proposta

La soluzione proposta in questo elaborato di tesi consiste nell'estensione dell'ambiente di sviluppo ufficiale per applicazioni mobili native Android, Android Studio, al fine di renderlo uno strumento ancor più competitivo ed unico nel suo genere, in grado di sopperire il più possibile ai maggiori punti di debolezza della programmazione nativa. Questa estensione è stata realizzata sotto forma di plugin, un programma non autonomo che interagisce con un altro programma al fine di estenderne o ampliarne le funzionalità originarie. Il plugin sviluppato sarà di tipo *tool-integration*; questa tipologia di plugin consente di manipolare strumenti e componenti di terze parti direttamente dall'IDE senza cambiare contesto. In particolare verrà implementata una nuova azione tramite il quale sarà possibile lanciare l'applicazione sviluppata. Questa azione sarà accessibile all'utente sotto forma di una nuova icona aggiunta alla toolbar di Android Studio.

Al lancio dell'azione del plugin verrà aperta un'applicazione composta da un tool grafico che permetterà all'utente di costruire un grafo (modello) delle varie schermate che comporranno l'applicazione che si intende sviluppare. Il modello potrà essere assemblato, mediante funzionalità di drag-and-drop, semplicemente trascinando col mouse all'interno dell'area di disegno le icone relative alle varie tipologie di schermate messe a disposizione. Le varie schermate messe a disposizione dell'utente, comunemente denominate activities nell'ambiente della programmazione Android, rappresenteranno non solo le schermate più comuni ma anche dei pattern più complessi. La realizzazione di tali pattern, benché molto diffusi nella programmazione Android, richiederebbe la stesura di codice nativo abbastanza elaborato da parte dei programmatori il che comporterebbe una notevole perdita di tempo. Col nostro strumento lo sviluppatore sarà in grado di inserire tali pattern semplicemente trascinando la relativa activity all'interno del grafo, il sistema penserà da solo a realizzare il codice dell'intera struttura del pattern.

Il tool grafico permetterà anche la progettazione dell'intero flusso di navigazione. Basterà infatti tracciare un link tra due activity per realizzare un flusso di navigazione fra esse, ovvero quello che in Android viene definito intent. Verranno messi a disposizione dell'utente diverse tipologie di intent con cui poter collegare le varie schermate, dai più comuni a soluzioni specifiche per ogni singola activity. I parametri di base relativi alle varie schermate e collegamenti potranno essere configurati mediante un attribute inspector integrato nel tool.

Una volta realizzato il modello l'utente avrà a disposizione una visione grafica completa dell'intera applicazione mobile progettata.

Per completare le funzionalità messe a disposizione dalla nostra estensione manca ancora un elemento fondamentale, quel tassello mancante che permette di collegare fra loro il processo di modellazione grafica con quello della codifica nativa. Questo elemento sarà rappresentato dal generatore di codice. Il generatore di codice avrà una funzionalità di tipo *model2text* ed andrà quindi a tradurre il modello grafico costruito restituendo il relativo codice nativo che sarà automaticamente aggiunto al progetto attualmente aperto nell'IDE.

Il risultato finale dovrà produrre uno strumento unico del suo genere in grado di ampliare ulteriormente le potenzialità del miglior ambiente di sviluppo disponibile in ambito Android e che permetta allo sviluppatore di poter generare per via grafica e con pochi click il codice relativo alla struttura base dell'intera applicazione mobile che intende realizzare, senza dover rinunciare alle potenzialità e alle funzioni messe a disposizione dall'IDE.

Capitolo 3

Architettura software

In questo capitolo verrà descritta l'architettura software del plugin realizzato per questo progetto di tesi. Inizialmente verrà fornita una panoramica sui software e le librerie utilizzate, a cui farà seguito un'analisi dettagliata della struttura del progetto. Successivamente si andrà ad analizzare l'architettura software vera e propria, descrivendone il processo di costruzione del modello grafico e di generazione del codice nativo.

3.1 Software e librerie utilizzati

L'applicazione realizzata consiste in un plugin che va ad implementare le funzionalità di Android Studio. Per la realizzazione di questo progetto è stato quindi necessario utilizzare degli strumenti che ne consentissero l'integrazione con questo IDE. Inoltre, dato l'approccio model-driven dell'applicazione sviluppata e le funzionalità di *drag-and-drop* presenti, sono stati utilizzati strumenti per la definizione dell'interfaccia grafica forniti dal linguaggio Java. In seguito verranno descritti nel dettaglio tutti gli strumenti software e le librerie utilizzate

IntelliJ IDEA

IntelliJ IDEA [29] è l'ambiente di sviluppo integrato (IDE) per il linguaggio di programmazione Java ideato da *JetBrains* [30] ed è stato utilizzato da Google come base per lo sviluppo di Android Studio. Oltre a fornire tutte le funzionalità tipiche di un IDE esso mette a disposizione dell'utente gli strumenti necessari per la realizzazione di un plugin compatibile con un qualsiasi prodotto

derivato dai software JetBrains. Tutte queste funzionalità per lo sviluppo di un plugin sono messe a disposizione dal *Plugin Development Kit*, già incorporato nelle recenti versioni dell'IDE. Grazie a questo componente sarà possibile specificare tutte le caratteristiche del nostro plugin, come il nome, i software supportati, l'icona, le informazioni sullo sviluppatore, la versione e la definizione della classe java che andrà ad implementarne l'azione. Tutte queste caratteristiche rendono IntelliJ IDEA l'ambiente di sviluppo ideale per la programmazione del nostro plugin di estensione di Android Studio.

Per la realizzazione di questo progetto di tesi si è scelto di utilizzare la Community Edition di IntelliJ IDEA, ovvero la versione gratuita ed open source.

JavaFX

JavaFX è una libreria grafica inclusa in Java SE (a partire dalla versione 7) che consente di sviluppare *Rich Client Application* e *Rich Internet Application*, ovvero applicazioni client e web dotate di un'interfaccia utente costruita attraverso un insieme di componenti UI (*User Interface*) predefiniti. Le API JavaFX sono integrate nel JDK (*Java Development Kit*) Java e, grazie al fatto che una JVM può essere eseguita su tutte le piattaforme desktop più diffuse, consentono la compilazione e la generazione di applicazioni eseguibili su diversi sistemi operativi. JavaFX rende libera la costruzione di un interfaccia consentendo di definire la disposizione dei componenti attraverso il codice di programmazione e un particolare file XML scritto in linguaggio FXML. L'uso di FXML permette di separare nettamente la progettazione grafica di una "Scena" dalla logica funzionale ad essa legata. Questa suddivisione tra controller e parte grafica resa possibile da FXML rende semplice l'implementazione di applicazioni secondo il pattern MVC (*Model-View-Controller*).

L'utente non sarà costretto ad editare manualmente i file FXML, JavaFX mette infatti a disposizione un particolare tool grafico chiamato ***Scene Builder*** [31], il quale permette di costruire graficamente ed in modo semplice un'interfaccia grafica tramite la generazione automatica di un file FXML. Scene Builder è compatibile con i principali IDE, compreso IntelliJ IDEA. Sarà perciò possibile editare i file FXML tramite questo strumento direttamente all'interno del nostro ambiente di sviluppo. JavaFX permette inoltre di modificare facilmente il *look and feel* dei nostri componenti UI, semplicemente utilizzando un foglio di stile CSS, senza la necessità di dover scrivere nulla nel codice che definisce l'aspetto grafico dei componenti visualizzati se non le chiamate ad una classe specifica o agli id CSS.

Le API di JavaFX mettono a disposizione dello sviluppatore tutto ciò che serve a realizzare le funzionalità di drag-and-drop di cui si serve la nostra applicazione.

Si è scelto di utilizzare questa libreria per la realizzazione dell'interfaccia e dei componenti grafici del nostro progetto con uno sguardo al futuro. Essa infatti è stata realizzata come evoluzione dell'attualmente più diffusa libreria grafica *Java Swift* ed è destinata a prenderne il posto come libreria di riferimento per la realizzazione di interfacce grafiche in Java.

3.2 Struttura del progetto

In questa sezione viene presentata l'architettura dell'applicazione sviluppata per questo lavoro di tesi. Si partirà dalla descrizione della gerarchia di cartelle all'interno dell'applicazione e per ogni elemento se ne descriverà la logica applicativa e la sua interazione con gli altri componenti.

La root directory dell'applicativo è suddivisa in due directory principali: ***resources*** e ***src***. La directory *resources* contiene tutte le risorse grafiche utilizzate per la creazione dell'interfaccia grafica, come immagini e icone. Inoltre nella sottocartella *META-INF* viene generato automaticamente dall'IDE il file *plugin.xml* che viene utilizzato per la definizione di tutti i parametri del plugin (nome, software supportati, icona, informazioni sullo sviluppatore, versione, definizione dell'azione implementata…).

Nella directory *src* è invece contenuta tutta la logica applicativa. In questa cartella si trova anche la classe *PluginAction*, responsabile della definizione del codice eseguito dall'azione sviluppata dal nostro plugin. Di seguito viene descritto il contenuto dei vari package presenti in questa directory, focalizzando l'attenzione sul ruolo che i vari file ricoprono nel funzionamento dell'applicazione. Va ricordato che l'utilizzo di JavaFX fa sì che la definizione di ogni elemento con un'interfaccia grafica sarà suddivisa tra la classe Java responsabile della logica applicativa ed il file FXML che ne definisce la struttura grafica.

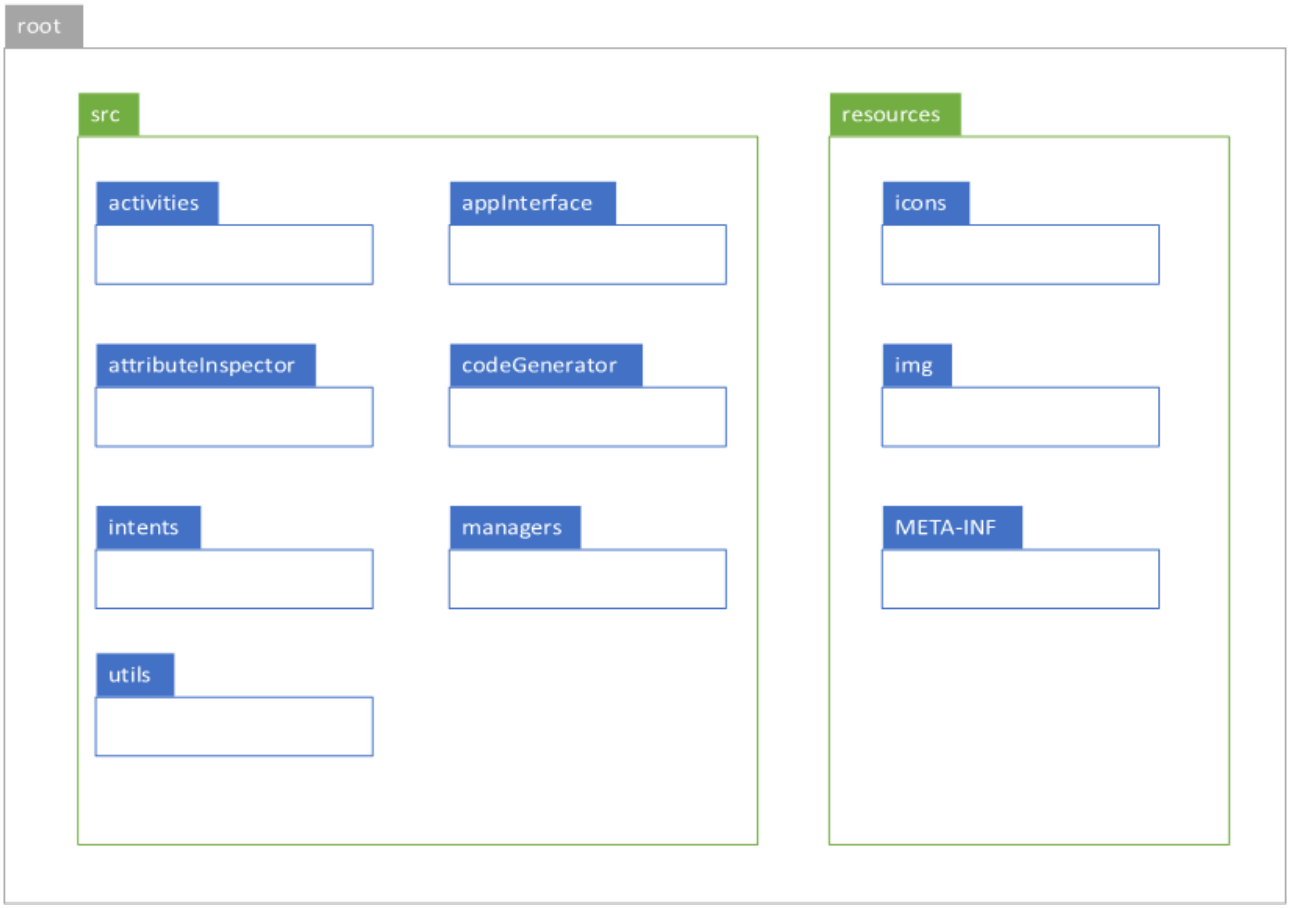

Figura 3.1: Struttura del progetto

AppInterface

In questa directory sono presenti le classi Java ed i file FXML responsabili della definizione della struttura dell'interfaccia grafica. Oltre all'aspetto puramente grafico vengono definite le logiche applicative di ogni elemento dell'interfaccia grafica che verrà descritta dettagliatamente nei prossimi paragrafi. Particolare attenzione merita la gestione della zona di disegno del modello grafico, responsabile della gestione e coordinazione dei vari elementi che la andranno a decorare. Sono inoltre presenti le classi relative ai vari componenti per la decorazione del modello dell'applicazione da sviluppare. In questi file verrà definita la struttura grafica di questi elementi, le loro funzionalità di drag-and-drop e la gestione degli eventi di selezione e cancellazione.

La directory contiene inoltre delle classi di supporto alla realizzazione del modello come *IsInitialActivity* e *SelectedItem*, due singleton che contengono il riferimento dell'activity iniziale della nostra applicazione e dell'elemento del grafo attualmente selezionato. In appInterface è presente anche il file *application.css*, responsabile del look-and-feel dell'interfaccia grafica.

Activities

Questa directory contiene tutte le classi relative alla logica applicativa di ogni tipologia di activity utilizzabile nel progetto. In particolare ogni classe conterrà il riferimento al relativo attribute inspector e i metodi per la decorazione dinamica dei template che verranno utilizzati dal generatore di codice per la generazione del codice nativo delle activities dell'applicazione mobile sviluppata.

Attribute Inspector

Contiene i file relativi alla logica applicativa ed alla definizione del layout degli attribute inspector di tutte le activity e gli intent configurabili nel progetto. In ogni classe sono presenti i metodi che consentono di leggere i valori degli attributi direttamente dalle classi dei componenti a cui si riferiscono e mostrarli nell'inspector. Le modifiche dei valori di un attribute inspector saranno invece gestite da dei listener che si occuperanno di aggiornare i relativi componenti.

