

Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione



Tesi di Laurea

Progettazione e realizzazione di un'app Android per la gestione dei clienti di un ristorante

Design and realization of an Android app for the management of clients in a restaurant

Relatore

Prof. Domenico Ursino

Candidato

Mirko Simoni

Anno Accademico 2018-2019

Indice

Introduzione	3
1 Introduzione alle app	7
1.1 Un po' di storia	7
1.1.1 Nascita	7
1.1.2 Sviluppo	7
1.1.3 Situazione attuale	8
1.2 Tipi di app	9
1.2.1 App native	9
1.2.2 App ibride	10
1.2.3 Web app	11
1.3 Principali sistemi operativi supportati e relativi distributori	11
1.3.1 Android - Google Play Store	12
1.3.2 iOS - Apple Store	14
1.3.3 Windows - Windows Store	15
2 Android	17
2.1 Un po' di storia	17
2.2 Descrizione	18
2.2.1 Open Source	18
2.2.2 Interfaccia utente	18
2.2.3 Ambiente di ripristino e modalità provvisoria	19
2.2.4 Aggiornamenti	19
2.2.5 Sicurezza	19
2.2.6 Firmware	20
2.3 Architettura Android	20
2.3.1 Il kernel Linux	20
2.3.2 L'Hardware Abstraction Layer	21
2.3.3 Il runtime e le librerie native	22
2.3.4 Il framework di API Java	23
2.3.5 Le app di sistema	24
2.4 Cronologia versioni	24

IV Indice

3	Analisi dei requisiti	27
3.1	Descrizione dell'app	27
3.2	Requisiti fondamentali	28
3.2.1	Requisiti di sistema	28
3.2.2	Requisiti funzionali	28
3.2.3	Requisiti non-funzionali	29
4	Progettazione	31
4.1	I linguaggi di programmazione per la realizzazione di app	31
4.2	Strumenti per la programmazione Android	33
4.3	I componenti di un'app Android	34
4.3.1	Le Activity	34
4.3.2	I Service	35
4.3.3	I Content Provider	36
4.3.4	I Broadcast Receiver	37
4.3.5	Gli Intent	37
4.4	L'interfaccia grafica di un'app	37
4.4.1	Il Layout	38
4.4.2	Le View	40
5	Implementazione	41
5.1	Strumenti utilizzati	41
5.1.1	Android SDK	41
5.1.2	Android Studio	41
5.2	Librerie utilizzate	42
5.2.1	Libreria Gson	43
5.2.2	Libreria Retrofit	43
5.3	Activity e relativi Layout	44
5.3.1	Activity per la registrazione e layout	44
5.3.2	Activity per la prenotazione e layout	48
5.3.3	Activity per il menù e layout	51
5.4	Interfaccia Api.java	55
5.5	Altre classi	56
5.5.1	Classe RetrofitClient.java	56
5.5.2	Classe User.java	57
5.5.3	Classe PagerViewAdapter.java	58
6	Manuale Utente	59
6.1	Funzionalità previste per l'utente	59
6.2	Funzionalità previste per l'amministratore	63
7	Discussione in merito al lavoro svolto	65
7.1	Lezioni apprese nell'utilizzo di Android Studio	65
7.2	Problemi riscontrati	67
8	Conclusioni e uno sguardo al futuro	69
	Ringraziamenti	71

Riferimenti bibliografici..... 73

Elenco delle figure

1.1	Primi anni di vita delle Applicazioni	8
1.2	App native, ibride e web app	10
1.3	App Ibride	11
1.4	Principali sistemi operativi per dispositivi mobili	12
1.5	Interfaccia Google Play	13
1.6	Interfaccia App Store	14
2.1	Andy Rubin	17
2.2	Architettura di Android	21
2.3	Versioni di Android	24
3.1	Requisiti non funzionali	29
4.1	Il logo di Java	31
4.2	Il logo di Xamarin	32
4.3	Il logo di Ionic	32
4.4	Il logo di Apache Cordova	33
4.5	Il logo di Android Studio	33
4.6	Il ciclo di vita di un'Activity	35
4.7	Lo stack delle Activity	35
4.8	Ciclo di vita delle due tipologie di Service	36
4.9	Esempio di Intent per il passaggio da un'Activity ad un'altra	37
4.10	Esempio di Linear, Relative e Table Layout	39
4.11	Esempio di FrameLayout	39
4.12	Esempio di ConstraintLayout	39
4.13	Diversi tipi di View	40
5.1	Un esempio di schermata di Android Studio	42
6.1	Schermata di login (a sinistra) e schermata di registrazione (a destra)	60
6.2	Schermata home (a sinistra) e schermata del menù laterale (a destra)	60
6.3	Schermata di prenotazione (a sinistra) e schermata di riepilogo della prenotazione (a destra)	61
6.4	Schermata del menù del ristorante	62

VIII Elenco delle figure

6.5	Schermata di modifica del profilo	62
6.6	Schermata di aggiunta di un piatto	63
7.1	Esempio spiegazione del metodo <code>setOnClickListener()</code>	66
7.2	Esempio di riferimento ad un errore (mancanza del punto e virgola) .	66
7.3	Esempio di emulatore	67

Elenco dei listati

5.1	Implementazione libreria Gson	43
5.2	Esempio di client REST	44
5.3	Implementazione libreria Retrofit2	44
5.4	Metodo onCreate() dell'activity registrazione	45
5.5	ID dell'elemento EditText per l'username	45
5.6	Struttura registraUtente()	45
5.7	Struttura openLogin()	47
5.8	Metodo onClick()	47
5.9	Layout dell'activity registrazione	47
5.10	onCreate() dell'activity per la prenotazione	48
5.11	Struttura handleDateButton()	49
5.12	Struttura handleHourButton()	49
5.13	Esempio di client REST	50
5.14	onCreate dell'activity menù	51
5.15	pagerViewAdapter	51
5.16	Metodo onPageSelected()	52
5.17	Struttura onChangeTab()	52
5.18	Layout della tab bar	54
5.19	Esempio di layout dei Fragment	55
5.20	Layout dell'activity menù	55
5.21	Alcune richieste nell'interfaccia Api.java	55
5.22	Definizione dell'URL, dell'oggetto mInstance e di un oggetto Retrofit	56
5.23	RetrofitClient()	56
5.24	Metodo getInstance()	57
5.25	Metodo getApi()	57
5.26	Esempio di Call<>	57
5.27	Modello User	57
5.28	Struttura getItem()	58
5.29	Metodo getCount()	58

Introduzione

La tecnologia dei computer mobili si distingue da quella dei computer portatili in quanto enfatizza la possibilità di usare il computer anche in movimento (per esempio in automobile, ma non alla guida). I computer mobili possono essere, dunque, dispositivi dedicati oppure general purpose, comunque di dimensioni e peso ridotti, tali da poter essere trasportati dall'utente. Storicamente i primi dispositivi di questo tipo sono stati i telefoni cellulari di prima generazione. Alcuni di questi dispositivi, quali smartphone e tablet, assumono importanza all'interno del contesto del cosiddetto web mobile, con fette di mercato in continua espansione.

Stiamo vivendo nell'era della tecnologia. Entriamo sempre di più nell'epoca superconnessa: tutti hanno uno smartphone e, bene o male, ovunque si può navigare su Internet, grazie a reti Wi-Fi, 3G, 4G, LTE e così via.

Stiamo assistendo ad una vera e propria trasformazione di esperienza di navigazione: ricerche con long tail (parole chiavi composte, ad esempio “acquisto casa Ancona?”) sempre più corte, tempo di permanenza sui siti in diminuzione e una grande fetta di mercato che ormai fa i propri acquisti solo attraverso lo smartphone. Il mobile ci permette di risparmiare tempo, di essere sempre in contatto con persone anche se sono dall'altra parte del mondo, ci semplifica moltissime attività e ci tiene sempre aggiornati. Ormai usiamo il nostro smartphone non solo per telefonare ma per fare tutto. Lo usiamo a lavoro, per scattare foto, per ascoltare musica, leggere libri, vedere film, fare jogging, e molto altro.

I primi veri e propri smartphone arrivarono intorno al 2003; essi integravano un semplice browser wap e consentivano l'installazione e l'esecuzione di applicazioni sviluppate in Java, pur con pesanti limitazioni. La vera rivoluzione nel mondo del mobile c'è stata nel 2007, anno in cui la Apple ha lanciato sul mercato il primo iPhone. Il primo smartphone targato Apple ha segnato una svolta nel mondo mobile, non tanto per la tecnologia innovativa (molti altri smartphone in commercio facevano le stesse cose), ma per una serie di accorgimenti hardware e software che tale dispositivo introduceva. La novità più importante è stata, sicuramente, la presenza di uno schermo multitouch, che introduceva un modo tutto nuovo di interfacciarsi con lo smartphone, e l'introduzione dell'App Store, un market digitale contentente migliaia di app di vario tipo.

Un punto di forza del mondo del mobile sono, per l'appunto, le applicazioni. Esse ci permettono di fare tutto ciò che abbiamo citato finora, semplificandoci davvero la

vita. Come Steve Jobs è riuscito ad unire un telefono cellulare con un iPod, quello che noi cerchiamo nelle app è di avere a portata di mano quante più cose possibili con un semplice tocco. È per questo, infatti, che le aziende di oggi, sia piccole che grandi, investono sulle applicazioni. Investire in un'applicazione creata su misura è una delle attività migliori che oggi può fare un'azienda moderna e al passo coi tempi. Molteplici sono le cose che un'app può fare; ad esempio può interagire con i clienti tramite il sistema di notifiche push oppure mediante i social network.

Un'app può essere semplice ed intuitiva. Può essere collegata al sito aziendale ed essere costantemente aggiornata con esso. Può essere efficace e veloce, può inviare offerte speciali o promozioni direttamente al cliente. Un'app può ridurre i costi fissi di un'azienda perché, ad esempio, permette di risparmiare su carta e stampa.

La principale motivazione che ci ha spinto nell'intraprendere questo progetto di tesi è stata, proprio, la volontà di affacciarsi a questo mondo. La volontà di imparare le basi della progettazione di un'app, nel nostro caso nativa, e di vedere più da vicino il sistema operativo Android, che non avevamo mai avuto modo di conoscere in precedenza ha rappresentato sicuramente uno stimolo importante per intraprendere questo percorso.

Dal momento che, nell'attuale fase storica, la cucina e la tecnologia sono due temi molto in voga, abbiamo pensato di creare quest'app per unire queste due cose, cercando di semplificare e velocizzare la permanenza al ristorante da parte di un cliente. L'obiettivo è sempre quello di migliorare l'esperienza del cliente. È questo il motivo per cui i ristoranti, nel corso del tempo, hanno adottato nuove tecnologie per velocizzare i tempi, per rendere i camerieri più efficienti e per migliorare il flusso del lavoro. Dai vecchi blocchetti per le comande si è passati ai palmari, e da quelli ci si sta via via spostando verso più innovativi sistemi di self ordering, talvolta con vere e proprie app per ordinazioni al ristorante. Una vasta gamma di persone utilizza le app del proprio ristorante preferito per ordinare a domicilio, effettuare una prenotazione di un tavolo e ottenere uno sconto in quanto cliente fidelizzato.

Le prenotazioni sono un problema per tutte le persone coinvolte: per i clienti, che devono sperare che qualcuno risponda alla chiamata, per i proprietari di ristoranti, che devono accertarsi di avere qualcuno a disposizione per rispondere alle chiamate. Un'app mobile può offrire ai clienti l'opportunità di prenotare un tavolo senza nemmeno entrare nel ristorante e, addirittura, senza neppure telefonare.

Il menù è sempre un gran problema sia per il ristorante, che deve effettuare una spesa sostanziosa per crearlo, sia per il cliente, il quale, fino a quando non mette piede nel ristorante, non è a conoscenza, a meno di qualche foto e di recensioni trovate su Internet, né del contenuto né dei prezzi dei vari piatti. Perciò, si è pensato di introdurre una visualizzazione, direttamente sull'app, del menù, sia per fare rendere conto al cliente del proprio contenuto, sia per farsi già un'idea su cosa ordinare.

Se il ristorante è piuttosto affollato, l'ultima cosa che si vuol fare è aspettare molto tempo per pagare o, da parte del proprietario, ingombrare la cassa con i clienti che attendono di pagare. Effettuare il pagamento attraverso l'app o alla cassa, ma già conoscendo la spesa e con i soldi già pronti in mano, ha diversi vantaggi: innanzitutto si può inserire un sistema che permette di far scegliere al cliente la modalità di pagamento (contanti, bancomat, carta di credito, etc.), il proprietario non perde tempo ogni volta a fare il conto e, se paga tramite app, si elimina la necessità di verificare se un ordine in fase di raccolta è già stato pagato.

Nella presente tesi, dopo una parte riguardante le applicazioni e Android, esporremo una panoramica della nostra app, soffermandoci sull'analisi dei requisiti e, in particolare, sulla parte di progettazione e implementazione di essa.

Più nello specifico, la tesi è strutturata come di seguito specificato:

- Nel primo capitolo verrà esposta inizialmente una panoramica sulle applicazioni, partendo dalla loro storia fino ai giorni d'oggi; successivamente parleremo dei vari tipi di applicazioni che possono essere sviluppate e dei vari sistemi operativi che le supportano.
- Nel secondo capitolo, invece, dopo un excursus sulla storia di Android, verrà proposta una descrizione generale soffermandoci, poi, in modo più approfondito, sulla sua architettura. Infine, verrà proposta una sintetica cronologia delle versioni con i relativi aggiornamenti.
- Nel terzo capitolo verrà fatta un'analisi dei requisiti, descrivendo, inizialmente, a grandi linee, l'app per poi passare ai requisiti fondamentali.
- Nel quarto capitolo sarà presentata nel dettaglio la progettazione, esponendo i linguaggi di programmazione più utilizzati per la realizzazione di applicazioni, i vari strumenti per la programmazione Android e le componenti fondamentali per la creazione di un'applicazione Android.
- Nel quinto capitolo verrà descritta l'implementazione presentando i vari strumenti e le varie librerie utilizzate, per poi fare un'accurata descrizione di alcune activity con i relativi layout, classi e interfacce da noi creati.
- Nel sesto capitolo verranno descritti un manuale utente con i vari screenshot delle schermate e una guida all'utilizzo dell'applicazione.
- Nel settimo capitolo verrà fatta una discussione in merito al lavoro svolto.
- Nell'ottavo capitolo, infine, verranno tratte le conclusioni e verranno delineati alcuni possibili sviluppi futuri.

Introduzione alle app

In questo capitolo presenteremo il concetto di applicazione, conosciuta da tutti con il termine app. Per fare ciò parleremo dapprima della loro nascita ed evoluzione negli anni fino ad oggi; successivamente illustreremo i vari tipi di app, per poi concludere con i principali sistemi operativi che le “ospitano” ed i relativi distributori.

1.1 Un po' di storia

1.1.1 Nascita

Le App nascono più di 10 anni fa. Già il primo telefono smart, realizzato dalla IBM, implementava al proprio interno 10 semplici app, come la rubrica, il calendario, la calcolatrice e l'orologio. Nokia fu, invece, la prima ad inserire un gioco per cellulari, ossia Snake, che abbiamo ritrovato con una veste completamente rinnovata sul nuovo Nokia 3310 2017. Sembra strano, ma è vero; tutto queste cose che abbiamo appena citato vengono considerate applicazioni a partire da un “semplicissimo” calendario fino ad arrivare ad una sveglia o una rubrica. Come già detto, quindi, le app esistono da molto più tempo di quello che immaginiamo. Ma il vero trampolino di lancio per esse lo ha dato nel 2008 Apple, con la Versione 2.0 del suo sistema operativo, che presentava l'inedito App Store.

Google nel medesimo anno commercializzò il primo smartphone, l'HTC Dream, dotato di sistema operativo Android, e del relativo negozio online per app chiamato, inizialmente, Android Market. La possibilità di scaricare in maniera semplice ed immediata tanti programmi diversi sul proprio dispositivo fece impennare il numero dei download effettuati, che, in un solo anno, nell'App Store, superarono la quota di 1 miliardo. Per Android ci vollero due anni per raggiungere lo stesso traguardo, un gap che, col passare degli anni, si assottigliò sempre di più fino al raggiungimento dei 50 miliardi di download nel 2013 per entrambi i principali store. Una rappresentazione grafica dell'evoluzione finora descritta viene mostrata nella Figura 1.1.

1.1.2 Sviluppo

Nel corso degli anni le persone hanno sempre più fatto affidamento alle app come mezzo per comunicare, per organizzarsi e per fare molto altro. Il primo ottobre 2001

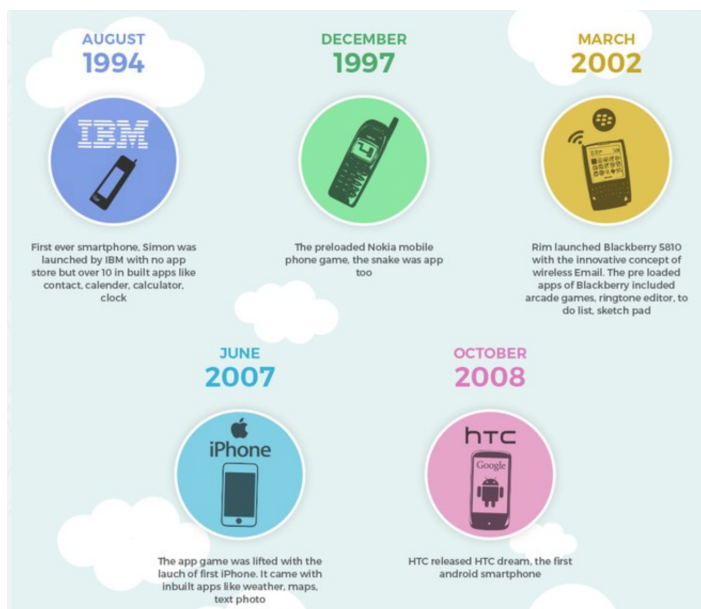


Figura 1.1. Primi anni di vita delle Applicazioni

è la data in cui nacque la prima generazione di iPod, un piccolo oggetto in cui si potevano scaricare migliaia di canzoni e dove si trovavano app come il solitario e bricks. Una delle grandi svolte si ebbe quando i produttori telefonici iniziarono ad inserire la tecnologia wireless nei cellulari. La Apple lanciò, di conseguenza, iTunes Music Store; come si può notare dal nome, si tratta di un negozio virtuale di musica (iTunes stesso è un'app). iTunes conteneva più di 200.000 brani venduti al prezzo di €0.99 ciascuno; già nella prima settimana si erano riusciti a vendere 1 milione di canzoni. Dopo il lancio di App Store già c'erano a disposizione 552 applicazioni delle quali 135 erano gratuite.

Nel 2009 abbiamo un altro grande avvenimento, ovvero la nascita dell'applicazione Whatsapp. Chi di noi oramai non la usa? Al tempo avere a disposizione un'applicazione che permetteva di messaggiare utilizzando semplicemente la connessione Internet e senza dover spendere soldi venne considerato qualcosa di geniale. Nei successivi anni vengono lanciati altri App Store, Windows Store e Amazon App Store, che divennero grandi rivali di Android Market e Apple Store. Le app di giochi vengono sempre più richieste. Tra i giochi mobile che hanno fatto la storia non possiamo non citare Temple Run, con i suoi 100 milioni di download nel 2011, Draw Something, che venne installato ben 1 milione di volte in appena 9 giorni, oppure Angry Birds, che in 4 anni ha raggiunto da solo i 2 miliardi di download.

