Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Progettazione e implementazione di un'app Android per la gestione di un calendario di ricorrenze

Design and implementation of an Android app for managing a recurrence calendar

Relatore

Prof. Domenico Ursino

Candidato

Luca Agostinelli

Anno Accademico 2018-2019

Indice

Elenco delle figure

Elenco dei listati

Introduzione

È sotto gli occhi di tutti il fatto che, da qualche decennio a questa parte, stiamo vivendo cambiamenti radicali sotto il punto di vista della tecnologia e del modo quotidiano di relazionarci con essa. La parola all'ordine del giorno è "semplificare".

Infatti, siamo passati ad utilizzare sempre più la tecnologia per migliorare la vita di tutti i giorni, semplificando operazioni che un tempo sarebbero state complicate. Basti pensare come è facile, al tempo d'oggi, cercare una notizia che magari un tempo richiedeva consultare anche decine di libri. La massima potenza di come è facile fare operazioni o cercare notizie dovunque siamo è racchiusa nel dispositivo che teniamo in tasca: lo smartphone.

Ancora non ci rendiamo conto del potenziale di questi dispositivi. Basta focalizzarsi sul fatto che i telefoni moderni sono migliaia di volte piu potenti dei calcolatori che hanno portato il primo uomo sulla Luna. I cellulari sono muniti, quindi, di processori molto potenti, interfacce facili da utilizzare e, cosa non da poco conto, riescono ad accedere alla rete da qualsiasi luogo in modo semplice e veloce, permettendo all'utente di avere le funzionalità di un personal computer nel palmo della propria mano, con la comodità di peso e dimensione ridotte. Non è da sottovalutare questo concetto, poichè è di grande rilievo il fatto che migliaia di utenti, grazie ai dispositivi mobili, stiano abbandonando sempre di più l'uso del computer, che sia un PC portatile o meno. Grazie alla praticità, alla velocità e ad altre caratteristiche, questi dispositivi possono essere usati come strumenti per la visualizzazione di dati ed informazioni riguardanti ogni ambito da tutte quelle persone che hanno la necessità di accedere ad informazioni sensibili anche in assenza di postazioni fisse; solo a titolo di esempio, pensiamo ad un medico che deve vedere i principi attivi di un medicinale o ad un camionista che non trova più la strada. Al giorno d'oggi tutti usufruiamo di queste tecnologie, e la maggior parte di noi non vorrebbe tornare indietro.

Tutte queste funzionalità sono utilizzabili tramite applicazioni, cioè software applicativi caratterizzati da una semplificazione ed eliminazione del non strettamente necessario, al fine di ottenere pulizia dell'interfaccia, leggerezza e velocità, in linea con le risorse hardware dei dispositivi. L'applicazione della presente tesi è stata sviluppata per smartphone e tablet, la cui continua evoluzione li ha portati a diventare uno strumento sostituivo, e a volte più efficace, di altre tecnologie già esistenti. Vengono in mente esempi banali, come la calcolatrice, il navigatore satellitare o la

macchinetta fotografica.

Nel nostro caso si è scelto di creare un'app per il sistema operativo Android, la cui natura open source e i bassi costi di produzione lo rendono un candidato ideale per i nostri obiettivi. Considerando che i dispositivi aventi Android come sistema operativo sono al momento i più diffusi al mondo, si è presentata, con il suo utilizzo, la possibilità di raggiungere il maggior numero possibile di potenziali utenti.

L'argomento trattato riguarda, in generale, gli eventi. Ci è mai capitato di dimenticarvi il compleanno di qualcuno o, più nello specifico, un anniversario? Da questa domanda è venuta fuori l'idea di progettare un'app che non solo ricordasse, tramite semplici notifiche, ricorrenze come compleanni, anniversari e onomastici, ma creasse eventi con data e luogo della festa. Anche in questo caso, l'evoluzione tecnologica ha dato la possibilità di sostituire il segnare le ricorrenze in una classica agenda con una piu pratica e compatta app in grado di svolgere le stesse funzionalità in modo pratico e veloce.

La tesi è strutturata come di seguito specificato:

- Nel primo capitolo verrà proposta una panoramica generale sulla storia della tecnologia. Verranno introdotti, poi, i sistemi operativi, in particolar modo quelli per dispositivi mobili, con un focus su Android e tutte le sue funzionalità.
- Nel secondo capitolo verrà descritta l'analisi dei requisiti, sia funzionali che non, dell'applicazione in questione, chiamata *Eventi*. È presente, anche, una trattazione della piattaforma di sviluppo utilizzata, ovvero Android Studio.
- Nel terzo capitolo verranno analizzati i componenti grazie ai quali è avvenuta la progettazione, come il layout e le viste. Inoltre, verranno illustrate la progettazione concettuale e quella logica, nonchè i mockup relativi all'app.
- Nel quarto capitolo verrà illustrata, invece, l'implementazione delle classi, descrivendo in dettaglio il codice corrispondente.
- Nel quinto capitolo, infine, verranno tratte le conclusioni, verranno esaminati i punti di forza e di debolezza dell'app realizzata e verranno delineati alcuni possibili sviluppi futuri.

1

Introduzione ad Android

In questo capitolo daremo un'occhiata allo scenario tecnologico di riferimento; in particolare, parleremo dei sistemi operativi per dispositivi fissi e mobili e analizzeremo ampiamente Android sotto tutti i punti di vista.

1.1 Storia della tecnologia

Il secolo scorso è stato caratterizzato dall'avvento di tecnologie che permettono la trasmissione di suoni ed immagini, tecnologie inimmaginabili per l'uomo di qualche secolo fa, ma che ora sono diventate strumento quotidiano per ciascuno di noi.

Quando pensiamo alla tecnologia pensiamo, probabilmente, ai più moderni prodotti del mercato industriale: computer sempre più potenti, versatili e leggeri; automobili più sofisticate; strumenti per riprodurre con elevata qualità suono e immagini. In realtà, questi sono più che altro i risultati dello sviluppo di nuove tecnologie, mentre la parola "tecnologia" ha un significato più ampio.

Con il termine "tecnologia", infatti, si indica, più che l'insieme di singoli oggetti, lo sviluppo di strumenti o di macchine con cui si è risolto un problema o è stato migliorato un aspetto della nostra vita quotidiana (Figura 1.1). In questo senso anche lo sviluppo dei primissimi utensili nella preistoria, dalla prima arma creata dal legno alla prima ruota, rappresenta un progresso tecnologico.

Oggi con la parola "tecnologia" intendiamo più in generale, l'utilizzo combinato di diverse discipline adottate per rendere quanto più efficiente ed economica possibile la produzione di nuovi beni e strumenti. Questa ultima definizione descrive, in particolare, il ruolo che la tecnologia ha nelle attuali società evolute e ci fa capire perché si può parlare anche di tecnologie in campi nei quali non c'è alcuno sviluppo di prodotti materiali, ma solo di procedure, come, ad esempio, nel campo informatico, la produzione di software, un prodotto immateriale.

Ed è qui che entrano in gioco macchine che utilizziamo quotidianamente, come, ad esempio, i personal computer. I computer vennero inventati per "computare", cioè per "risolvere problemi matematici complessi", come il dizionario definisce tuttora quella parola. Fu proprio per risolvere equazioni che venne inventato il computer elettronico digitale. La parola "computer" nella lingua inglese, in origine, definiva

Figura 1.1. Evoluzione della tecnologia

una persona che risolveva equazioni. Fu soltanto intorno al 1945 che il nome venne esteso alle macchine.

Per il primo microprocessore, però, dobbiamo aspettare fino al 1975 quando la Intel produsse l'Intel 4004 (Figura 1.2). Questo processore, abbinato ad altri dispositivi, ha aperto la strada dell'informatica ad appassionati anche con pochi soldi.

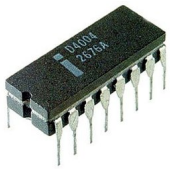

Figura 1.2. Microprocessore Intel 4004

Un altro snodo fondamentale è rappresentato dall'annuncio del primo personal computer nel 1981 da parte di IBM, un prodotto che non aveva nulla di geniale; era, in fondo, un assemblato di parti esistenti, quali la CPU Intel. Questo segnava l'inizio di una rivoluzione secolare per il computer.

Nel 1984 Apple annuncia il personal computer Macintosh. Sony e Philips introducono i primi CD-ROM, che forniscono una enorme capacità di registrazione dei dati (fino a 640 MB).

Nel 1990 nasce l'Internet dei nostri giorni: Berners Lee scrive il prototipo iniziale per il WWW, che usa le altre sue creazioni: URL, HTML e HTTP; nel 1995 la Microsoft lancia Windows 95 (Figura 1.3) che include il browser Internet Explorer.

Dal 2000 al 2010 Internet è ovunque attraverso connessioni via cavo, satellite o Wi-Fi: da ogni dispositivo è possibile accedere ad Internet. I telefoni cellulari si sono trasformati in dei mini PC tascabili connessi permanentemente alla rete e capaci di eseguire video in tempo reale. È anche il periodo in cui Apple ritorna alla ribalta creando prodotti rivoluzionari come l'iPhone e l'iPad, basati su un'interfaccia di estrema qualità e usabilità, utilizzando un sistema operativo mobile nuovo e innovativo chiamato iOS. A questo si contrappone Google che lancerà il sistema operativo concorrente, chiamato Android.

Figura 1.3. Sistema operativo Windows 95

1.2 Sistemi operativi

Il software può essere diviso un due grandi classi:

- i programmi di sistema, che gestiscono le operazioni del sistema di elaborazione;
- i programmi applicativi, che risolvono i problemi dei loro utilizzatori.

L'insieme dei programmi di sistema viene comunemente identificato con il nome di sistema operativo (SO).

Lo scopo del SO è quello di gestire le risorse del sistema di elaborazione e rendere agevole l'interfaccia tra l'uomo e la macchina. Si occupa, dunque, di gestire la memoria di massa, la memoria RAM e i processi all'interno del dispositivo. I SO sono generalmente costituiti da un insieme di moduli, ciascuno dedicato a svolgere una determinata funzione. I vari moduli del SO interagiscono tra di loro secondo regole precise al fine di realizzare le funzionalità di base dalla macchina.