Code Generator

In questo package sono contenuti tutti i template utilizzati per la generazione del codice nativo dell'applicazione mobile. I templates sono costituiti da file testuali contenenti la struttura del codice nativo dei vari componenti, decorati con dei *token* che verranno sostituiti con le informazioni generate dinamicamente da metodi dedicati presenti nelle classi delle varie activities (descritte in precedenza). All'interno della directory troviamo inoltre il file *CodeGenerator* che costituisce il cuore della generazione del codice nativo. Esso si occupa infatti di generare tutte le classi, i file di layout e l'AndroidManifest dell'applicazione sviluppata e di salvarli negli appositi path del progetto corrente di Android Studio. Questa classe si occupa inoltre della gestione di eventuali file sovrascritti. Tali file saranno salvati in un'apposita directory di Android Studio accessibile dall'utente, in modo tale da non cancellare accidentalmente dei file e da permettere all'utente di riportare nei file appena generati del codice presente in quelli sovrascritti mediante copia-incolla.

Intents

In questa directory sono presenti tutte le classi relative alla logica applicativa di ogni tipologia di intent utilizzabile nel progetto. In particolare ogni classe conterrà il riferimento al relativo attribute inspector e i metodi per la decorazione dinamica dei template che saranno a loro volta utilizzati nella decorazione dei template delle activities di cui gli intent fanno parte.

Managers

Le classi contenute in questa directory si occupano della gestione di determinati componenti dell'interfaccia grafica. In particolare:

- *AttributeInspectorManager*: si occupa della gestione dell'attribute inspector, caricandone il contenuto in base all'elemento del modello grafico attualmente selezionato e aggiornando i vari componenti in base alle modifiche effettuate sui relativi attributi.
- *GraphHandler*: gestisce l'area di disegno dell'interfaccia grafica e mette a disposizione i metodi per recuperare gli id di tutti i componenti del modello grafico. In fase di generazione del codice, si occupa di verificare che gli attributi dei componenti del grafo abbiano un valore valido, che non esistano due elementi con lo stesso id e che sia stata selezionata una schermata iniziale dell'applicazione che si intende sviluppare.
- *ProjectHandler*: ha il compito di gestire il progetto di Android Studio da cui viene lanciato il nostro plugin. In fase di generazione del codice nativo fornisce al *CodeGenerator* i path in cui andare a salvare le classi, i file di layout e l'AndroidManifest dell'applicazione generata.
- *StructureTreeManager*: mette a disposizione una serie di metodi per la creazione, cancellazione e ricerca degli elementi del Project Tree.

Utils

Contiene una serie di classi che definiscono enumerators e costanti utilizzati per varie funzioni del sistema.

3.3 Generazione modello astratto

In questa sezione verrà fornita una descrizione dettagliata del processo di generazione del modello astratto. Data la natura model-driven del progetto realizzato, questa funzionalità ne rappresenta un elemento fondamentale. La generazione del modello astratto consiste nella realizzazione di un modello grafico rappresentante l'applicazione mobile che si intente sviluppare. In seguito verrà fornita una descrizione approfondita della composizione dell'interfaccia grafica, specificando le funzionalità messe a disposizione da ogni suo elemento. Dopodiché verranno presentati tutti gli elementi utilizzabili per la composizione del modello, specificandone in primo luogo la composizione della struttura grafica, il loro comportamento all'interno dell'area di disegno e come essi possono essere utilizzati per comporre il modello grafico; verrà poi fornito un elenco di tutte le tipologie di activities e intents presenti nel sistema, di cui sarà fornita una descrizione dettagliata delle funzionalità e del layout.

3.3.1 Interfaccia grafica dell'applicazione

L'interfaccia grafica dell'applicazione è suddivisa in quattro parti. La parte centrale rappresenta l'area di disegno, ovvero la zona destinata a contenere il modello dell'applicazione ideata. Il modello verrà costruito trascinando le varie activities all'interno di quest'area dove potranno essere posizionate liberamente e collegate tra loro. L'area di disegno non ha una dimensione fissa e questa caratteristica le permette di adattarsi alla costruzione di modelli grafici di qualsiasi dimensione. Le activities con cui decorare il modello sono messe a disposizione dell'utente attraverso un menu di selezione posto nella parte in basso a sinistra dell'interfaccia grafica. In questo menu saranno elencate le varie activities utilizzabili, rappresentate da un'icona raffigurante una schematizzazione del layout di quella tipologia di activity e da una breve descrizione. Per aggiungere un'activity al modello grafico basterà quindi cliccare sulla relativa icona presente nel menu di selezione e trascinarla all'interno dell'area di disegno.

Sempre nella parte sinistra dell'interfaccia, al di sopra del menu di selezione, troviamo il Project Tree. Questo componente mostra una panoramica di tutti gli elementi che compongono il modello mediante una struttura ad albero, come quelle presenti nei comuni IDE. In questa struttura ad albero il livello più alto sarà costituito dall'elenco di tutte le activities inserite del modello, per ciascuna delle quali saranno elencati i link uscenti e i relativi intents che compongono ogni link. Ognuno di questi elementi sarà contraddistinto da una relativa icona per facilitarne l'utilizzo. Tramite il Project Tree sarà possibile selezionare e cancellare qualsiasi elemento del modello.

Nella parte destra dell'editor grafico si trova invece un Attribute Inspector. Si tratta di un pannello che presenta all'utente gli attributi relativi all'elemento selezionato del modello grafico, consentendogli di settarli a suo piacimento.

Una volta realizzato il modello grafico sarà possibile generare il corrispondente codice nativo premendo il tasto "Generate Code" posizionato nella parte in alto a sinistra dell'area di disegno.

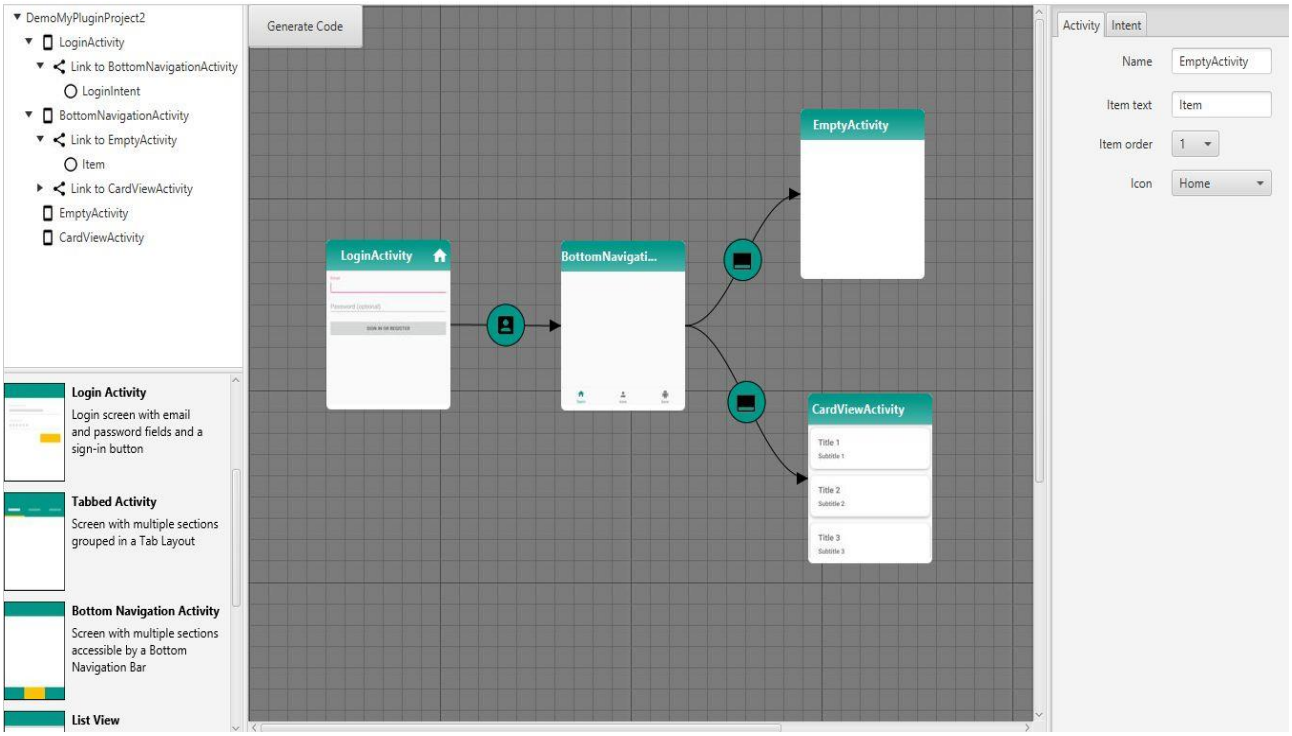

Figura 3.2: Interfaccia grafica dell'applicazione

3.3.2 Composizione del modello

In questa sezione verranno descritti gli elementi utilizzabili per la composizione del modello grafico che la nostra applicazione utilizzerà per la generazione del codice nativo. Per ogni elemento verrà fornita una descrizione della composizione grafica, delle funzionalità e delle interazioni con gli altri componenti del modello.

Il modello generato assumerà le sembianze di un grafo, in cui i nodi saranno rappresentati dalle **activities**, ovvero le varie schermate della nostra applicazione, i collegamenti, che chiameremo **link**, costituiranno il flusso di navigazione fra esse mentre gli **intent** rappresenteranno il tipo di interazione presente fra le varie schermate.

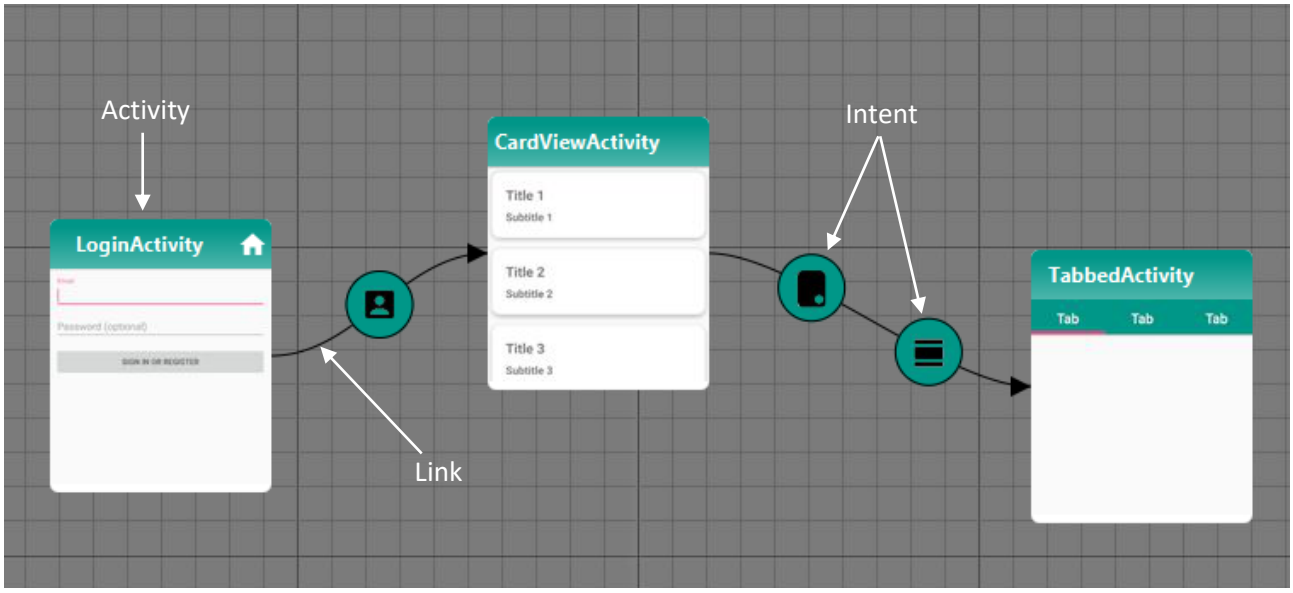

Figura 3.3: Componenti del modello grafico

3.3.2.1 Struttura delle activities

Come detto le activities costituiranno i nodi del nostro modello e rappresentano le varie schermate dell'applicazione che si intende sviluppare. Il loro aspetto grafico richiama quello di una reale schermata di un applicazione mobile ed è composto da un header ed un corpo. L'header è una barra posta all'estremità superiore dell'activity ed ha due funzioni: contiene il nome identificativo dell'activity e permette, tenendo premuto il pulsante sinistro del mouse su di esso, di spostare l'activity in qualsiasi posizione all'interno dell'area di disegno. Inoltre se l'activity è stata impostata come schermata iniziale dell'applicazione, nell'header verrà mostrata un'icona a forma di casetta. Il corpo mostrerà una rappresentazione grafica che richiama il layout dell'activity in base alla tipologia e permetterà quindi di identificare facilmente la tipologia delle activities presenti nel modello.

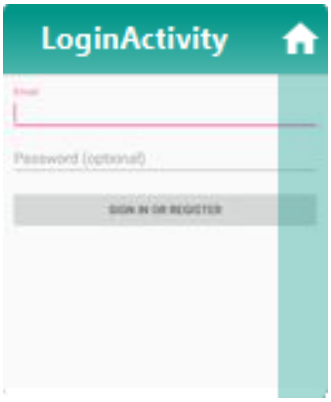

Figura 3.4: Struttura di una activity

Il corpo dell'activity è a sua volta suddiviso in una zona centrale e due bande laterali. Le due bande laterali non sono inizialmente distinguibili dal resto del corpo, ma verranno evidenziate da un ombreggiatura solo quando il puntatore del mouse si troverà sopra ad esse. La loro funzione è quella di consentire la creazione dei link tramite funzionalità drag-and-drop. Infatti, per collegare due activity con un link, basterà cliccare su una di queste bande della schermata di partenza, trascinare il mouse tenendo premuto il tasto sinistro per tracciare una linea che colleghi l'activity di partenza con quella di destinazione e rilasciare il tasto del mouse nel corpo dell'activity di destinazione. A questo punto apparirà un Context Menu che permetterà all'utente di scegliere con quale intent fra quelli a disposizione si vogliono collegare le due activity. Una volta selezionato il tipo di intent desiderato, se non esiste già un link tra le due schermate, il sistema traccerà automaticamente un link tra esse e vi aggiungerà l'intent selezionato; se invece è già presente un link, verrà solo aggiunto un nuovo intent. La seguente immagine mostra il procedimento appena descritto per la creazione di un link.