1.1.3 Situazione attuale

Attualmente esistono moltissimi tipi di applicazioni a partire dalla più sciocca e inutile fino all'app più di successo del momento. Ma le app più utilizzate, e certamente presenti in tutti gli smartphone moderni, sono quelle per la messaggistica

istantanea, prime fra tutte Whatsapp e Messenger, nonché le app per l'accesso ai social network Instagram, Snapchat, Facebook, Twitter, e tanti altri. Le app sono arrivate ad occupare una grande parte della nostra vita. Ci permettono di fare qualsiasi cosa in qualsiasi momento e ad una velocità elevata. Ci troviamo nel bel mezzo di uno sciopero dei mezzi, al centro di una grande città e non sappiamo cosa fare? Basta accendere la connessione dati, scaricare un'app come "Uber" e farci portare dove vogliamo senza nemmeno doverci spostare a piedi da dove siamo. A volte non sappiamo come arrivare in un qualche luogo, ed ecco che entra in gioco Google Maps, o qualsiasi altra App di mappe, con i suoi vari percorsi, sia a piedi che con i mezzi pubblici o in macchina, e anche con consigli su altri metodi da adottare in caso di necessità, come ad esempio utilizzare Uber.

Insomma, al giorno d'oggi, le app sono veramente moltissime e, come si diceva all'inizio, di qualsiasi tipo e genere. Certo a molti può anche non piacere che molte app sostituiscono oggetti che per anni hanno fatto parte del nostro ufficio o della nostra vita, come il calendario sopra la scrivania, l'orologio appeso al muro o una cartina da viaggio. Tuttavia nessuno di noi può negare l'utilità e la comodità che abbiamo nell'avere a portata di mano migliaia e migliaia di "piccoli oggetti" che non occupano troppo spazio, la maggior parte dei quali sono gratuiti e fanno ciò che noi facevamo una volta in modo molto migliore, efficiente e veloce.

1.2 Tipi di app

Sviluppare un'app è davvero affascinante ma anche complicato. Per entrare un pò più nello specifico è bene conoscere la terminologia e le basi. Iniziamo col dire che esistono diversi tipi di app (Figura 1.2) ed ognuno ha le sue particolarità. Più specificatamente le app si dividono in:

- app native;
- app ibride;
- web app.

Il tipo di app che scegliamo di sviluppare è dettato da una serie di esigenze, come vedremo nel seguito.

1.2.1 App native

Le app native sono quelle applicazioni sviluppate utilizzando uno specifico linguaggio di programmazione per ogni tipologia di sistema operativo (Objective-C/Swift per iOS, Java per Android, C per Windows Phone, etc.). Un'app nativa consiste quindi in uno strumento informatico che si installa e si utilizza interamente sul proprio dispositivo mobile, vale a dire un programma progettato con lo scopo di rendere possibile un servizio, o una serie di servizi o strumenti ritenuti utili o desiderabili dall'utente; il programma è stato creato appositamente per uno specifico sistema operativo. L'interazione diretta con le API messe a disposizione dal costruttore del sistema operativo garantirà accesso immediato a tutte le funzionalità del dispositivo, oltre a permettere prestazioni ottimali e a migliorare sensibilmente l'usabilità.



Figura 1.2. App native, ibride e web app

Vediamo più nello specifico quali sono i principali vantaggi nel progettare app native:

- Maggiore velocità, affidabilità e migliore reattività, oltre che una risoluzione superiore che assicura un'esperienza migliore all'utente.
- Risoluzione superiore per una migliore esperienza utente.
- Accesso facilitato alle funzionalità del telefono.
- Invio di notifiche push per comunicazioni con gli utilizzatori.
- Utilizzo on e off-line, permettere agli utenti di accedere all'app senza connessione; si tratta di un grosso punto di forza da non sottovalutare.

A differenza degli altri tipi di app, le app native godono di prestazioni ottime ma ciò comporta costi elevati. Il budget da mettere in conto per lo sviluppo di app native è, in media, 4/5 volte superiore rispetto a quello per le applicazioni ibride.

1.2.2 App ibride

Simili nella progettazione a una web app (scritte, cioè, con linguaggi web HTML5, CSS3 e JavaScript), le app ibride sono per definizione una soluzione cross-platform (Figura 1.3). Il codice scritto, cioè, è uno solo, ma viene adattato facilmente per i diversi sistemi operativi attraverso un opportuno middleware.

Queste app vengono dette, anche, multipiattaforma e sono più rapide da realizzare e meno costose rispetto alle app native. Allo stesso tempo offrono più possibilità rispetto ad una web app. I più semplici esempi di app ibride consistono nell'uso di un componente nativo in grado di visualizzare pagine web (ad esempio, nel caso di Android, una WebView), e di una web application che viene automaticamente visualizzata tramite tale componente nativo. La web application può essere sia embedded (ovvero inclusa nell'app, in forma di sito statico e in locale), oppure remota (cioè installata su un web server, ed accessibile solo mediante una connessione web). Quest'ultimo caso permette, inoltre, di visualizzare la stessa web application di volta in volta su piattaforme diverse, dal momento che un'app "contenitore" (sia essa su Android o su iOS) può accedervi facilmente via HTTP.



Figura 1.3. App Ibride

L'approccio ibrido ha il vantaggio di limitare al minimo la necessità di manutenzione dei componenti nativi (dal momento che tutta la logica è implementata nella web application), mentre il grosso della manutenzione è (almeno idealmente) unico per entrambe le piattaforme.

1.2.3 Web app

Una web application è, in poche parole, un'applicazione che gira su un server web e che viene utilizzata attraverso un browser web, a differenza di una tradizionale applicazione desktop. Quindi, mentre un'app nativa o ibrida è installata fisicamente e interamente sul dispositivo dell'utente, una web app è sostanzialmente un collegamento verso un applicativo remoto, scritto in un linguaggio cross-platform come HTML5, con il codice dell'interfaccia utente che può risiedere sul dispositivo mobile oppure essere anch'esso in remoto.

Questa soluzione comporta delle importanti conseguenze in termini di funzionamento. Il vantaggio principale di una web app consiste nel fatto di non incidere in alcun modo, o in maniera minima, sulle capacità di memoria e di calcolo del dispositivo, in quanto il nucleo elaborativo e/o l'interfaccia utente dell'applicazione si trova su server remoti. Tuttavia, per funzionare, una web app richiede il costante accesso a Internet e le sue prestazioni dipenderanno in modo sensibile dalla velocità di connessione.

1.3 Principali sistemi operativi supportati e relativi distributori

Un sistema operativo (abbreviato in SO), in informatica, è un software di sistema che gestisce le risorse hardware e software della macchina, fornendo servizi di base ai software applicativi. Tra i sistemi operativi per i dispositivi mobili, quali smartphone e tablet, vi sono Android, iOS e Windows Phone (Figura 1.4).



Figura 1.4. Principali sistemi operativi per dispositivi mobili

1.3.1 Android - Google Play Store

Nel 2017 Android è diventato il sistema operativo più diffuso al mondo, superando anche Windows che, da sempre, deteneva questo ambito scettro. Complice l'enorme diffusione degli smartphone e la presenza del robottino verde in numerose categorie di prodotti, Android è indiscutibilmente il sistema operativo più utilizzato e più importante.

Dal punto di vista tecnico Android è costituito, nella parte più bassa, da un kernel Linux, che funziona da abstraction layer tra l'hardware e il software, e dai driver per la gestione delle varie componenti hardware. Al di sopra troviamo un livello che comprende un insieme di librerie native, scritte in C e C++, che rappresentano il cuore vero e proprio di Android. Nel febbraio del 2009 Google rilascia il primo aggiornamento, Android 1.1, con alcuni miglioramenti alla fluidità, alla sicurezza e altri cambiamenti.

È solo dalla versione 1.5 che Google decide di adottare un nome in codice per ogni release di Android, partendo dalla lettera C e abbinandola al nome di un dolce. Ad oggi Android è arrivato alla tredicesima versione, partendo da Cupcake fino a Pie.

Il relativo distributore di app che troviamo nei dispositivi Android è Google Play, detto anche, "Google Play Store" o, più semplicemente, "Play Store" (precedentemente Android Market). Si tratta di un servizio di distribuzione digitale gestito e sviluppato da Google.

Google Play si è originato da tre prodotti distinti: Android Market, Google Music e Google eBookstore.

Android Market venne annunciato da Google il 28 agosto 2008, e venne reso disponibile agli utenti il 22 Ottobre. A Dicembre 2010, il filtro dei contenuti venne aggiunto ad Android Market, la pagina dei dettagli di ciascuna app iniziò a mostrare un grafico promozionale nella parte superiore, e la dimensione massima di un'app venne aumentata da 25 megabyte a 50 megabyte.

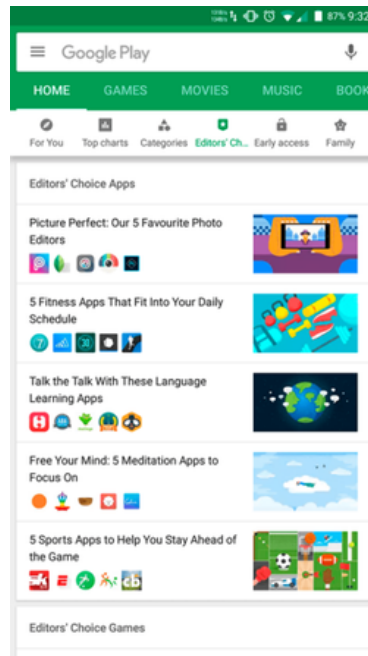


Figura 1.5. Interfaccia Google Play

Google eBookstore è stato lanciato il 6 Dicembre 2010, debuttando con tre milioni di ebook, rendendolo la più grande raccolta di ebook al mondo.

Nel Novembre 2011 Google ha annunciato Google Music, una sezione del Play Store che offre acquisti di musica. A Marzo 2012, Google ha aumentato la dimensione massima consentita di un'app permettendo agli sviluppatori di allegare due file di espansione al download di base di un'app; ogni file di espansione può avere una dimensione massima di 2 gigabyte, dando agli sviluppatori di app un totale di 4 gigabyte. Sempre a Marzo 2012, l'Android Market è stato "rebrandizzato" come Google Play.

A Maggio 2016 è stato annunciato che il Google Play Store, incluse tutte le app Android, sarebbe arrivato su Chrome OS a Settembre 2016.

Play Store è l'app store preinstallato ufficiale di Google su dispositivi con certificazione Android. Esso fornisce l'accesso ai contenuti sul Google Play Store, incluse app, libri, riviste, musica, film e programmi televisivi.

Play Store filtra l'elenco di app fornendo a quelle compatibili con il dispositivo dell'utente. Gli sviluppatori possono scegliere come target componenti hardware specifici (come la bussola), componenti software (come i widget) e versioni di Android (come, ad esempio, 7.0 Nougat).

Non è necessario che le applicazioni Android vengano acquisite utilizzando il Play Store. Gli utenti possono scaricare le applicazioni Android anche dal sito web di uno sviluppatore o tramite app store di terze parti.

1.3.2 iOS - Apple Store

Il sistema operativo iOS è stato sviluppato da Apple per iPhone, iPod touch e iPad. È un sistema operativo che viene presentato il 9 Gennaio 2007 al Macworld Conference & Expo di San Francisco e debutta nel mercato con il primo iPhone il 29 Giugno dello stesso anno. L'ultimo rilascio del sistema operativo è avvenuto il 13 Settembre 2016 con iOS 10.

Una delle versioni più importanti è la iOS 7 che ha portato un grande cambiamento nella grafica che viene rinnovata in chiave minimale; essa presenta icone molto più semplici e colorate, meno reminiscenti di elementi del mondo reale. Altro importante punto di rottura con le precedenti versioni di iOS è la rimozione della barra di sblocco presente fin dalla prima release di iOS, che viene sostituita con una schermata di sblocco più semplice e minimalista. Altro rinnovamento sostanziale è la totale revisione del multitasking, di cui vengono modificati il look e le funzionalità, rendendole più attuali e al pari dei sistemi concorrenti. Lo strumento che utilizza iOS per la diffusione di app è chiamato App Store la cui interfaccia viene mostrata in Figura 1.6.

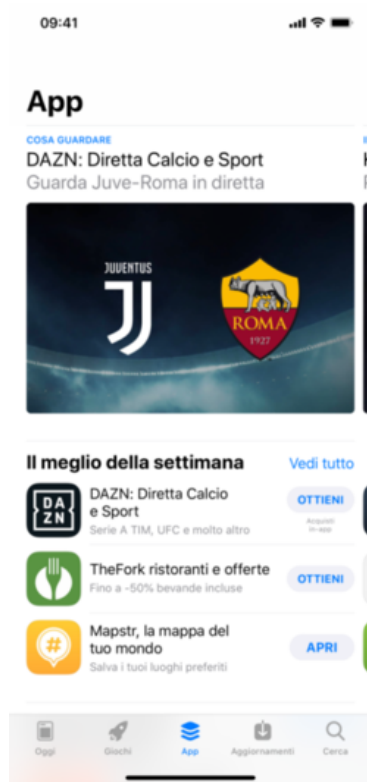


Figura 1.6. Interfaccia App Store

L'App Store è uno strumento realizzato da Apple disponibile per iPhone, iPod touch e iPad che permette agli utenti di scaricare e acquistare applicazioni disponi-

bili in iTunes Store. Le applicazioni possono essere sia gratuite che a pagamento, e possono essere scaricate direttamente dal dispositivo o su un computer.

L'App Store è stato aperto il 10 luglio 2008 tramite un aggiornamento software di iTunes. Al 13 Giugno 2016 sono disponibili in App Store più di due milioni di applicazioni sviluppate da terze parti, con oltre 130 miliardi di download.

Per prevenire gli abusi, le applicazioni devono essere preventivamente approvate da Apple. Il processo di approvazione prevede che due operatori provino in modo indipendente l'applicazione al fine di fornire un giudizio obiettivo e imparziale. Le applicazioni possono essere rifiutate per una serie di motivazioni di carattere tecnico/commerciale.

1.3.3 Windows - Windows Store

Windows Phone è una famiglia di sistemi operativi per smartphone realizzati da Microsoft, presentata per la prima volta al Mobile World Congress il 15 Febbraio 2010. La prima versione, Windows Phone 7, è stata lanciata sul mercato il 21 Ottobre del 2010 (l'8 Novembre negli Stati Uniti), seguita, il 29 Ottobre 2012 dalla seconda versione, Windows Phone 8. Con l'uscita di Windows 10, Microsoft ha cambiato il nome del sistema operativo per smartphone e tablet in Windows 10 Mobile.

Uno dei problemi principali per gli utenti di Windows Phone è sempre stata la presenza di un esiguo numero di applicazioni nello store rispetto ai principali concorrenti, dovuta sia al tardo arrivo del sistema sul mercato sia al mancato supporto applicativo da parte di alcune grandi aziende del web (ad esempio, Facebook) con un contestuale proliferarsi di applicazioni di terze parti, anche qualitativamente scadenti.

Dopo una prima timida ripresa nel corso del 2013 e del 2014, il divario del Marketplace è aumentato in maniera esponenziale decretando il fallimento di Windows Phone su scala mondiale.

Il Windows Phone Store (Windows Phone Marketplace su Windows Phone 7.x) viene usato per distribuire e vendere musica, video ed applicazioni per Windows Phone, è accessibile dall'hub Store (Marketplace su Windows Phone 7) sul telefono oppure dal sito internet www.windowsphone.com/store.

Lo store è amministrato da Microsoft, che deve prima approvare i software così da prevenire ogni tipo di malware. A Luglio 2014 Windows Phone Store contava oltre 350.000 applicazioni e il traguardo delle 500.000 è stato raggiunto il 16 Novembre dello stesso anno.

Per pubblicare un'applicazione sul Windows Phone Store, essa deve essere inviata a Microsoft per l'approvazione. Microsoft stessa ha delineato i contenuti che non saranno approvati, tra questi i software a sfondo sessuale; essi includono contenuti che mostrano nudità (capezzoli, genitali, natiche e peli pubici), prostituzione e feticismi sessuali.

Android

In questo capitolo parleremo di Android esponendo brevemente la sua storia e la cronologia delle versioni, concentrandoci successivamente su una descrizione generale del software.

2.1 Un po' di storia

Nell'Ottobre 2003 Andrew Rubin (meglio conosciuto come Andy Rubin, Figura 2.1), Rich Miner, Nick Sears e Chris White fondarono una società, la Android Inc. per lo sviluppo di quello che Rubin definì «... dispositivi cellulari più consapevoli della posizione e delle preferenze del loro proprietario».



Figura 2.1. Andy Rubin

Inizialmente la società operò in segreto, rivelando solo di progettare software per dispositivi mobili. Durante lo stesso anno il budget iniziale si esaurì, motivo per

cui fu fondamentale un finanziamento di 10.000 dollari da parte di Steve Perlman (amico intimo di Rubin) per poter continuare lo sviluppo. Steve Perlman consegnò a Rubin il denaro in una busta ma rifiutò ogni proposta di partecipazione al progetto.

Nel 17 Agosto 2005 Google acquistò l'azienda, in vista del fatto che la società di Mountain View desiderava entrare nel mercato della telefonia mobile. In questi anni il team di Rubin iniziò a sviluppare un sistema operativo basato sul kernel Linux. Il "robotto verde" venne ufficialmente presentato il 5 novembre 2007 dalla neonata OHA (Open Handset Alliance), un consorzio di aziende del settore Hi Tech che include Google, produttori di smartphone come HTC e Samsung, operatori di telefonia mobile come Sprint Nextel e T-Mobile, e produttori di microprocessori come Qualcomm e Texas Instruments Incorporated. Il 22 Ottobre 2008 venne lanciato sul mercato il primo dispositivo mobile che conteneva Android, l'HTC Dream.

Dal 2008 gli aggiornamenti di Android per migliorarne le prestazioni e per eliminare eventuali problemi di sicurezza delle precedenti versioni sono stati molti.

Nel Marzo 2013 Larry Page annuncia che Andy Rubin ha lasciato la presidenza di Android per dedicarsi ad altri progetti di Google. Viene rimpiazzato da Sundar Pichai.

2.2 Descrizione

2.2.1 Open Source

Android adotta una politica di licenza di tipo open source (escluse alcune versioni intermedie) e si basa sul kernel Linux. La licenza (Licenza Apache) sotto cui è distribuito consente di modificare e distribuire liberamente il codice sorgente. Inoltre, Android dispone di una vasta comunità di sviluppatori che realizzano applicazioni con l'obiettivo di aumentare le funzionalità dei dispositivi. Tali applicazioni sono scritte soprattutto in linguaggio di programmazione Java.

Nell'Ottobre 2012 le applicazioni disponibili presenti sul market ufficiale Android (Google Play) hanno raggiunto le 700 000 unità. Questi fattori hanno permesso ad Android di diventare il sistema operativo più utilizzato in ambito mobile, oltre a rappresentare, per le aziende produttrici, la migliore scelta in termini di costo, personalizzazione e leggerezza del sistema operativo stesso, senza dover scrivere un proprio sistema operativo da zero.

2.2.2 Interfaccia utente

L'interfaccia utente di Android è basata sul concetto di direct manipulation, per cui si utilizzano gli ingressi mono e multi-touch come strisciate, tocchi e pizzichi sullo schermo per manipolare gli oggetti visibili sullo stesso. La risposta all'input dell'utente è stata progettata per essere immediata e tentare di fornire un'interfaccia fluida.

Sensori hardware interni come accelerometri, giroscopi e sensori di prossimità sono utilizzati da alcune applicazioni per rispondere alle azioni da parte dell'utente, ad esempio la regolazione dello schermo da verticale a orizzontale a seconda di come il dispositivo è orientato o che consentono all'utente di guidare un veicolo in una corsa virtuale ruotando il dispositivo, simulando il controllo di un volante.

2.2.3 Ambiente di ripristino e modalità provvisoria

Su Android è semplice poter accedere all'ambiente di ripristino (recovery mode), come accade per qualsiasi distribuzione Linux o per Windows. L'ambiente è come sempre archiviato in una partizione di sistema nascosta. L'accesso avviene accendendo o riavviando il dispositivo tenendo premuti contemporaneamente, per qualche secondo, anche altri tasti (ad esempio tasti "home", "volume su" e "power", in altri casi "home" e "volume giù").