Le due principali macrocategorie di sistemi operativi (Figura 1.4) sono:

- SO per dispositivi fissi, come personal computer;
- SO per dispositivi mobili, come smartphone e tablet.

Figura 1.4. Dispositivi fissi e mobili

1.2.1 Sistemi operativi per dispositivi fissi

Abbiamo detto che un sistema operativo è un software che gestisce le risorse hardware e software della macchina, fornendo servizi di base ai software applicativi. Tra

i sistemi operativi per computer desktop (Figura 1.5) si citano macOS, Microsoft Windows, le distribuzioni GNU/Linux, sistemi Unix-like e Chrome OS.

La forza dominante rimane Windows, che si mantiene al 90,61% del mercato a livello mondiale. Il sistema operativo di Microsoft detiene una componente molto salda sul mercato, con Windows 10 che cresce al 29%, Windows 7 che scende al 47,21% e sia Windows XP che Windows 8.1 che vanno al di sotto del 6%.

MacOS possiede, invece, una fetta decisamente più bassa: circa il 6,35% degli utenti. Il dato non è un massimo storico (era al 9,57% nell'aprile 2016) e si pensa sia in calo a causa delle politiche di Apple, più incentrate sui dispositivi mobili.

Discorso differente, invece, per Linux. Il sistema del Pinguino è stato storicamente relegato ad una posizione di netta minoranza, spesso sotto il punto percentuale. Però, negli ultimi anni, ha conosciuto una rinascita che l'ha portato, secondo alcune stime, sopra il 2% a livello mondiale.

Il dato più importante, però, non riguarda l'utilizzo dei sistemi operativi, ma l'utilizzo dei computer in generale, il quale ha segnato un calo negli ultimi anni che andrà sempre più ad aumentare, secondo gli analisti, favorendo il mercato dei dispositivi mobili, dei quali parleremo nella prossima sottosezione.

Figura 1.5. Loghi di Linux, MacOS e Windows

1.2.2 Sistemi operativi per dispositivi mobili

Negli smartphone, il sistema operativo è il software preinstallato che gestisce tutte le funzioni che lo smartphone è in grado di svolgere.

Il market share riguardante i sistemi operativi mobili ha subito una forte scossa negli ultimi anni. Abbiamo assistito alla nascita di nuovi SO, alla morte di altri e alla lunga strada verso la prima posizione percorsa da Android, il più utilizzato sistema operativo per smartphone degli ultimi sette anni.

Nel corso degli anni i sistemi operativi più diffusi sono stati cinque: Android, Apple iOS, Symbian OS, Windows Mobile, Blackberry OS. I due colossi, come tutti sappiamo, ad oggi, sono i primi due (Figura 1.6).

Il sistema operativo mobile di Google (Android) è installato su decine di migliaia di modelli di smartphone. La concorrenza di Apple (iOS) non lo può scalfire. Gli ultimi dati sui sistemi operativi per smartphone rivelano che, recentemente, la

concorrenza all'interno del mondo Android ha continuato ad intensificarsi raggiungendo l'88% circa del mercato mondiale, mentre il rimanente è quasi tutto di Apple, circa 11,9%, con uno 0,1% relativo ad altri sistemi operativi.

Sfida sproporzionata quindi, visto che il sistema operativo della Apple gira solo sui telefonini della Apple. La scelta di Google, invece, è stata quella di puntare su un SO open source molto flessibile, pressoché gratuito e facilmente personalizzabile.

Figura 1.6. Loghi di iOS e Android

1.3 Android

1.3.1 Origine di Android

La strada per il successo, però, non è mai facile; infatti Android non era così amato ed utilizzato ai suoi inizi.

Partiamo dalle origini, nel 2009, dove Symbian (Figura 1.7) governava il mercato dei telefoni cellulari, con una percentuale di presenza del 48,8%. A fargli concorrenza, erano presenti RIM (Blackberry) col 20,6% del market share, iOS, nato nel 2007 col 10,5%, a ruota Windows Mobile 6 col 10,2% e, infine, l'appena nato Android, nella sua prima versione, all'1,6%. Ma se c'è qualcosa che l'informatica ci ha insegnato nel corso della sua storia è che niente di ciò che viene amato dura per sempre.

Figura 1.7. Logo del sitema operativo Symbian

Il primo grande scossone in questo mercato è arrivato, paradossalmente, da Bada, un sistema operativo ideato da Samsung per diversi suoi dispositivi (Figura 1.8). Seppur questo abbia avuto una vita decisamente breve, è servito come punto di

svolta in questo mercato, proponendo qualcosa di diverso che potesse invogliare le persone a utilizzarlo. Bada, nato nel 2010, riuscì subito a sottrarre terreno a Symbian, raggiungendo lo 0,9% di market share e ferendolo. Il capolista, anche a causa della fantastica galoppata di Android che lo aveva portato al 17,2%, iniziò a perdere terreno scendendo al 40,91%: sempre primo, ma con qualche difficoltà in più.

Figura 1.8. Logo del sitema operativo Bada

Il picco massimo mai raggiunto dal sistema operativo di Samsung è stato nel secondo quarto del 2012, col 2,7% del market share. In questo periodo, al contrario dei precedenti, i dispositivi mobili più venduti al mondo possedevano il sistema operativo Android, che raggiunse il 64,2%. Symbian si avviava all'estinzione con un market share pari al 5,9%, mentre Microsoft, Blackberry e Apple restavano mediamente costanti. La fine economica di Bada avvenne poi nel 2013, mentre quella di Symbian fu nel 2014, a causa di un Android ormai all'80,8%.

La situazione nei successivi anni fu più o meno stabile per Android e iOS, mentre pessima per Blackberry e Microsoft, i cui sistemi operativi si avviarono verso la fine. Windows Mobile e Blackberry OS sparirono a tutti gli effetti dal mercato a partire dal 2017 dando inizio al duopolio Android-iOS. Ad oggi, troviamo Android dominatore del market share, mentre iOS al secondo e ultimo posto.

1.3.2 Architettura di Android

Android comprende tutto lo stack degli strumenti per la creazione di applicazioni, tra cui un sistema operativo, un insieme di librerie native per le funzionalità core della piattaforma, un'implementazione della macchina virtuale (VM) e un insieme di librerie Java.

L'architettura di Android è un'architettura a strati dove i livelli inferiori offrono servizi ai livelli superiori, permettendo un più alto grado di astrazione (Figura 1.9).

Analizziamo i livelli partendo dal più basso fino ad arrivare al più alto:

Il kernel di Linux

Il layer di più basso livello è rappresentato dal kernel Linux (Figura 1.10). La necessità era, infatti, quella di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante attraverso la definizione di diversi driver, il cui nome è completamente esplicativo.

In particolare, possiamo notare la presenza di driver per la gestione delle periferiche multimediali, del display, della connessione Wi-Fi e dell'alimentazione.

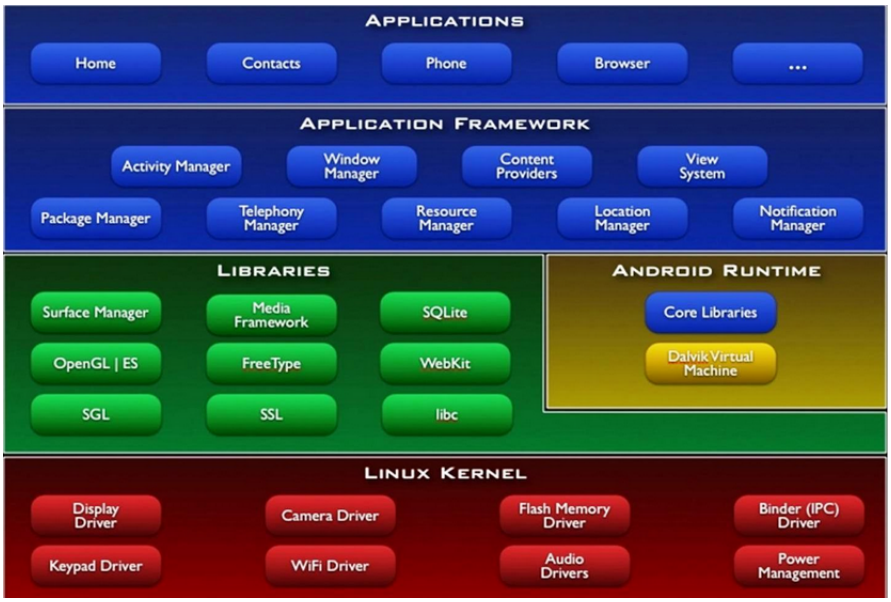

Figura 1.9. Architettura Android

Figura 1.10. Il kernel di Linux

È da notare, anche, la presenza di un driver dedicato alla gestione della comunicazione tra processi diversi (IPC). La sua importanza è fondamentale per far comunicare componenti diversi in un ambiente in cui ciascuna applicazione viene eseguita all'interno di un proprio processo.

La scelta verso l'utilizzo di un kernel Linux è stata la conseguenza della necessità di avere un SO che fornisse tutte le caratteristiche di sicurezza, gestione della memoria, gestione dei processi, power management e che fosse affidabile e testato.

Librerie native

Al di sopra del kernel di Linux abbiamo un livello che contiene un insieme di librerie native realizzate in C e C++, che rappresentano il core vero e proprio di Android (Figura 1.11).

Si tratta di librerie che fanno riferimento ad un insieme di progetti open source; di seguito vengono elencate le più importanti:

- *Surface Manager (SM)*: ha la responsabilità di gestire le view, ovvero ciò da cui un'interfaccia grafica è composta.

Figura 1.11. Le librerie native

Il compito di SM è quello di prendere le diverse finestre e di disegnarle sul buffer da visualizzare, attraverso la tecnica del double buffering. In questo modo non si avranno finestre che si accavallano in modo scoordinato sul display. SM permetterà, inoltre, la visualizzazione contemporanea di grafica 2D e 3D dalle diverse applicazioni.