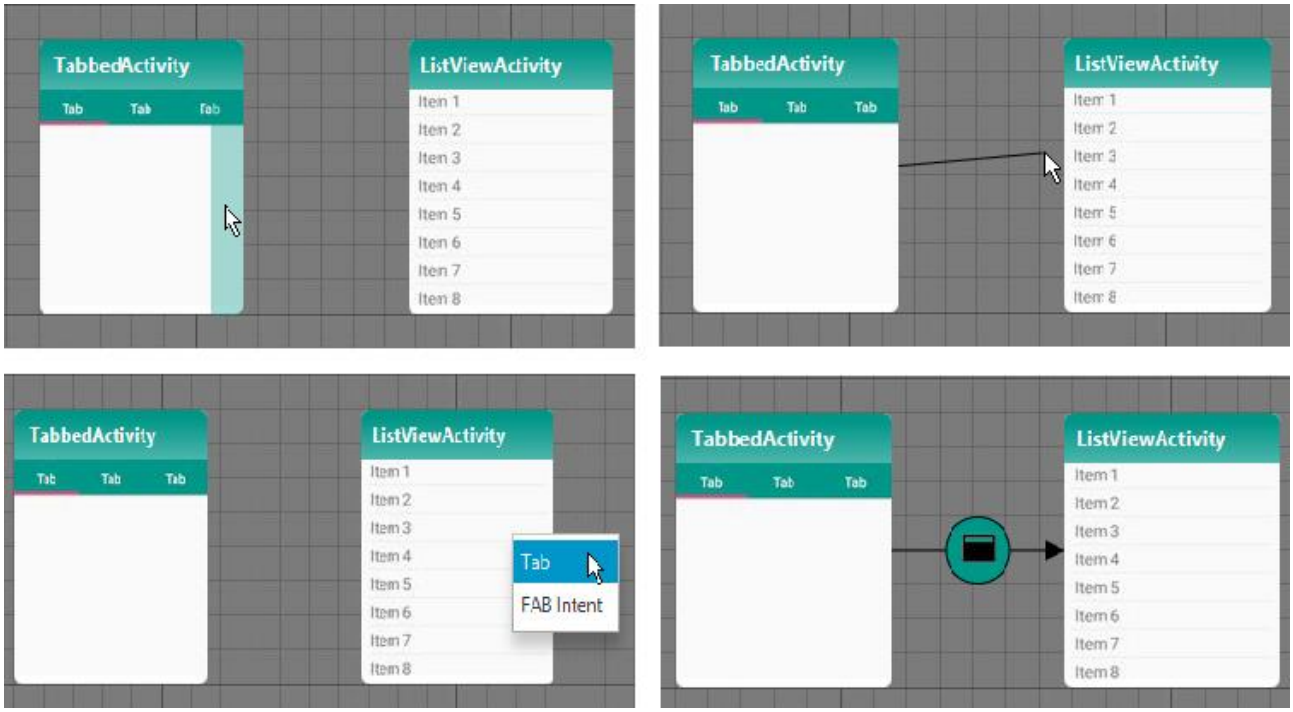

Figura 3.5: Creazione di un link fra due activities

La parte centrale del corpo permetterà all'utente di selezionare l'activity mediante doppio click. L'activity selezionata sarà evidenziata con un bordo azzurro. Per deselezionarla basterà selezionare un altro elemento del modello o fare doppio click in uno spazio vuoto dell'area di disegno. Una volta selezionata, potrà essere cancellata dal grafo premendo il tasto "*canc*" della tastiera. La cancellazione di un'activity comporterà la cancellazione dal modello di tutti i link e gli intent ad essa collegati.

I vari tipi ti activity forniti dall'applicazione e le loro proprietà verranno descritti del dettaglio nella sezione 3.3.3.

3.3.2.2 Struttura dei link

I link rappresenteranno il flusso di navigazione fra due activity. Essi hanno per lo più una funzione grafica, il loro scopo è infatti quello di raggruppare in unico collegamento tutti gli intent presenti fra due activity, rendendo più pulito e meno confusionario il modello grafico. Sono rappresentati graficamente da una *curva di Bèzier* [32] con una freccia posta sull'estremità finale per indicare la direzione del flusso di navigazione.

Grazie al *Property Binding System* [33] fornito da JavaFX, in seguito allo spostamento di una delle activity a cui il link è collegato, è possibile aggiornare in real time la forma del link, i punti di ancoraggio del link sulle activites che collega, la posizione degli intent contenuti e la direzione del link.

Cliccando due volte sul link esso verrà selezionato insieme a tutti gli intent che contiene. Il link selezionato sarà evidenziato con un bordo azzurro. Una volta selezionato, nell'attribute inspector verranno caricati gli attributi relativi a tutti i link in esso contenuti. Per deselezionarlo basterà selezionare un altro elemento del modello o fare doppio click in uno spazio vuoto dell'area di disegno. Premendo il tasto "*canc*" della tastiera sarà possibile rimuovere dal grafo il link selezionato e tutti gli intent ad esso associati.

3.3.2.3 Struttura degli intents

Gli intents definiscono il modo in cui due activity interagiscono fra loro. Ad esempio un *Button Click intent* tra due activity indicherà che sarà possibile passare dalla prima activity alla seconda mediante il click di un bottone. Sono rappresentati da un'icona circolare raffigurante la tipologia di intent e vengono posti lungo i link. È possibile selezionare un intent mediante doppio click del tasto sinistro del mouse e una volta selezionato sarà evidenziato con bordo color azzurro. Come sempre per deselezionarlo basterà selezionare un altro elemento del modello o fare doppio click su una zona vuota dell'area di disegno. Premendo il tasto "*canc*" della tastiera l'intent selezionato verrà cancellato dal grafo. Se l'intent rimosso era l'unico presente nel rispettivo link, la sua cancellazione comporterà anche la cancellazione di quest'ultimo.

Gli intent messi a disposizione dall'applicazione, le loro proprietà e le varie tipologie di activity a cui possono essere associati verranno descritti del dettaglio nella sezione 3.3.4.

3.3.3 Activities implementate

Nelle sezioni precedenti sono stati descritti tutti gli elementi del modello fra cui le activities. Si è visto che per inserire una determinata schermata nella applicazione mobile da sviluppare basterà trascinare l'activity del tipo desiderato nell'area di disegno, aggiungendola così al modello. In questo progetto di tesi sono state realizzate varie tipologie di activities che permettono all'utente di modellare le schermate della propria applicazione mobile utilizzando le strutture e i pattern più comuni e diffusi nell'ambiente Android. Verranno ora elencate tutte le tipologie di activity realizzate, descrivendone le funzionalità messe a disposizione, il layout, gli intent che possono generare e gli attributi che è possibile configurare mediante il relativo attribute inspector. L'attribute inspector di un'activity è composto da due sezioni suddivise mediante un TabPane, una relativa all'activity stessa e una agli intent che in essa vengono generati. In questo capitolo verranno presi in considerazione gli attributi modificabili nel tab relativo alla schermata, lasciando la descrizione di quelli relativi agli intents al prossimo capitolo ad essi dedicato.

Empty Activity

È il tipo di activity più semplice presente in Android ed è costituito da una schermata vuota. L'Empty Activity viene utilizzata quando l'utente vuole sviluppare l'activity da zero, servendosi della schermata vuota generata come base da decorare a proprio piacimento.

Intents

- Button Click
- Button Click With Result

Attributi

Attributo	Tipo	Descrizione	Default value
Name	Stringa	Attributo che rappresenta l'id dell'activity. Questo valore sarà utilizzato come nome della	"EmptyActivity"

		classe che conterrà il codice dell'activity.	
Is Initial Activity	Booleano	Indica se l'activity rappresenta la schermata iniziale dell'applicazione.	Vero se non sono presenti altre activities nel modello, Falso altrimenti

Tabella 3.1: Tabella degli attributi di una Empty Activity

Basic Activity

L'elemento caratteristico della Basic Activity è un Floating Action Button posto nella parte in basso a destra dello schermo. Questo bottone può essere utilizzato per la navigazione verso un'altra schermata mediante l'apposito intent, oppure potrà essere implementato in seguito dall'utente. Anche questo tipo di activity può essere utilizzato come base per lo sviluppo di una schermata.

Intents

- Button Click
- Button Click With Result
- Floating Action Button Click

Attributi

Attributo	Tipo	Descrizione	Default value
Name	Stringa	Attributo che rappresenta l'id dell'activity. Questo valore sarà utilizzato come nome della classe che conterrà il codice dell'activity.	"BasicActivity"

Is Initial Activity	Booleano	Indica se l'activity rappresenta la schermata iniziale dell'applicazione.	Vero se non sono presenti altre activities nel modello, Falso altrimenti

Tabella 3.2: Tabella degli attributi di una Basic Activity

Login Activity

Rappresenta una tipica schermata di login, contenente i campi per l'inserimento di email e password e un bottone per il sign-in. Implementa una funzione di auto compilazione per la mail e i controlli sulla validità del contenuto dei campi mail e password (lunghezza dei contenuti, corretta formattazione della mail, campi vuoti).

Intents

- Login Intent

Attributi

Attributo	Tipo	Descrizione	Default value
Name	Stringa	Attributo che rappresenta l'id dell'activity. Questo valore sarà utilizzato come nome della classe che conterrà il codice dell'activity.	"LoginActivity"
Is Initial Activity	Booleano	Indica se l'activity rappresenta la schermata iniziale dell'applicazione.	Vero se non sono presenti altre activities nel modello, Falso altrimenti

Tabella 3.3: Tabella degli attributi di una Login Activity

List View

Questo tipo di activity consente di creare schermate per la visualizzazione di un elenco di elementi, come ad esempio l'elenco degli utenti registrati, dei contatti salvati sullo smartphone, piuttosto che l'elenco dei messaggi ricevuti in un'applicazione di messaggistica e così via. L'utente potrà scorrere verticalmente la lista mediante gesture. Attraverso l'attribute inspector vengono messi a disposizione dell'utente due tipi di configurazione: String e Custom. Nel primo caso il contenuto delle righe della List View sarà di tipo testuale, nel secondo sarà rappresentato da un oggetto che l'utente potrà poi personalizzare con gli elementi desiderati.

Intents

- Floating Action Button Click
- List Item Click

Attributi

Attributo	Tipo	Descrizione	Default value
Name	Stringa	Attributo che rappresenta l'id dell'activity. Questo valore sarà utilizzato come nome della classe che conterrà il codice dell'activity.	"ListViewActivity"
Is Initial Activity	Booleano	Indica se l'activity rappresenta la schermata iniziale dell'applicazione.	Vero se non sono presenti altre activities nel modello, Falso altrimenti
Content	Enum {String, Custom}	Permette di scegliere il tipo di contenuto delle righe della List View. Esso potrà essere di tipo testuale	String

		o costituito da un oggetto personalizzabile successivamente dall'utente.	

Tabella 3.4: Tabella degli attributi di una List View

Grid View

La Grid View activity permette di creare schermate contenenti un insieme di elementi disposti secondo un layout a griglia. Un classico esempio di utilizzo è quello di una schermata di un comune sito di e-commerce dove vengono mostrati i vari prodotti disponibili o quelli risultanti da una particolare ricerca. L'utente potrà scorrere verticalmente la lista mediante gesture. Attraverso l'attribute inspector vengono messi a disposizione dell'utente due tipi di configurazione: String e Custom. Nel primo caso il contenuto delle celle della Grid View sarà di tipo testuale, nel secondo sarà rappresentato da un oggetto che l'utente potrà in seguito personalizzare con gli elementi desiderati. Sarà inoltre possibile configurare il numero di colonne della tabella.

Intents

- Floating Action Button Click
- Grid Item Click

Attributi

Attributo	Tipo	Descrizione	Default value
Name	Stringa	Attributo che rappresenta l'id dell'activity. Questo valore sarà utilizzato come nome della classe che conterrà il codice dell'activity.	"GridViewActivity"

Is Initial Activity	Booleano	Indica se l'activity rappresenta la schermata iniziale dell'applicazione.	Vero se non sono presenti altre activities nel modello, Falso altrimenti
Content	Enum {String, Custom}	Permette di scegliere il tipo di contenuto delle righe della Grid View. Esso potrà essere di tipo testuale o costituito da un oggetto personalizzabile successivamente dall'utente.	String
Columns	Intero	Consente di configurare il numero di colonne della Grid View.	2

Tabella 3.5: Tabella degli attributi di una Grid View

Card View

La Card View activity si basa su un nuovo widget introdotto in Android con l'avvento del *Material Design*, che permette la visualizzazione di un elenco di elementi chiamati Cards. Una Card è un elemento grafico contraddistinto da bordi arrotondati, un'ombreggiatura e un effetto rilievo, che può contenere elementi di tipo testuale, immagini o link. I possibili scenari di utilizzo sono gli stessi descritti per le List View e Grid View, con la differenza che le righe e le celle di tali activities saranno sostituite dalle Cards che consentono di dare alla schermata un look-and-feel migliore. Attraverso l'attribute inspector l'utente potrà configurare il tipo di layout secondo cui disporre le Cards nella schermata. Vengono messi a disposizione due tipi di layout, List e Grid.

Nel primo caso le Cards verranno disposte verticalmente ed andranno a formare una lista, nel secondo caso invece saranno disposte secondo un layout a griglia, di cui sarà possibile impostare il numero di colonne. Anche in questo caso l'utente potrà scorrere l'elenco delle Cards mediante gestures.

Intents

- Floating Action Button Click
- Card Click

Attributi

Attributo	Tipo	Descrizione	Default value
Name	Stringa	Attributo che rappresenta l'id dell'activity. Questo valore sarà utilizzato come nome della classe che conterrà il codice dell'activity.	"CardViewActivity"
Is Initial Activity	Booleano	Indica se l'activity rappresenta la schermata iniziale dell'applicazione.	Vero se non sono presenti altre activities nel modello, Falso altrimenti
Layout	Enum {List, Grid}	Permette di scegliere il tipo di layout della schermata. Il layout di tipo "List" permette di visualizzare le Cards in una struttura a lista, mentre nel layout di tipo "Grid" le Cards saranno disposte in una struttura a griglia.	List

Columns	Intero	Consente di configurare il numero di colonne della Card View. Questo attributo sarà visibile e configurabile solo se il layout scelto è di tipo “Grid”.	2

Tabella 3.6: Tabella degli attributi di una Card View

Tabbed Activity

Questo tipo di activity permette di creare una schermata a sezioni multiple, mediante l’utilizzo di una struttura a tab. Ogni sezione sarà accessibile premendo il relativo tab o scorrendo orizzontalmente le varie sezioni via gestures. I vari tab saranno contenuti nella Toolbar dell’activity. Questo tipo di schermata viene solitamente utilizzato per raggruppare diverse schermate in una singola activity o quando è necessario suddividere il contenuto di un’activity in diverse sezioni.

Tipici esempi di utilizzo di questo tipo di struttura sono il Play Store, dove le varie categorie di applicazioni sono accessibili mediante il click del relativo tab oppure la rubrica, le cui varie sezioni (elenco contatti, gruppi, preferiti…) sono contenute in una struttura a tab.

La Tabbed Activity funge dunque da contenitore di altre schermate, che potranno essere istanziate e visualizzate sotto forma di fragments, che saranno descritti nella parte conclusiva di questa sezione.

Le activity che possono essere contenute sotto forma di fragment nella Tabbed Activity sono: Empty Activity, Basic Activity, List View, Grid View e Card View.