Il menù mette a disposizione diversi comandi tra i quali:

- ripristino alle condizioni di fabbrica (hard reset);
- eliminazione dei dati presenti nella memoria cache;
- lancio di eseguibili da percorso esterno (ad esempio, da un PC connesso a un dispositivo) o memoria flash;
- attivazione della modalità di download (reboot to bootloader), metodo alternativo all'ADB per eseguire operazioni sul caricatore di avvio, ad esempio installare un firmware di tipo custom.

L'ambiente di ripristino non va confuso con la modalità provvisoria, o modalità sicura (safe mode), analoga a quella di Windows, anch'essa lanciabile (in accensione) attraverso una combinazione di tasti (ad esempio, tasto di accensione, prima schermata di avvio/apparizione logo, tasto volume giù tenendolo premuto). Con la modalità provvisoria sono eseguiti solo i servizi essenziali e, quindi, si possono tentare prove per risolvere malfunzionamenti o altre procedure diagnostiche.

2.2.4 Aggiornamenti

Android ha un rapido ciclo di aggiornamento, con la distribuzione di nuove versioni ogni sei-nove mesi. Gli aggiornamenti sono in genere di natura incrementale, apportando miglioramenti del software a intervalli regolari, piuttosto che revisioni complete del sistema ogni due o tre anni (pratica comune per i sistemi operativi desktop come Windows).

Tra una major release e l'altra vengono messi a disposizione aggiornamenti intermedi per risolvere problemi di sicurezza e altri bug del software. La maggior parte dei dispositivi Android sono in grado di ricevere gli aggiornamenti in modalità "OTA" (over the air), ovvero senza necessità di un collegamento a un PC.

Nel 2011, Google ha siglato un accordo con un certo numero di produttori annunciando l' "Android Update Alliance" e impegnandosi a fornire aggiornamenti tempestivi a ogni dispositivo per 18 mesi dalla sua immissione in commercio.

2.2.5 Sicurezza

Uno dei più grandi problemi di sicurezza è la mancanza della pubblicazione di aggiornamenti da parte dei produttori. I produttori dei dispositivi (OEM) devono implementarli per ogni singolo dispositivo (spesso questo succede soltanto per i primi 12 mesi dopo il rilascio). In Italia si teme che oltre il 50% degli smartphone in uso non ricevano più aggiornamenti di sicurezza.

Diversi marchi commerciali hanno prodotto software antivirus per dispositivi Android. Al fine di migliorare la sicurezza del sistema, Google ha introdotto dei meccanismi automatici di analisi del software per bloccare eventuali applicazioni malevole presenti nel market Google Play. Queste soluzioni hanno ridotto il problema ma non risolto. Un'analisi di McAfee stima che nel 2012 l'85% dei virus per dispositivi mobili sia stato sviluppato per dispositivi Android.

2.2.6 Firmware

I telefoni che utilizzano Android come sistema operativo possono ottenere (grazie al lavoro di alcune comunità, come quella di XDA) i permessi di root, essendo Android basato su kernel Linux. Questo “sblocco” permette ad essi di accedere a funzioni avanzate, come gestire direttamente CPU e app di sistema, altrimenti inaccessibili, ma anche all'utente di cambiare il firmware del telefono. Senza i permessi aggiuntivi è, comunque, solitamente possibile installare eventuali aggiornamenti firmware ufficiali del produttore del dispositivo, senza far decadere la garanzia dello stesso.

A oggi la completezza dei firmware preinstallati dai produttori non spinge gli utenti a sostituire il firmware con altri creati dalle comunità online, ma rimane comunque molto popolare la ROM LineageOS.

2.3 Architettura Android

Android ha un'architettura di tipo gerarchico, strutturata a layer a complessità crescente dal basso verso l'alto, come mostrato nella Figura 2.2. I layer comprendono un sistema operativo, un insieme di librerie native per le funzionalità core della piattaforma, una implementazione della Virtual Machine e un insieme di librerie Java.

2.3.1 Il kernel Linux

Il livello più basso è rappresentato dal kernel Linux nella Versione 2.6 e 3.x (da Android 4.0 in poi) che costituisce il livello di astrazione di tutto l'hardware sottostante. Si noti la presenza di driver per la gestione delle periferiche multimediali, del display, delle connessione Wi-Fi e Bluetooth, dell'alimentazione, del GPS, della fotocamera. I produttori di telefoni possono quindi, intervenire già a questo livello per personalizzare i driver di comunicazione con i propri dispositivi. Grazie all'astrazione dell'hardware, infatti, i livelli soprastanti non si accorgono dei cambiamenti hardware, permettendo una programmazione ad alto livello omogenea ed una user experience indipendente dal dispositivo.

La scelta verso l'utilizzo di un kernel Linux si spiega attraverso la necessità di avere un'alta affidabilità. L'affidabilità è la più importante delle prestazioni in un dispositivo mobile che deve principalmente garantire il servizio di telefonia; gli utenti si aspettano quindi tale affidabilità, ma, allo stesso tempo, hanno bisogno di un dispositivo che possa garantire servizi più evoluti. Linux permette di raggiungere entrambi gli scopi.

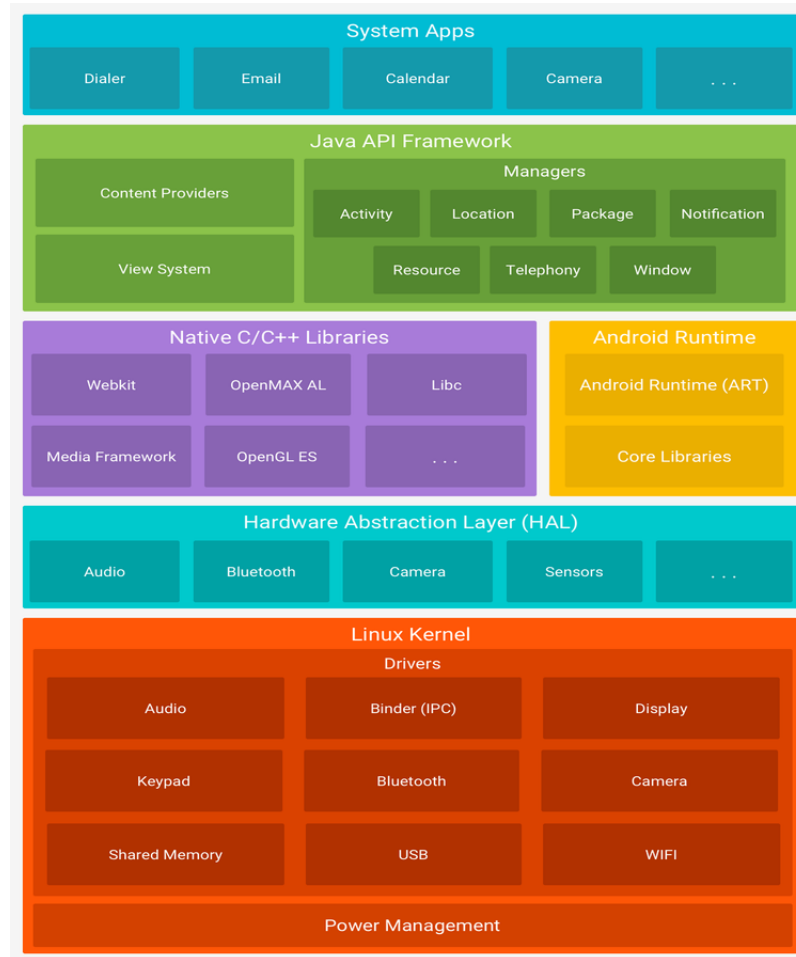


Figura 2.2. Architettura di Android

Android utilizza la ART (Android RunTime) che ha rimpiazzato la vecchia e obsoleta Dalvik virtual machine. Questa nuova virtual machine, sviluppata da Google, utilizza un compilatore Ahead-of-time. La piattaforma hardware principale di Android è l'architettura ARM. L'architettura x86 è supportata grazie al progetto Android x86, mentre Google TV utilizza una speciale versione x86 di Android.

2.3.2 L'Hardware Abstraction Layer

Appena sopra il kernel Linux è presente l'Hardware Abstraction Layer (HAL). Questo offre un'interfaccia standard che espone le funzionalità hardware del dispositivo ai livelli superiori, in particolare al framework Java. Esso è composto da una serie di moduli, ognuno dei quali implementa un'interfaccia per uno specifico componente hardware, come possono essere i sensori, la fotocamera o il modulo Bluetooth.

Quando un'API produce una chiamata per accedere ad un componente hardware, il sistema carica il modulo di quel componente.

2.3.3 Il runtime e le librerie native

Salendo nella gerarchia troviamo un insieme di librerie native realizzate in C e C++ e l'Android Runtime. L'ambiente di runtime è costituito dalle librerie core Java e dalla macchina virtuale ART: insieme costituiscono la piattaforma di sviluppo per Android.

Le librerie, invece, fanno riferimento a un insieme di progetti open source e sono descritte di seguito:

- *Surface Manager (SM)*, componente fondamentale in quanto ha la responsabilità di gestire le View, ovvero i componenti dell'interfaccia grafica. Il suo compito è quello di coordinare e impostare i diversi layer delle finestre, utilizzando il double buffering. Il Surface Manager permette l'accesso alle funzionalità del display e la visualizzazione contemporanea di grafica 2D e 3D delle diverse applicazioni.
- *OpenGL ES (for Embedded Systems)*, una versione ridotta di OpenGL per sistemi embedded. Essa comprende un insieme di API multipiattaforma che forniscono l'accesso a funzionalità 2D e 3D nei dispositivi mobili.
- *SGL (Scalable Graphics Library)*, libreria in C++ che costituisce il motore grafico di Android insieme alle OpenGL. Se per la grafica 3D ci si appoggia all'OpenGL, per quella 2D viene utilizzato, invece, un motore ottimizzato chiamato SGL. È una libreria utilizzata principalmente dal Window Manager e dal Surface Manager all'interno del processo di renderizzazione grafica.
- *Media Framework*, API in grado di gestire i diversi codec per i vari formati di acquisizione e riproduzione audio e video. Si basa su una libreria open source OpenCore di PacketVideo, uno dei membri fondatori dell'OHA. I codec gestiti dal Media Framework permettono la gestione dei formati più importanti, tra cui MPEG4, H.264, MP3, AAC, AMR, oltre quelli per la gestione delle immagini, come JPG e PNG.
- *FreeType*, libreria di piccole dimensioni, molto efficiente e altamente personalizzabile per il rendering dei font. Una caratteristica importante è quella di fornire un insieme di API semplici per ciascun tipo di font in modo indipendente dal formato del corrispondente file.
- *SSL (Secure Socket Layer)*, libreria per la sicurezza della comunicazione su Internet.
- *SQLite*, libreria scritta in C che implementa un DBMS20.
- *WebKit*, framework per la navigazione web utilizzato già da diversi browser come Safari e Chrome. Si tratta di un browser, engine open-source basato sulle tecnologie HTML, CSS, JavaScript e DOM. Un aspetto da sottolineare è che WebKit si comporta come un browser engine, quindi può essere integrato in diversi tipi di applicazioni. Esso è caratterizzato dal fatto di essere compatto, diretto, di non necessitare alcuna configurazione e, soprattutto, di essere transazionale. Permette di creare una base di dati incorporata in un unico file, ed è diretto, in quanto non utilizza un processo standalone, ma può essere incorporato all'interno dell'applicazione che lo usa.

- *LibC*, implementazione della libreria standard C ottimizzata per dispositivi basati su Linux embedded.

2.3.4 Il framework di API Java

Il livello per noi più importante, e quello su cui faremo maggiore affidamento, è il framework delle API Java, anche detto Application Framework (AF). Tale livello contiene delle API scritte in Java che espongono al programmatore tutte le funzioni che il sistema operativo offre. Grazie ad esse abbiamo tutto il necessario per sviluppare le nostre app ed accedere a tutto ciò che offrono i livelli inferiori dello stack, senza preoccuparci della loro implementazione. Qui troviamo gli strumenti per costruire le interfacce delle nostre app, accedere alle risorse non di codice, come elementi grafici, layout e stringhe localizzate, utilizzare le notifiche, gestire il ciclo di vita e la navigazione delle app e accedere ai dati di altre app, nonché condividere dati con altre app.

Queste API sono le medesime sfruttate dalle app di sistema; quindi non ci sarà alcuna differenza tra queste ultime e le nostre app. Nello specifico qui possiamo trovare:

- *Activity Manager*, la cui responsabilità è l'organizzazione delle varie schermate di un'applicazione in uno stack a seconda dell'ordine di visualizzazione delle stesse sullo schermo dei diversi dispositivi. Più genericamente l'Activity Manager è lo strumento fondamentale attraverso il quale l'utente interagisce con l'applicazione.
- *Package Manager*, il quale gestisce il ciclo di vita delle applicazioni nei dispositivi, come il processo d'installazione, aspetti grafici dell'applicazione, diverse activity o aspetti di sicurezza.
- *Window Manager*, il quale permette di gestire le finestre delle varie applicazioni, gestite da processi diversi, sullo schermo del dispositivo.
- *Telephony Manager*, il quale consente una maggiore interazione con le funzionalità caratteristiche di un telefono, come una banale chiamata o la verifica dello stato della stessa telefonata.
- *Content Provider*, il quale ha la responsabilità di gestire la condivisione di informazioni tra i vari processi.
- *Resource Manager*, il quale è un componente che ha il compito di ottimizzare le risorse mettendo a disposizione una serie di API di semplice utilizzo.
- *View System*, il quale si occupa della gestione della renderizzazione dei componenti e degli eventi associati. L'interfaccia grafica di un'applicazione per Android è composta da specializzazioni della classe View, ciascuna caratterizzata da una particolare forma e da un diverso modo di interagire con essa attraverso un'accurata gestione degli eventi associati.
- *Location Manager*, il quale mette a disposizione delle API che vengono utilizzate da applicazioni relative la localizzazione e la gestione delle mappe.
- *Notification Manager*, rende disponibili un insieme di strumenti che l'applicazione può utilizzare per inviare una particolare notifica al dispositivo, che la dovrà presentare all'utente con i meccanismi che conosce.

2.3.5 Le app di sistema

Nel livello più elevato troviamo le app di sistema. Queste forniscono un insieme di app fondamentali, quali browser, client email, messaggistica tramite SMS, dialer, etc.

Come già detto, le app di sistema non hanno accesso ad API dedicate; di conseguenza anche le nostre app saranno su questo livello e potranno essere usate al posto di quelle fornite, se compiono la stessa funzione. Inoltre, le app di sistema possono fornire alcune funzionalità chiave che possono essere sfruttate anche dalle nostre app, permettendoci, così, di riutilizzare ciò che è già presente.

2.4 Cronologia versioni

I nomi delle versioni di Android sono abbastanza semplici da ricordare, in quanto procedono in ordine alfabetico e seguono la numerazione data. L'elemento in comune dei nomi delle versioni Android è l'essere tutti il nome di un dolce (Figura 2.3). Questo, però, a partire da Android 1.5.



Figura 2.3. Versioni di Android

Cos'è successo prima di quella numerazione e perché iniziare dalla lettera C? In rete ci sono molte versioni riguardanti i nomi delle prime versioni di Android. Ecco tutti i nomi dei sistemi operativi Android in ordine cronologico con i principali aggiornamenti:

- *Android 1.5 - Cupcake (2009)*: introduzione della classica tastiera sullo schermo, Android Market.
- *Android 1.6 - Donut (2009)*: casella di ricerca, possibilità di installare il sistema operativo in qualsiasi smartphone, e non solo in quelli con risoluzione 320 x 480 pixel.
- *Android 2.0 / 2.1 - Eclair (2009)*: comparsa delle mappe interattive di Google Maps, personalizzazione della schermata della home page, introduzione dei sistemi vocali.

- *Android 2.2 - Froyo (2010)*: problemi legati alla sicurezza; diventa possibile eseguire veloci ricerche sul web con la voce, scrivere appunti, impostare l'allarme della sveglia, segnare eventi sul calendario, e molto altro ancora; compare infine l'hotspot portatile.
- *Android 2.3 - Gingerbread (2010)*: grazie a una nuova grafica gli sviluppatori possono progettare giochi efficienti e di alta qualità; permette di visualizzare il consumo della batteria per ogni app e integra il supporto a NFC.
- *Android 3.0 - Honeycomb (2011)*: si adatta facilmente a qualsiasi dispositivo, specialmente a quelli di grandi dimensioni, progettati per leggere, guardare video ad alta risoluzione e lavorare. I controlli diventano completamente touch screen.
- *Android 4.0 - Ice Cream Sandwich (2011)*: si possono vedere le applicazioni aperte di recente e si ha la possibilità di sincronizzare i segnalibri del browser.
- *Android 4.1 / 4.2 / 4.3 - Jelly Bean (2012)*: compaiono le notifiche interattive, l'assistente vocale Google Now, il supporto all'OpenGL ES 3.0 e al Bluetooth 4.0, il supporto multiutente, la scrittura a gesti e le impostazioni rapide.
- *Android 4.4 - KitKat (2013)*: compaiono l'hotword "Ok Google" per poter richiamare l'assistente vocale e l'Open Storage Framework per migliorare la gestione file.
- *Android 5.0 / 5.1 - Lollipop (2014)*: funziona egregiamente su ogni terminale, smartphone, Android TV, tablet, e anche sui orologi Google Wear, ottimizzando automaticamente ogni pagina in base alle caratteristiche dello schermo.
- *Android 6.0 - Marshmallow (2015)*: introduce Doze, che mira a migliorare ulteriormente la gestione energetica.
- *Android 7.0 / 7.1 - Nougat (2016)*: vengono migliorate l'implementazione della realtà virtuale, rendendo l'utilizzo di tale tecnologia più pragmatica; viene integrata la compatibilità attiva a schermi sensibili alla pressione; vengono aumentate le azioni disponibili per le notifiche; viene inserito il tasto hamburger (tre linee orizzontali impilate) alle impostazioni, che permette, dopo aver modificato un sottomenù, di ritornare al menù principale con meno tocchi rispetto al tasto indietro; viene introdotto il multi-window, che permette di utilizzare due o quattro applicazioni alla volta a seconda del dispositivo, smartphone o tablet; viene migliorata la funzionalità Doze; vengono migliorati il risparmio dati per le applicazioni in background, nonché Android for Work, che permette di gestire il telefono durante l'orario lavorativo.
- *Android 8.0 / 8.1 - Oreo (2017)*: vengono introdotti il Picture-in-Picture (PiP, ovvero immagine nell'immagine), una diversa gestione dei font, dei canali di notifica, permettendo una migliore e dettagliata gestione delle stesse, Autofill, che permette alle app che lo supportano l'immissione automatica di campi di testo quali le password, codec LDAC per migliorare l'audio in Bluetooth, Wi-Fi Aware, conosciuto anche come Neighbor Awareness Networking (NAN), e vari miglioramenti di minore rilevanza.
- *Android 9.0 - Pie (2018)*: vengono introdotti molteplici modifiche al design, come un nuovo indicatore fluttuante del volume, un centro notifiche ridisegnato e nuove icone nel menu impostazioni. È anche stato aggiunto il supporto nativo alle API Multi-Camera e al notch, la tacca presente sul lato superiore dello schermo di alcuni smartphone che contiene vari sensori.

Prima della Versione 1.5, però, sono state ufficialmente presentate altre due versioni di Android, la 1.0 (mai montata su un telefono commerciale) e la 1.1 (gli utenti HTC Dream TIM sono gli unici “fortunati” ad essere ancora fermi a questa versione). Queste due versioni non sono caratterizzate da nessun nome di un dolce. Perché? Wikipedia (in versione inglese) riporta che Android 1.0 si chiamasse Astro e la versione 1.1 Petit Four. La versione italiana dell’enciclopedia riporta, invece, Apple Pie per la versione 1.0 e Bender per la versione 1.1. Tempo fa la stessa Wikipedia riportava Banana Bread come la versione 1.1 di Android. Se i nomi Apple Pie e Banana Bread sono totalmente frutto dell’immaginazione di qualche editor “burlone”, gli altri nomi hanno tutti un fondamento di verità.