- *Open GL ES*: è la libreria utilizzata per la grafica 3D, la quale permette di accedere alle funzionalità di un eventuale acceleratore grafico hardware. Si tratta di una versione ridotta di OpenGL, specializzata per dispositivi mobili.
  Essa consiste in un insieme di API multipiattaforma che forniscono l'accesso a funzionalità 2D e 3D in dispositivi embedded. Come avvenuto per le J2ME di Java, anche in relazione alla grafica su dispositivi con risorse relativamente ridotte, si è deciso di creare un sottoinsieme delle API dell'OpenGL, da cui, appunto, l'OpenGL ES.
- *Scalable Graphics Library (SGL)*: è una libreria in C++ che, insieme alle OpenGL, costituisce il motore grafico di Android. Mentre per la grafica 3D ci si appoggia all'Open GL, per quella 2D viene utilizzato un motore ottimizzato chiamato, appunto, SGL.
  Si tratta di una libreria utilizzata principalmente dal Window Manager e dal Surface Manager all'interno del processo di renderizzazione grafica.
- *Media Framework*: componente in grado di gestire i diversi CODEC per i vari formati di acquisizione e riproduzione audio e video.
- *FreeType*: serve per gestire i font. Si è deciso di utilizzare questo motore perché è di piccole dimensioni, molto efficiente, altamente customizzabile e, soprattutto, portabile. Attraverso FreeType, le applicazioni di Android saranno in grado di visualizzare immagini di alta qualità.
- *SQLite*: è una libreria in-process che implementa un DBMS relazionale caratterizzato dal fatto di essere molto compatto, diretto, di non necessitare alcuna configurazione e, soprattutto, transazionale.
- *WebKit*: si tratta di un browser engine open source basato sulle tecnologie HTML, CSS, JavaScript e DOM. Un aspetto da sottolineare è che WebKit non è un browser, ma un browser engine; quindi, andrà integrato in diversi tipi di applicazioni.
- *SSL*: è una libreria per la gestione dei Secure Socket Layer. Anche gli aspetti legati alla sicurezza non potevano, di certo, essere trascurati da Android.
- Libc: è un'implementazione della libreria standard C `libc` ottimizzata per i dispositivi basati su Linux embedded, come Android.

Application Framework

Tutte le librerie viste finora vengono, poi, utilizzate da un insieme di componenti di più alto livello che costituiscono l'Application Framework (AF) (Figura 1.12). Si tratta di un insieme di API e componenti per l'esecuzione di funzionalità ben precise e di fondamentale importanza in ciascuna applicazione Android.

Tutte le applicazioni per Android utilizzano lo stesso AF e, come tali, possono essere estese, modificate o sostituite. Da qui il motto che possiamo trovare sul sito di Android, ovvero: "All applications are equals".

Figura 1.12. Application Framework

Activity Manager

Quando inizieremo la progettazione e lo sviluppo delle applicazioni per Android, vedremo come assuma fondamentale importanza il concetto di Activity.

Si tratta di un qualcosa che possiamo associare inizialmente ad una schermata; essa, quindi, permette non solo la visualizzazione o la raccolta di informazioni ma, in modo più generico, è lo strumento fondamentale attraverso il quale l'utente interagisce con l'applicazione.

Comprendere a fondo, pertanto, il funzionamento di questo componente è fondamentale per riuscire a realizzare delle applicazioni.

Package Manager

Un aspetto non trascurabile di un sistema come Android è la gestione del processo di installazione delle applicazioni nei dispositivi.

Come vedremo, ciascuna applicazione dovrà fornire, al dispositivo che la dovrà eseguire, un determinato insieme di informazioni che descriveremo attraverso un opportuno file XML di configurazione. Tale file prende il nome di `AndroidManifest`.

Si tratta di informazioni di vario genere, per esempio, relative agli aspetti grafici (il layout) dell'applicazione, alle diverse Activity, o ad aspetti di sicurezza. La responsabilità del Package Manager sarà, quindi, di gestire il ciclo di vita delle applicazioni nei dispositivi.

Window Manager

Un componente molto importante è il Window Manager, che permette di gestire le finestre delle diverse applicazioni, gestite da processi diversi, sullo schermo del dispositivo.

Telephony Manager

Il Telephony Manager permette una maggiore interazione con le funzionalità caratteristiche di un telefono, come la semplice possibilità di iniziare una chiamata o di verificare lo stato della chiamata stessa.

Content Provider

Il Content Provider (CP) è un componente fondamentale nella realizzazione delle applicazioni per Android, poiché ha la responsabilità di gestire la condivisione di informazioni tra i vari processi.

Il funzionamento è simile a quello di un repository condiviso con cui le diverse applicazioni possono interagire inserendo o leggendo informazioni.

Resource Manager

Come vedremo nel dettaglio, un'applicazione è composta, oltre che da codice, anche da un insieme di file di tipo diverso, per esempio immagini, file di configurazione o di properties per la internazionalizzazione (I18N), file di definizione del layout, e così via. La responsabilità di gestire questo tipo di informazioni è stata affidata al Resource Manager, che metterà a disposizione una serie di API di semplice utilizzo. Si tratta di un componente con responsabilità di ottimizzazione delle risorse.

View System

La gestione della renderizzazione dei componenti, nonché della gestione degli eventi associati, è di responsabilità di un componente che si chiama View System (VS).

Location Manager

Le applicazioni che gestiscono, tra le informazioni disponibili, quelle relative alla localizzazione si chiamano Location Based Application (LBA) e possono essere realizzate utilizzando le API messe a disposizione dal Location Manager.

Notification Manager

Un altro servizio molto utile è quello fornito dal Notification Manager, che mette a disposizione un insieme di strumenti che l'applicazione può utilizzare per inviare una particolare notifica al dispositivo, il quale la dovrà presentare all'utente con i meccanismi che conosce. La notifica potrà essere un segnale attraverso la vibrazione, un LED, o un'icona.

1.3.3 Differenti versioni di Android

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 fra tutti i nomi delle versioni Android è l'essere il nome di un dolce. Questo, però, a partire da Android 1.5. Cos'è successo prima di quella numerazione e perchè iniziare dalla lettera C?

In rete ci sono molte versioni riguardanti i nomi delle prime versioni di Android non presenti nella lista seguente. Invece, niente è cambiato a partire dalla versione di Android 1.5, dove la numerazione è stata accompagnata dall'ordine alfabetico con nomi di dolci:

- Android 1.5 - Cupcake (2009);
- Android 1.6 - Donut (2009);
- Android 2.0 / 2.1 - Eclair (2009);
- Android 2.2 - Froyo (2010);
- Android 2.3 - Gingerbread (2010);
- Android 3.0 - Honeycomb (2011);
- Android 4.0 - Ice Cream Sandwich (2011);
- Android 4.1 / 4.2 / 4.3 - Jelly Bean (2012);
- Android 4.4 - KitKat (2013);
- Android 5.0 / 5.1 - Lollipop (2014);
- Android 6.0 - Marshmallow (2015);
- Android 7.0 / 7.1 - Nougat (2016);
- Android 8.0 / 8.1 - Oreo (2017);
- Android 9.0 - Pie (2018) (Figura 1.12).

Figura 1.13. Logo di Android Pie (2018)

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.

Diverse fonti riportano che Android 1.0 si chiamasse Astro e la Versione 1.1 Bender. Così, anche a causa di codici numerici confusi, si decise di ricorrere a una catalogazione progressiva alfabetica (in uso ancora ora), iniziando dalla lettera C,

in quanto terza release ufficiale di Android. Non è stato ancora, però, svelato quale fossero i veri nomi delle versioni pre-1.5.

Per la versione successiva a Marshmallow, Google ha, poi, scelto di lasciare che fossero gli utenti stessi a scegliere quale fosse il nuovo nome di Android. Android N è stato poi svelato il 30 giugno 2016 e il suo nome ufficiale è Android Nougat (ovvero mandorlato/torrone). Il 21 agosto 2017 è stato, invece, svelato Android 8.0 Oreo. Si tratta del famoso biscotto americano. Nel 2018 Google si è trattenuta dal voler giocare con i propri utenti e, senza grande sorpresa, ha annunciato Android 9.0 Pie.

1.3.4 Fattori di successo

Sicuramente uno dei punti chiave del successo di Android è il fatto che dietro ci sia un'azienda come Google. In teoria è una piattaforma open source basata su Linux, ma Linux non ha avuto un grande impatto sul mercato dei personal computer, quindi, dal punto di vista dell'immagine, non ha contribuito molto a rendere famoso Android. Infatti la gente che compra un telefono Android non pensa a Linux, ma pensa a Google.

Inoltre, essendo open source, il prodotto può essere trasformato e modificato in molti modi diversi. Questo permette ai programmatori e ai diversi produttori di smartphone e tablet di modificarlo liberamente secondo le proprie necessità, che vanno via via cambiando con il tempo.

Android è stato progettato principalmente per smartphone e tablet, ma il carattere aperto e personalizzabile del sistema operativo ne consente l'utilizzo anche su altri strumenti elettronici, tra cui portatili, netbook, ebook reader, smart watch, smart TV, autoradio e, addirittura, occhiali (Google Glass).

Un altro fattore chiave risiede nel fatto che è basato sul concetto di manipolazione diretta "utente-dispositivo", per cui bastano semplici tocchi sul display per interagire con le applicazioni. La risposta agli input dell'utente è stata progettata per essere immediata e fornisce un'interfaccia fluida. Il tutto per semplificare al massimo l'utilizzo da parte del singolo.

L'assenza di un numero adeguato di alternative nel mercato della telefonia è una cosa difficile da vedere sotto l'aspetto positivo. Meno concorrenza significa, anche, meno proposta e meno impegno da parte del produttore nello sviluppare qualcosa di nuovo e importante per gli utenti. Come detto, Android è riuscito a proporre qualcosa di nuovo portando a sé utenti da altri sistemi operativi, mentre iOS ha visto continuamente un via vai delle stesse persone.

Fino a che non ci sarà un nuovo terzo concorrente che genererà un terremoto in questo mercato, difficilmente vedremo novità rivoluzionarie da parte di entrambe le aziende.