<u>Intents</u>

- Floating Action Button Click
- Tab Intent

Attributi

Attributo	Tipo	Descrizione	Default value
Name	Stringa	Attributo che rappresenta l'id dell'activity. Questo valore sarà utilizzato come nome della classe che conterrà il codice dell'activity.	"TabbedActivity"
Is Initial Activity	Booleano	Indica se l'activity rappresenta la schermata iniziale dell'applicazione.	Vero se non sono presenti altre activities nel modello, Falso altrimenti

Tabella 3.7: Tabella degli attributi di una Tabbed Activity

Bottom Navigation Activity

La Bottom Navigation Activity permette di creare una schermata a sezioni multiple, accessibili mediante una Bottom Navigation Bar, posta nella parte inferiore nella schermata. Per accedere ad ogni sezione basterà premere il relativo item della Bottom Navigation Bar che sarà costituito da un nome identificativo e da un icona. Questo tipo di activity viene utilizzato per raggruppare diverse schermate in una singola activity o per organizzare e divedere il contenuto e le funzionalità di una schermata in più sezioni. In particolare gli sviluppatori Android ne consigliano l'uso per la realizzazione di strutture con un numero di sezioni compreso tra tre e cinque.

Un tipico esempio di utilizzo è rappresentato dall'applicazione orologio, in cui le varie funzionalità messe a disposizione (impostazione ora, sveglia, fusi orari, cronometro…) sono suddivise in diverse sezioni accessibili mediante una Bottom Navigation Bar.

Anche la Bottom Navigation Activity funge dunque da contenitore di altre schermate che, come nel caso della Tabbed Activity, potranno essere aggiunte sotto forma di fragments.

Le activity che possono essere contenute sotto forma di fragment sono: Empty Activity, Basic Activity, List View, Grid View e Card View.

Intents

- Bottom Navigation Item

Attributi

Attributo	Tipo	Descrizione	Default value
Name	Stringa	Attributo che rappresenta l'id dell'activity. Questo valore sarà utilizzato come nome della classe che conterrà il codice dell'activity.	"BottomNavigationActivity"
Is Initial Activity	Booleano	Indica se l'activity rappresenta la schermata iniziale dell'applicazione.	Vero se non sono presenti altre activities nel modello, Falso altrimenti

Tabella 3.8: Tabella degli attributi di una Bottom Navigation Activity

Fragments

Nelle descrizioni delle varie tipologie di activities appena esposte si è visto che Tabbed Activity e Bottom Navigation Activity rappresentano una sorta di contenitore di altre schermate, le quali possono essere aggiunte a queste strutture sotto forma di fragment.

Un fragment è una sorta di sub-activity che rappresenta una porzione dell'user interface di una schermata. Si può pensare ad un fragment come se fosse una sezione di un'activity con il proprio ciclo di vita, in grado di ricevere i propri eventi di input e che può essere aggiunta o rimossa mentre l'activity è in esecuzione. Un fragment può esistere solamente se associato ad un'activity e il suo ciclo di vita dipende da quello dell'activity che lo contiene.

Nella creazione del modello grafico del nostro progetto la gestione dei fragment viene realizzata in modo automatico e trasparente all'utente che continuerà a vederle come normali activities. Infatti l'utente potrà aggiungere le sub-activities di una Tabbed Activity o di una Bottom Navigation Activity semplicemente collegando mediante l'apposito intent queste schermate con le activities che dovranno contenere. Solo nella fase di generazione del codice il sistema andrà a controllare se quella determinata schermata dovrà essere sviluppata come activity o come fragment.

Le uniche caratteristiche che differenzieranno i fragment dalle normali activities nel modello grafico saranno rappresentate dal non poter impostare quella schermata come schermata iniziale o come destinazione di un intent (fatta ovviamente eccezione per gli intent che permettono a Tabbed e Bottom Navigation Activity di contenerli). Se la schermata impostata come iniziale viene "trasformata" in fragment, l'activity che la contiene diventerà automaticamente la nuova schermata iniziale.

Le activites che possono essere utilizzate come fragment sono: Empty Activity, Basic Activity, List View, Grid View e Card View. Fatta eccezione per l'attributo Is Initial Activity, esse mantengono tutti gli attributi precedentemente descritti.

3.3.4 Intents implementati

In questo progetto di tesi sono state realizzate varie tipologie di intents che permettono all'utente di creare un flusso di navigazione articolato. Oltre ai vari intent che permettono di passare da una schermata all'altra cliccando un particolare elemento grafico (bottoni, floating action button, elementi di liste e griglie…) sono stati realizzati intent ad hoc come quelli che permettono l'inserimento di una determinata activity in una Tabbed Activity o in una Bottom Navigation Activity o l'intent incaricato di gestire la risposta al click di una Card. Questo tipo di interazioni nello sviluppo di applicazioni Android mediante codice nativo non sono realizzabili mediante l'utilizzo di semplici intent, ma necessitano di strutture elaborate. Come per le activities, anche per quanto riguarda gli intents, poter creare questo genere di interazioni complesse in modo semplice e grafico, rappresenta uno dei principali obiettivi e punti di forza ed innovazione di questo progetto di tesi.

Di seguito verranno elencati tutti gli intents realizzati, descrivendone il funzionamento, le activity da cui possono essere creati e gli attributi che è possibile configurare mediante il relativo attribute inspector.

Button Click

Questo intent consente la navigazione verso una nuova schermata dell'applicazione mediante la pressione di un bottone. Si tratta di uno degli intent più comuni e può essere utilizzato praticamente in ogni tipo di app mobile. L'aggiunta di un Button Click intent comporta l'inserimento di un bottone nel layout della relativa activity.

Da un'activity possono essere generati uno o più Button Click intent.

Activities

- Empty Activity
- Basic Activity

Attributi

Attributo	Tipo	Descrizione	Default value
Button ID	Stringa	Consente di definire l'ID del bottone creato	"button"
Button Text	Stringa	Permette di definire il testo contenuto nel bottone creato	
Extras	Enum {None, String, Boolean, Integer, Float, Double}	Questo attributo consente di specificare il tipo di Extras aggiunto all'intent. Se l'intent non contiene Extras sarà impostato su "None"	None

Tabella 3.9: Tabella degli attributi di un Button Click

Floating Action Button Click

Consente la navigazione verso una nuova schermata dell'applicazione mediante la pressione di un Floating Action Button. Questo tipo di intent modifica il layout dell'activity che lo contiene andando ad aggiungere un Floating Action Button nella parte in basso a destra della schermata, in primo piano rispetto al resto del contenuto.

Un utilizzo tipico di questo intent si ha in activities che presentano una lista di elementi, in cui cliccando il Floating Action Button è possibile aprire una schermata che permette all'utente di creare un nuovo elemento da aggiungere alla lista. Possiamo trovare un'interazione del genere ad esempio in applicazioni che permettono di aggiungere una recensione di un determinato prodotto o servizio oppure in applicazioni di messaggistica dove il floating action button viene utilizzato per creare un nuovo messaggio.

Da un'activity può essere generato uno e un solo Floating Action Button Click intent.

Activities

- Basic Activity
- List View
- Grid View
- Card View
- Tabbed Activity

Attributi

Attributo	Tipo	Descrizione	Default value
Extras	Enum {None, String, Boolean, Integer, Float, Double}	Questo attributo consente di specificare il tipo di Extras aggiunto all'intent. Se l'intent non contiene Extras sarà impostato su "None"	None

Tabella 3.10: Tabella degli attributi di un Floating Action Button Click

Login Intent

Questo intent consente la navigazione verso una nuova schermata dell'applicazione mediante la pressione del tasto sign-in di una Login Activity. Viene solitamente utilizzato per caricare la home di un'applicazione o una schermata ad accesso riservato per cui è stata richiesta l'autenticazione dell'utente.

Da una Login Activity può essere generato uno ed un solo Login Intent.

Activities

- Login Activity

Attributi

Attributo	Tipo	Descrizione	Default value
Button Text	Stringa	Permette di definire il testo contenuto nel bottone per il sign-in	"Sign in or register"
Extras	Enum {None, String, Boolean, Integer, Float, Double}	Questo attributo consente di specificare il tipo di Extras aggiunto all'intent. Se l'intent non contiene Extras sarà impostato su "None"	None

Tabella 3.11: Tabella degli attributi di un Login Intent

Button Click With Result

Si tratta di un tipo particolare di button click intent. A differenza del Button Click precedentemente descritto, questo intent genera un duplice flusso di navigazione fra due schermate. Infatti, mediante la pressione di un bottone l'activity *A* potrà lanciare un'activity *B*, restando in attesa di una risposta da parte di quest'ultima. A sua volta l'activity *B* mediante il click di un bottone lancerà *A*, includendo nell'intent una risposta. A questo punto *A* andrà ad estrarre dall'intent la risposta di *B*.

Questo intent andrà ad aggiungere un bottone al layout della schermata di partenza ed uno a quella di destinazione.

Un utilizzo tipico di questo tipo di interazione si ha in applicazioni che permettono di accedere alla fotocamera per scattare una fotografia. In questo caso cliccando un bottone di una schermata si aprirà la fotocamera e la schermata chiamante resterà in attesa della foto scattata. Una volta scattata la foto verrà richiamata la schermata di partenza a cui viene restituita la foto scattata.

Un'activity può generare Button Click With Result intents verso una o più activities, ma fra ogni coppia di activities può esistere uno e un solo intent di questo tipo.

Activities

- Empty Activity
- Basic Activity

Attributi

Attributo	Tipo	Descrizione	Default value
Button ID	Stringa	Consente di definire l'ID del bottone creato	"button" o "resultButton"
Button Text	Stringa	Permette di definire il testo contenuto nel bottone creato	
Extras	Enum {None, String, Boolean, Integer, Float, Double}	Questo attributo consente di specificare il tipo di Extras aggiunto all'intent. Se l'intent non contiene Extras sarà impostato su "None"	None

Tabella 3.12: Tabella degli attributi di un Button Click With Result

Questi attributi si riferiscono sia al bottone utilizzato per la chiamata sia a quello utilizzato per restituire la risposta.

List Item Click

Permette la navigazione verso una nuova schermata dell'applicazione cliccando un elemento di una List View. Questa funzionalità è spesso utilizzata in costrutti di tipo master-detail. Ad esempio, in una List View contenente l'elenco degli utenti registrati di un'applicazione, cliccando su una riga dell'elenco verrà visualizzata la schermata contenente i dati relativi all'utente selezionato dalla lista.

Da una List View può essere generato uno ed un solo List Item Click intent.

Activities

- List View

Attributi

Attributo	Tipo	Descrizione	Default value
Extras	Enum {None, String, Boolean, Integer, Float, Double}	Questo attributo consente di specificare il tipo di Extras aggiunto all'intent. Se l'intent non contiene Extras sarà impostato su "None"	None

Tabella 3.13: Tabella degli attributi di un List Item Click

Grid Item Click

Permette la navigazione verso una nuova schermata dell'applicazione cliccando un elemento di una Grid View. Questa funzionalità è spesso utilizzata in costrutti di tipo master-detail.

Ad esempio, in una Grid View utilizzata in un'app di e-commerce per mostrare il catalogo dei prodotti acquistabili, cliccando su una cella contenente un determinato prodotto verrà visualizzata la schermata contenente la scheda prodotto dell'elemento selezionato.

Da una Grid View può essere generato uno ed un solo Grid Item Click intent.

Activities

- Grid View

Attributi

Attributo	Tipo	Descrizione	Default value
Extras	Enum {None, String, Boolean, Integer, Float, Double}	Questo attributo consente di specificare il tipo di Extras aggiunto all'intent. Se l'intent non contiene Extras sarà impostato su "None"	None

Tabella 3.14: Tabella degli attributi di un Grid Item Click

Card Click

Questo intent viene generato in risposta al click di una Card in una Card View e permette la navigazione verso una nuova schermata. Come precedentemente descritto per List Item Click e Grid Item Click, anche questo intent è spesso utilizzato per funzionalità di tipo master-detail.

Da una Card View può essere generato uno ed un solo Card Click intent.

Activities

- Card View

Attributi

Attributo	Tipo	Descrizione	Default value
Extras	Enum {None, String, Boolean, Integer, Float, Double}	Questo attributo consente di specificare il tipo di Extras aggiunto all'intent. Se l'intent non contiene Extras sarà impostato su "None"	None

Tabella 3.15: Tabella degli attributi di un Card Click

Tab Intent

Questo tipo di intent permette di aggiungere l'activity destinataria, sotto forma di fragmet, come sezione di una Tabbed Activity. A livello di layout il Tab Intent andrà quindi ad aggiungere un nuovo tab alla Toolbar della schermata.

Ad una Tabbed Activity possono essere aggiunti da 1 a 5 Tab Intents.

Activities

- Tabbed Activity

Attributi

Attributo	Tipo	Descrizione	Default value
Tab Text	String	Consente di specificare il testo mostrato nel tab	"Tab"
Tab Order	Intero	Permette di configurare l'ordine in cui il tab viene	N° di tab presenti + 1

		visualizzato nella toolbar	

Tabella 3.16: Tabella degli attributi di un Tab Intent

Bottom Navigation Item

Grazie a questo intent è possibile aggiungere un'activity (sotto forma di fragment) come sezione di una Bottom Navigation Activity. Questo tipo di intent comporta l'aggiunta di un nuovo elemento alla Bottom Navigation Bar.

Ad una Bottom Navigation Activity possono essere aggiunti da 1 a 5 Bottom Navigation Item intents.

Activities

- Bottom Navigation Activity

Attributi

Attributo	Tipo	Descrizione	Default value
Item Text	String	Consente di specificare il testo mostrato nel relativo item della Bottom Navigation Bar	"Item"
Item Order	Intero	Permette di configurare l'ordine in cui l'item viene visualizzato nella Bottom Navigation Bar	N° di item presenti + 1

Icon	Enum {Android, Build, Dashboard, Edit, Home, Notifications, Person, Share}	Permette di scegliere l'icona mostrata nell'item della Bottom Navigation Bar	Android 

Tabella 3.17: Tabella degli attributi di un Bottom Navigation Item

Si riporta di seguito un modello logico che riassume le relazioni fra activities e link, specificando quali e quanti intents possono essere generati dai vari tipi di activity.