Sono gli ingegneri Google Jean-Baptiste Queru, Dianne Hackborn e Romain Guy, che, tramite Google+, cercano di chiarire la situazione. Inizialmente i rilasci interni di Android erano segnati con delle milestone, seguiti da un numero progressivo. Così avevamo nomi come m3-rc20a e m5-rc15.

I problemi sono iniziati quando gli sviluppatori nelle varie email facevano riferimento solo al codice numerico (es rc47a), creando confusione, in quanto non si riusciva a capire a quale milestone si facesse riferimento. È così che si decise di ricorrere a una catalogazione progressiva alfabetica (in uso ancora ora), iniziando dalla lettera C, in quanto terza release ufficiale di Android. Il primo firmware a seguire questa numerazione fu il CRC29.

3

Analisi dei requisiti

In questo capitolo proponiamo una descrizione e un'analisi dei requisiti della nostra app.

3.1 Descrizione dell'app

Quest'app da noi progettata è pensata per la gestione di un ristorante. L'app può essere utilizzata sia dai clienti che dal ristorante, attraverso un amministratore.

Il cliente, una volta registrato, può:

- prenotare un tavolo;
- visualizzare il menù direttamente sull'app;
- scegliere la modalità di pagamento.

Il cameriere può:

- visualizzare i tavoli a cui è assegnato;
- prendere gli ordini;
- ricevere notifiche di ordine pronto, relative al proprio tavolo.

Il cuoco può:

- visualizzare l'ordine;
- inviare una notifica al cameriere quando il piatto è pronto.

Infine, l'amministratore può:

- visualizzare e gestire le prenotazioni;
- modificare il menù;
- gestire i pagamenti;
- gestire le ordinazioni;
- registrare un utente come cuoco o cameriere;
- assegnare i tavoli ai camerieri.

La nostra app permette di ridurre i tempi di attesa, consentendo al cliente di visualizzare il menù direttamente sull'app, facendosi un'idea sui costi e sull'ordine da effettuare. Così facendo, il personale di sala non deve più fornire il menù ai clienti, ma bensì rimanere a disposizione di essi per chiarimenti ed effettuare l'ordine più velocemente. Inoltre, vi è più flessibilità per ogni cliente nel pagamento e nelle prenotazioni.

Il proprietario del locale non deve più pensare a registrare le prenotazioni o a fare il conto di ogni tavolo, perchè è già tutto integrato nell'app.

3.2 Requisiti fondamentali

In Ingegneria del Software, l'analisi dei requisiti è un'attività preliminare allo sviluppo di un sistema software, il cui scopo è quello di definire le funzionalità che il nuovo prodotto deve offrire, ovvero i requisiti che devono essere soddisfatti dal software sviluppato per venire incontro alle esigenze del committente.

L'analisi dei requisiti è una fase presente essenzialmente in tutti i modelli di ciclo di vita del software, pur con diverse enfasi e diverse connotazioni.

3.2.1 Requisiti di sistema

L'applicazione è di tipo nativo per Android e può essere utilizzata in smartphone contenenti il sistema operativo Android con versione Marshmallow (livello API 23) o superiore.

3.2.2 Requisiti funzionali

I requisiti funzionali si presentano come elenchi di funzionalità o servizi che il sistema deve fornire. Essi descrivono anche il comportamento del sistema a fronte di particolari input e come esso dovrebbe reagire in determinate situazioni. Che si tratti di utente o di sistema, un requisito funzionale potrà essere formulato a diversi livelli di dettaglio, dovendo preservare, naturalmente, la precisione della specifica.

Due importanti caratteristiche delle specifiche dei requisiti sono la completezza e la coerenza.

I requisiti funzionali che devono essere forniti dall'app sono:

- RF1: l'app deve fornire all'amministratore, o admin, il pieno controllo;
- RF2: l'app gestisce le attività CRUD (Create, Read, Update e Delete) del titolare/amministratore;
- RF3: l'app gestisce le attività CRUD dei clienti;
- RF4: l'app gestisce le attività CRUD dei dipendenti;
- RF5: l'app dovrà prevedere accesso all'area riservata del cliente e dell'amministratore;
- RF6: l'app dovrà consentire all'amministratore di creare, modificare o rimuovere l'account di un dipendente;
- RF7: l'app dovrà consentire all'amministratore di visualizzare le prenotazioni;

- RF8: l'app dovrà consentire all'amministratore di gestire gli ordini;
- RF9: l'app dovrà consentire all'amministratore di modificare il menù;
- RF10: l'app dovrà consentire all'amministratore di gestire i pagamenti;
- RF11: l'app dovrà consentire all'amministratore di assegnare i tavoli ai camerieri;
- RF12: l'app dovrà consentire al cliente di registrarsi in modo tale da avere un account;
- RF13: l'app dovrà consentire al cliente di visualizzare il menù;
- RF14: l'app dovrà consentire al cliente di prenotare un tavolo;
- RF15: l'app dovrà consentire al cliente di scegliere la modalità di pagamento;
- RF16: l'app dovrà consentire al cuoco di visualizzare l'ordine;
- RF17: l'app dovrà consentire al cuoco di inviare una notifica al cameriere;
- RF18: l'app dovrà consentire al cameriere di visualizzare a quali tavoli è stato assegnato e le notifiche da parte del cuoco.

3.2.3 Requisiti non-funzionali

I requisiti non funzionali rappresentano i vincoli e le proprietà/caratteristiche relative ad sistema, come vincoli di natura temporale, vincoli sul processo di sviluppo e sugli standard da adottare. Tali requisiti non riguardano solo il sistema software che si sta sviluppando; alcuni, infatti, possono vincolare il processo usato per sviluppare il sistema (Figura 3.1).

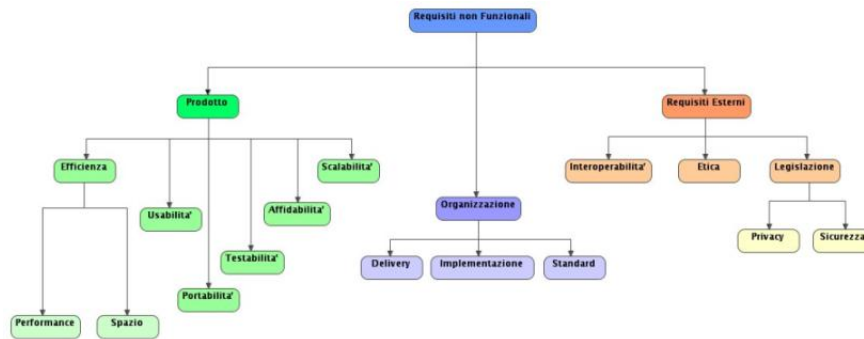


Figura 3.1. Requisiti non funzionali

I requisiti non funzionali relativi alla nostra app sono:

- L'app dovrà richiedere ai clienti un username di almeno 6 caratteri, che dovrà essere diverso da quello degli altri utenti, e una password di almeno 6 caratteri, oltre a vari dati personali.
- Viene interamente sfruttata tecnologia open source per implementare l'app.
- Il linguaggio di programmazione utilizzato è Java, con una piattaforma di backend in PHP, sviluppata su un Webserver Apache.
- La piattaforma utilizzata per la stesura del codice è Android Studio.
- Verranno utilizzate varie librerie e API per semplificare la stesura del codice.
- Il software che si utilizzerà per la gestione del database è MySQL.

- phpMyAdmin è l'applicazione web che ci permetterà di gestire l'amministrazione di MySQL.
- Il software utilizzato per la gestione delle API relative alle Query è Postman.
- Il software per la creazione dei mockup è Balsamiq Mockups 3.

Progettazione

In questo capitolo daremo uno sguardo ravvicinato alle applicazioni Android, parlando degli strumenti necessari per la programmazione fino ad arrivare alle vere e proprie componenti che fanno parte della struttura di un'app.

4.1 I linguaggi di programmazione per la realizzazione di app

Le app native sono realizzate utilizzando il linguaggio della piattaforma su cui andranno a girare. Nel caso di Android, il linguaggio adottato per sviluppare app di questo tipo è Java.

Java (Figura 4.1) non è JavaScript, per non confondere le due cose, anche se in questo caso entrambi i linguaggi sono utili per la realizzazione e lo sviluppo di applicazioni per i vari sistemi operativi mobile. Java è un linguaggio di programmazione orientato agli oggetti, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione.



Figura 4.1. Il logo di Java

Xamarin (Figura 4.2) è un framework per lo sviluppo di app native e cross-platform con C#. Il framework si basa sul progetto open source Mono offrendo pieno supporto non solo alle piattaforme Android e iOS ma anche a Windows Phone. Si tratta di un framework abbastanza avanzato, che consente agli sviluppatori

l'utilizzo trasparente delle API native nell'app prodotta. Tramite Xamarin.Forms, gli sviluppatori hanno, inoltre, a disposizione un vasto numero di componenti per la realizzazione dell'interfaccia, in grado di adattarsi facilmente ai sistemi operativi mobili supportati.

Xamarin offre la possibilità di eseguire il codice su un framework .NET, potendo simulare l'esecuzione dell'app sia su iOS che su Android. Risulta versatile soprattutto per grandi team, in cui molte persone partecipano allo stesso progetto. L'intera piattaforma può essere utilizzata tramite l'IDE Microsoft Visual Studio, le cui funzionalità sono ben note anche a chi non si occupa di sviluppo mobile.



Figura 4.2. Il logo di Xamarin

Ionic (Figura 4.3), considerato da molti “il Bootstrap del mobile”, è un framework che permette di creare app per Android e iOS basate su HTML5. Esso consente di sfruttare al meglio le tecnologie Web per creare applicazioni mobile con “look and feel” simile a quelle native e, in questo ambito, si rivela tra le soluzioni di maggior successo.



Figura 4.3. Il logo di Ionic

Apache Cordova (Figura 4.4) permette ai programmatori di creare applicazioni mobili usando CSS3, HTML5 e Javascript, invece di affidarsi ad API specifiche delle piattaforme Android, iOS o Windows Phone. Il framework incapsula, poi, il codice CSS, HTML e JavaScript generato all'interno delle predette piattaforme. Le applicazioni generate dal framework non possono né considerarsi puramente native (il rendering della struttura grafica è fatto con visualizzazioni web) né basate completamente sul web (il programma viene impacchettato come un'applicazione per la distribuzione e ha accesso alle API native dei dispositivi mobili).

Apache Cordova comprende un migliaio di plugin-in scritti in Java che utilizzano le API Android per essere integrate, attraverso il linguaggio Javascript, all'interno del codice HTML; di conseguenza, esso permette agli sviluppatori di accedere all'hardware dello smartphone in modo efficace. È possibile controllare la fotocamera,

il sensore GPS, il Bluetooth, il giroscopio e ogni altro dispositivo hardware dello smartphone.



Figura 4.4. Il logo di Apache Cordova

4.2 Strumenti per la programmazione Android

Per sviluppare un'applicazione Android, abbiamo bisogno di alcuni strumenti software, tra cui un JDK e un IDE. Il JDK (Java Development Kit) è il kit di sviluppo per la programmazione in Java; difatti, come detto nei paragrafi precedenti, è il linguaggio con il quale vengono realizzate le app native Android.

Per quanto riguarda l'IDE (ambiente di sviluppo integrato), quello più utilizzato è, senza dubbio, Android Studio (Figura 4.5). Questo IDE, scaricabile dal sito ufficiale di Android, è stato sviluppato e pensato da Google appositamente per la realizzazione di applicazioni Android e, negli anni, è diventato l'ambiente di sviluppo di riferimento, superando Eclipse.



Figura 4.5. Il logo di Android Studio

Con Android Studio, è possibile scaricare il pacchetto che costituisce il cuore dello sviluppo di app Android, l'Android SDK. L'SDK (Software Development Kit) è un pacchetto che include una serie di strumenti di sviluppo, tra cui programmi, librerie, emulatori, piattaforme per ogni versione di Android, che facilitano lo sviluppo di app.

4.3 I componenti di un'app Android

Indipendentemente dallo scopo di un'applicazione Android, essa basa le sue funzionalità su 5 componenti fondamentali: *Activity*, *Service*, *Content Provider*, *BroadcastReceiver* e *Intent*. Essi verranno esaminati in dettaglio nelle prossime sottosezioni.

4.3.1 Le Activity

Un'Activity è una finestra che contiene l'interfaccia utente di un'applicazione; il suo scopo è quello di permettere un'interazione con gli utenti. Un'app è generalmente composta da una o più Activity; il sistema operativo mantiene in primo piano (in foreground) una sola Activity alla volta, lasciando tutte le altre in secondo piano (background). Dal momento in cui un'Activity è in primo piano al momento in cui va in background, essa attraversa una serie di stati, che descrivono il ciclo di vita di un'Activity.

Per creare un'Activity è necessario definire una classe Java che estende la classe base `Activity`; essa definisce una serie di eventi che governano il ciclo di vita dell'Activity stessa (Figura 4.6). Tali eventi sono:

- **onCreate**: l'Activity viene creata. Il programmatore deve assegnare le configurazioni di base e definire quale sarà il layout dell'interfaccia.
- **onStart**: l'Activity diventa visibile. È il momento in cui si possono attivare funzionalità e servizi che devono offrire informazioni all'utente.
- **onResume**: l'Activity diventa la destinataria di tutti gli input dell'utente.
- **onPause**: l'inverso di **onResume**; notifica la cessata interazione dell'utente con l'Activity.
- **onStop**: opposto di **onStart**; segna la fine della visibilità dell'Activity.
- **onDestroy**: contrapposto a **onCreate**; segna la distruzione dell'Activity.
- **onRestart**: evento che si verifica nel momento in cui l'Activity viene arrestata e riavviata.

Le varie Activity sono “impilate” una sopra l'altra; con il tasto “back” possiamo passare alla schermata precedente, mettendo in background l'Activity attuale e ponendo in foreground quella precedente. Questa “sequenza” di Activity è detta Activity Stack (Figura 4.7) e viene gestita dall'Activity Manager, uno dei componenti del middleware della piattaforma.

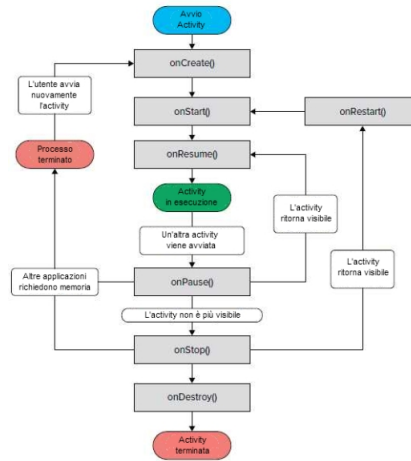


Figura 4.6. Il ciclo di vita di un'Activity

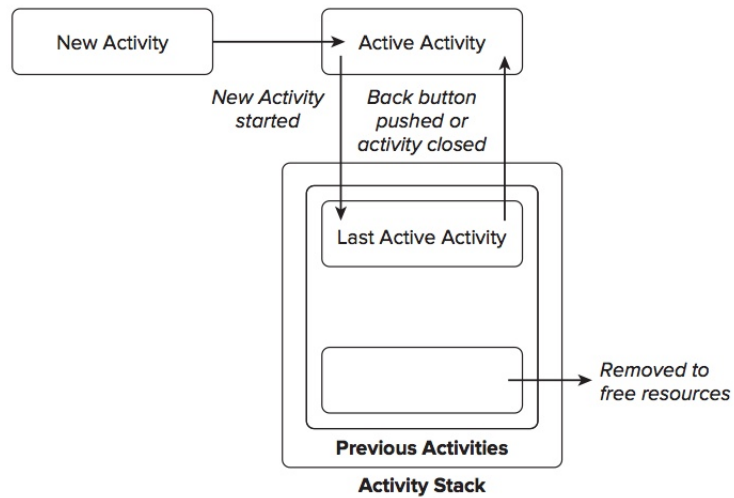


Figura 4.7. Lo stack delle Activity

4.3.2 I Service

Un componente di fondamentale importanza nell'architettura Android è il Service. Esso è distinto dall'Activity, in quanto non necessita di una UI, avendo il compito di restare in esecuzione in background a prescindere dall'interfaccia grafica. Un componente (ad esempio una Activity) potrà connettersi ad un particolare servizio con l'obiettivo di avviarlo o fermarlo. Un esempio di servizio può essere un lettore musicale.

I Service sono di due tipi:

- *Service Started o Locale*: sono Service che vengono eseguiti perennemente in background, anche se la componente che li ha avviati viene terminata. Questo

tipo di Service viene utilizzato per operazioni indipendenti da altre applicazioni, come aggiornamenti dati in background, download di un file, etc.

- *Service Bound*: un Service è “bound” quando un componente si lega ad esso attraverso il metodo `bindService()`. Un Service di questo tipo può essere visto come la parte server di un’interfaccia client-server che permette ai componenti di interagire con il Service, inviare richieste, ottenere risposte, il tutto in maniera trasversale ai diversi processi attraverso l’IPC (Inter-Process Communication). Un Service bound vive fin quando un componente è legato ad esso. Più componenti possono legarsi al Service, ma quando tutti non sono più legati allo stesso, il Service viene eliminato. A differenza del Service locale esso non è pensato per girare in background con un tempo indefinito.

La differenza tra le due tipologie si riflette anche sul ciclo di vita. Il diagramma in Figura 4.8 li mette a confronto.

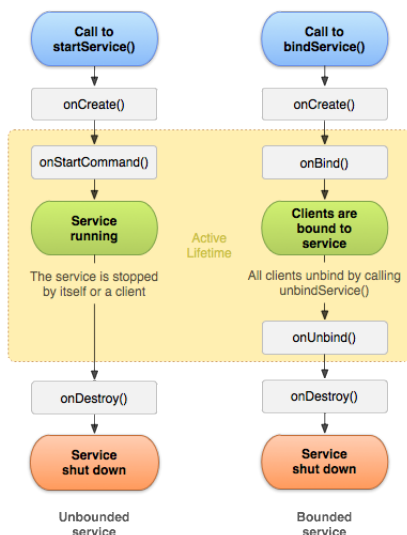


Figura 4.8. Ciclo di vita delle due tipologie di Service

4.3.3 I Content Provider

Ogni applicazione Android memorizza i dati che utilizza in file e database privati; di conseguenza, solo l’applicazione stessa può accedere a questi dati. Si ha come risultato che nessun’app può accedere a dati di altre applicazioni.

Un meccanismo di condivisione esiste ed è rappresentato dai ContentProvider, una delle quattro componenti delle app Android. Essi permettono di leggere e modificare dati presenti in una determinata sorgente, inviando chiamate ad un indirizzo univoco, detto URL. I comandi passati rispecchiano le quattro operazioni CRUD eseguibili sui database (Create, Read, Update, Delete).

I Content Provider vengono utilizzati in vari ambiti:

- Per condividere basi di dati di utilità generale presenti nel sistema operativo: si pensi ai Contatti, al dizionario utente, al Calendario, e molto altro.
- Per fare in modo che più applicazioni possano condividere gli stessi dati, cosa che capita spesso quando più app provengono dallo stesso produttore.
- Come elemento architetturale interno ad un'app articolata in cui più componenti accedono agli stessi dati.

4.3.4 I Broadcast Receiver

Per Broadcast Receiver si intende quel componente che si mette in “ascolto” e, quando riceve un determinato tipo di messaggio, effettua una determinata azione (ad esempio lancia un'attività, comunica con un servizio, etc.). La classe da estendere per creare un Broadcast Receiver è la classe `android.content.BroadcastReceiver`; creiamo un Broadcast Receiver quando vogliamo intercettare un determinato evento sul sistema, come quando si scatta una foto o quando parte la segnalazione di batteria scarica.