1.4 Applicazioni Android

Le applicazioni (o app) sono la forma più generica per indicare i software applicativi installabili su Android.

Per motivi di sicurezza informatica, le applicazioni sono fornite di un sistema di certificazione che verifica l'integrità del pacchetto da installare e la paternità rispetto a un distributore di software accreditato presso Google.

Android è fornito di una serie di applicazioni preinstallate che vanno dal browser alla radio analogica FM, dal calendario all'applicazione Gmail, dalla calcolatrice al navigatore satellitare, e comprende anche la ricerca vocale.

In media, ogni persona ha nel proprio smartphone circa 80 applicazioni, anche se poi non tutte vengono regolarmente usate. La maggior parte, però, sono davvero comode e permettono di rendere la vita più semplice consentendo, per esempio, di fare la spesa direttamente dal proprio cellulare, di consultare in un attimo il meteo, di fare shopping dal divano con prodotti che arrivano comodamente a casa. E si tratta di un mercato che, come indicato dalle statistiche, è in netta crescita. Come ben noto per i tantissimi utenti, i due store più grandi fanno riferimento ai due principali sistemi operativi più diffusi, App Store di Apple e Google Play di Google.

1.4.1 Store

Lo store ufficiale di Android è Google Play (Figura 1.14). Il nome attuale è stato adottato a partire dal 6 marzo 2012 mentre la denominazione precedente era Android Market. Il servizio offre la possibilità di acquistare non solo applicazioni, ma, dal 2012, anche libri e musica, rispettivamente Play Libri e Play Musica. Dal 2013 anche film, e più tardi anche riviste.

Figura 1.14. Logo di Google Play

Tutti o quasi i dispositivi Android hanno preinstallata una icona denominata Google Play. Per accedere a Google Play è necessario possedere un account Google. Le applicazioni possono essere scaricate sia dal catalogo ufficiale Google Play sia da altri cataloghi, come l'Amazon Appstore di Amazon.com, o F-Droid, che contiene solo software su licenza libera. In alternativa, le app possono essere installate direttamente a partire da un file APK fornito dal distributore del software.

Attualmente il numero di applicazioni disponibili su questo store sfiora i 4 milioni e, come già detto, è un mercato in crescita; basti pensare che ogni mese vengono aggiunte circa 50 mila app.

1.5 Perchè utilizzare Android

Come detto ampiamente, sono molti i fattori positivi che mettono in risalto l'utilizzo di Android da parte degli utenti, a partire dal fatto che è open source, passando dalla semplicità e fluidità dell'interfaccia ed arrivando alla mancanza di alternative valide, soprattutto in certe aree di prezzo.

Infatti, cosa che fino ad ora non è stata discussa, Android, oltre ad essere una potenza per i top di gamma, riesce a sbaragliare totalmente la concorrenza per gli smartphone di fascia medio-bassa. Per quanto riguarda il passaggio da iOS ad Android, la maggior parte degli utenti ha effettuato il cambio per una questione di prezzo. In commercio ci sono diversi terminali Android che costano molto meno di un nuovo iPhone, e questo è un fattore chiave nella decisone su quale telefono acquistare e, implicitamente, quale sistema operativo adottare.

2

Analisi dei requisiti

In questo capitolo saranno illustrate la componente dati e i requisiti, funzionali e non, per l'implementazione dell'applicazione Eventi.

2.1 Introduzione all'applicazione

La programmazione Android e i suoi sviluppatori stanno avendo negli ultimi anni un grandissimo successo.

Infatti, con i suoi 1.4 miliardi di installazioni mensili attive, Android è il sistema operativo più diffuso per dispositivi mobili. E con un numero così elevato di installazioni, l'attività di sviluppo delle app è un'opportunità divertente, stimolante e redditizia. Per sviluppare applicazioni Android è necessario possedere nozioni di programmazione Java e familiarità con il linguaggio XML.

Prima di vedere la progettazione di un'app con il relativo ambiente di sviluppo e tutti i suoi strumenti, concentrimoci sull'analisi dei requisiti dell'applicazione in questione. Abbiamo scelto per tale applicazione il nome "*Eventi*".

L'idea dell'app *Eventi* nasce dal fatto che ciascuno di noi nella vita si dimentica spesso di qualcosa: che sia una cosa semplice, come una faccenda, oppure un oggetto dimenticato da qualche parte, o magari un augurio non fatto ad una persona a cui si tiene molto.

Ed è proprio quest'ultimo motivo che ci ha spinto a voler creare quest'applicazione.

2.2 Raccolta informazioni

Quando si ha una nuova idea è meglio metterla in pratica al più presto per evitare che qualcuno lo faccia prima di noi. Ma ancor prima di attuarla è bene prendersi qualche momento per assicurarsi che possa essere competitiva sul mercato.

A volte capita che, purtroppo, facendo le cose di fretta anche la migliore delle idee venga rovinata. Che si voglia sviluppare un'applicazione su piattaforma mobile, desktop o entrambe, la prima cosa da fare è partire da alcune buone prassi. Una di questa è, sicuramente, l'analisi dei requisiti.

2.2.1 Studio fattibilità

Una delle prime cose da fare è capire se lo sviluppo dell'applicazione è profittevole: lo studio della fattibilità aiuta a comprendere se l'app renderà partendo dalla definizione delle spese e dagli sforzi tecnici che gli sviluppatori dovranno sostenere. Successivamente dovrà essere affrontata un'analisi del mercato; bisognerà capire, infatti, quali sono i competitor (Figura 2.1), come agiscono online e offline, quali sono gli elementi con cui differenziarsi. In secondo luogo si dovrà identificare un target ben preciso, al quale l'applicazione mirerà per soddisfare certe specifiche.

Figura 2.1. Idee concorrenti

È ovvio che un'app semplice come *Eventi* non ha un entourage dietro e non ha un team di sviluppatori, ma, nel suo piccolo, può essere comunque realizzata applicando i criteri definiti dall'Ingegneria del Software. Infatti, a nostro avviso, la richiesta da parte degli utenti di avere un calendario di ricorrenze per non dimenticarsi gli eventi è in aumento, e di competitor sul Play Store o sull'App Store ne sono presenti abbastanza, ma non un numero così elevato come per altre app, ad esempio per quelle di messaggistica. Inoltre, l'idea di poter mandare inviti per un evento creato direttamente dall'applicazione *Eventi* è una caratteristica che la fa differenziare dalla concorrenza. Ma di questa funzionalità tratteremo in dettaglio nel seguito.

2.2.2 Tipologia di applicazione

Individuato il target, occorre identificare la tipologia di applicazione più adatta a questo tipo di pubblico che, quindi, sia in grado di soddisfarlo. Si tratta di decidere se svilupperemo un software per app native, web app o app ibride (Figura 2.2).

Figura 2.2. Loghi per tipo di applicazione

Nel nostro caso si è optato per creare un'applicazione nativa sfruttando diversi vantaggi, tra i quali:

- Le app native non necessitano obbligatoriamente di Internet per funzionare, il che costituisce certamente un vantaggio. Nonostante siamo nel 2019, esistono ancora zone poco coperte dalla rete Internet, e permettere agli utenti di accedere all'app senza connessione è un grosso punto di forza da non sottovalutare.
- Le notifiche push sono possibili solo nel caso di applicazioni native. Tali notifiche permettono di avvisare gli utenti e di attirare la loro attenzione ogni volta che lo desideriamo.
- Le app native garantiscono maggiore velocità, affidabilità e migliore reattività, oltre che una risoluzione superiore che assicura un'esperienza migliore all'utente.
- Le app native permettono un accesso più facile a tutte le funzionalità del telefono.

2.3 Descrizione della componente dati

Alla creazione dell'applicazione la componente dati dell'app *Eventi* è vuota ed è costituita soltanto dalla corrispettiva struttura.

Infatti è possibile aggiungere i contatti, da cui poi deriveranno gli eventi, soltanto durante il runtime dell'applicazione. Ovviamente solo allora la componente dati si popolerà delle informazioni immesse dall'utente.

Per quanto riguarda la navigazione all'interno delle schermate, si deve tener traccia dei dati collegati ad un contatto, che verranno visualizzati ed elaborati a seconda delle esigenze.

2.4 Requisiti funzionali

I requisiti funzionali descrivono le interazioni tra il sistema e il suo ambiente indipendentemente dalla sua implementazione. L'ambiente include l'utente e ogni altro sistema esterno.

2.4.1 Caricamento della lista di contatti

Una volta caricate nella memoria del telefono informazioni riguardanti diversi contatti, questi potranno essere visualizzati nella schermata principale tramite una lista scorrevole. La lista è ordinata alfabeticamente e dovrà essere composta, ovviamente, da almeno un contatto.

2.4.2 Aggiunta di un contatto

Grazie ad una schermata apposita si potrà popolare il database di contatti, completi di nome, cognome, numero di telefono, data di nascita e anniversario.

2.4.3 Vista di un contatto

È chiaro che ogni contatto può essere visualizzato con tutti i suoi campi tramite un tap su di esso.

2.4.4 Rimozione di un contatto

Gestire la lista dovrà essere semplice; quindi, l'aggiunta di un pulsante con cui poter rimuovere un elemento dalla lista è fondamentale.

2.4.5 Creazione evento

Nel caso in cui si voglia creare una ricorrenza è sufficiente selezionare gli invitati per un determinato evento, scrivere data, luogo ed ora, e gli inviti verrano spediti tramite l'applicazione relativa ai messaggi presente nello smartphone dell'utente.

2.4.6 Gestione delle notifiche

Le ricorrenze potrebbero voler essere visualizzate dal possessore dell'applicazione in un certo orario. Così, tramite la schermata "impostazioni", è possibile attivare e disattivare le notifiche e, nel caso in cui esse fossero attive, impostare un orario di ricezione.

2.5 Requisiti non funzionali

I requisiti non funzionali descrivono gli aspetti del sistema che non sono direttamente legati al comportamento, cioè alle funzionalità, del sistema. Essi includono una grande varietà di richieste che si riferiscono a diversi aspetti del sistema, ad esempio alle performance.