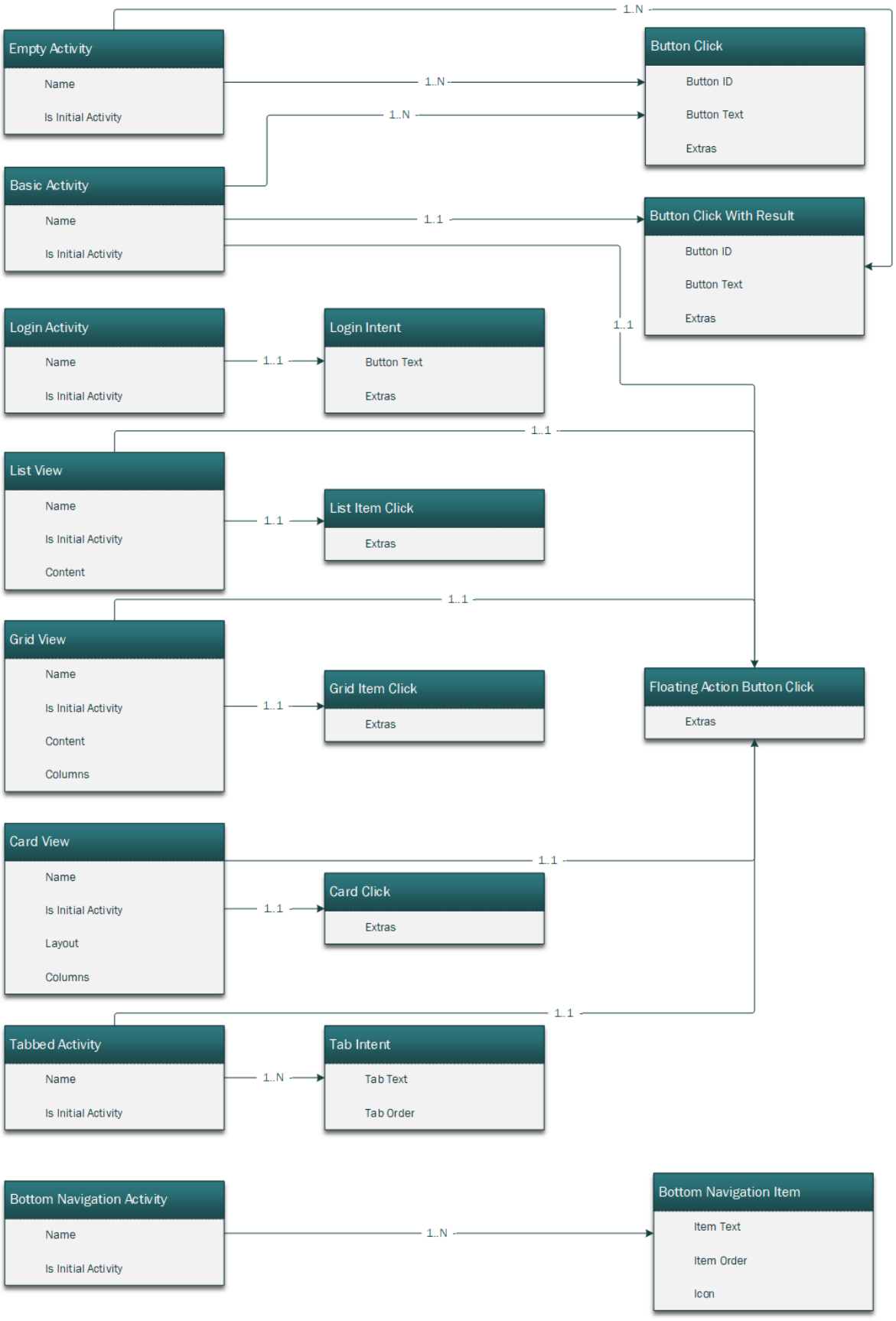

Figura 3.6: Modello logico delle relazioni tra activities e intents

3.4 Generazione codice nativo

In questa sezione verrà descritto il meccanismo di traduzione in codice nativo del modello grafico realizzato. Questa funzionalità permette al sistema di generare il codice nativo relativo ai vari componenti utilizzati per la decorazione del modello ed aggiungere i file generati al progetto di Android Studio. In questo modo l'IDE conterrà tutto il codice perfettamente funzionante dell'applicazione realizzata mediante modello, che l'utente potrà utilizzare come struttura di base per lo sviluppo del prodotto finale.

Di seguito saranno descritti tutti i passaggi della procedura di traduzione del modello grafico in codice nativo, dopodiché verrà fatta un'analisi del codice generato per la realizzazione di tutte le activities e gli intent messi a disposizione in questo progetto di tesi.

3.4.1 Traduzione del modello grafico in codice nativo

Dopo aver realizzato il modello grafico dell'applicazione che si intende sviluppare mediante le varie activities e i vari tipi di intents messi a disposizione sarà possibile generare il corrispondente codice nativo premendo il tasto "*Generate Code*" situato nell'editor grafico. Come è stato spiegato nelle precedenti sezioni di questo elaborato, il processo di generazione del codice nativo si baserà sull'utilizzo di templates che conterranno il codice dei vari file (classi Java o file XML di layout). Il codice contenuto nei template rappresenterà la struttura di base dei vari file e sarà decorato in modo dinamico durante il processo di generazione del codice. A tal proposito i template conterranno dei *token* che saranno utilizzati come "segnaposto" per indicare al sistema in che punto del file andare ad inserire una determinata porzione di codice. Durante la creazione dinamica delle varie porzioni di codice verranno utilizzate le informazioni fornite dall'utente tramite attribute inspector. Una volta terminata la decorazione dei template essi conterranno il codice finale e completo di un determinato file e saranno salvati come classe Java o XML nell'apposito path del progetto di Android Studio da cui è stato lanciato il nostro plugin.

Questa procedura verrà eseguita per ogni activity presente nel modello grafico e sarà gestita dal *CodeGenerator*. Esso per prima cosa andrà a decorare i template dei file da generare servendosi di appositi metodi forniti dalla classe dell'activity. In particolare ogni classe activity metterà a disposizione i seguenti metodi:

- *createJavaCode( )*: permette di generare le classi Java contenenti il codice nativo dell'activity.
- createFragmentCode( ): permette di generare le classi Java contenenti il codice nativo dell'activity nel caso in cui essa dovrà essere realizzata sotto forma di fragment.
- createXMLCode( ): genera i file di layout (XML) dell'activity.

In questi metodi le porzioni di codice riguardanti gli intents contenuti nell'activity saranno generate a loro volta mediante l'uso di specifici template o l'invocazione degli appositi metodi della classe intent *getIntentCode( )* e *getLayoutCode( ).*

Dopo aver decorato i vari template grazie ai metodi esposti, il CodeGenerator andrà a formattare i file creati, in modo tale che il codice in essi contenuto presenti una corretta indentazione. A questo punto i template, decorati e formattati, verranno utilizzati per la creazione dei file veri e propri che saranno salvati negli appositi path del progetto di Android Studio. Nel caso fossero già presenti dei file con lo stesso nome, il CodeGenerator provvederà a salvare i vecchi file in un apposito package */overwrited*, evitando così sovrascritture accidentali e consentendo all'utente di poter riportare le modifiche effettuate nei vecchi file in quelli nuovi mediante copia-incolla.

Queste operazioni verranno effettuate per ogni activity presente nel modello.

L'ultimo step del processo di generazione del codice riguarda la creazione dell'AndroidManifest. Il CodeGenerator raccoglierà i vari elementi di tipo <activity> da inserire nel Manifest invocando i metodi *getManifest( )* di tutte le activities che compongono il modello. Sarà compito di questi metodi andare a verificare se la relativa activity è impostata come schermata iniziale dell'applicazione e in tal caso aggiungere il necessario <intent-filter> all'elemento <activity>. Infine, una volta aggiunte anche le necessarie permissions, l'AndroidManifest viene salvato nel relativo path del progetto.

Di seguito viene riportato un diagramma di sequenza (sequence diagram) in cui vengono mostrati tutti i passaggi dello scenario di generazione del codice nativo.

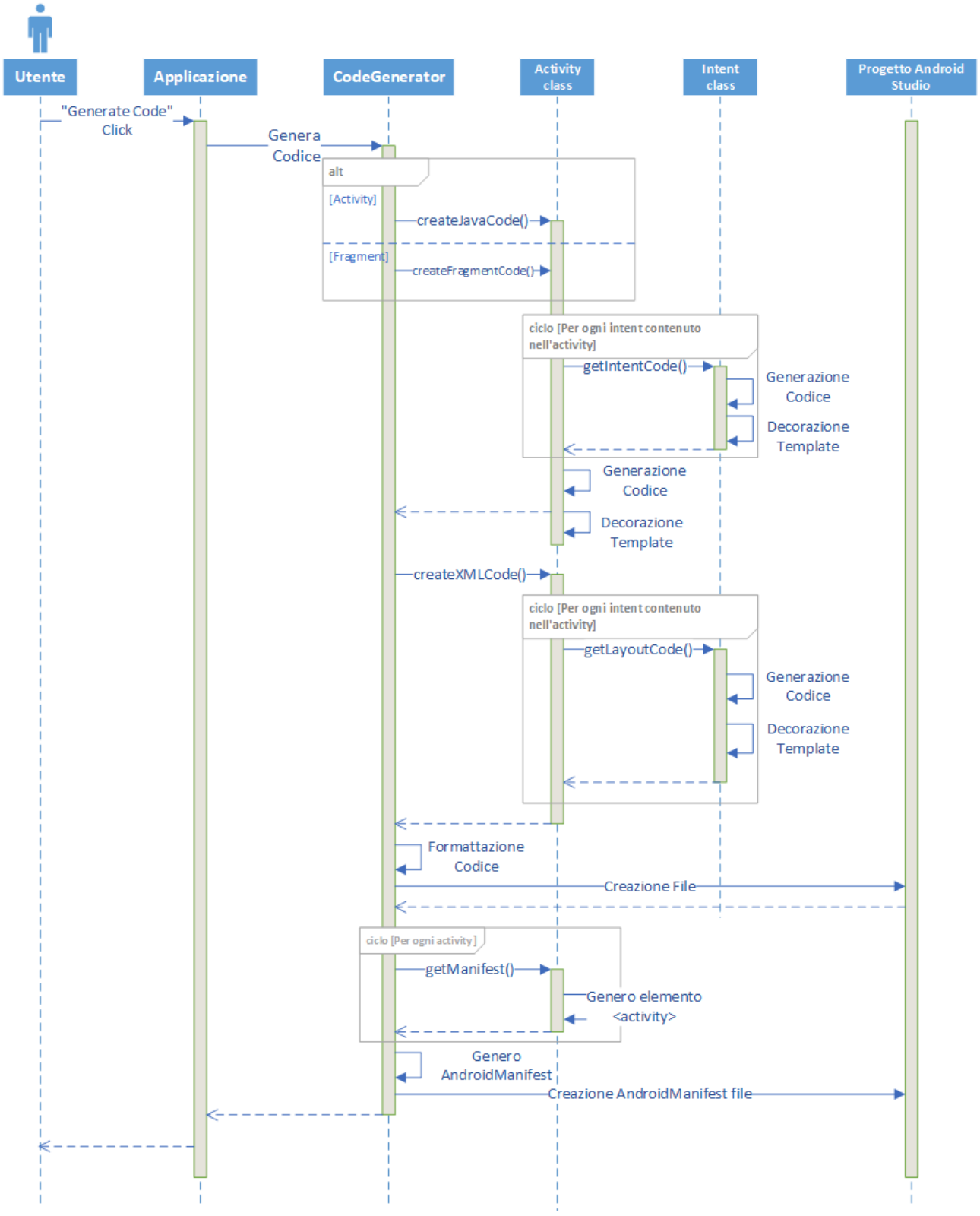

Figura 3.7: Diagramma di sequenza dello scenario di generazione del codice nativo

3.4.2 Analisi del codice generato

A conclusione della descrizione dell'architettura software del progetto realizzato, si procederà ora con un'analisi del codice generato nel processo di traduzione del modello grafico in codice nativo. A tale scopo verrà fornito al lettore un elenco delle classi Java e dei file di layout generati per ogni tipologia di activity, di cui sarà descritto il contenuto. Dopodiché per ogni tipologia di intent verrà descritto il codice generato nelle classi e nei file di layout delle activities che lo contengono.

Da notare che per ogni classe Java realizzata verrà generato automaticamente il blocco di generazione degli import necessari.

3.4.2.1 Codice generato dalle Activities

Empty Activity

EmptyActivity.java

Implementa la logica applicativa dell'activity. Presenta un metodo *onCreate()* che viene chiamato alla creazione dell'activity.

activity_empty.xml

Definisce il layout dell'activity ed è costituito da un RelativeLayout vuoto.

Basic Activity

BasicActivity.java

Implementa la logica applicativa dell'activity. Presenta un metodo *onCreate(),* invocato alla creazione dell'activity, al cui interno viene dichiarato un Floating Action Button con il relativo metodo *onClick()*. Se all'activity non è stato aggiunto nessun Floating Action Button Click intent, il metono *onClick()* genererà una Snackbar che mostrerà il messaggio "Replace with your own action".

activity_basic.xml

Definisce il layout dell'activity ed è costituito da un RelativeLayout contenente un Floating Action Button posto nella parte in basso a destra dello schermo.

Login Activity

LoginActivity.java

Implementa la logica applicativa dell'activity. Presenta un metodo *onCreate()* che viene chiamato alla creazione dell'activity e contiene i listener relativi ai campi per l'inserimento di email e password. Per quanto riguarda questi campi, vengono inoltre implementati tutti i metodi di controllo della validità dei valori inseriti (lunghezza, formato della mail, validità delle informazioni, campi vuoti) e tutta la logica applicativa relativa all'auto completamento della mail. Viene inoltre implementata un'AsyncTask fittizia che sarà avviata al click del pulsante di sign-in e potrà essere sviluppata dall'utente.

activity_login.xml

Il layout prende la forma di un LinearLayout, al cui interno vengono definiti una ProgressBar (utilizzata dalla AsyncTask fittizia), un TextInputLayout contenente la AutoCompleteTextView utilizzate per l'inserimento della mail, un TextInputLayout contenente un EditText utilizzato per l'inserimento della password ed un Button per la realizzazione del pulsante di sign-in.

List View

ListView.java

Implementa la logica applicativa dell'activity. Presenta un metodo *onCreate()* che viene chiamato alla creazione dell'activity e si occuperà di creare la lista caricare il contenuto di ogni riga.

CustomAdapter.java

Viene realizzato nel caso in cui l'utente, tramite attribute inspector, abbia selezionato un contenuto di tipo "Custom" per le righe ed implementerà un ArrayAdapter di un oggetto custom definito in *CustomModel.java*. Contiene il metodo *getView()* che restituisce la View di una riga della List View. La View restituita sarà strutturata secondo il layout definito in *activity_custom_row.xml*. Questo metodo utilizzerà un pattern di tipo ViewHolder che permette di evitare ritardi di caricamento del contenuto in seguito allo scrolling della lista. Infatti, salvando le varie View in un ViewHolder esse saranno subito a disposizione del sistema, permettendo all'applicazione di risparmiare il tempo impiegato a rigenerarle.

CustomModel.java

Implementa l'oggetto utilizzato dall'ArrayAdapter e rappresenta il modello del contenuto di una riga della ListView. Contiene due attributi di tipo String, chiamati title e subtitle, il costruttore ed i relativi metodi di getter e setter.

activity_list_view.xml

Definisce il layout dell'activity ed è costituito da un RelativeLayout in cui è racchiusa una ListView che verrà utilizzata per mostrare il contenuto delle varie righe della List View Activity.

activity_list_view_row.xml

Definisce il layout di una riga della List View. Nel caso in cui il tipo di contenuto selezionato è "String" esso contiene una TextView in cui sarà mostrato il contenuto testuale della riga. Se il contenuto richiesto è invece di tipo "Custom", sarà composto da un LinearLayout contenente due TextView destinate a mostrare il valore degli attributi title e subtitle definiti in *CustomModel.java*.