4.3.5 Gli Intent

Gli Intent sono strumenti che consentono a un componente di richiedere una funzionalità che viene implementata da un altro componente. Essi vengono utilizzati principalmente per gestire le Activity (Figura 4.9); difatti, un'app è generalmente composta da più Activity, e l'Intent consente il passaggio da una schermata all'altra.

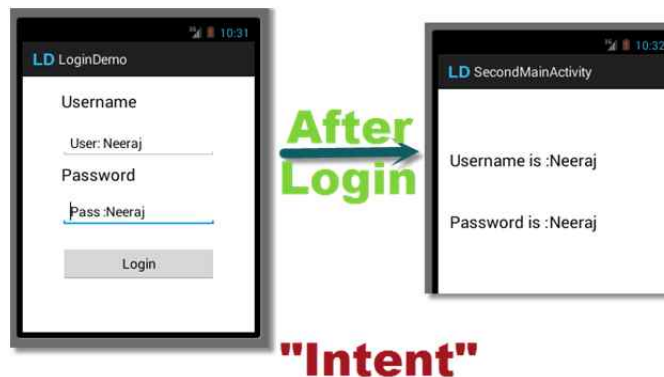


Figura 4.9. Esempio di Intent per il passaggio da un'Activity ad un'altra

4.4 L'interfaccia grafica di un'app

Una “User Interface”, o brevemente UI, indica la piattaforma attraverso la quale l'utente interagisce con la macchina. Si tratta, infatti, dell'interfaccia che permette di

utilizzare un computer, effettuare un ordine su un negozio online, oppure utilizzare un'app sullo smartphone. L'interfaccia utente comprende tutti gli strumenti di una superficie che l'utente vede e tramite i quali effettua azioni, dalle semplici righe di comando basate su testo alle strutture complicate delle interfacce grafiche. Al contempo, l'UI fa anche sì che la macchina invii un feedback all'utente di modo che quest'ultimo possa avere la conferma che la sua azione sia stata effettuata con successo.

Un'Activity ha bisogno di un volto, di un suo aspetto grafico. Sempre. Anche nei casi più semplici, come quando si limita a stampare la stringa "Hello World!". Scrivere interfacce tramite XML è più semplice e consente di definire layout su base dell'orientamento dello schermo (verticale o orizzontale), definire layout per schermi a risoluzioni diverse e definire layout per lingue differenti, lasciando che Android carichi automaticamente ciò che corrisponde alla lingua di sistema impostata dall'utente. Inoltre, scrivere layout in XML permette su molti IDE (tra cui Android Studio) di vedere un'anteprima del proprio lavoro.

Scrivere interfacce in codice Java ha, invece, il vantaggio di avere performance leggermente migliori, permette di gestire gli eventi associati all'interfaccia, consente di rendere l'interfaccia dinamica.

4.4.1 Il Layout

La struttura grafica di un'Activity è chiamata layout. Un layout viene progettato in XML, in una modalità che ricorda molto l'uso di HTML per le pagine web.

Nel framework Android sono stati definiti vari tipi di layout:

- *LinearLayout* (Figura 4.10): contiene un insieme di elementi che distribuisce in maniera sequenziale dall'alto verso il basso (se definito con orientamento verticale) o da sinistra a destra (se ha orientamento orizzontale, il valore di default). È un layout molto semplice e piuttosto naturale per i display di smartphone e tablet.
- *TableLayout* (Figura 4.10): altro layout piuttosto semplice, inquadra gli elementi in una tabella e, quindi, è particolarmente adatto a mostrare strutture regolari suddivise in righe e colonne, come form o griglie. È piuttosto semplice da usare e ricorda molto le tabelle HTML nelle pagine web, con i ben noti tag `<table>` `<tr>` `<td>`.
- *RelativeLayout* (Figura 4.10): sicuramente il più flessibile e moderno. Adatto a disporre in maniera meno strutturata gli elementi, ricorda un po' il modo di posizionare `<div>` flottanti nelle pagine web. Essendo "relative" gli elementi si posizionano in relazione l'uno all'altro o rispetto al loro contenitore, permettendo un layout fluido, che si adatta bene a display diversi. Rispetto agli altri due è ricco di attributi XML che servono ad allineare e posizionare gli elementi tra loro.
- *AbsoluteLayout*: è il layout più semplice da utilizzare, ma anche il più vecchio e deprecato in quanto non permette un facile adattamento ai diversi schermi dei dispositivi.
- *FrameLayout* (Figura 4.11): è il layout che permette la sovrapposizione degli oggetti contenuti all'interno dello schermo. È spesso utilizzato per creare dei semplici popup e messaggi di alert.

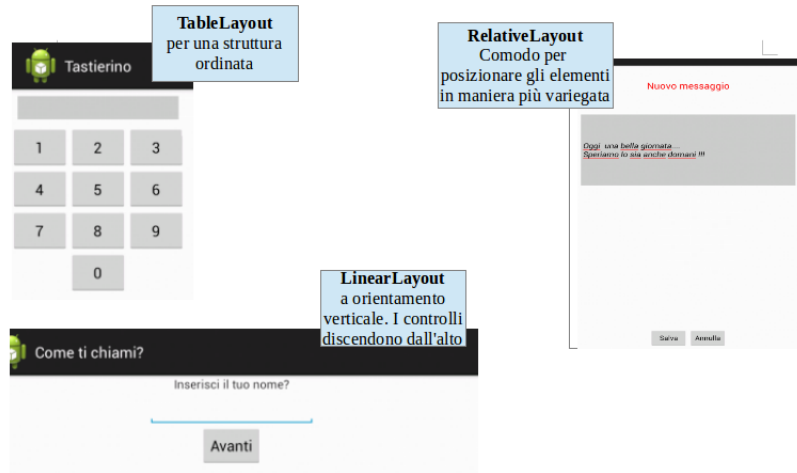


Figura 4.10. Esempio di Linear, Relative e Table Layout

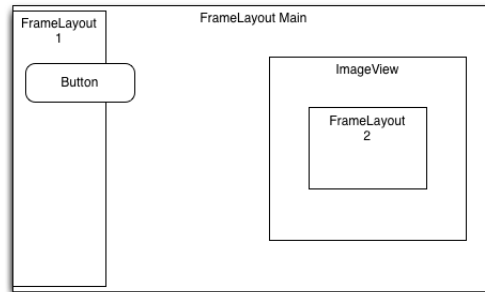


Figura 4.11. Esempio di FrameLayout

- *ConstraintLayout* (Figura 4.12): questo layout è molto simile al *RelativeLayout*, ma permette di definire la posizione degli elementi in maniera più flessibile e tramite l'interfaccia grafica.

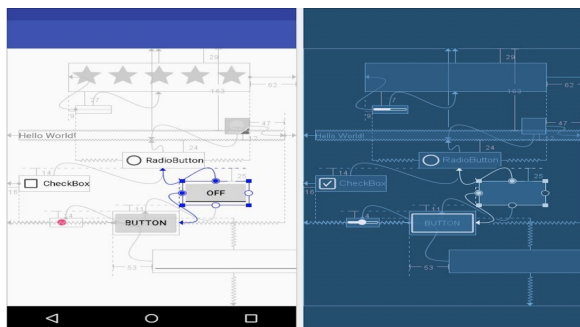


Figura 4.12. Esempio di ConstraintLayout

4.4.2 Le View

Dopo aver analizzato i vari tipi di layout che è possibile utilizzare per posizionare le viste all'interno di un'activity, è il momento di dare un'occhiata ai vari tipi di View che possiamo utilizzare per realizzare l'interfaccia utente delle nostre applicazioni.

Una View è un elemento grafico (o widget) di un'interfaccia utente che appare su uno schermo. L'SDK di Android prevede diverse View di default, derivate dalla classe di base. Analizziamo più in dettaglio alcune di esse:

- *TextView*: viene utilizzata per mostrare un testo all'utente;
- *EditText*: viene utilizzata per consentire all'utente di inserire testo;
- *Button*: rappresenta un pulsante;
- *ImageButton*: rappresenta un pulsante con un'immagine;
- *Spinner*: consente la scelta di un'opzione tra una lista a comparsa;
- *RadioButton*: rappresenta una casella che può essere selezionata o meno;
- *CheckBox*: rappresenta un quadratino con due possibili stati: selezionato (checked) o non selezionato (unchecked).

Nella Figura 4.13 possiamo vedere alcuni tipi di View.



Figura 4.13. Diversi tipi di View

Implementazione

Questo capitolo è particolarmente tecnico; in esso si affronterà l'approfondimento delle specifiche strutturali delle singole componenti, evidenziando parti di codice e dettagli sugli stessi.

5.1 Strumenti utilizzati

5.1.1 Android SDK

Il kit di sviluppo software Android (SDK) include una serie completa di strumenti di sviluppo. Tra questi sono presenti: un debugger, librerie, un emulatore portatile basato su QEMU, documentazione, codici esemplificativi e tutorial. Le piattaforme di sviluppo attualmente supportate sono i computer che eseguono Linux (qualsiasi moderna distribuzione Linux), macOS 10.5.8 o versioni successive, e Windows XP o versioni successive. A partire da marzo 2015, l'SDK non è disponibile su Android in sé, ma lo sviluppo del software rimane possibile utilizzando applicazioni Android specializzate.

Fino alla fine del 2014 l'ambiente di sviluppo integrato ufficialmente supportato (IDE) è stato Eclipse, utilizzando gli Android Development Tools (ADT) Plugin, anche se, ad esempio, gli ambienti IntelliJ IDEA e NetBeans supportavano pienamente lo sviluppo di Android. A partire dal 2015 Android Studio, realizzato da Google e alimentato da IntelliJ, è l'IDE ufficiale di Android.

5.1.2 Android Studio

Android Studio (Figura 5.1), il nuovo IDE (Integrated Development Environment) di Android, è l'ambiente di sviluppo integrato per tutti gli sviluppatori e per chi fa coding avanzato su Android. Android Studio è un nuovo ambiente di sviluppo Android basato su IntelliJ IDEA, che fornisce nuove funzionalità e miglioramenti rispetto ad Eclipse ADT. Sviluppare con questa nuova piattaforma è più semplice e funzionale.

Android Studio offre:

- un sistema di compilazione flessibile basato Gradle;
- la possibilità di costruire APK varianti e multipli;
- un esteso supporto ai servizi Google e a vari tipi di dispositivi;
- un ricco ambiente di layout con editing a tema;
- strumenti per la cattura di prestazioni, usabilità, compatibilità di versione, e altri problemi;
- funzionalità ProGuard e app-signing;
- il supporto per Google Cloud Platform e la possibilità di integrare Google Cloud Messaging e App Engine.

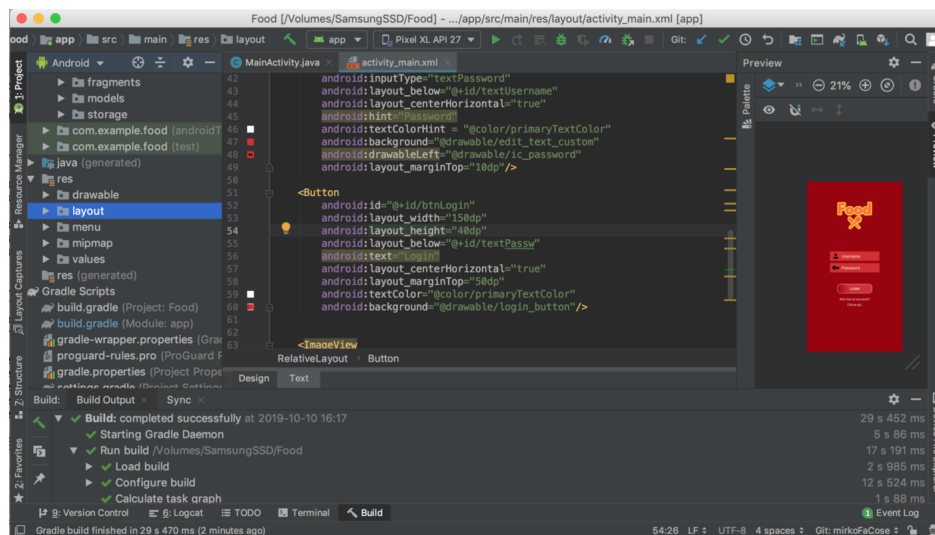


Figura 5.1. Un esempio di schermata di Android Studio

5.2 Librerie utilizzate

Una libreria, in Informatica, è un insieme di funzioni o strutture dati predefinite e predisposte per essere collegate ad un programma software attraverso un opportuno collegamento. Quest'ultimo può essere statico o dinamico; nel secondo caso si parla di dynamic-link library ("libreria a collegamento dinamico").

Lo scopo delle librerie software è quello di fornire una collezione di entità di base pronte per l'uso, evitando al programmatore di dover riscrivere ogni volta le stesse funzioni o strutture dati e facilitando, così, le operazioni di sviluppo e manutenzione.

Quasi tutti i linguaggi di programmazione supportano il concetto di libreria e moltissimi includono delle librerie standardizzate (spesso chiamate, proprio, librerie standard del linguaggio in questione); si tratta di un insieme di funzioni e/o strutture dati che permettono di risolvere i problemi di programmazione più comuni. Ad esempio, molti linguaggi di programmazione hanno una libreria matematica che

consente di eseguire elevamenti a potenza, il calcolo dei logaritmi, etc: essi hanno, anche, funzioni di I/O, funzioni e strutture dati per la gestione di collezioni di oggetti, etc. Java e Dotnet dispongono di una moltitudine di librerie (package) per le più svariate funzioni, importabili e attivabili nell'ambiente di sviluppo.

5.2.1 Libreria Gson

In Android Studio, per poter usufruire di una libreria esterna, bisogna interagire con il Gradle. Una delle librerie utilizzate per la realizzazione dell'app è “Gson”. Essa è stata incorporata nel Gradle tramite la seguente riga di codice riportata nel Listato 5.1.

```

1      dependencies {
2          ..
3          implementation 'com.squareup.retrofit2:converter-gson:2.6.0'
4          ..
5      }
```

Listato 5.1. Implementazione libreria Gson

Gson è una libreria sviluppata da Google che rende incredibilmente semplice e veloce la conversione tra gli oggetti Java e la loro rappresentazione JSON. Originariamente la libreria fu creata per essere utilizzata esclusivamente per l'uso interno nei progetti Google; in seguito è stata resa disponibile pubblicamente per gli sviluppatori. Essa può quindi, essere utilizzata nelle nostre applicazioni in alternativa ad altre librerie analoghe (come ad esempio *Jettison* o *Jabsorb*).

Le caratteristiche principali di Gson sono, in particolare, le seguenti:

- È semplice e veloce da utilizzare.
- È indipendente dai file sorgenti. Quasi tutte le librerie analoghe, invece, richiedono l'inserimento di alcune annotazioni nei bean che occorre serializzare. Questa procedura è improponibile se non si ha a disposizione il codice sorgente, personalizzabile.
- Consente di creare degli adapter personalizzati per definire la modalità di serializzazione degli oggetti.
- Fornisce un ampio supporto per Java Generics.

5.2.2 Libreria Retrofit

La libreria Retrofit nasce da una compagnia, Square Open Source, che da tempo offre strumenti per una programmazione efficace ed efficiente sulle principali piattaforme: Java, Android, iOS, Javascript e Ruby. Retrofit ha saputo conquistare i favori degli sviluppatori Java/Android grazie alla semplicità con cui permette di integrare delle API REST nelle proprie app. L'utente non dovrà più maneggiare esplicitamente thread e richieste HTTP; la libreria maschera il tutto dietro delle apposite annotation Java tramite le quali è possibile definire l'indirizzo della chiamata REST, il metodo HTTP da usare e parametri vari.

Nel Listato 5.2 viene riportato un client REST definito tramite una delle annotazioni di Retrofit.

```

1 public interface UserRestService {
2     @GET("users")
3     Call<List<User>> listUsers();
4 }

```

Listato 5.2. Esempio di client REST

Per poter utilizzare Retrofit nel suo funzionamento base è sufficiente includere nel Gradle la libreria tra le dipendenze del progetto (Listato 5.3).

```

1 dependencies {
2     ..
3     implementation 'com.squareup.retrofit2:retrofit:2.6.0'
4     ..
5 }

```

Listato 5.3. Implementazione libreria Retrofit2

Il programmatore non deve far altro che predisporre delle interfacce che definiscano i metodi per l'interazione. Retrofit ha da poco raggiunto la Versione 2.0, anche se ancora in beta. Grazie a questa nuova versione dovrebbero pervenire agli sviluppatori alcune integrazioni particolarmente desiderate, tra cui:

- Viene cambiata la modalità di effettuare richieste. Da ora, una chiamata sincrona ed una asincrona saranno “esteriormente” identiche: entrambe genereranno un oggetto che implementa l'interfaccia `Call<T>` sul quale si invocherà il metodo `execute`, per una chiamata sincrona, oppure il metodo `enqueue` per una chiamata asincrona. Quest'ultima dovrà ricevere come parametro un riferimento al listener che gestirà il risultato ottenuto sia in caso di successo che di fallimento.
- Quanto detto al punto precedente avrà come diretto vantaggio la possibilità di interrompere una richiesta in corso. Si tratta di una funzionalità da tempo desiderata dagli sviluppatori. Il metodo `cancel`, previsto sempre nell'interfaccia `Call<T>`, interromperà l'attività in rete relativa alla specifica richiesta.

5.3 Activity e relativi Layout

In questa sezione entreremo più nel dettaglio in merito all'attività di implementazione da noi condotta. Verranno descritte le varie Activity presenti con i relativi layout. Inoltre, verrà presentata una panoramica su altre classi e interfacce da noi implementate.

5.3.1 Activity per la registrazione e layout

La registrazione del cliente è stata implementata nell'activity `Registrazione.java` e il relativo layout viene mostrato in `activity_registrazione.xml`.

Il primo metodo è `onCreate()`, il quale è richiamato quando un'activity viene avviata per la prima volta. Il metodo accetta un parametro che può essere nullo, oppure può ritornare le informazioni salvate precedentemente con il metodo `onSaveInstanceState()`. Troviamo il suo codice nel Listato 5.4.

```

1      @Override
2      protected void onCreate(Bundle savedInstanceState) {
3          super.onCreate(savedInstanceState);
4          setContentView(R.layout.activity_registrazione);
5
6          editTextEmail = findViewById(R.id.editTextEmail);
7          editTextPassword = findViewById(R.id.editTextPassword);
8          editTextUsername = findViewById(R.id.editTextUsername);
9          editTextName = findViewById(R.id.editTextName);
10         editTextSurname = findViewById(R.id.editTextSurname);
11
12         findViewById(R.id.textViewLogin).setOnClickListener(this);
13
14
15         btnRegistrati = findViewById(R.id.btnRegistrati);
16
17         btnRegistrati.setOnClickListener(new View.OnClickListener() {
18             @Override
19             public void onClick(View v) {
20                 registraUtente();
21             }
22         });
23     }

```

Listato 5.4. Metodo `onCreate()` dell'activity registrazione

Il metodo `setContentView()` definisce il layout grafico da visualizzare sullo schermo del dispositivo. Esso riceve un intero che deve identificare la risorsa XML dove è definito il layout grafico. La costante `R.layout.activity_registrazione` identifica tale risorsa. Un altro metodo molto importante è `findViewById(int n)` che, in questo caso, ricerca l'elemento in base all'ID definito nella risorsa XML tramite la tag nel Listato 5.5.

```

1      <EditText
2          ....
3          android:id="@+id/editTextUsername"
4          ....
5      />

```

Listato 5.5. ID dell'elemento `EditText` per l'username

In questo esempio identifichiamo tramite l'ID il campo di testo per inserire lo username.