2.5.1 Affidabilità

L'affidabilità è la capacità di un sistema o di una componente di fornire la funzione richiesta sotto determinate condizioni. La persona che utilizzerà questa applicazione vuole che faccia esattamente quello che gli viene richiesto, per cui sarà necessario che sia progettata in modo affidabile.

2.5.2 Usabilità

Il sistema deve essere facile da usare per controllori esperti, e deve essere tale da minimizzare gli errori degli utenti. Per cui, se si volessero usare certi campi di un contatto che non sono stati riempiti, il sistema non andrà in crash ma verrà visualizzato un messaggio di errore.

Inoltre, l'interfaccia deve essere semplice e intuitiva nonostante le diverse funzionalità.

2.5.3 Prestazioni

Il progetto è indicato per ogni tipo di persona, per cui l'utilizzo su diversi dispositivi deve essere un obiettivo. Di conseguenza, le prestazioni su telefoni di qualunque fascia con installate versioni differenti di Android devono essere accettabili.

2.6 Strumenti per la programmazione Android

2.6.1 Il linguaggio di programmazione Java

Come già accennato in precedenza, nella programmazione di app Android bisogna avere una infarinatura sul linguaggio di programmazione Java.

Java (Figura 2.3) è il linguaggio di programmazione più popolare al mondo ed è anche il più usato nel mondo del lavoro; infatti è in prima posizione come richiesta nel settore informatico. Viene usato praticamente ovunque: sviluppo web, mobile, software. Ha una comunità di sviluppatori enorme e questo si traduce in tante risorse di qualità sia a pagamento che gratuite, come il programma Android Studio, introdotto nella prossima sottosezione.

Figura 2.3. Logo del linguaggio Java

2.6.2 Android Studio

Il migliore strumento per creare app Android tramite il linguaggio Java, al giorno d'oggi, è ormai divenuto Android Studio. Si tratta dell'ambiente di sviluppo integrato (IDE) ufficiale di Google ed è ormai giunto alla Versione 3.4.

Android Studio (Figura 2.4) sostituisce il precedente software basato su Eclipse e porta la creazione di app Android a un livello nettamente superiore.

Basato sull'editor IntelliJ IDEA, ancora oggi piuttosto popolare nel mondo Java, Android Studio è completamente gratuito e si potrà essere sin da subito operativi. Grazie a IntelliJ si può trarre vantaggio dall'autocomposizione: verrà automaticamente suggerita la sintassi Java da usare, così da evitare di scrivere inutilmente tutte le funzioni o i riferimenti.

Le ultime versioni di Android Studio, tra l'altro, integrano anche il pacchetto JDK (Java Development Kit) cosicché non si debba neppure scaricarlo e installarlo in proprio. Il JDK è l'insieme degli strumenti per sviluppare programmi da parte dei programmatori Java.

Figura 2.4. Logo di Android Studio

3

Progettazione

Questo capitolo è dedicato, inizialmente, a tutte le funzionalità che Android Studio offre, partendo dalle componenti, passando per l'interfaccia e arrivando, infine, alla gestione dei database. Successivamente, si parlerà della progettazione concettuale e logica nonchè della progettazione delle funzionalità dell'applicazione Eventi.

3.1 Componenti principali per la progettazione

È possibile identificare quattro componenti principali che caratterizzano un'applicazione per il sistema operativo Android. Ciascun componente è un punto diverso attraverso cui il sistema può interagire con l'applicazione. Non tutti, però, sono punti di accesso reali per l'utente, ma ciascuno di essi possiede una propria entità e svolge un determinato ruolo, contribuendo a definire il comportamento generale dell'applicazione.

Essi sono: le Activity, i Service, i Content Provider e i Broadcast Receiver.

3.1.1 Le Activity

Un'Activity in Android è essenzialmente una finestra che contiene l'interfaccia utente di un'applicazione; il suo scopo è quello di permettere un'interazione con gli utenti.

Un'applicazione Android può avere zero o più activity, anche se, solitamente, almeno una è presente.

Per creare un'Activity è necessario fare due cose, ovvero:

- estendere la classe `Activity`, appartenente al framework Android;
- registrare l'Activity nell'`AndroidManifest.xml` mediante l'uso dell'apposito tag XML <`activity`>.

Questo modo di procedere vale per tutte le quattro componenti fondamentali di un'applicazione.

Il ciclo di vita di un'Activity

Dal momento in cui un'Activity compare sullo schermo al momento in cui scompare essa passa attraverso una serie di stati, che, nel loro complesso, rappresentano il ciclo di vita dell'Activity. Comprendere il ciclo di vita di un'Activity è fondamentale per assicurarci che le nostre applicazioni Android funzionino correttamente (Figura 3.1).

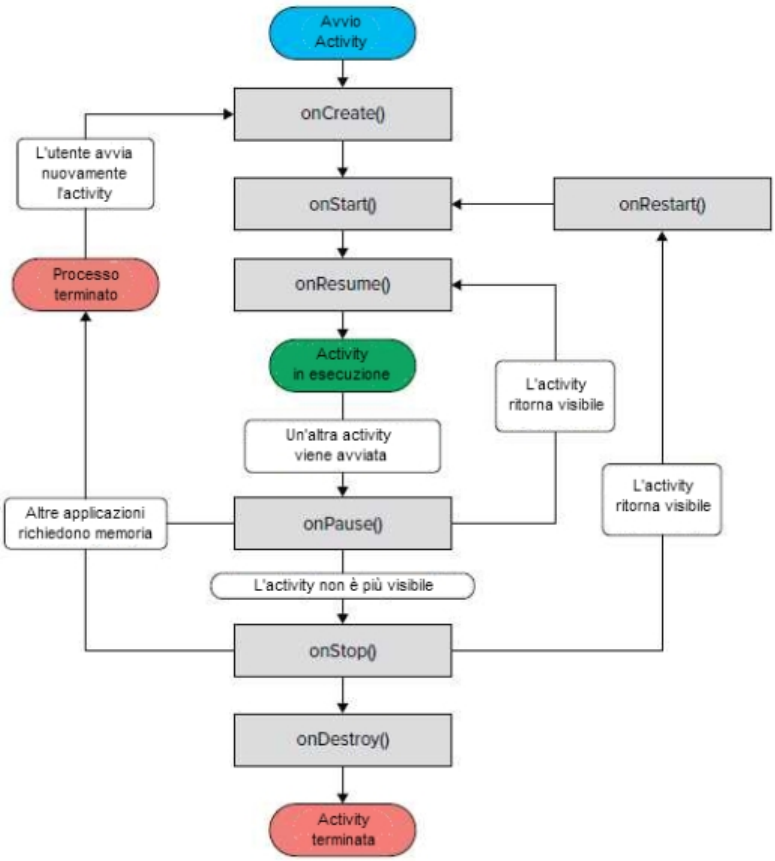

Figura 3.1. Il ciclo di vita di un'Activity

Il primo metodo che viene eseguito quando l'Activity viene invocata è `onCreate()`, che ha lo scopo di definire gli elementi di base per il funzionamento. Se al suo interno si fa utilizzo anche del metodo `setContentView()` è possibile definire un layout personalizzato. Una volta che ne viene portata a termine l'esecuzione, viene eseguito il metodo `onStart()`, che corrisponde al momento in cui l'Activity diventa visibile all'utente. Quando egli inizierà l'interazione con l'applicazione verrà richiamato il metodo `onResume()`. Da questo momento in poi verrà eseguito normalmente il codice contenuto al suo interno (visualizzazione di file con informazioni o file multimediali, pressione di pulsanti, etc.), fino a quando non sopraggiungerà un altro

evento come la chiamata ad un'altra Activity. In questo caso verrà invocato il metodo `onPause()` dove, solitamente, si salvano le modifiche ai dati e si bloccano le animazioni per ridurre il carico di lavoro della CPU. Se essa, poi, dovesse venire richiamata, riprenderà la sua esecuzione dal metodo `onResume()`. Se, invece, diventerà non più visibile all'utente verrà chiamato il metodo `onStop()`, il quale provvede, in caso di mancanza di memoria, ad eliminare le Activity in sospeso non necessarie. Nel caso l'applicazione venga riaperta dopo `onStop()`, questa dovrà ricominciare la sua esecuzione, tramite il metodo `onRestart()`, dal metodo `onStart()`. Infine, è presente il metodo `onDestroy()`, che permette di svolgere le ultime operazioni prima che l'app venga terminata o dall'utente o per mancanza di memoria.

Gli Intent

Gli Intent sono fondamentali per la creazione di una rete di Activity; essi, infatti, svolgono il ruolo di collante. `Intent` è una classe fornita col framework di Android. Questa prevede dei meccanismi per instaurare una sorta di messaggistica, gestita dal sistema operativo, con cui una componente può richiedere l'esecuzione di operazioni ad altre componenti. Gli Intent (Figura 3.2) vengono utilizzati per lanciare Activity, Service, oppure per inviare un messaggio di broadcast che può essere ricevuto da qualsiasi app. Esistono due tipi di Intent: espliciti ed impliciti.

Utilizzando il primo tipo, come dice il nome stesso, il programmatore ha bisogno di indicare esplicitamente quale Activity (classe) dovrà ricevere i dati, o quale Activity, in generale, dovrà essere richiamata.

Il meccanismo degli Intent impliciti, invece, permette di comunicare non solo con altre Activity interne all'applicazione, ma soprattutto con altre applicazioni o servizi.

La grossa differenza rispetto al meccanismo degli Intent espliciti è che, negli Intent impliciti, non sappiamo a priori quale sarà l'applicazione o il componente con cui l'Activity andrà a comunicare. Quindi, conosciamo l'eventuale dato da passare e l'azione, ma non sappiamo a chi esattamente verrà passato il dato. Sembra una situazione assurda, ma, nella realtà, il 90% dello scambio di informazioni tra le Activity di un'applicazione, avviene in modo "implicito".