Grid View

GridView.java

Implementa la logica applicativa dell'activity. Presenta un metodo *onCreate()* che viene chiamato alla creazione dell'activity e si occuperà di creare la griglia e caricare il contenuto di ogni cella.

CustomAdapter.java

Viene realizzato nel caso in cui l'utente, tramite attribute inspector, abbia selezionato un contenuto di tipo "Custom" per le celle ed implementerà un ArrayAdapter di un oggetto custom definito in *CustomModel.java*. In essa troveremo il metodo *getView()* che restituisce la View di una cella della Grid View. La View restituita sarà strutturata secondo il layout definito in *activity_custom_cell.xml*. Anche in questo caso viene utilizzato un pattern di tipo ViewHolder che permette di evitare ritardi di caricamento del contenuto in seguito allo scrolling della schermata. Salvando le varie View in un ViewHolder esse saranno subito a disposizione del sistema, permettendo all'applicazione di risparmiare il tempo impiegato a rigenerarle.

CustomModel.java

Implementa l'oggetto utilizzato dall'ArrayAdapter e rappresenta il modello del contenuto di una cella della GridView. Presenta due attributi di testuali, chiamati title e subtitle, l'implementazione del costruttore ed i relativi metodi di getter e setter.

activity_grid_view.xml

Definisce il layout dell'activity ed è costituito da un RelativeLayout in cui è racchiusa una GridView che verrà utilizzata per mostrare il contenuto delle varie celle della Grid View Activity. Il numero delle colonne della GridView sarà ricavato dal valore impostato dall'utente per mezzo dell'attribute inspector.

activity_grid_view_cell.xml

Definisce il layout di una cella della Grid View. Se il tipo di contenuto selezionato è "String" esso conterrà una TextView in cui sarà mostrato il contenuto testuale della cella. Atrimenti, in caso di contenuto "Custom", sarà composto da un LinearLayout contenente due TextView destinate a mostrare il valore degli attributi title e subtitle definiti in *CustomModel.java*.

Card View

CardView.java

Contiene la logica applicativa dell'activity. Presenta un metodo *onCreate()* che sarà invocato alla creazione dell'activity e si occuperà di creare le varie Card da mostrare nella schermata. Per fare ciò istanzierà un LinearLayoutManager o GridLayoutManager (in base alla tipologia di layout impostata mediante attribute inspector) ed una RecyclerView che fornirà l'Adapter utilizzato per decorare le varie Cards. Questo Adapter sarà implementato in *CardViewAdapter.java* di cui segue una descrizione.

CardViewAdapter.java

Come detto questo Adapter estende una classe definita nell'ambito della RecyclerView. La RecyclerView è stata introdotta in Android in seguito alla nascita del *Material Design* e viene utilizzata in particolare per l'ottimizzazione delle Card View. Essa infatti, unita al pattern ViewHolder, andrà a ridurre i rallentamenti provocati dalla continua rigenerazione di elementi necessaria per le funzioni di scrolling della schermata. In particolare, la RecyclerView ricicla il più

possibile le view che crea per mostrarne gli elementi mentre il ViewHolder conserva i riferimenti ai widget interni ad ogni elemento.

In questa classe saranno implementati i metodi *onCreateViewHolder()* e *onBindViewHolder()* che rispettivamente rappresentano il momento in cui un elemento della RecyclerView viene creato ed il momento in cui vengono recuperati i riferimenti ai suoi elementi interni da popolare con i nuovi dati. Sarà in fine presente un metodo *getItemCount()* che restituirà il numero delle Cards generate.

CardViewModel.java

Implementa l'oggetto utilizzato dall'Adapter e rappresenta il modello del contenuto di una Card. È formato da due attributi di tipo String, chiamati title e subtitle, il costruttore ed i relativi metodi di getter e setter.

activity_card_view.xml

Definisce il layout dell'activity ed è costituito da un RelativeLayout in cui è racchiusa la RecyclerView.

activity_card_view_card.xml

Definisce il layout delle Card. La struttura è realizzata tramite un'oggetto CardView, che contiene due TextView per la visualizzazione degli attributi title e subtitle, le quali sono organizzate in un LinearLayout.

Tabbed Activity

TabbedActivity.java

Presenta un metodo *onCreate()*, chiamato alla creazione dell'activity, al cui interno viene implementata tutta la logica applicativa dell'activity. La logica applicativa si baserà sull'utilizzo di due elementi portanti: un ViewPager e un FragmentPagerAdapter. Il ViewPager si occuperà dell'implementazione della funzionalità di scrolling orizzontale che potrà essere utilizzata per la navigazione fra le varie sezioni (fragment) della schermata, mentre il FragmentPagerAdapter gestirà la creazione del contenuto delle varie sezioni e la relativa visualizzazione a schermo in seguito alla pressione del corrispondente tab o allo scrolling.

activity_tabbed.xml

Definisce il layout dell'activity. La struttura di base è costituita da un CoordinatorLayout in cui sono definiti il ViewPager e un AppBarLayout, composta da Toolbar e TabLayout contenente tutti i TabItem relativi alle varie sezioni dell'activity.

Bottom Navigation Activity

BottomNavigationActivity.java

Definisce la logica applicativa dell'activity. In questa classe troviamo un onNavigationItemSelectedItem, messo a disposizione dalla BottomNavigationView, che si occuperà di gestire la risposta al click dei vari items, mostrando a video i relativi fragment. Per la realizzazione delle funzionalità dell'activity esso si servirà di un FragmentManager ed un FragmentTransaction. All'interno del metodo *onCreate()* dell'activity, viene invece definito il codice che definisce quale delle sezioni visualizzare all'apertura della schermata.

activity_bottom_navigation.xml

Definisce l'aspetto grafico dell'activity. Il layout è di tipo ConstraintLayout e contiene un FrameLayout utilizzato per visualizzare il contenuto della sezione selezionata ed una BottomNavigationView che consente di visualizzare la barra con i vari items, la cui struttura specifica è definita nel file *navigation.xml*.

navigation.xml

Definisce la struttura della BottomNavigationBar, specificando tutti gli tems in essa contenuti, ognuno con il proprio testo e la propria icona.

Icone

Nel path */res/drawable/* verranno creati i file XML relativi alle varie icone utilizzate negli items della BottomNavigationActivity

Fragment

Nelle precedenti sezioni è stato spiegato che le activities utilizzate per realizzare le varie sezioni presenti in TabbedActivity e BottomNavigationActivity vengono realizzate sotto forma di fragment.

Questa distinzione è ininfluente nella realizzazione del modello grafico ma assume importanza in fase di generazione del codice dove, a seconda che la classe realizzata sia un'activity oppure un fragment, il codice da creare sarà parzialmente diverso. In particolare una classe contenete la logica applicativa di un fragment dovrà estendere la classe Fragment, mentre il metodo *onCreate()* presente nelle activities sarà rimpiazzato da *onCreateView()*, utilizzato dall'activity che contiene il fragment per crearlo. La View restituita da questo metodo sarà inoltre utilizzare per la definizione degli elementi presenti nel fragment (bottoni, floating action buttons, list view…).

I file di layout resteranno invece invariati.

3.4.2.2 Codice generato dagli Intents

In questa sezione vedremo come viene modificato il codice di un'activity dalla presenza dei vari tipi di intents. In particolare per ogni tipo di intents verrà descritto il codice che esso andrà ad aggiungere alle classi Java e ai file di layout delle activities che li implementano.

Button Click

La presenza di un Button Click intent va ad aggiungere alla classe dell'activity la dichiarazione di un oggetto di tipo Button, il cui ID sarà quello impostato dall'utente tramite attribute inspector, e il recupero di un riferimento alla View del bottone tramite il metodo *findViewById()*. L'Intent vero e proprio verrà creato nel metodo *setOnClickListener()*, permettendo così di avviare l'activity di destinazione con un click del bottone.

Se è stato impostato mediante attribute inspector un particolare tipo di Extras, verranno dichiarate una variabile statica di tipo Stringa chiamata EXTRA_MESSAGE che costituirà l'identificativo dell'Extras ed una variabile del tipo specificato che costituirà il contenuto dell'Extras. Tali variabili verranno utilizzate per aggiungere l'Extras all'intent generato, mediante il metodo *intent.putExtra()*. Nell'activity lanciata dall'intent verrà generato il codice per la lettura dell'Extras inviato.

Per quanto riguarda il file di layout, verrà aggiunto un elemento di tipo Button, che mostrerà il testo specificato dall'utente nell'attribute inspector. Se l'activity implementa più di un Button Click intent, i vari Button creati verranno inseriti in un LinearLayout con orientamento verticale.

Floating Action Button Click

La presenza di un Floating Action Button Click intent va ad aggiungere alla classe dell'activity il codice relativo al recupero della View di un oggetto di tipo FloatingActionButton, mediante il metodo *findViewById()*. L'Intent vero e proprio verrà creato nel metodo *setOnClickListener()*, permettendo così di avviare l'activity di destinazione con un click del FloatingActionButton.

Anche in questo caso, la presenza di un Extras comporterà l'aggiunta della dichiarazione delle variabili relative all'id e al contenuto dell'Extras. Il metodo *intent.putExtra()* permetterà di aggiungere l'Extras all'intent generato. Nell'activity lanciata dall'intent verrà generato il codice per la lettura dell'Extras inviato.

Nel file di layout sarà invece presente un FloatingActionButton, posto nella parte in basso a destra della schermata.

Login Intent

Permette la creazione di un intent all'interno del codice che gestisce le operazioni da svolgere in seguito al click del pulsante di sign-in.

La presenza di un Extras comporterà l'aggiunta della dichiarazione delle variabili relative all'id e al contenuto dell'Extras. Il metodo *intent.putExtra()* permetterà di aggiungere l'Extras all'intent generato. Nell'activity lanciata dall'intent verrà generato il codice per la lettura dell'Extras inviato.

Il Login Intent non apporta nessun cambiamento al layout della schermata.

Button Click With Result

Questo tipo di intent andrà ad aggiungere del codice sia nella classe Java dell'activity chiamante, sia in quella dell'activity destinataria. In particolare, nell'activity chiamante verrnno generati:

- La dichiarazione di un oggetto di tipo Button, il cui ID sarà quello impostato dall'utente tramite attribute inspector, e il recupero di un riferimento alla View del bottone tramite il metodo *findViewById()*.
- Nel metodo *setOnClickListener()* del bottone verrà implementato il codice dell'intent vero e proprio che permetterà di lanciare l'activity destinataria mediante il metodo *startActivityForResult()*.

- Le variabili per la generazione di un'eventuale Extras e il metodo *intent.putExtra()* per allegarlo all'intent.
- Il metodo *onActivityResult()* che permetterà di analizzare la risposta ricevuta dall'activity destinataria ed estrarne il valore di un eventuale Extras ad essa allegato.

Nell'activity destinataria verranno invece generati:

- La dichiarazione di un oggetto di tipo Button, utilizzato per inviare l'intent di risposta all'activity chiamante. L' ID di questo bottone sarà quello impostato dall'utente tramite attribute inspector. Tramite il metodo *findViewById()* verrà recuperato il riferimento alla View del bottone.
- Il codice per leggere un'eventuale Extras allegato all'intent proveniente dall'activity chiamante.
- L'intent vero e proprio viene aggiunto nel metodo *setOnClickListener()* del bottone dove, mediante il metodo *setResult()* verrà aggiunta la risposta che sarà inviata all'activity chiamante all'invocazione del metodo *finish()*.
- Le variabili per la generazione di un'eventuale Extras e il metodo *intent.putExtra()* per allegarlo all'intent.

A livello grafico verrà aggiunto un bottone nel file di layout dell'activity chiamante ed uno in quello dell'activity ricevente. Entrambi i bottoni conterranno il testo impostato mediante il relativo attribute inspector.

List Item Click e Grid Item Click

La presenza di un List Item Click modificherà la List View che lo contiene andando ad aggiunge all'interno dell'*onCreate()* un listener che permette di invocare un Intent mediante il click di una riga della lista. Questo listener sarà realizzato mediante il metodo *setOnItemClickListener()* messo a disposizione dall'oggetto ListView. Anche in questo caso la presenza di un Extras genererà il relativo codice delle variabili necessarie e dei metodi per allegarlo all'intent descritti nei precedenti casi.

Nell'activity destinataria verrà inserito il codice per consentire la cosiddetta "navigazione all'indietro". Questa funzionalità è realizzata mediante l'inserimento di una piccola freccetta nella parte sinistra dell'Action Bar. Cliccando questa freccetta si ritornerà all'activity chiamante (ovvero l'activity che ha lanciato la schermata contenente la freccetta mediante un List Item Click intent). Questo costrutto permette di realizzare un pattern molto utilizzato per funzionalità di tipo master-detail, permettendo attraverso il click della freccetta di richiamare la schermata master da quella detail. L'inserimento di questo pattern comporta una modifica dell'AndroidManifest file, in cui andrà specificata la parent activity (master) nell'elemento <activity> dell'activity destinataria (detail).

Questo tipo di intent non apporta alcuna modifica ai file di layout.

Il Grid Item Click andrà a generare lo stesso tipo di codice descritto per il View Item Click, ovvero un listener per invocare intent mediante il click di una cella della griglia realizzato mediante il metodo *setOnItemClickListener()* messo a disposizione dall'oggetto GridView, il codice relativo a un eventuale Extras e l'aggiunta della freccetta nell'activity destinataria con conseguente aggiornamento dell'AndroidManifest.

Card Click

La presenza di un Card Click intent non solo andrà a modificare la classe dell'activity in cui viene implementato ma genererà anche un nuova classe CardViewClickListener.java che andrà ad implementare il listener necessario per gestire le gestures ed in particolare il click delle Cards presenti nella Card View. Questo listener sarà utilizzato nel metodo *onCreate()* dell'activity per lanciare l'intent vero e proprio.

Anche in questo caso la presenza di un Extras genererà il relativo codice delle variabili necessarie e dei metodi per allegarlo all'intent descritti in precedenza.

Come si è visto per gli intent di tipo List Item Click e Grid Item Click, anche questo intent aggiunge all'activity destinataria il codice per la realizzazione della freccetta posta nell'ActionBar ed utilizzata per la navigazione all'indietro.

I file di layout non subiscono alcuna modifica.

Tab Intent

Un Tab Intent va a modificare il FragmentPagerAdapter, responsabile della gestione delle varie sezioni contenute nella relativa Tabbed Activity, andando ad aggiungervi un nuovo tab.

Per quanto riguarda l'aspetto grafico verrà aggiunto un nuovo TabItem al TabLayout contenuto nel file di layout della Tabbed Activity.

Bottom Navigation Item

Un Bottom Navigation Item intent va a modificare l'oggetto onNavigationItemSelectedItem, contenuto nella relativa Bottom Navigation Activity, andando ad aggiungere una nuova sezione a quelle già presenti.