Nell'ultima parte del codice possiamo notare un altro metodo molto importante, ovvero `setOnClickListener()`, il quale è un "gestore di evento". Esso permette di rendere "intelligente" il pulsante `btnRegistrati`, e fare in modo che si accorga se qualcuno lo tocca. All'interno delle parentesi è stata inserita un'istruzione che verrà eseguita solo al verificarsi dell'evento monitorato. L'istruzione di cui abbiamo appena parlato è `registraUtente()`, la quale ci permetterà di effettuare la registrazione. Vediamo nel Listato 5.6 com'è strutturata.

```

1      private void registraUtente(){
2
3          String email = editTextEmail.getText().toString().trim();
4          String password = editTextPassword.getText().toString().trim();
5          String username = editTextUsername.getText().toString().trim();
6          String surname = editTextSurname.getText().toString().trim();
7          String name = editTextName.getText().toString().trim();
8
9          /*Verifica campi registrazione, e registrazione utente*/
10
11         if(username.isEmpty()){
12             editTextUsername.setError("Campo Username vuoto!");
13             editTextUsername.requestFocus();
14             return;
15         }
16
17         if(email.isEmpty()){
18             editTextEmail.setError("Campo Email vuoto!");
19             editTextEmail.requestFocus();
20             return;

```

```

21     }
22
23     if(!Patterns.EMAIL_ADDRESS.matcher(email).matches()){
24         editTextEmail.setError("Inserire una Email valida!");
25         editTextEmail.requestFocus();
26         return;
27     }
28
29     if(password.isEmpty()){
30         editTextPassword.setError("Campo Password vuoto!");
31         editTextPassword.requestFocus();
32         return;
33     }
34
35     if(password.length() < 6){
36         editTextPassword.setError("La password deve essere di almeno
37         6 caratteri!");
38         editTextPassword.requestFocus();
39         return;
40     }
41
42     if(name.isEmpty()) {
43         editTextName.setError("Campo Name vuoto!");
44         editTextName.requestFocus();
45         return;
46     }
47
48     if(surname.isEmpty()) {
49         editTextSurname.setError("Campo Surname vuoto!");
50         editTextSurname.requestFocus();
51         return;
52     }
53
54
55     Call<User> call = RetrofitClient
56         .getInstance()
57         .getApi()
58         .createUser(username, email,
59         password, name, surname);
60
61     call.enqueue(new Callback<User>() {
62         @Override
63         public void onResponse(Call<User> call, Response<User> response) {
64             if(!response.isSuccessful()){
65                 Log.v("reg", "response.code=" + response.code());
66             }else{
67                 Log.v("reg", "ok=" + response.code());
68                 openLogin();
69             }
70         }
71
72         @Override
73         public void onFailure(Call<User> call, Throwable t) {
74             Log.v("reg", "fail: " + t.getMessage());
75         }
76     });
77
78 }

```

Listato 5.6. Struttura `registraUtente()`

Nella prima parte del codice abbiamo inserito una serie di validazioni per i vari campi di testo, come il controllo del riempimento dei vari campi o del pattern dell'email.

In seguito possiamo notare la `Call<User>`, messa a disposizione da Retrofit, la quale, tramite una richiesta al database presente nell'interfaccia `Api.java` che abbiamo chiamato `createUser()`, e passando i vari parametri, ci permette di effettuare la registrazione (il modello `User` e la classe `RetrofitClient` verranno spiegate nella Sezione 5.5).

Nella `Call` possiamo notare due metodi, ovvero `onResponse()` e `onFailure`. Il primo si avvierà in caso di registrazione avvenuta con successo, aprendo, di conseguenza, l'activity relativa al Login con `openLogin()`, tramite `Intent` (Listato 5.7).

```

1 private void openLogin(){
2     Intent intent = new Intent(this, MainActivity.class);
3     startActivity(intent);
4 }

```

Listato 5.7. Struttura openLogin()

Una volta effettuata l'apertura dell'activity, visualizzerà un messaggio nel Logcat grazie alla classe `Log.v`. Il secondo, invece, si avvierà in caso di fallimento e riporterà nel Logcat il tipo di errore.

Infine, abbiamo aggiunto un ultimo metodo, ovvero `onClick()` (Listato 5.8).

```

1 @Override
2 public void onClick(View v) {
3     switch (v.getId()) {
4         case R.id.textViewLogin:
5             startActivity(new Intent(this, MainActivity.class));
6             break;
7     }
8 }

```

Listato 5.8. Metodo onClick()

Questo ci permette di aprire l'activity di Login, cliccando nella relativa `TextView`, se siamo già in possesso di un account.

Il layout di questa activity si trova nel file `activity_registrazione.xml`. Tale file è strutturato come specificato nel Listato 5.9.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".activities.Registrazione"
8     android:background="@color/primaryDarkColor">
9
10 <EditText
11     android:id="@+id/editTextUsername"
12     android:layout_above="@+id/btnRegistrati"
13     android:layout_width="200dp"
14     android:layout_height="30dp"
15     android:layout_centerHorizontal="true"
16     android:layout_marginBottom="230dp"
17     android:ems="10"
18     android:inputType="textPersonName"
19     android:hint="Username"
20     android:background="@drawable/edit_text_custom"
21     android:textColorHint="@color/primaryTextColor"/>
22
23 <EditText
24     android:id="@+id/editTextPassword"
25     android:layout_width="200dp"
26     android:layout_height="30dp"
27     android:ems="10"
28     android:inputType="textPassword"
29     android:layout_above="@+id/btnRegistrati"
30     android:layout_marginBottom="150dp"
31     android:layout_centerHorizontal="true"
32     android:hint="Password"
33     android:background="@drawable/edit_text_custom"
34     android:textColorHint="@color/primaryTextColor"/>
35
36
37 <EditText
38     android:id="@+id/editTextName"
39     android:layout_width="200dp"
40     android:layout_height="30dp"
41     android:ems="10"
42     android:layout_above="@+id/btnRegistrati"
43     android:layout_marginBottom="110dp"
44     android:layout_centerHorizontal="true"
45     android:hint="Nome"
46     android:background="@drawable/edit_text_custom"
47     android:textColorHint="@color/primaryTextColor"/>
48
49 <EditText
50     android:id="@+id/editTextSurname"

```

```

51         android:layout_width="200dp"
52         android:layout_height="30dp"
53         android:ems="10"
54         android:layout_above="@+id/btnRegistrati"
55         android:layout_marginBottom="70dp"
56         android:layout_centerHorizontal="true"
57         android:hint="Cognome"
58         android:background="@drawable/edit_text_custom"
59         android:textColorHint="@color/primaryTextColor"/>
60
61     <EditText
62         android:id="@+id/editTextEmail"
63         android:layout_width="200dp"
64         android:layout_height="30dp"
65         android:layout_above="@+id/btnRegistrati"
66         android:layout_marginBottom="190dp"
67         android:layout_centerHorizontal="true"
68         android:ems="10"
69         android:inputType="textEmailAddress"
70         android:hint="Email"
71         android:background="@drawable/edit_text_custom"
72         android:textColorHint="@color/primaryTextColor"/>
73
74     <Button
75         android:id="@+id/btnRegistrati"
76         android:layout_width="170dp"
77         android:layout_height="40dp"
78         android:layout_centerInParent="true"
79         android:text="Registrati"
80         android:textColor="@color/primaryTextColor"
81         android:background="@drawable/login_button"/>
82
83     <TextView
84         android:id="@+id/textViewLogin"
85         android:layout_width="wrap_content"
86         android:layout_height="wrap_content"
87         android:layout_below="@+id/btnRegistrati"
88         android:layout_centerHorizontal="true"
89         android:layout_marginTop="50dp"
90         android:lineSpacingExtra="7dp"
91         android:text="Hai già un Account?\nClicca qui."
92         android:textAlignment="center"
93         android:textColor="@color/primaryTextColor"/>
94
95 </RelativeLayout>
96

```

Listato 5.9. Layout dell'activity registrazione

Il tipo di layout utilizzato è il `RelativeLayout`, il quale ci ha aiutato a semplificare la disposizione delle `EditText`, del `Button` e della `TextView`. Inoltre, possiamo notare il background per il pulsante, che troviamo nella cartella `Drawable`, con il nome `login_button`, e dei campi di testo chiamati `edit_text_custom`.

5.3.2 Activity per la prenotazione e layout

Nell'activity `PrenotazioneActivity.java` sono state implementate le funzionalità della prenotazione. Mentre nel file `activity_prenotazione.xml` si trova definito il relativo layout.

Come nella precedente activity il primo metodo che esaminiamo è `onCreate()`, mostrato nel Listato 5.10.

```

1     @Override
2     protected void onCreate(Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         setContentView(R.layout.activity_prenotazione);
5
6         editTextnposti = findViewById(R.id.numberEditText);
7
8         confirmButton = findViewById(R.id.confirmButton);
9         dateButton = findViewById(R.id.dateButton);
10        hourButton = findViewById(R.id.hourButton);
11
12        dateTextView = findViewById(R.id.dataTextView);
13        hourTextView = findViewById(R.id.hourTextView);
14
15        dateButton.setOnClickListener(new View.OnClickListener() {
16            @Override

```



```

17         public void onClick(View v) {
18             handleDateButton();
19         }
20     });
21
22     hourButton.setOnClickListener(new View.OnClickListener() {
23         @Override
24         public void onClick(View v) {
25             handleHourButton();
26         }
27     });
28
29     confirmButton.setOnClickListener(new View.OnClickListener() {
30         @Override
31         public void onClick(View v) {
32             booking();
33         }
34     });
35 }

```

Listato 5.10. onCreate() dell'activity per la prenotazione

Notiamo subito che sono stati impostati tre metodi `setOnClickListener()`. Il primo è relativo al `dateButton`; esso ci permette di aprire il calendario e scegliere il giorno; a tal fine utilizza il metodo `handleDateButton()` come mostrato nel Listato 5.11.

```

1 private void handleDateButton() {
2
3     Calendar calendar = Calendar.getInstance();
4
5     int YEAR = calendar.get(Calendar.YEAR);
6     int MONTH = calendar.get(Calendar.MONTH);
7     int DAY = calendar.get(Calendar.DATE);
8     DatePickerDialog datePickerDialog = new DatePickerDialog
9     (this, new DatePickerDialog.OnDateSetListener() {
10         @Override
11         public void onDateSet(DatePicker view, int year, int month, int date){
12             String datestring = year + "-" + month + "-" + date;
13             dateTextView.setText(datestring);
14         }
15     }, YEAR, MONTH, DAY);
16     datePickerDialog.show();
17 }

```

Listato 5.11. Struttura `handleDateButton()`

In questo metodo creiamo un oggetto di nome `calendar` e definiamo tre variabili intere: `YEAR` (anno), `MONTH` (mese) e `DAY` (giorno). Successivamente, notiamo un listener, `DatePickerDialog.OnDateSetListener()`, il quale riconosce che l'utente ha selezionato la data. Poiché vogliamo stampare la data selezionata nella `dateTextView`, abbiamo creato il metodo `onDateSet()`, il quale ci permette di fare ciò memorizzando la data nella stringa `datestring` e passandola, poi, alla variabile `dateTextView`, tramite il metodo `setText(datestring)`.

Il secondo `setOnClickListener()` è relativo a `hourButton`; esso avvia, tramite il solito `onClick()`, il metodo `handleHourButton()`. Il comportamento implementato è analogo a quello visto per la data ma permetterà di selezionare un orario. Il codice corrispondente è mostrato nel Listato 5.12.

```

1 private void handleHourButton() {
2
3     Calendar calendar = Calendar.getInstance();
4
5     int HOUR = calendar.get(Calendar.HOUR);
6     int MINUTE = calendar.get(Calendar.MINUTE);
7
8     boolean is24HourFormat = DateFormat.is24HourFormat(this);
9
10    TimePickerDialog timePickerDialog =
11    new TimePickerDialog(this,
12    new TimePickerDialog.OnTimeSetListener() {
13        @Override

```

```

14     public void onTimeSet(TimePicker view, int hour, int minute) {
15
16         String timeString = hour + ":" + minute;
17         hourTextView.setText(timeString);
18     }
19     }, HOUR, MINUTE, is24HourFormat);
20     timePickerDialog.show();
21 }
22 }

```

Listato 5.12. Struttura handleHourButton()

Infine, l'ultimo metodo è il `confirmBooking()`, assegnato a `confirmButton`; esso ci permette di confermare la prenotazione. Il codice è molto simile a quello della registrazione (Listato 5.13).

```

1 private void booking(){
2
3     if (dateTextView.getText().toString().equals("Seleziona data")) {
4         dateTextView.setError("Seleziona una data!");
5         dateTextView.requestFocus();
6         return;
7     }
8
9     if (hourTextView.getText().toString().equals("Seleziona orario")) {
10        hourTextView.setError("Seleziona un orario!");
11        hourTextView.requestFocus();
12        return;
13    }
14
15    if (editTextnposti.getText().toString().isEmpty()) {
16        editTextnposti.setText("Inserire il numero di persone!");
17        editTextnposti.requestFocus();
18        return;
19    }
20
21    int numeroposti = Integer.parseInt(editTextnposti.getText().toString().trim());
22    String data = dateTextView.getText().toString().trim();
23    String ora = hourTextView.getText().toString().trim();
24    int cliente_id = 1;
25
26    Call<Prenotazione> call = RetrofitClient.getInstance().getApi().putBooking(numeroposti,
27    data, ora, cliente_id);
28
29    call.enqueue(new Callback<Prenotazione>() {
30        @Override
31        public void onResponse(Call<Prenotazione> call, Response<Prenotazione> response) {
32            if (!response.isSuccessful()){
33                try {
34                    Log.v("prenotazione", "response.code=" + response.code()
35                    + " response.status"
36                    + response.errorBody().toString());
37                } catch (IOException e) {
38                    e.printStackTrace();
39                }
40            }else{
41                openRiepilogo();
42            }
43        }
44
45        @Override
46        public void onFailure(Call<Prenotazione> call, Throwable t) {
47            Log.v("prenotazione", "fail=" + t.getMessage());
48        }
49    });

```

Listato 5.13. Esempio di client REST

Nelle prime righe notiamo una serie di istruzioni `if()` le quali ci servono per eseguire i controlli sull'inserimento di data, ora e numero di posti. Essendo la variabile `numeroposti` un intero, abbiamo utilizzato il metodo `Integer.parseInt()` per poterlo convertire in una stringa al fine di avere una più agevole estrapolazione del dato.

Il valore `cliente_id` è un valore relativo all'id del cliente che ha effettuato il login; per semplicità abbiamo assegnato ad esso un valore in modo tale da non dover

ogni volta effettuare il login per effettuare i vari test. Nel nostro caso il cliente che sta effettuando la prenotazione è colui che ha come ID il numero 1.

Nella nostra interfaccia `Api.java` abbiamo creato la richiesta `putBooking()`, che ci permette di inserire, nelle varie colonne della tabella relativa alle prenotazioni, i vari parametri (numero di posti, data e ora). Abbiamo creato, in questo caso, un nuovo modello che troviamo nella `Call<>` di nome `Prenotazione`, presente nel file `Prenotazione.java`.

In caso di successo, nell'`onResponse()` troviamo il metodo `openRiepilogo()` il quale ci permette, tramite `Intent`, di aprire un riepilogo della prenotazione.

Per concludere, evidenziamo che la nostra activity ha un layout presente nel file `activity_prenotazione.xml`. Non inseriamo il codice di tale layout perchè molto simile a quello della registrazione.

Anche qui abbiamo utilizzato un `RelativeLayout` e inserito dei `Button`, delle `TextView` e un `EditText`.

5.3.3 Activity per il menù e layout

Nell'activity `MenuActivity.java` abbiamo implementato ciò che riguarda la visualizzazione del menù. Come nelle precedenti activity, iniziamo illustrando il metodo più importante, cioè `onCreate()` (Listato 5.14).

```

1      @Override
2      protected void onCreate(Bundle savedInstanceState) {
3          super.onCreate(savedInstanceState);
4          setContentView(R.layout.activity_menu);
5
6          antipasti = findViewById(R.id.antipastiText);
7          primi = findViewById(R.id.primiText);
8          secondi = findViewById(R.id.secondiText);
9          dolci = findViewById(R.id.dolciText);
10         bevande = findViewById(R.id.bevandeText);
11         viewPager = findViewById(R.id.fragment_container);
12
13         ....

```

Listato 5.14. `onCreate` dell'activity menù

Come possiamo notare abbiamo una serie di oggetti che vengono “estrapolati” dal layout tramite il metodo `findViewById(R.id.id_oggetto)`. I primi 5 sono delle `TextView`, mentre l'ultimo è un `ViewPager`. Per creare il layout del menù, che vedremo nel dettaglio in seguito, abbiamo utilizzato i `Fragment` (delle piccole activity).

La logica di gestione dei `Fragment` si basa sempre sul concetto di `Adapter`. Abbiamo creato, infatti, una classe `PagerViewAdapter.java` (che vedremo in dettaglio nella Sezione 5.5), estensione di `FragmentPagerAdapter`. Nel metodo `onCreate()` dell'activity verrà dapprima istanziato un `PagerViewAdapter`, che verrà subito assegnato al `viewPager` tramite il metodo `setAdapter()`; quest'ultimo ci permette di visualizzare le varie `View` (Listato 5.15).

```

1      pagerViewAdapter = new PagerViewAdapter(getSupportFragmentManager());
2      viewPager.setAdapter(pagerViewAdapter);

```

Listato 5.15. `pagerViewAdapter`

Ad ogni `TextView` abbiamo assegnato il metodo `setOnClickListener()` al fine di poter gestire con maggiore facilità il cambio dei `Fragment`, tramite il metodo `setCurrentItem()`, definito nella classe `ViewPager`.

L'ultimo metodo che troviamo nell'`onCreate()` è l'`onPageSelected()` che permette di gestire i vari `Fragment` quando questi vengono selezionati (Listato 5.16).

```

1      @Override
2      public void onPageSelected(int i) {
3          onChangeTab(i);
4      }

```

Listato 5.16. Metodo `onPageSelected()`

All'interno di tale metodo troviamo il metodo `onChangeTab()`, che ci permette di gestire il `Fragment` relativo alla `TextView` che selezioniamo (Listato 5.17).