Basti pensare, ad esempio, a quando si vuole condividere qualcosa in una pagina Internet: tutti i browser danno la possibilità di selezionare un menù, dal quale scegliere il famoso pulsante "condividi".

3.1.2 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.

Secondo quanto fornito nella documentazione di Android, un servizio può assumere due forme (Figura 3.3):

Figura 3.2. Esempio di un Intent

- *Started o Locale*: un servizio è "avviato" quando un componente richiama su di esso il metodo `start Service()`. Una volta avviato, esso può girare in background in maniera indefinita anche se il componente che lo ha avviato viene distrutto. Di solito, un servizio avviato esegue una singola operazione e non ritorna un risultato al chiamante. L'operazione eseguita potrebbe essere, ad esempio, il download o l'upload di un file in rete. Quando l'operazione è terminata il servizio è in grado di fermarsi da solo.
- *Bound*: un servizio è "bound" quando un componente si lega ad esso attraverso il metodo `bindService()`. Un servizio di questo tipo può essere visto come la parte server di una interfaccia client-server che permette ai componenti di interagire con il servizio, inviare richieste e ottenere risposte, il tutto in maniera trasversale ai diversi processi attraverso l'IPC (Inter Process Communication). Un servizio "bound" vive fin quando un componente è legato ad esso. Più componenti possono legarsi al servizio; quando ad un servizio non è legato alcun componente, il servizio viene eliminato. A differenza del servizio locale esso non è pensato per girare in background con un tempo indefinito.

3.1.3 I Content Provider

Tutte le applicazioni Android devono avere cura dei dati che gestiscono, in qualunque maniera siano immagazzinati. Considerando che nessuna app può accedere alle informazioni custodite da un'altra, è previsto un meccanismo ufficiale di sistema per la condivisione di dati; tale meccanismo si basa sul concetto di Content Provider.

I Content Provider sono componenti che permettono di leggere e modificare dati presenti in una determinata sorgente, inviando chiamate ad un indirizzo univoco, detto URI. 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 ancora;
- per fare in modo che più applicazioni possano condividere gli stessi dati, cosa che capita spesso quando più app provengono dallo stesso produttore;

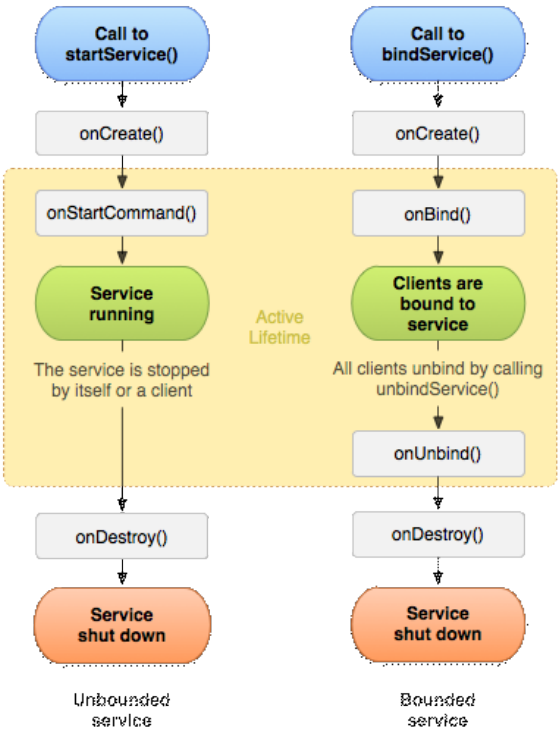

Figura 3.3. Il ciclo di vita dei due tipi di Service

- come elemento architetturale interno ad un'app articolata in cui più componenti accedono agli stessi dati.

3.1.4 I Broadcast Receiver

Un Broadcast Receiver è un componente Android che consente a un'applicazione di rispondere ai messaggi trasmessi dal sistema operativo Android o da un'altra applicazione. Una trasmissione viene generata da Android al verificarsi di alcune azioni; la classe `BroadcastReceiver` consente allo sviluppatore di gestire la situazione in caso di evento o azione. L'azione può essere l'arrivo di un messaggio o una chiamata, un download completo, un avvio completato, etc.

3.2 L'interfaccia grafica di un'app

Abbiamo capito come creare la nostra Activity, ma non siamo ancora in grado di assorciarci un layout corretto e capire come strutturarlo. Qualsiasi applicazione Android si voglia iniziare a sviluppare, ha sempre bisogno di un proprio aspetto grafico.

In Android Studio le interfacce possono essere classificate in interfacce testuali e interfacce grafiche.

Utilizzare le interfacce testuali comporta implementare l'interfaccia grafica scrivendo le classiche righe di codice (per questo motivo esse sono chiamate testuali) a cui un programmatore standard è abituato. Questa tipologia di interfaccia potrebbe essere di difficile lettura per un utente non-programmatore che si appresta ad iniziare questo nuovo percorso.

Ecco perchè, negli ultimi anni, hanno fatto la loro comparsa le interfacce grafiche. A differenza delle interfacce testuali, queste ultime facilitano, e di molto, la creazione di un'interfaccia utente. In questo caso non è più necessario scrivere righe di codice sconosciute a molti, ma si crea il layout desiderato semplicemente inserendo e spostando con l'aiuto del mouse i vari componenti che dovranno far parte dell'applicazione finale.

Per meglio comprendere l'interfaccia grafica è opportuno introdurre il concetto di view e layout.

3.2.1 Il layout

Un layout definisce la struttura visibile dell'interfaccia utente. In Android, con il termine "layout" si identificano tutte le ViewGroup che determinano il posizionamento degli widget sullo schermo. Questo tipo di ViewGroup prende il nome di LayoutManager. È possibile definire un layout sia dinamicamente, tramite Java, che mediante file XML. Quest'ultimo è il metodo più utilizzato, in quanto rende il codice più manutenibile a livello architetturale.

Le tipologie di layout più utilizzate sono: LinearLayout, TableLayout, RelativeLayout e ConstraintLayout, come vedremo nelle prossime sottosezioni. Non si può dire che esista un layout migliore di un altro; in base alla situazione e al tipo di oggetto da posizionare, deve essere il programmatore a scegliere quello che più si adatti. L'obiettivo da perseguire è che le informazioni si collochino in modo opportuno rispetto alla superficie dello schermo e che lo facciano sui dispositivi più disparati, aumentando la possibilità che gli utenti apprezzino il lavoro di progettazione. Ovviamente, per creare layout complessi, è possibile inserire più layout, anche di diverso tipo, uno dentro l'altro.

LinearLayout

Il LinearLayout (Figura 3.4) è il layout manager più utilizzato; dispone ogni elemento in maniera sequenziale su una riga o su una colonna in base al valore `android:orientation="horizontal"` o `android:orientation="vertical"`, impostato nell'`activity_main.xml`.

TableLayout

Il TableLayout (Figura 3.5) dispone i componenti in forma tabellare ed è particolarmente adatto a mostrare strutture regolari suddivise per righe e colonne, come se fossimo in presenza di una griglia.

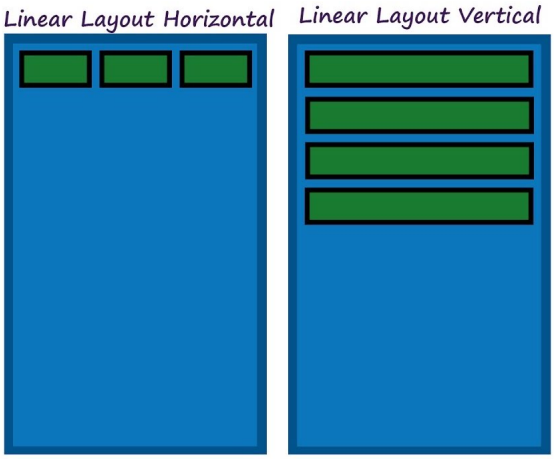

Figura 3.4. Esempio di LinearLayout

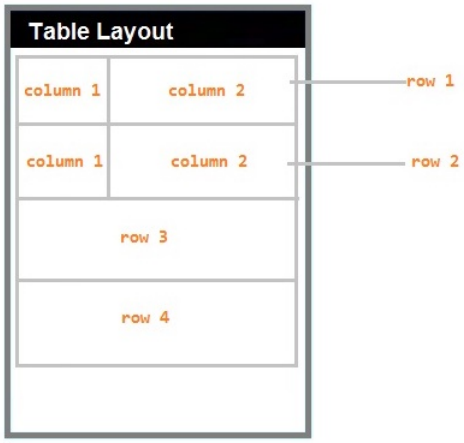

Figura 3.5. Esempio di TableLayout

RelativeLayout

Il RelativeLayout (Figura 3.6) è, sicuramente, il layout più moderno e flessibile tra quelli disponibili. È diventato il layout manager di default nei progetti Android. La parola "Relative" deriva dal fatto che gli elementi si dispongono in maniera relativa tra di loro, o rispetto al loro contenitore, adattandosi, quindi, a ogni tipo di display.

ConstraintLayout

Questo layout è molto simile al RelativeLayout, ma permette di definire la posizione degli elementi in maniera abbastanza flessibile tramite l'interfaccia grafica.

Grazie al ConstraintLayout (Figura 3.7) è possibile inserire un numero indeterminato di elementi, che possono anche essere ancorati tra di loro; nelle proprietà

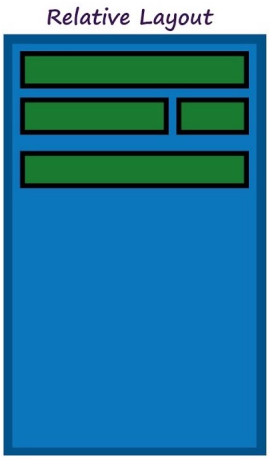

Figura 3.6. Esempio di RelativeLayout

potrà essere stabilita una distanza e come si dovrà comportare il presente "collegamento". Questo permetterà di mantenere la coerenza dell'interfaccia nelle varie situazioni.