Per quanto riguarda i file di layout verrà aggiunto un nuovo <item> al file incaricato della costruzione del menu caricato nella Bottom Navigation Bar (*navigation.xml*) e verrà creato un nuovo file XML che definisce l'icona del nuovo item aggiunto.

Nel prossimo capitolo verrà mostrata un'analisi quantitativa dello sviluppo di un'applicazione mobile tramite l'utilizzo del plugin di estensione di Android Studio descritto fino ad ora, al fine di dimostrare le reali capacità di questo strumento nella fase iniziale dello sviluppo di applicazioni mobili.

Capitolo 4

Valutazioni

In questo capitolo viene presentato un esempio di processo di sviluppo di un'applicazione mobile tramite l'utilizzo dell'estensione di Android Studio realizzata per questo progetto di tesi. Dopo aver descritto le funzionalità dell'applicazione ed il modello utilizzato per crearla verrà presentata un'analisi quantitativa del codice sorgente che è stato possibile generare nella fase iniziale dello sviluppo mediante il modello grafico del plugin realizzato.

4.1 Applicazione realizzata

Per analizzare i risultati prodotti dall'estensione di Android Studio è stata creata un'applicazione mobile per smartphone chiamata *Ricettario*. L'applicazione presenta una serie di ricette di cucina suddivise per categoria che l'utente può consultare ed aggiungere alle ricette preferite. Per ogni ricetta l'applicazione mostra le informazioni di base comprendenti una valutazione compresa tra 1 e 5 stelle, l'elenco degli ingredienti necessari, il procedimento di preparazione e le recensioni pubblicate dagli altri utenti. L'utente può pubblicare a sua volta una recensione, votando l'applicazione ed aggiungendo un proprio commento. La valutazione andrà a sommarsi a quella fornita dagli altri utenti aggiornando così la valutazione media complessiva della ricetta. L'applicazione presenterà inoltre una schermata in cui verranno mostrate le ricette in evidenza, ovvero le ricette più apprezzate dagli utenti. Si presenta di seguito un grafico che descrive il modello dell'applicazione generato mediante l'estensione di Android Studio.

Figura 4.1: Modello dell'applicazione Ricettario

Come si evince dal modello appena proposto, la schermata iniziale dell'applicazione è costituita da una schermata di login che permette l'autenticazione e la registrazione degli utenti. Una volta eseguita l'autenticazione apparirà la schermata principale dell'applicazione che sarà divisa in tre sezioni, accessibili attraverso una Bottom Navigation Bar, premendo la relativa icona. La prima sezione "Ricette" mostra l'insieme delle categorie di ricette disponibili, disposte in una struttura a griglia. Ogni elemento della griglia è costituito da un'immagine identificativa e dal nome della categoria. Premendo una determinata cella si aprirà una schermata contenente l'elenco delle ricette della categoria selezionata. La sezione "Preferiti" mostra l'elenco delle ricette selezionate come preferite dall'utente, mentre nella sezione "In evidenza" vengono elencate le cinque ricette con la valutazione più elevata. Gli elementi degli elenchi visualizzati nelle sezioni "Preferiti" e "In evidenza" così come l'elenco delle ricette mostrato in seguito al click di una determinata categoria, sono stati realizzati mediante liste di Card. Ogni Card è costituita dalla foto della ricetta, dal nome e dalla valutazione in numero di stelle.

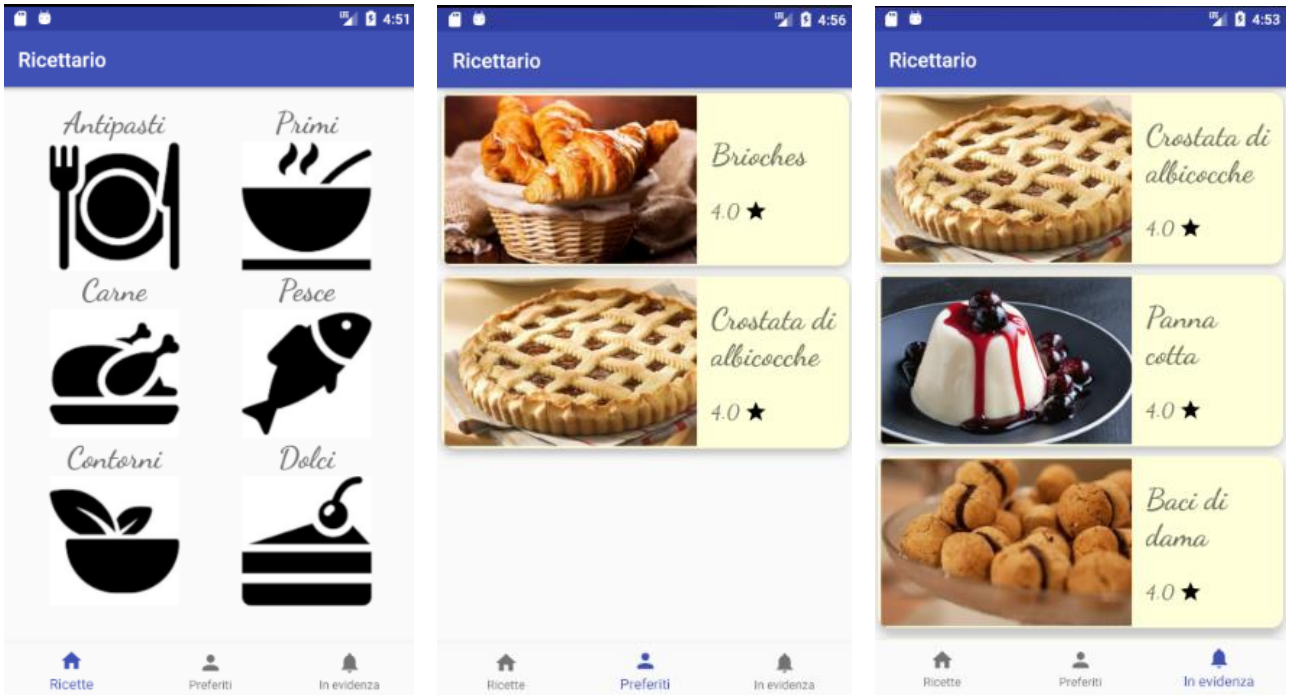

Figura 4.2: Sezioni della schermata Main dell'applicazione Ricettario

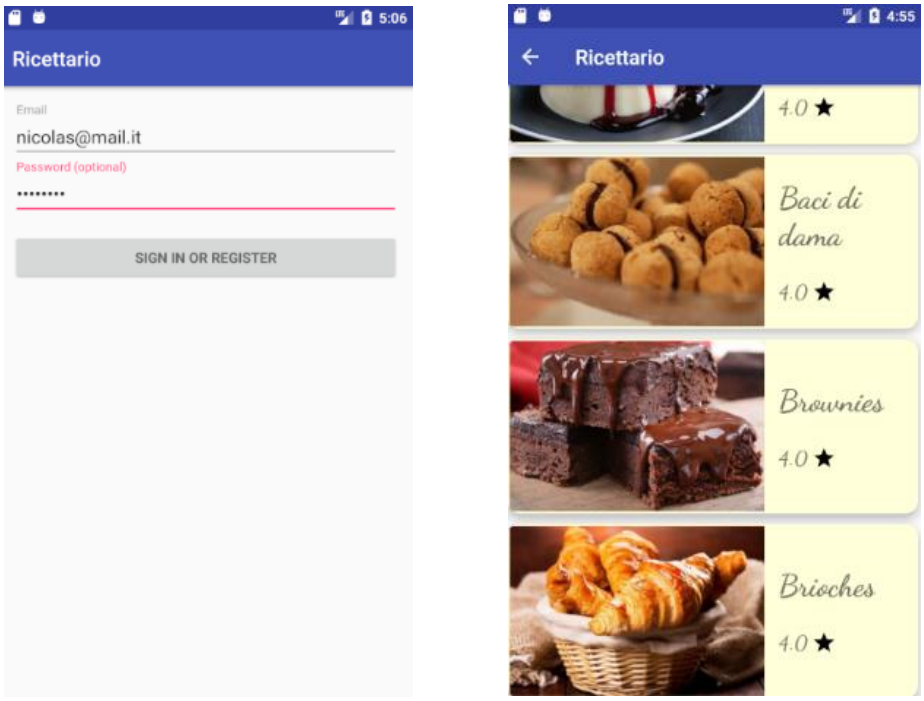

Figura 4.3: Schermate di login ed elenco ricette per categoria dell'applicazione Ricettario

Cliccando su una di queste card (in ciascuna delle tre schermate a elenco appena descritte), verrà aperta la schermata relativa alla ricetta selezionata. Questa schermata presenta quattro sezioni raggruppate in una struttura a tab, posta nella Toolbar. È possibile navigare fra le varie sezioni premendo il relativo tab o mediante gestures orizzontali. Nell'Action Bar di questa schermata si trova inoltre la freccetta che consente la navigazione all'indietro verso l'elenco da cui è stata lanciata questa schermata. La prima sezione presenta la scheda della ricetta selezionata, composta da un'intestazione, che ne mostra l'immagine e il nome, e da informazioni di base come la difficoltà di preparazione, il tempo di cottura, il numero di dosi e la valutazione. Posizionato in basso a destra troviamo inoltre un Floating Action Button con un icona a forma di cuore che ci permette di aggiungere o rimuovere la ricetta dal nostro elenco dei preferiti.

Le sezioni "Ingredienti" e "Ricetta" contengono rispettivamente l'elenco degli ingredienti, composto dal nome di ogni ingrediente con il rispettivo dosaggio, e quello dei vari passaggi del procedimento di preparazione. L'ultima sezione presenta invece le recensioni degli utenti riguardanti la ricetta selezionata. Anche in questo caso la schermata sarà formata da una lista di Cards rappresentanti le varie recensioni. Ogni Card conterrà lo username dell'utente che ha pubblicato la recensione, la valutazione da esso data alla ricetta e il suo commento.

Per poter aggiungere la propria recensione basterà cliccare il Floating Action Button con un icona a forma di *più*, posto nella parte in basso a destra della schermata, aprendo così una nuova activity che permetterà di scrivere il proprio commento e di valutare la ricetta selezionando il relativo numero di stelle. Premendo un Button con la scritta “Pubblica” la recensione verrà pubblicata e si verrà reindirizzati alla schermata della scheda della ricetta che presenterà ora una valutazione aggiornata tenendo conto della nostra votazione.

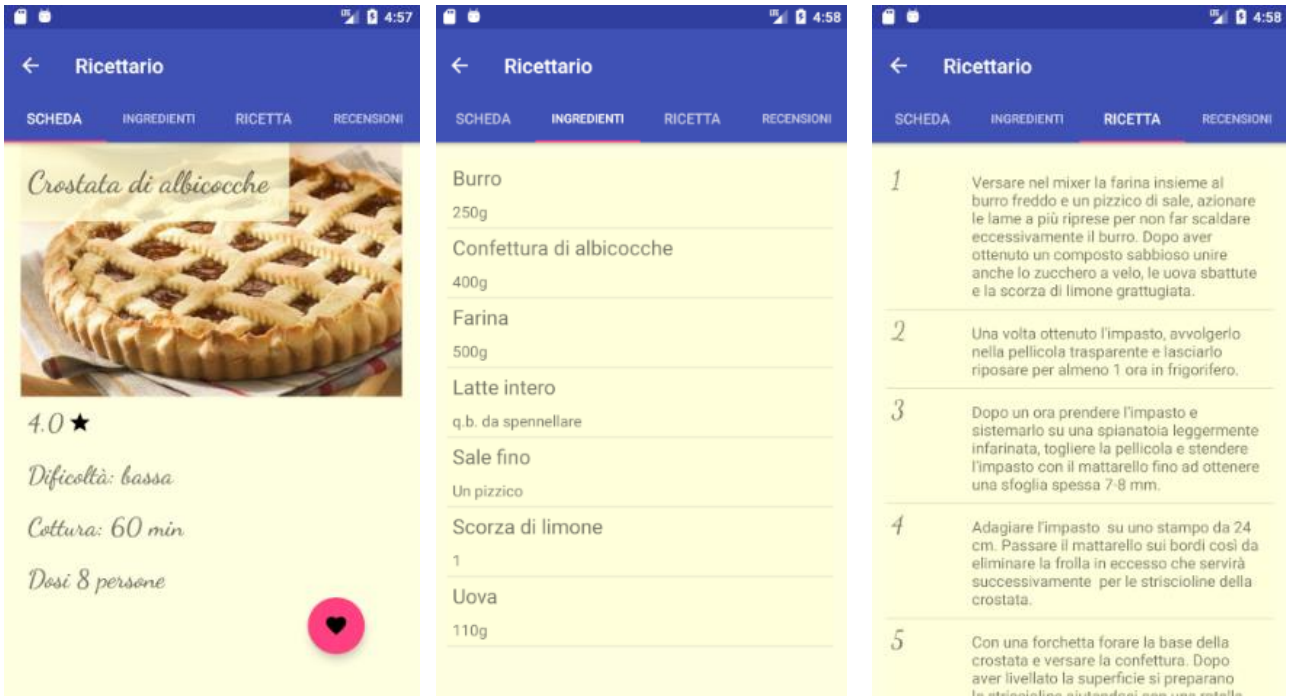

Figura 4.4: Sezioni dell’activity Recipe dell’applicazione Ricettario

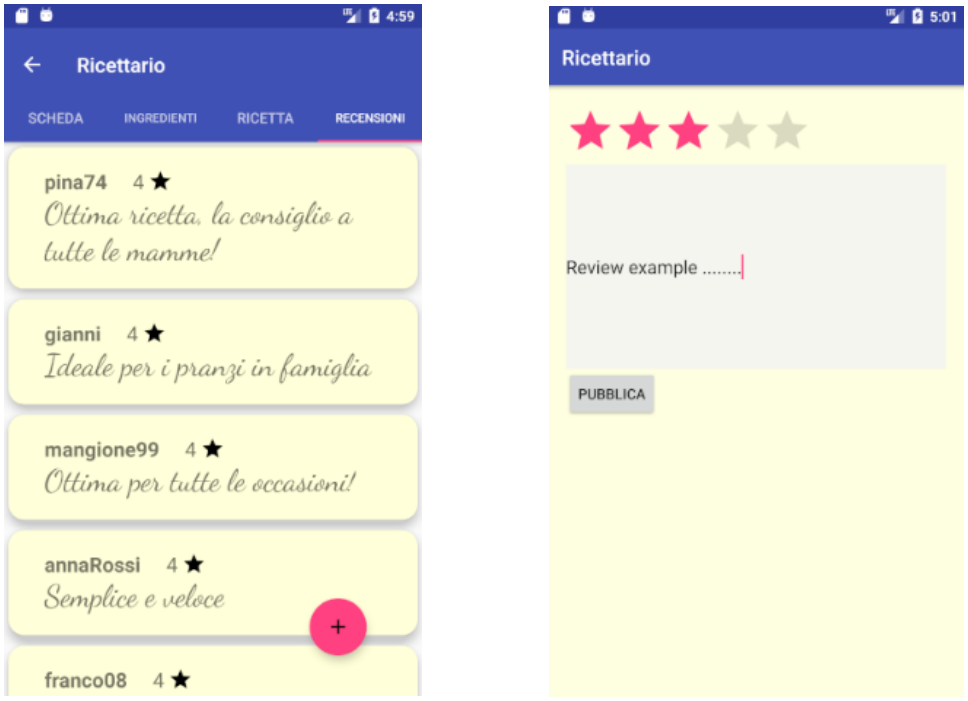

Figura 4.5: Schermate elenco e aggiunta recensioni dell'applicazione Ricettario

4.2 Analisi quantitativa e valutazioni finali

Al fine di analizzare i risultati prodotti da questa ricerca verrà ora fornita un'analisi quantitativa del codice nativo dell'applicazione *Ricettario* che è stato possibile generare nella fase iniziale dello sviluppo mediante l'utilizzo dell'estensione di Android Studio realizzata per questo progetto di tesi.