```

1      private void onChangeTab(int i) {
2
3          if(i == 0) {
4
5              final LinearLayout linearLayoutAntipasti =
6                  findViewById(R.id.linearLayoutAntipasti);
7
8              linearLayoutAntipasti.removeAllViews();
9
10             Call<ArrayList<Piatto>> call = RetrofitClient
11                 .getInstance()
12                 .getApi()
13                 .getPiattoByCategoria(3);
14
15             call.enqueue(new Callback<ArrayList<Piatto>>() {
16                 @Override
17                 public void onResponse(Call<ArrayList<Piatto>> call,
18                     Response<ArrayList<Piatto>> response) {
19                     if(!response.isSuccessful()){
20                         try {
21                             Log.v("errorecat", "response.code=" + response.code()
22                                 + "response.errorBody="
23                                 + response.errorBody().string());
24                         } catch (IOException e) {
25                             e.printStackTrace();
26                         }
27                     } else {
28                         for (Piatto p : response.body()){
29                             TextView tv = new TextView(getApplicationContext());
30                             tv.setText("\n\t" + p.getNome() + "\t\t\t\t\t" + p.getPrezzo() +
31                                 "\neuro \n");
32                             tv.setTextSize(15);
33                             tv.setTextColor(getColor(R.color.primaryDarkColor));
34                             linearLayoutAntipasti.addView(tv);
35                         }
36                     }
37                 }
38
39                 @Override
40                 public void onFailure(Call<ArrayList<Piatto>> call, Throwable t) {
41                     Log.v("fail", "fail=" + t.getMessage());
42                 }
43             });
44
45             antipasti.setTextSize(18);
46             antipasti.setTextColor(getColor(R.color.primaryTextColor));
47
48             primi.setTextSize(13);
49             primi.setTextColor(getColor(R.color.primaryLightColor));
50
51             secondi.setTextSize(13);
52             secondi.setTextColor(getColor(R.color.primaryLightColor));
53
54             dolci.setTextSize(13);
55             dolci.setTextColor(getColor(R.color.primaryLightColor));
56
57             bevande.setTextSize(13);
58             bevande.setTextColor(getColor(R.color.primaryLightColor));
59         }
60
61         if(i == 1) {
62
63             final LinearLayout linearLayoutPrimi =
64                 findViewById(R.id.linearLayoutPrimi);
65
66             linearLayoutPrimi.removeAllViews();
67

```

```

68 Call<ArrayList<Piatto>> call =
69 RetrofitClient.getInstance()
70 .getApi()
71 .getPiattibyCategoria(1);
72
73 call.enqueue(new Callback<ArrayList<Piatto>>() {
74 @Override
75 public void onResponse(Call<ArrayList<Piatto>> call,
76 Response<ArrayList<Piatto>> response) {
77 if(!response.isSuccessful()){
78 try {
79 Log.v("errorecat", "response.code=" + response.code()
80 + "response.status" + response.errorBody().string());
81 } catch (IOException e) {
82 e.printStackTrace();
83 }
84 } else {
85 for (Piatto p : response.body()){
86 TextView tv = new TextView(getApplicationContext());
87 tv.setText("\n\t" + p.getNome() + "\t\t\t\t\t" + p.getPrezzo() +
88 "\neuro \n");
89 tv.setTextSize(15);
90 tv.setTextColor(getColor(R.color.primaryDarkColor));
91 linearLayoutPrimi.addView(tv);
92 }
93 }
94 }
95
96 @Override
97 public void onFailure(Call<ArrayList<Piatto>> call, Throwable t) {
98 Log.v("fail", "fail=" + t.getMessage());
99 }
100 });
101
102 antipasti.setTextSize(13);
103 antipasti.setTextColor(getColor(R.color.primaryLightColor));
104
105 primi.setTextSize(18);
106 primi.setTextColor(getColor(R.color.primaryTextColor));
107
108 secondi.setTextSize(13);
109 secondi.setTextColor(getColor(R.color.primaryLightColor));
110
111 dolci.setTextSize(13);
112 dolci.setTextColor(getColor(R.color.primaryLightColor));
113
114 bevande.setTextSize(13);
115 bevande.setTextColor(getColor(R.color.primaryLightColor));
116 }
117
118 ....
119
120 }

```

Listato 5.17. Struttura onChangeTab()

In questo metodo, come possiamo, notare abbiamo inserito una serie di istruzioni `if()` le quali ci permettono di gestire i `Fragment` e le `TextView` che selezioniamo. La prima parte è la solita `Call<>` messa a disposizione da `Retrofit`, che ci permette di stampare nel `fragment` i vari piatti presenti nel database di una certa categoria. Ad esempio, nel caso in cui `i==0` ci troviamo nella sezione degli antipasti; infatti, passiamo alla `getPiattibyCategoria()` l'ID 3 che è, appunto, l'ID della categoria relativa agli antipasti. In caso di successo, grazie all'oggetto `tv`, possiamo stampare tutta la lista dei piatti presenti nella categoria degli antipasti utilizzando un `ArrayList`, come possiamo notare all'interno della `Call<ArrayList<Piatto>>`.

Alla fine dell'`if()` utilizziamo due metodi per ogni `TextView`. Il primo metodo, `setTextSize()`, ci permette di impostare la dimensione del testo ad un certo valore. Il secondo metodo, `setTextColor()`, ci consente di impostare il colore. Questo perchè, riprendendo il caso di prima, nel momento in cui ci troviamo nella sezione antipasti, vogliamo che il testo "Antipasti" abbia un font più grande rispetto al resto del testo (18 invece che 13) e un colore differente (`primaryTextColor` invece che `primaryLightColor`). Così facendo, mettiamo in evidenza la sezione in cui ci troviamo. Lo stesso discorso vale nel caso in cui ci troviamo nelle altre categorie: primi, secondi, dolci, bevande.

Il layout del menù comprende più file poiché, come abbiamo già anticipato, sono stati utilizzati i Fragment per strutturarli. Il primo file che analizziamo è `tab_bar.xml` (Listato 5.18).

```

1      <LinearLayout
2          xmlns:android="http://schemas.android.com/apk/res/android"
3          android:layout_width="match_parent"
4          android:layout_height="wrap_content"
5          android:orientation="horizontal">
6
7          <TextView
8              android:id="@+id/antipastiText"
9              android:layout_width="match_parent"
10             android:layout_height="50dp"
11             android:text="Antipasti"
12             android:textSize="15dp"
13             android:textStyle="bold"
14             android:gravity="center"
15             android:textColor="@color/primaryLightColor"
16             android:layout_weight="1"/>
17
18         <TextView
19             android:id="@+id/primiText"
20             android:layout_width="match_parent"
21             android:layout_height="50dp"
22             android:text="Primi"
23             android:textSize="15dp"
24             android:textStyle="bold"
25             android:gravity="center"
26             android:textColor="@color/primaryLightColor"
27             android:layout_weight="1"/>
28
29         <TextView
30             android:id="@+id/secondiText"
31             android:layout_width="match_parent"
32             android:layout_height="50dp"
33             android:text="Secondi"
34             android:textSize="15dp"
35             android:textStyle="bold"
36             android:gravity="center"
37             android:textColor="@color/primaryLightColor"
38             android:layout_weight="1"/>
39
40         <TextView
41             android:id="@+id/dolciText"
42             android:layout_width="match_parent"
43             android:layout_height="50dp"
44             android:text="Dolci"
45             android:textSize="15dp"
46             android:textStyle="bold"
47             android:gravity="center"
48             android:textColor="@color/primaryLightColor"
49             android:layout_weight="1"/>
50
51         <TextView
52             android:id="@+id/bevandeText"
53             android:layout_width="match_parent"
54             android:layout_height="50dp"
55             android:text="Bevande"
56             android:textSize="15dp"
57             android:textStyle="bold"
58             android:gravity="center"
59             android:textColor="@color/primaryLightColor"
60             android:layout_weight="1"/>
61
62     </LinearLayout>

```

Listato 5.18. Layout della tab bar

Il layout utilizzato è il `LinearLayout` e, al suo interno, abbiamo inserito delle `TextView` con, appunto, i nomi delle varie categorie. Questo è il layout della barra per la scelta della categoria nel menù.

I vari fragment hanno un layout davvero semplice. Sono composti da due `LinearLayout` uno dentro l'altro. Abbiamo utilizzato questo tipo di layout per semplificarci la stampa dei piatti, poiché, con gli altri tipi di layout, riscontravamo dei problemi. Esponiamo di seguito il codice di uno dei Fragment, dato che sono tutti molto simili (Listato 5.19).

```

1 <LinearLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:tools="http://schemas.android.com/tools"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   android:background="@drawable/bg_color">
7
8   <LinearLayout
9     android:id="@+id/linearLayoutDolci"
10    android:layout_width="match_parent"
11    android:layout_height="500dp"
12    android:layout_marginTop="10dp"
13    android:layout_marginLeft="10dp"
14    android:layout_marginRight="10dp"
15    android:background="@drawable/fragment_bg"
16    android:orientation="vertical"
17    tools:layout_editor_absoluteX="0dp"
18    tools:layout_editor_absoluteY="0dp">
19
20   </LinearLayout>
21
22 </LinearLayout>

```

Listato 5.19. Esempio di layout dei Fragment

Infine, abbiamo il layout principale che è contenuto in `activity_menu.xml` (Listato 5.20).

```

1 <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:app="http://schemas.android.com/apk/res-auto"
4   xmlns:tools="http://schemas.android.com/tools"
5   android:layout_width="match_parent"
6   android:layout_height="match_parent"
7   android:background="@drawable/bg_color"
8   tools:context=".activities.MenuActivity">
9
10  <include layout="@layout/tab_bar"/>
11  <android.support.v4.view.ViewPager
12    android:id="@+id/fragment_container"
13    android:layout_width="match_parent"
14    android:layout_height="500dp"
15    android:layout_marginTop="52dp"/>
16
17  <TextView
18    android:id="@+id/totalTextView"
19    android:layout_width="wrap_content"
20    android:layout_height="wrap_content"
21    android:layout_below="@+id/fragment_container"
22    android:text="Note:"
23    android:textColor="@color/primaryTextColor"
24    android:textSize="20dp"/>
25
26 </RelativeLayout>

```

Listato 5.20. Layout dell'activity menù

In questo layout abbiamo incluso la `tab_bar` e abbiamo inserito il `ViewPager`, già visto prima, al quale abbiamo, appunto, assegnato l'ID `fragment_container` perchè è quella parte del layout che dovrà contenere tutti i vari `Fragment`.

5.4 Interfaccia Api.java

`Api.java` è un'interfaccia creata per richiedere una REST API. Essa è un'interfaccia per gestire le query (GET, POST, etc.) e viene richiamata nelle activity tramite il metodo `getApi()`. Nel Listato 5.21 una parte della nostra interfaccia con tre esempi di richieste.

```

1 public interface Api {
2
3     ....

```

```

4
5     @FormUrlEncoded
6     @POST("user")
7     Call<User> createUser(
8         @Field("username") String username,
9         @Field("email") String email,
10        @Field("password") String password,
11        @Field("nome") String name,
12        @Field("cognome") String surname
13    );
14
15    @GET("user/{id}")
16    Call<User> getIdUtente(
17        @Path("id") int id
18    );
19
20    @FormUrlEncoded
21    @PATCH("user/{id}")
22    Call<User> patchDatiUtente(
23        @Path("id") int id,
24        @Field("email") String email,
25        @Field("nome") String name,
26        @Field("cognome") String surname
27    );
28
29    ....
30
31 }

```

Listato 5.21. Alcune richieste nell'interfaccia `Api.java`

La prima è una richiesta di tipo `POST`, la quale si trova nell'activity, per la registrazione, `Registrazione.java` e viene invocata tramite il metodo `createUser()`. Essa ci permette di inserire le varie stringhe (username, email, password, nome e cognome) nella relativa tabella `User` del database. La `GET`, invece, ci restituisce l'ID relativo ad un utente. La `PATCH`, infine, serve per modificare delle stringhe (nel nostro caso email, nome e cognome) relative ad un utente.

5.5 Altre classi

In questa sezione descriveremo alcune delle classi più importanti da noi utilizzate.

5.5.1 Classe `RetrofitClient.java`

Questa classe è stata creata perchè, utilizzando `Retrofit` nell'app, è una buona pratica creare una classe che ci permette di interfacciarci velocemente al nostro database e alle nostre API in ogni activity. Innanzitutto, abbiamo definito l'URL delle nostre API, un oggetto `mInstance` e un oggetto `Retrofit` di nome `retrofit` (Listato 5.22).

```

1     private static final String BASE_URL = "http://134.209.194.10/index.php/api/";
2     private static RetrofitClient mInstance;
3     private Retrofit retrofit;

```

Listato 5.22. Definizione dell'URL, dell'oggetto `mInstance` e di un oggetto `Retrofit`

Successivamente abbiamo creato un costruttore così da avere il nostro oggetto `Retrofit` (Listato 5.23).

```

1     private RetrofitClient(){
2
3         retrofit = new Retrofit.Builder()
4             .baseUrl(BASE_URL)

```



```

5     .addConverterFactory(GsonConverterFactory.create())
6     .build();
7     }

```

Listato 5.23. RetrofitClient()

A questo punto è stato necessario inserire un metodo `synchronized` per avere una singola istanza della nostra classe `RetrofitClient` (Listato 5.24).

```

1     public static synchronized RetrofitClient getInstance(){
2         if(mInstance == null){
3             mInstance = new RetrofitClient();
4         }
5         return mInstance;
6     }

```

Listato 5.24. Metodo getInstance()

Nel momento in cui abbiamo `mInstance==null` vuol dire che la nostra istanza non è stata ancora creata e, quindi, si procede alla sua creazione tramite l'istruzione `mInstance = new RetrofitClient()`.

Infine, abbiamo bisogno di un ultimo metodo per “prendere” le nostre API (Listato 5.25).

```

1     public Api getApi(){
2         return retrofit.create(Api.class);
3     }

```

Listato 5.25. Metodo getApi()

Quindi, nelle varie `activity`, grazie a questa classe, possiamo fare, ad esempio, una richiesta di tipo `GET` (Listato 5.26).

```

1     Call<User> call = RetrofitClient.getInstance().getApi().getIdUtente(idUser);

```

Listato 5.26. Esempio di Call<>

5.5.2 Classe User.java

In questa classe abbiamo definito il modello di un utente che, appunto, è costituito da `username`, `nome`, `cognome`, `email` e `password`, creando un costruttore e dei `Getter` (Listato 5.27).

```

1     public class User {
2
3         private String username, email, password, nome, cognome;
4
5
6         public User(String username, String nome, String cognome, String email,
7             String password) {
8
9             this.password = password;
10            this.nome = nome;
11            this.cognome = cognome;
12            this.email = email;
13            this.username = username;
14        }
15
16        public String getPassword() {
17            return password;
18        }
19    }

```

```

20     public String getNome() {
21         return nome;
22     }
23
24     public String getCognome() {
25         return cognome;
26     }
27
28     public String getEmail() {
29         return email;
30     }
31
32     public String getUsername() {
33         return username;
34     }
35 }

```

Listato 5.27. Modello User

Altre classi simili sono state create per il modello dei piatti, della prenotazione, etc.

5.5.3 Classe PagerViewAdapter.java

Questa classe è stata creata per scorrere le varie view dei Fragment. Semplicemente abbiamo creato uno `switch-case-break` che, a seconda del Fragment che abbiamo selezionato, ci restituisce la relativa vista (Listato 5.28).

```

1     @Override
2     public Fragment getItem(int position) {
3
4         Fragment fragment = null;
5         switch (position){
6             case 0:
7                 fragment = new Fragment_Antipasti();
8                 break;
9             case 1 :
10                fragment = new Fragment_Primi();
11                break;
12            case 2 :
13                fragment = new Fragment_Secondi();
14                break;
15            case 3 :
16                fragment = new Fragment_Dolci();
17                break;
18            case 4 :
19                fragment = new Fragment_Bevande();
20                break;
21        }
22        return fragment;
23    }

```

Listato 5.28. Struttura getItem()

Inoltre, abbiamo aggiunto un contatore, che restituisce il numero di Fragment che abbiamo, nel nostro caso 5 (Listato 5.29).