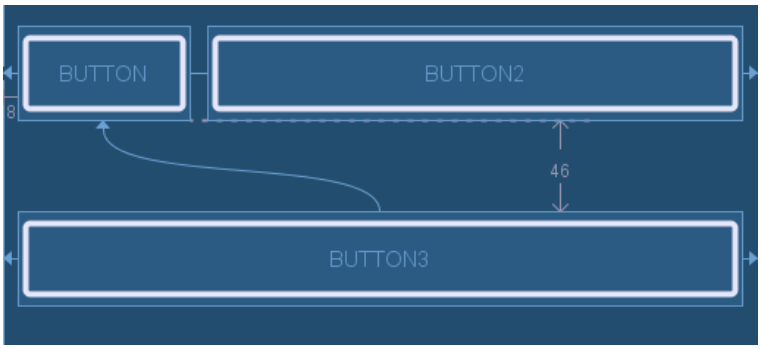

Figura 3.7. Esempio di ConstraintLayout

3.2.2 Le View

`Android.view.View` è la classe di base per la definizione dell'interfaccia grafica e la gestione degli eventi nati dall'interazione dell'utente con la UI. L'interfaccia grafica di un'Activity viene, in genere, definita da una o più View, elementi grafici (o widget) che appaiono su uno schermo. Una View può, quindi, essere considerata come una porzione dell'interfaccia dell'Activity e, spesso, consiste in controlli grafici, come Button, TextView o EditText (Figura 3.8).

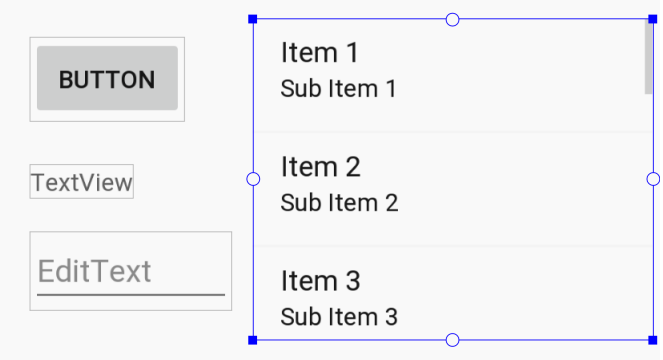

Figura 3.8. Esempi di Button, TextView, EditText e ListView

Button

Il Button (pulsante) è uno dei componenti grafici classici presenti in molti programmi. L'utente interagisce con esso tramite un tocco. Può essere costituito da testo, da un'icona, o da entrambi. L'`ImageButton` è, appunto, la classe che permette di realizzare un pulsante con un'immagine di sfondo.

TextView

Una TextView è un componente molto importante di un'applicazione Android; infatti, questa permette di mostrare all'utente un qualsiasi testo, il che è fondamentale per l'interazione con l'utilizzatore. Essa viene definita nel file `.xml` del layout e può essere manipolata dal file Java attraverso le funzioni ad essa associate, come, ad esempio, `getText()` o `setText()`.

La TextView ha degli attributi che vanno definiti in fase di creazione della stessa, e che servono a manipolare la grafica, cioè il modo in cui l'utente vede il testo.

EditText

In molte applicazioni, spesso, è presente la necessità di inserire del testo, ad esempio il proprio nome, una password o altre informazioni; per fare ciò è presente l'EditText. Tramite questo componente grafico è possibile inserire del testo e gestire gli eventi relativi al suo contenuto.

ListView

La ListView è una componente che, come dice il nome stesso, permette di visualizzare una lista di elementi. Le voci contenute vengono caricate in automatico da un Adapter, ovvero un oggetto che si occupa sia dell'accesso ai dati che della loro visualizzazione.

3.3 Gestione dei database in Android

La piattaforma Android fornisce diversi metodi e strumenti per salvare i dati delle applicazioni in modo permanente. Un primo metodo è quello di salvare i file all'interno della memoria condivisa (SD card). Un'altra possibilità potrebbe essere quella di memorizzare i dati sul web attraverso una connessione di rete, in modo da renderli accessibili anche ad altri dispositivi. In alternativa, i dati da salvare si possono organizzare in file di testo o in un database. I vantaggi di memorizzare i dati sul web sono la semplicità e il poco sforzo a livello di codice. Se, però, emerge la necessità di trattare dati più numerosi e complessi, la potenza e la versatilità dei database sono da preferire. Questo è proprio quello che è successo per la nostra app, per la quale è risultato di estremo aiuto il DBMS (Database Management System) SQLite, integrato all'interno del sistema operativo Android.

3.3.1 Database Management System

In informatica, un Database Management System (Figura 3.9), abbreviato in DBMS, è un sistema software progettato per consentire la creazione, la manipolazione (da parte di un amministratore) e l'interrogazione efficiente (da parte di uno o più utenti client) di database (ovvero di collezioni di dati strutturati). È ospitato su architettura hardware dedicata, come un server, oppure su un semplice computer. I DBMS svolgono un ruolo fondamentale in numerose applicazioni informatiche, dalla contabilità, alla gestione delle risorse umane, fino a contesti tecnici come la gestione di rete o la telefonia.

Il DBMS integrato in Android proviene dal mondo open source. Si tratta di SQLite, che si distingue per leggerezza e facilità di integrazione.

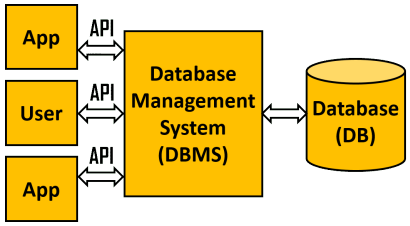

Figura 3.9. Funzionamento di un Database Management System

3.3.2 SQLite

Per poter utilizzare SQLite (Figura 3.10) non è necessario nè compilare nè installare alcun software; è tutto compreso nel sistema operativo stesso. Come detto, essendo leggero (la memoria sul disco varia da appena 4KB a circa 350KB), è la tecnologia perfetta per creare e gestire database in un ambiente come quello degli applicativi mobile, dove le risorse sono molto limitate e, dunque, è molto importante ottimizzarne l'utilizzo. Esso possiede, inoltre, praticamente tutti gli strumenti principali

che caratterizzano i più importanti database SQL (tabelle, viste, indici, trigger). Il codice è distribuito gratuitamente sia per scopi commerciali che privati. La sua diffusione è più ampia di quanto ci si possa aspettare; è, infatti, utilizzato in moltissimi applicativi e dispositivi che utilizziamo quotidianamente, come l'iPhone della Apple, o Mozilla Firefox; esso sarà, anche, utilizzato nell'applicazione oggetto della presente tesi.

Figura 3.10. Logo di SQLite

3.4 Diagramma dei casi d'uso

Nel percorso di progettazione di un'applicazione, il passo successivo alla stesura dei requisiti funzionali e non, è la rappresentazione dei diagrammi dei casi d'uso (Use Case Diagram). Gli elementi principali che vengono utilizzati nei diagrammi dei casi d'uso sono tre: lo scenario, gli attori e il caso d'uso. Lo scenario viene rappresentato tramite un rettangolo, al cui interno verranno inseriti i model element, che rappresentano caratteristiche del sistema. Gli attori sono rappresentati graficamente da un'icona che raffigura un uomo stilizzato. L'attore è colui che interagirà con il sistema. Un caso d'uso è raffigurato tramite un'ellisse contenente il nome; in particolare, un caso d'uso rappresenta una funzione o un servizio offerto dal sistema a uno o più attori.

In un diagramma di casi d'uso si mettono in evidenza tutte le funzionalità con cui l'utente potrà interagire, e le modalità con cui il sistema gestirà le varie situazioni.

In particolare, nel diagramma dei casi d'uso dell'applicazione *Eventi* (Figura 3.11), l'utente potrà, per prima cosa, visualizzare la lista degli eventi. Egli potrà, inoltre, creare una ricorrenza inserendo tutti i dati richiesti e visualizzare tutti i dati di una persona. Successivamente alla creazione di una ricorrenza, egli potra creare l'evento (che sia un compleanno, un onomastico o un anniversario) e selezionare gli invitati da aggiungere alla festa. Inoltre, egli potrà modificare a suo piacimento le notifiche, accedendo alla schermata delle impostazioni.

3.5 Progettazione concettuale

Lo scopo della progettazione concettuale è quello di creare uno schema di alto livello che rappresenti i dati e le relazioni tra di essi relativi al contesto analizzato. Partendo dalle informazioni raccolte durante l'analisi dei requisiti, avremo che i dati verranno suddivisi in due categorie, ovvero entità e relazioni. Un'entità rappresenta

Figura 3.11. Diagramma dei casi d'uso per l'app *Eventi*

un oggetto (fisico o astratto) del progetto analizzato, descritto tramite attributi. Una relazione, invece, rappresenta i legami logici e le associazioni che sussistono tra le varie entità.

Per quanto riguarda l'applicazione *Eventi*, le entità che serivranno a formare lo schema finale sono: "Persona", "Ricorrenza" e "Invitato". Invece, le relazioni che faranno da collante tra le entità sono: "Invita" e "Festeggia". Nella Figura 3.12 viene presentato lo schema concettuale (Diagramma Entità-Relazione) relativo alla realtà di interesse.