Nella fase di composizione del modello grafico le varie activities messe a disposizione hanno permesso di realizzare tutte le schermate presenti nell'applicazione. Nello specifico la schermata di login è stata realizzata mediante una Login Activity, per le schermate Main e Recipe sono state usate rispettivamente una Bottom Navigation Activity e una Tabbed Activity, le cui sezioni sono state automaticamente realizzate sotto forma di Fragment. Per le varie schermate contenenti i diversi elenchi di Cards relative alle ricette e alle recensioni sono state usate delle Card View con layout di tipo lista. Una Grid View con contenuto custom ha permesso di realizzare il menu di scelta delle categorie, mentre grazie a delle List View con contenuto custom sono stati realizzati gli elenchi degli ingredienti e dei passaggi del procedimento di preparazione.

Per quanto riguarda infine la scheda della ricetta e la form per la creazione di una nuova recensione, dato il livello di personalizzazione richiesto da queste schermate sono state utilizzate rispettivamente una Empty Activity e una Basic Activity che forniscono all'utente la struttura base delle activities da sviluppare successivamente in base al contenuto richiesto.

Grazie alla visione d'insieme fornita dall'estensione realizzata e alle varie tipologie di intents messi a disposizione è stato possibile realizzate tutte le interazioni fra le varie schermate, generando così l'intero flusso di navigazione. In particolare, l'interazione fra la schermata di login e la Main activity è stata realizzata mediante un Login Intent. Le varie sezioni delle schermate Main e Recipe sono state aggiunte mediante intents di tipo Bottom Navigation Item e Tab Intent che attraverso i rispettivi attribute inspector hanno consentito di impostare le icone e scritte presenti nei vari items e tab, nonché l'ordine in cui inserire le varie sezioni. Per quanto riguarda la navigazione mediante click delle Cards delle varie ricette sono stati utilizzati dei Card Click intents mentre per passare dalla schermata contenente le varie categorie di ricette all'elenco delle ricette di una determinata categoria è stato necessario l'utilizzo di un Grid Item Click. Un Floating Action Button Click è stato utilizzato nella schermata contenente l'elenco delle recensioni per aprire la form dedicata alla creazione di una nuova activity. Per ritornare alla scheda della ricetta dopo aver confermato con l'apposito bottone il completamento della recensione è invece stato usato un Button Click intent. Infine, i vari tipi di informazioni passate da una schermata all'altra sono stati definiti settando il tipo di Extras desiderato mediante attribute inspector dei relativi intents.

Il codice nativo generato da questo modello implementa la logica applicativa e il layout di base di tutte le schermate e tutto il codice relativo al flusso di navigazione. I vari file generati sono stati salvati nei relativi path di Android Studio e sono quindi già a disposizione dell'utente per essere sviluppati o per il building dell'applicazione.

A partire dai file generati mediante il modello grafico sono state necessarie le seguenti aggiunte e modifiche al fine di ottenere l'applicazione *Ricettario* finale:

- Sono stati modificati i file XML di layout relativi alle Cards delle varie ricette, inserendo una ImageView per poter visualizzare la foto della ricetta e il codice relativo alle migliorie dell'aspetto grafico e del *look-and-feel* delle Cards.
- Sono stati sviluppati la logica applicativa ed il layout delle schermate riguardanti la scheda della ricetta e la form per l'inserimento di una nuova recensione.

- Sono state sviluppate tutte le classe necessarie alla realizzazione di una base di dati mediante *SQLite*, utilizzata per contenere le informazioni relative alle varie ricette, alle recensioni e i dati degli utenti registrati (username, email, password). In particolare, è stata creata la classe *MyDatabase.java* per la creazione della struttura della base di dati e di tutte le sue tabelle, *RecipeSQLiteRepository.java* per la gestione delle operazioni CRUD (Create Read Update Delete) sulla base di dati e *DbService.java* utilizzata per il popolamento di tutte le tabelle con i dati relativi a ricette, utenti e recensioni. Sono infine state create le varie classi *Cursor* necessarie per la lettura dei risultati delle varie query.

- Nelle classi delle schermate è stato inserito il codice relativo alla chiamata delle varie query per le operazioni sulla base di dati.

- Altre piccole modifiche di vario genere come sostituire i valori generici associati agli Extras con le informazioni effettive da allegare agli intents o l'aggiunta di una ImageView nelle celle della schermata inerente alle varie categorie di ricette.

Al fine di avere una valutazione quantitativa del risultato ottenuto è stata svolta un'analisi sulle *Logical SLOC* dell'applicazione, ovvero le righe di codice che compongono il progetto senza considerare i commenti e gli spazi vuoti fra gli *statements*. Di seguito viene riportata una tabella dei risultati ottenuti.

	App Generata	**App Completa**	**Percentuale**
Mobile XML	*520*	*705*	*73,8%*
Mobile Java SLOC	*1143*	*2087*	*54,8%*
Mobile Java SLOC senza codice Database	*1143*	*1450*	*78,8%*

Tabella 4.1: Copertura codice sorgente dell'applicazione Ricettario

I risultati riportati in tabella permettono di trarre una serie di conclusioni sullo strumento realizzato; in primo luogo possiamo vedere come i file XML relativi al layout di base delle varie schermate siano stati sufficienti a coprire un'ampia porzione del codice XML dell'intera applicazione. La parte non coperta riguarda le varie personalizzazioni alle Cards e al layout di alcune particolari schermate, finalizzate a migliorarne l'aspetto grafico ed il *look-and-feel*. Per quanto concerne invece il codice Java si può notare che l'aggiunta di una base di dati abbia influito significativamente sul risultato ottenuto, riducendo così la porzione di codice generato automaticamente a circa la metà del codice complessivo. Su questo dato pesano in particolare le classi utilizzati per costruire la struttura della base di dati, per la gestione delle operazioni CRUD e la classe dedicata al popolamento del database con tutte le informazioni riguardanti le diverse ricette e i dati fittizi dei vari utenti con le relative recensioni.

Considerando che l'obiettivo iniziale della ricerca era quello di fornire uno strumento destinato a generare il codice di base delle varie schermate dell'applicazione e del relativo flusso di navigazione, è stata aggiunta un ulteriore analisi delle righe di codice Java escludendo il codice relativo a creazione, popolamento e definizione delle funzioni per le operazioni CRUD del database (tenendo invece in considerazione i vari statements di interrogazione della base di dati presenti nelle activities), ottenendo un notevole incremento della percentuale di copertura. Ritengo che questo dato sia particolarmente significativo in quanto fornisce una visione più coerente rispetto agli obiettivi preposti. La parte di codice non coperta in questo caso riguarda le interazioni con la base di dati per l'inserimento o la lettura di dati, il codice delle due schermate contenenti la scheda della ricetta e il form per l'aggiunta di una recensione e l'aggiornamento delle classi Adapter e Model in seguito all'aggiunta di campi nelle Cards e nella Grid View delle categorie.

Per un'analisi a livello di tempistica invece, si può innanzitutto escludere il tempo necessario all'integrazione tra l'applicativo e la base di dati, poiché si tratta di una fase inevitabile sia attraverso l'uso del normale approccio via IDE sia in seguito alla generazione del codice mediante il plugin realizzato. Dunque, sarebbe possibile in linea teorica valutare quantomeno il tempo necessario a riprodurre lo stesso risultato ottenuto utilizzando il plugin, attraverso la stesura di codice nativo mediante IDE oppure valutando il tempo necessario per modificare ed ampliare il codice generato al fine di ottenere l'applicazione finale. Tuttavia si tratterebbe di una valutazione fortemente legata alle capacità dello sviluppatore e alle dimensioni dell'applicazione.

Ciò che si può affermare per certo sulla tempistica è la rapidità con cui il plugin realizzato permette di realizzare in pochi click la struttura di base e il flusso di navigazione dell'intera applicazione, generando un app mobile completa e perfettamente funzionante. Questo processo è reso più veloce, semplice e intuitivo dalla visione d'insieme dell'applicazione fornita all'utente che permette di progettare direttamente l'applicazione nel suo insieme evitando le perdite di tempo e le complicazioni presenti negli strumenti che permettono di lavorare solo sulle singole schermate. In generale per la realizzazione dell'applicazione *Ricettario*, l'abbattimento dei tempi ottenuto è stimabile ad almeno il 50-60% del tempo totale di sviluppo mediante codice nativo.

Capitolo 5

Conclusioni

Come conclusione di questa ricerca si può affermare che lo strumento realizzato per questo progetto di tesi soddisfi gli obiettivi preposti. Esso infatti estende le funzionalità di Android Studio aggiungendo la possibilità di generare il codice di un'applicazione mediante un approccio model-driven, permettendo di creare in modo semplice ed immediato un modello grafico in grado di fornire una visione d'insieme dell'intera applicazione e di poterne configurare l'intero flusso di navigazione, andando così a colmare il più possibile il gap con la relativa controparte dell'ambiente iOS rappresentata da Storyboard. Al tempo stesso, benché risulti oggettivamente impossibile stabilire numericamente i vantaggi forniti in termini di tempi di sviluppo, poiché sarebbe necessario avere a disposizione un grosso campione di applicazioni e sviluppatori, è ragionevole pensare che la possibilità di realizzare gran parte della logica applicativa (operazione che comporterebbe l'impiego di una notevole quantità di tempo indipendentemente dall'abilità dello sviluppatore) in pochi minuti per mezzo dell'editor grafico permetta di garantire l'abbattimento dei tempi e costi di sviluppo dell'approccio nativo pur mantenendone intatte tutte le potenzialità.

5.1 Sviluppi futuri

Per quanto riguarda gli eventuali sviluppi futuri dello strumento realizzato è possibile individuare diversi possibili scenari di sviluppo. Il primo riguarda sicuramente l'ampliamento dei componenti messi a disposizione dell'utente, andando ad aggiungere altre tipologie di activities e intents ed ampliando il numero di configurazioni e personalizzazioni disponibili per quelli già presenti. Questo consentirebbe di poter creare applicazioni ancora più complesse e di aumentare ulteriormente la percentuale di codice coperto.

A tal proposito, risulta evidente dall'analisi quantitativa descritta nel precedente capitolo che l'elemento che più ha inciso nel calcolo delle *Logical SLOC* dell'applicazione sia stato il codice relativo all'integrazione e sviluppo di una base di dati. Partendo da questo dato sarebbe interessante provvedere ad introdurre una sezione all'interno del plugin che permetta all'utente di creare le varie tabelle *SQLite* e le loro proprietà relazionali, per poi generarle automaticamente in aggiunta al codice attualmente generato. Questo andrebbe ad aumentare considerevolmente la copertura di codice e a ridurre ulteriormente i tempi e costi di sviluppo.

Infine, potrebbe essere utile affiancare alla natura *model2text*, fornita dall'approccio model-driven, un meccanismo *text2model* che permetta di aggiornare in real time il modello grafico del plugin in base alle modifiche effettuate direttamente sul codice dall'utente.

Bibliografia

[1] **Apple Inc**. - Cupertino, California, USA
http://www.apple.com

[2] **Samsung Group** - Samsung Town, Seoul, Corea del Sud
http://www.samsung.com

[3] **Huawei Technologies Co. Ltd.** - Shenzhen, Guandong, Cina
http://www.huawei.com

[4] **International Data Corporation** – Smartphone Vendor Market Share, 2017 Q1
https://www.idc.com/promo/smartphone-market-share/vendor

[5] **iOS** - Apple Inc. - http://www.apple.com/ios/

[6] **Android** - Google Inc. - http://www.android.com

[7] **Google Inc.** - Mountain View, California, USA
http://www.google.com

[8] **Windows Phone Operating System** - Microsoft Corporation
http://www.microsoft.com/it-it/windows/phones

[9] **BlackBerry Operating System** - BlackBerry Ltd.
https://us.blackberry.com/software/smartphones/blackberry-10-os

[10] **Gartner** - Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter Of 2017. 23 Maggio 2017.
https://www.gartner.com/newsroom/id/3725117

[11] **Microsoft Windows** - Microsoft Corporation - http://www.microsoft.com

[12] **TechCrunch** – Android Overtakes Windows As The Internet's Most Used Operating System. 3 Aprile 2017.
https://techcrunch.com/2017/04/03/statcounter-android-windows/

[13] **Android Studio** – Google Inc.

https://developer.android.com/studio/index.html

[14] **Xcode** – Apple Inc. - https://developer.apple.com/xcode/

[15] **Storyboard** – Apple Inc.

https://developer.apple.com/library/content/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

[16] **Material Design** - Google Inc. - http://material.google.com/

[17] **FluidUI** - Fluid Software - http://www.fluidui.com

[18] **Proto.io** - Labs Division of SNQ Digital - http://proto.io

[19] **InVisionApp** - InVision - http://www.invisionapp.com

[20] **Marvel App** - Marvel Prototyping LTD - https://marvelapp.com

[21] **Eclipse** - Eclipse Foundation - http://www.eclipse.org

[22] **AIDE** - https://play.google.com/store/apps/details?id=com.aide.ui

[23] **AngularJS** - Google Inc. - http://www.angularjs.org

[24] **Apache Cordova** - Apache Cordova - http://cordova.apache.org

[25] **Andromo** – Indigo Rose Software - https://www.andromo.com/

[26] **Appy Pie** - Pie Appy Inc. - https://it.appypie.com/

[27] **Buzztouch** - https://www.buzztouch.com/

[28] **Neonto Studio** - Neonto Ltd. - http://www.neonto.com/

[29] **IntelliJ IDEA** - JetBrains - https://www.jetbrains.com/idea/

[30] **JetBrains** - JetBrains s.r.o. - https://www.jetbrains.com/

[31] **Scene Builder** – Oracle

http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html

[32] **Curva di Bèzier** - Wikipedia - https://it.wikipedia.org/wiki/Curva_di_B%C3%A9zier

[33] **Oracle** - Using JavaFX Properties and Binding

https://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm

Appendice A

Lista degli acronimi

API	Application Programming Interface
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
HTML	Hyper-Text Markup Language
IDE	Integrated Development Environment
JDK	Java Development Kit
XML	eXtensible Markup Language
SLOC	Source Lines Of Code