```

1     @Override
2     public int getCount() {
3         return 5;
4     }

```

Listato 5.29. Metodo getCount()

Manuale Utente

In questo capitolo verranno presentate tutte le funzionalità dell'applicazione sviluppata; a tal fine saranno proposti gli screenshot che descrivono le varie interfacce pensate per gli utenti.

6.1 Funzionalità previste per l'utente

Come già detto nei capitoli precedenti quest'applicazione è di tipo nativo, quindi supportata da smartphone con sistema operativo Android con versione Marshmallow o superiore.

Per il corretto funzionamento dell'applicazione è necessario che lo smartphone sia collegato ad una rete Wi-Fi o tramite connessione dati cellulare.

Nel momento in cui apriamo l'app ci troveremo davanti la schermata di Login (Figura 6.1). Se si è già in possesso di uno username e una password, basterà inserire nei relativi campi di testo le credenziali e cliccare il pulsante "LOGIN" (1) per effettuare l'accesso. Nel caso in cui non si è in possesso di uno username e una password, cliccando il testo "Non hai un account? Clicca qui." (2) verremmo reindirizzati alla pagina di registrazione (Figura 6.1).

Per effettuare la registrazione basterà compilare ogni campo di testo presente in questa schermata e cliccare sul pulsante "REGISTRATI" (3). Se, invece, si è in possesso di un account basterà cliccare sul testo "Hai già un account? Clicca qui." (4) per tornare alla schermata di login.

Dopo aver effettuato il login, verremo reindirizzati alla schermata Home (Figura 6.2). In questa schermata possiamo trovare delle informazioni relative al ristorante. Cliccando nelle tre barrette laterali in alto a sinistra possiamo aprire un menù laterale (Figura 6.2) con una serie di operazioni da poter effettuare.

La prima voce che troveremo nel menù è la "Home" (5); cliccando su tale voce possiamo riaccedere alla schermata della home. Cliccando, invece, nella seconda voce "Prenotazione" (6) accediamo alla schermata relativa alla prenotazione (Figura 6.3).

In tale schermata è possibile effettuare una prenotazione. In essa dobbiamo scegliere la data, l'ora e il numero di persone.

Cliccando sul pulsante "DATA" (11) apparirà un calendario grazie al quale possiamo scegliere il giorno. Il pulsante "ORARIO" (12) farà apparire un orologio con

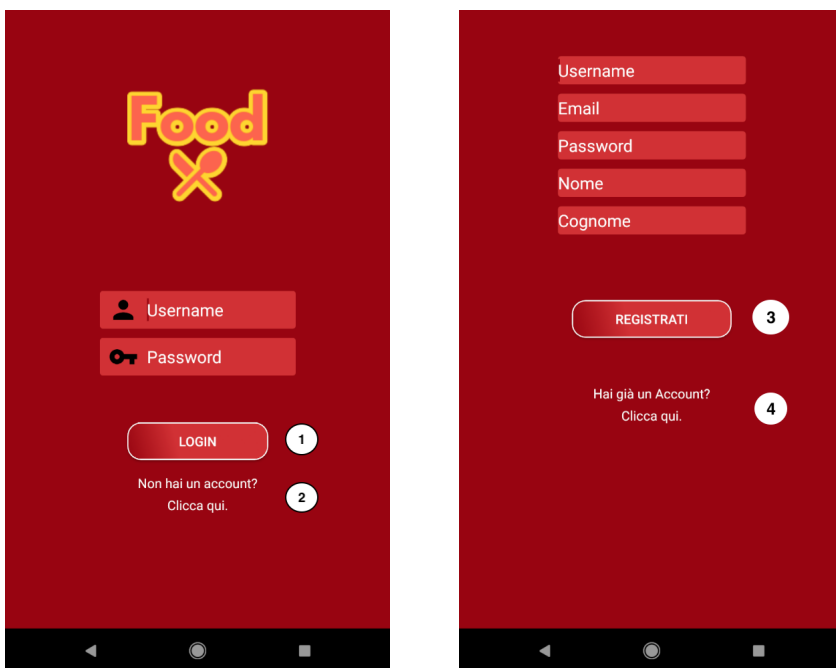


Figura 6.1. Schermata di login (a sinistra) e schermata di registrazione (a destra)



Figura 6.2. Schermata home (a sinistra) e schermata del menù laterale (a destra)

il quale possiamo scegliere l'ora e, nella casella a fianco della scritta "Inserisci il numero di posti" si dovrà, appunto, inserire il numero di partecipanti. Una volta fatto ciò, basterà cliccare sul pulsante "CONFERMA" (13) per effettuare la prenotazione. Successivamente, si verrà reindirizzati alla schermata di riepilogo della prenotazione (Figura 6.3).



Figura 6.3. Schermata di prenotazione (a sinistra) e schermata di riepilogo della prenotazione (a destra)

In questa schermata è possibile visualizzare in dettaglio la prenotazione effettuata. Il pulsante "OK" (14) ci riaprirà la schermata Home, mentre con il pulsante "CANCELLA" (15) è possibile cancellare la prenotazione. Si può tornare a questa schermata cliccando nel menù laterale della Home sulla voce "Riepilogo prenotazione" (7).

Tornando nel menù laterale della Home abbiamo un'altra voce che è il "Menù" (8) la quale ci farà accedere al menù del ristorante (Figura 6.4).

Qui abbiamo la possibilità di visualizzare il menù del ristorante; per navigare tra le varie categorie basterà cliccare sulle relative voci in alto e visualizzare i piatti con i relativi prezzi di ogni categoria. Inoltre, in basso, possiamo trovare delle "Note" da parte del ristorante, come, ad esempio, piatti mancanti, allergeni, etc. Cliccando nella freccia indietro (16) torneremo alla schermata Home.

La penultima voce è "Account" (9) che aprirà una schermata per modificare le nostre credenziali (Figura 6.5).

In questa schermata troveremo già le nostre credenziali impostate nei relativi campi di testo (email, nome e cognome) e potremo effettuare modifiche ad esse

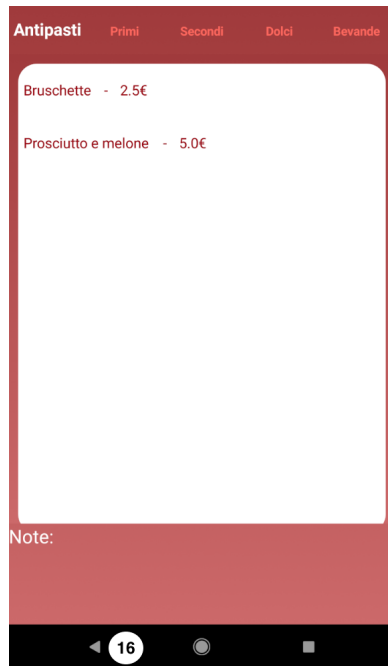


Figura 6.4. Schermata del menù del ristorante

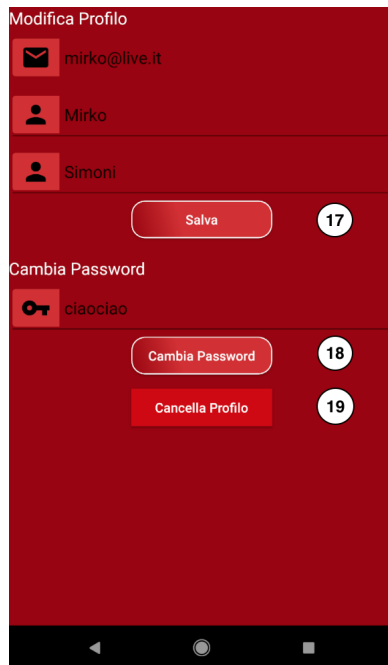


Figura 6.5. Schermata di modifica del profilo

cliccando sul pulsante “SALVA” (17). Lo stesso vale per la password ma, per modificarla, dovremmo cliccare nel pulsante “CAMBIA PASSWORD” (18). Invece, per cancellare il nostro account, basterà cliccare sull'ultimo pulsante “CANCELLA PROFILO” (19) e verremo reindirizzati alla schermata di registrazione.

Infine, nel menù laterale della Home, abbiamo la possibilità di effettuare il logout cliccando nell'apposita voce “logout” (10) tornando, così, alla schermata di login.

6.2 Funzionalità previste per l'amministratore

L'amministratore potrà, una volta effettuato il login, accedere alla schermata per l'aggiunta dei piatti al menù. A tal punto utilizzerà la schermata in Figura 6.6.

Figura 6.6. Schermata di aggiunta di un piatto

In questa schermata è necessario inserire il nome e il prezzo del piatto nelle relative caselle di testo, e scegliere nel menù a tendina (20) la relativa categoria a cui assegnare il piatto. Cliccando sul pulsante “Aggiungi” (21), il piatto verrà aggiunto al menù nella relativa sezione.

Discussione in merito al lavoro svolto

In questo capitolo verranno esposte delle considerazioni sulla nostra esperienza relativa all'utilizzo di Android Studio. Successivamente, verranno mostrate le difficoltà e le problematiche riscontrate durante il percorso di progettazione.

7.1 Lezioni apprese nell'utilizzo di Android Studio

Così come dimostrato nella presente tesi, il mondo delle applicazioni native gode di un insieme esteso e vario di librerie di supporto e strumenti, come Android Studio, per facilitare lo sviluppo software. In particolare, anche e soprattutto dalla nostra esperienza, abbiamo potuto appurare quanto Android Studio sia uno strumento molto potente, che consente di realizzare app con facilità, grazie al mix di tool che garantisce per aiutarci con la programmazione in Java. La chiarezza e la semplicità con cui possiamo utilizzare questi tool è davvero strabiliante. Infatti, iniziando a sviluppare il primo codice con Android Studio, si nota subito che l'IDE è molto comodo e veloce nell'interazione con l'utente. La parte di layout di Android Studio mette a disposizione una componente di design, dove possiamo divertirci a dare un'interfaccia grafica alla nostra app manualmente; chi lo preferisce può, invece, giostrarsi utilizzando codice XML. Altra particolarità è che esiste un efficiente meccanismo di preview dei layout. L'aspetto dell'interfaccia viene velocemente mostrato in anteprima, non in maniera generica ma adattato realisticamente ai dispositivi.

Grazie alle tantissime librerie al proprio interno e alla documentazione, messa a disposizione da Android Studio, si viene aiutati molto nell'utilizzo di Java in svariati modi diversi: Widget per le View e tanto altro. Nel momento in cui non si conosce, ad esempio, cosa faccia un metodo, Android Studio mette a disposizione una finestra, di lato al metodo scelto, con una dettagliata spiegazione (Figura 7.1).

Inoltre, quando riscontriamo errori sul codice, abbiamo nel nostro "Build" un riferimento all'errore (Figura 7.2) e, con il doppio click su di esso, siamo rimandati al file e alla riga in cui l'errore stesso si trova.

Nel menù dei tool abbiamo, inoltre, a disposizione un AVD Manager che si occupa di gestire i dispositivi virtuali, utilizzabili per il test delle applicazioni. Una volta invocato si può richiedere la creazione di un nuovo emulatore scegliendone le caratteristiche. Al termine lo si può avviare ed utilizzare per l'esecuzione della

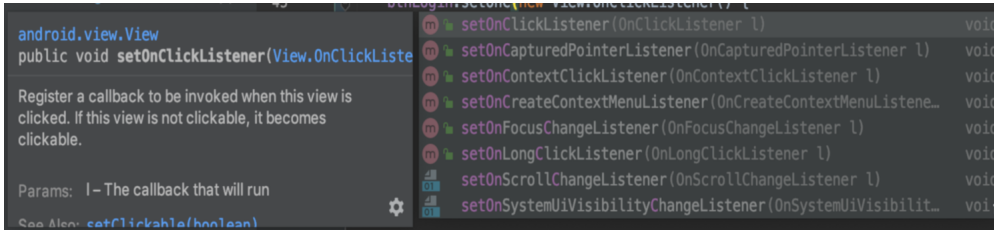


Figura 7.1. Esempio spiegazione del metodo `setOnClickListener()`

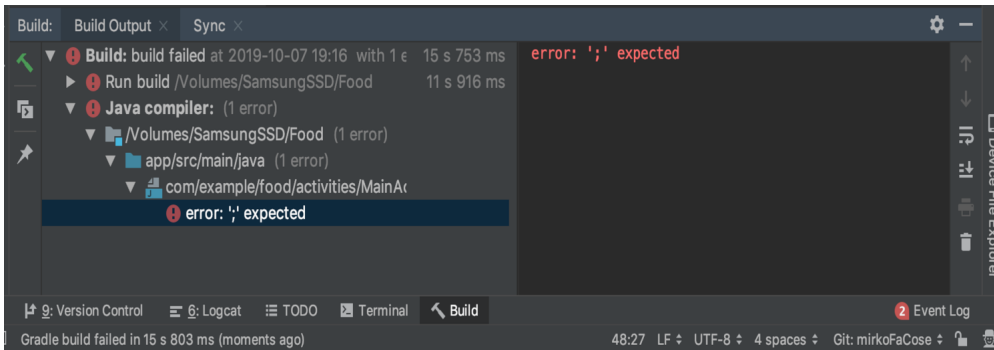


Figura 7.2. Esempio di riferimento ad un errore (mancanza del punto e virgola)

propria app. Questo tool è molto comodo per programmatori che non possiedono ancora un dispositivo Android o per testare l'applicazione anche su dispositivi che non possediamo; inoltre, esso ci consente di programmare solo con il PC, senza bisogno di altro. Sicuramente è meglio testare la propria app su uno dispositivo fisico, ma l'emulatore è pur sempre comodo. L'usabilità degli emulatori al giorno d'oggi è davvero eccellente. Possiamo vedere un esempio di emulatore nella Figura 7.3.

Uno degli strumenti con cui si viene a contatto presto è il Gradle. Si tratta di un prodotto di build automation molto diffuso nel mondo Java. La sua presenza si nota subito nella struttura del progetto. Si vede, a parte, una cartella denominata Gradle Scripts. Al suo interno ci sono due file `build.gradle`: uno generale per tutto il progetto ed uno più specifico per ogni modulo che stiamo sviluppando. Il linguaggio in cui sono scritti i file Gradle è Groovy; è importante prenderci confidenza soprattutto con le direttive di maggiore importanza come le dependencies, necessarie a risolvere le dipendenze del progetto.

Spesso il programmatore sarà chiamato ad introdurre modifiche ad uno o più file `build.gradle`; in questi casi sarà molto importante premere il pulsante *Sync...* dopo il salvataggio affinché i cambiamenti apportati vengano assimilati dal progetto.

L'IDE dispone, anche, di molti template per applicazioni. Ogni volta che inizieremo un nuovo progetto ci verrà offerta la possibilità di impiantarli su una struttura già pronta per le tipologie di app più comuni. Non solo in questo modo risparmieremo del tempo ma avremo un'architettura di base rispettosa di tutte le principali best practice.

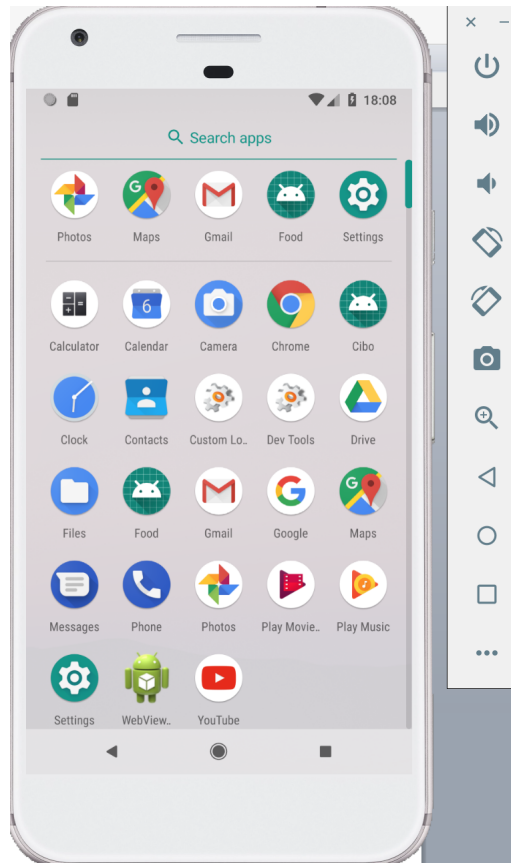


Figura 7.3. Esempio di emulatore

Android Studio è un ambiente ricco e veloce che non costituisce l'unico modo per realizzare app Android, ma, sicuramente, rappresenta quello più efficiente e sempre al passo con i tempi.

7.2 Problemi riscontrati

Le problematiche riscontrate erano dovute, in primo luogo, all'inesperienza, dovuta al lungo intervallo di tempo passato senza programmare in Java. Buona parte di tali lacune è stata colmata seguendo le lezioni del corso di Programmazione Mobile. Capire come implementare l'intero meccanismo di chiamate sincrone e asincrone di Retrofit, in modo efficace e con un'adeguata gestione degli errori, ha richiesto un adeguato sforzo di apprendimento. Una volta superato tale scoglio, le successive iterazioni si sono state sviluppate sempre più velocemente.

Per la creazione delle classi, dell'interfaccia e delle varie activity, siamo quasi sempre partiti da una base vuota per poi creare interamente il tutto, avvalendoci di numerosi esempi sui vari siti web con cui siamo potuti confrontare. Come ben

sappiamo, sviluppando un'applicazione non si riesce mai a trovare ciò che si vuole, ma soltanto materiale sparso per il web che bisogna comprendere e riadattare totalmente al proprio progetto.

Di quando in quando, inoltre, abbiamo avuto modo di colloquiare con altri colleghi sviluppatori Android all'interno dell'università, con i quali abbiamo discusso in merito ai problemi riscontrati ricavandone dalle discussioni dei buoni spunti per le diverse correzioni/soluzioni.

Per quanto riguarda l'implementazione dei layout, abbiamo scritto l'intero codice XML per tutte le schermate, abbiamo creato le varie icone tramite i `vector assets` che si trovano dentro la cartella `drawable` e i vari background dei pulsanti e campi di testo prestando particolare attenzione a come esse figurassero con i dispositivi. Tale attività si è rivelata, anch'essa, piuttosto complessa, almeno all'inizio. Infatti, non è semplice prendere confidenza con l'XML. Tuttavia, una volta individuato il giusto tipo di layout con i vari strumenti messi a disposizione, è tutto in discesa. Anche in questo caso abbiamo preso spunto dagli innumerevoli esempi presenti nella rete.

Sarebbe stato interessante implementare una grafica basata sul Material Design, ma è sorto un grande problema di compatibilità con i fragment che componevano il menù. Ciononostante è giusto sottolineare che la grafica progettata è di per sé molto basilare, senza particolari espedienti. Inoltre, utilizzando il designer presente all'interno di Android Studio, abbiamo potuto avere una visione in tempo reale di quanto stavamo producendo, senza dover testare il tutto su un dispositivo reale per ciascuna modifica.

Conclusioni e uno sguardo al futuro

Nella presente tesi abbiamo discusso della progettazione e realizzazione di un'applicazione Android per la gestione dei clienti di un ristorante. All'inizio abbiamo presentato un'introduzione alle applicazioni raccontando in parte la loro storia, i vari tipi e i sistemi operativi supportati con i relativi distributori.

Successivamente abbiamo analizzato Android esponendo la storia, fornendo una dettagliata descrizione di alcune sue componenti e sulla sua architettura ed, infine, presentando un piccolo riassunto sulla cronologia delle versioni.

Si è, quindi, pensato di realizzare un'applicazione Android attraverso l'uso di un IDE, in particolare Android Studio, di cui ne abbiamo analizzato le caratteristiche principali, i suoi punti di forza e la facilità con cui consente la realizzazione di applicazioni.

Per garantire una maggiore qualità, come in ogni progetto, è stato necessario effettuare una prima parte dedicata alla progettazione. In particolare, abbiamo effettuato un'analisi dei requisiti, per individuare le funzionalità che dovevano essere implementate nell'applicazione; successivamente, siamo passati alla progettazione esponendo i vari strumenti utilizzati, i linguaggi di programmazione per realizzare applicazioni Android e fornendo una panoramica sulle varie componenti di un'applicazione.

Infine, siamo passati alla descrizione della fase implementativa, durante la quale, oltre a descrivere gli strumenti messi a disposizione da Android Studio per la programmazione di un'applicazione, abbiamo illustrato l'implementazione di alcune sue funzionalità e descritto le librerie da noi utilizzate.

L'applicazione da noi realizzata ha ancora tanta strada da fare. Ci sono alcune funzioni presenti nell'analisi dei requisiti che non siamo riusciti ad implementare come, ad esempio, la parte relativa al personale e alla gestione dei pagamenti, le quali verranno implementate nelle versioni successive. Inoltre abbiamo pensato che si potrebbe implementare un sistema che permette non solo di visualizzare un menù ma anche di ordinarlo. Pensiamo, infine, che sarebbe possibile espandere l'applicazione per più ristoranti, inserendo un'activity per la ricerca di essi.

Ringraziamenti

Questo elaborato segna la fine del mio percorso di studi relativo alla laurea triennale in Ingegneria dell'Informatica e dell'Automazione e l'inizio della magistrale.

Desidero ringraziare tutti coloro che mi hanno aiutato nella stesura di questa tesi di laurea e nel relativo progetto ad essa collegato, in particolare Andrea ed Edoardo. Un grande ringraziamento a mia madre e mio padre che, con il loro dolce e instancabile sostegno, sia morale che economico, mi hanno permesso di arrivare fin qui contribuendo alla mia formazione personale.

Un sincero ringraziamento va al mio relatore, il Professore Domenico Ursino, che mi ha permesso di intraprendere questo progetto ed il relativo lavoro di stesura della tesi e mi ha, anche, aiutato considerevolmente durante questi mesi, essendo sempre molto cordiale e disponibile.

Un ringraziamento speciale va a mia sorella Alice che, come i miei genitori, mi è stata sempre vicina e aiutato ad affrontare anche i momenti più difficili.

Infine, un ringraziamento va a tutti i miei amici e colleghi che hanno reso questi anni davvero interessanti. Ringrazio, quindi, i miei colleghi Riccardo, Lorenzo, Roman, Ornald, Bruno, Francesco, Carlo, Matteo e Radek, con cui ho condiviso momenti di gioia, che mi hanno permesso di staccare momentaneamente dai libri, e di studio. Ringrazio, anche, la mia amica Giulia, con la quale sono riuscito a superare il nostro ultimo esame, Martina, che con molta pazienza ha riletto tutti i miei capitoli e mi ha aiutato nella correzione, e Melanie, per l'aiuto con InDesign e per avermi sempre spronato a fare nuove esperienze.

Dire grazie non è mai un gesto scontato, non è mai inutile. Quindi, di nuovo, un grazie di cuore a tutti!

Riferimenti bibliografici

1. The history of apps. An infographic approach. <https://www.techaheadcorp.com/blog/the-history-of-apps-an-infographic-approach/>, 2015.
2. Guida Android. <https://www.html.it/guide/guida-android/>, 2016.
3. Sviluppo di software Android. https://it.wikipedia.org/wiki/Sviluppo_di_software_Android, 2016.
4. App native e app ibride: che differenze ci sono? <https://www.kaleidoscope.it/magazine/app-native-e-app-ibride>, 2017.
5. Come funzionano le librerie Java. <https://www.nextre.it/funzionano-le-librerie-java/>, 2017.
6. Un'infografica mostra l'evoluzione delle app nel tempo, dall'IBM Simon a Snapchat. <https://www.tuttoandroid.net/news/uninfografica-mostra-levoluzione-delle-app-nel-tempo-dallibm-simon-snapchat-480714/>, 2017.
7. Iniziare con Retrofit2. <https://riptutorial.com/it/retrofit2>, 2018.
8. L'architettura della piattaforma Android. <https://www.decodexlab.com/zero/3-1-larchitettura-della-piattaforma-android/>, 2018.
9. Android. <https://it.wikipedia.org/wiki/Android>, 2019.
10. Android: origini, storia, versioni varianti del robottino verde. <https://www.tuttoandroid.net/android/>, 2019.
11. App mobile ibride, native o web: le differenze. <https://www.html.it/faq/app-mobile-ibride-native-o-web-le-differenze/>, 2019.
12. Applicazione mobile. https://it.wikipedia.org/wiki/Applicazione_mobile, 2019.
13. J. Bloch. *Effective Java*. Addison-Wesley Professional, 2017.
14. M. Bonifazi, F. Collini, A. Martellucci, and S. Sanna. *Android. Programmazione avanzata*. Edizioni LSWR, 2015.
15. C. De Sio Cesari. *Manuale di Java 9. Programmazione orientata agli oggetti con Java standard edition 9*. Hoepli, 2018.
16. I. F. Darwin. *Android Cookbook*. O Reilly, 2011.
17. M. Delisle. *PhpMyAdmin e MySQL. Guida pratica*. Mondadori Informatica, 2005.
18. M. Gargenta and M. Nakamura. *Sviluppare con Android: Realizzare le applicazioni mobili con Java ed Eclipse*. OàReilly, 2014.
19. G. Grandinetti. *Android studio. Sviluppare vere applicazione Android partendo da zero*. Edizionifutura, 2014.
20. A. Lorenzi and A. Rizzi. *Java. Programmazione ad oggetti e applicazioni Android*. Atlas, 2013.
21. W. Savitch. *Programmazione di base e avanzata con Java*. Pearson, 2018.
22. H. Schildt. *Java. La guida completa*. McGraw-Hill, 2014.

23. D. Shepherd. *XML*. Apogeo, 2002.
24. N. Smyth. *Android Studio 3.5 Development Essentials - Java Edition*. Payload Media, 2019.