3.5.1 Entità

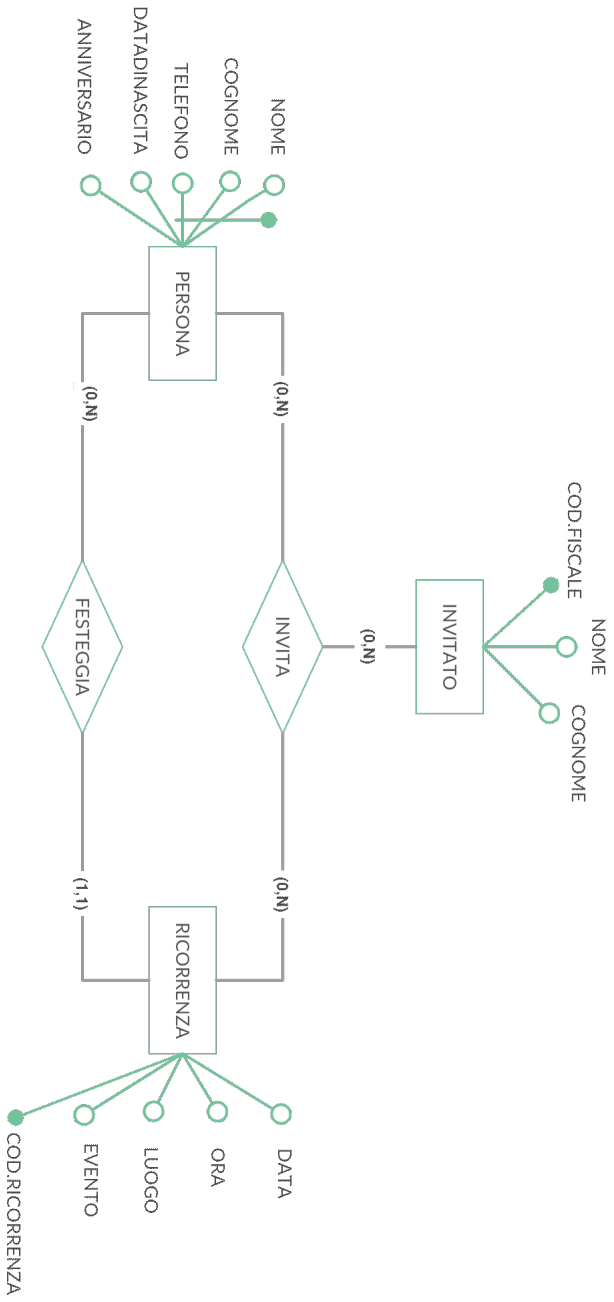

Figura 3.12. Diagramma E/R dell'applicazione *Eventi*

Persona

L'entità "Persona" è la più significativa poichè senza le persone non ci sarebbero le ricorrenze oggetto della nostra app. Gli attributi di questa entità sono i dati

anagrafici della persona, con relativo numero di telefono, ed eventualmente, una data di anniversario.

Ricorrenza

L'entità "Ricorrenza", invece, possiede gli attributi relativi ad una determinata ricorrenza, ovvero data, ora, luogo ed evento in questione.

Invitato

L'entità "Invitato" è più semplice delle altre entità, poichè non ha bisogno di attributi ridondanti, come la data di nascita o l'anniversario.

3.5.2 Relazioni

Invita

La relazione "Invita" è una relzione ternaria e fa da tramite tra le entità "Persona", "Ricorrenza" e "Invitato". Le cardinalità associate alle varie entità che partecipano a tale relazione sono sempre di tipo (0,N).

Festeggia

La relazione "Festeggia", collega, invece, solo "Persona" e "Ricorrenza". Una persona può festeggiare numerose ricorrenze, mentre una ricorrenza è collegata ad una e una sola persona.

3.6 Progettazione logica

L'obiettivo della progettazione logica è quello di costruire uno schema logico in grado di descrivere, in maniera corretta ed efficiente, tutte le informazioni contenute nello schema E/R, prodotto durante la fase di progettazione concettuale.

Lo schema logico complessivo per la realtà di interesse viene riportato nella Figura 3.13.

3.7 Progettazione dei mockup

Per progettare le schermate dell'app *Eventi* si è fatto uso di una serie di mockup.

Un mockup è la riproduzione di un oggetto o modello; esso viene utilizzato ogniqualvolta sia necessario avere un'idea che assomigli all'originale o che dia una rappresentazione visiva, eventualmente dettagliata, di come sarà o dovrà essere la schermata finale.

RELAZIONE
PERSONA (Nome, Cognome, Telefono, DataDiNascita, Anniversario)
RICORRENZA (Cod.Ricorrenza, Data, Ora, Luogo, Evento)
INVITATO (Cod.Fiscale, Nome, Cognome)
FESTEGGIA (Cod.Ricorrenza, Nome, Cognome, Telefono)
INVITA (Cod.Ricorrenza, Nome, Cognome, Telefono, Cod.Fiscale)

Figura 3.13. Schema logico complessivo della realtà di interesse

3.7.1 Schermata degli eventi

Nella prima schermata (Figura 3.14) è possibile visualizzare la lista scorrevole di persone al quale sono associati i rispettivi dati. Da questa schermata l'utente può eseguire tre operazioni, ovvero:

- visualizzare la scheda di una persona, premendo sul nome della stessa;
- aprire la schermata di aggiunta di un contatto, tramite il pulsante "+", posizionato in basso a destra;
- aprire le impostazioni tramite l'icona in alto a destra.

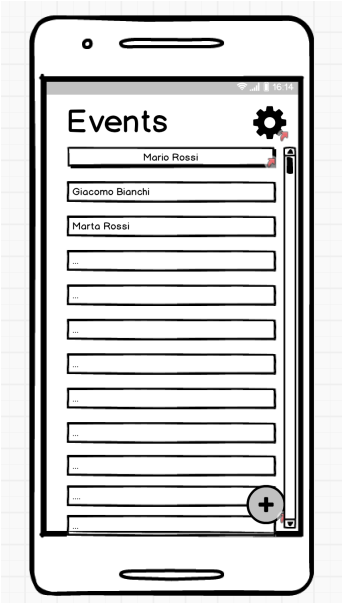

Figura 3.14. Mockup della schermata relativa agli eventi

3.7.2 Schermata di creazione delle ricorrenze

In questa schermata (Figura 3.15) è possibile, dopo aver inserito tutti i dati richiesti, creare una nuova ricorrenza, da aggiungere alla lista vista in precedenza. Inoltre, è presente il pulsante "Indietro" per ritornare alla schermata precedente.

Figura 3.15. Mockup della schermata per la creazione delle ricorrenze

3.7.3 Schermata di visualizzazione dei dati di una persona

Dopo aver premuto sul nome di una persona, si aprirà una schermata (Figura 3.16) completa di tutti i dati della persona selezionata. Da questa schemata, l'utente, oltre a tornare indietro, potrà accedere alla schermata di creazione degli eventi, tappando, a seconda dell'esigenza, su uno dei tre eventi: "compleanno", "anniversario" od "onomastico".

3.7.4 Schermata di creazione eventi

In questa schermata (Figura 3.17) è possibile, dopo aver inserito il luogo, la data e l'ora dell'evento scelto precedentemente, selezionare gli invitati da una lista scorrevole. Fatto ciò, se si vuole creare l'evento, è possibile farlo tramite il pulsante "Crea Evento". Altrimenti, si può tornare indietro con il solito pulsante "Indietro".

3.7.5 Schermata relativa alle impostazioni

L'ultima schermata che ci rimane da analizzare è quella relativa alle impostazioni (Figura 3.18). In essa si potranno gestire le notifiche; queste ultime potranno essere

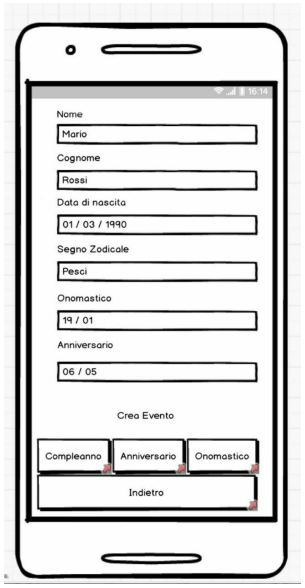

Figura 3.16. Mockup della schermata per la visualizzazione dei dati di una persona

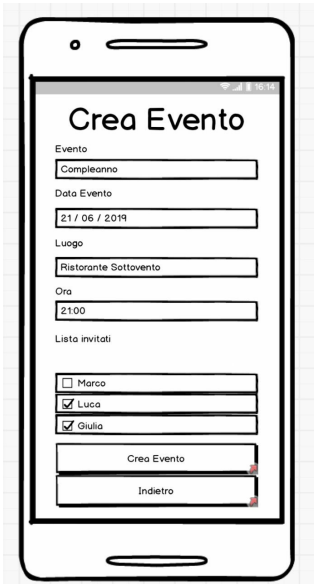

Figura 3.17. Mockup della schermata per la creazione degli eventi

attivate o disattivate. Qualora siano state attivate sara possibile impostare un orario predefinito.

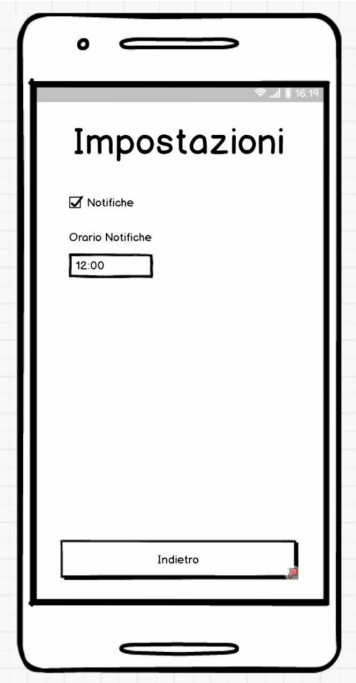

Figura 3.18. Mockup della schermata per impostare le notifiche

4

Implementazione

In questo capitolo vengono mostrate le implementazioni e le funzionalità di tutte le classi ed i metodi che compongono, nella sua interezza, l'applicazione Eventi.

4.1 Implementazione della base di dati

Come detto nei precedenti capitoli, una delle caratteristiche che può rivelarsi molto utile nelle nostre applicazioni Android è l'utilizzo di un database relazionale interno al dispositivo. SQLite permette di avere a disposizione un database relazionale salvato in un unico file.

Un modo comodo per gestire il nostro dabatase SQLite viene offerto direttamente da Android: si tratta della classe `SqliteOpenHelper`, che si preoccuperà di cercare per noi il database su disco, grazie a nome e versione che noi indicheremo, e di offrircelo nel codice Java, come oggetto della classe `SqliteDatabase`.

4.1.1 Classe `DatabaseHelper`

È buona pratica creare una classe "helper" per semplificare le interazioni con il database ed introdurre, così, un livello di astrazione che fornisca metodi intuitivi, flessibili e robusti per inserire, eliminare e modificare i record del database. In questo caso la classe è `DatabaseHelper` dichiarata nel Listato 4.1.

Questa è una classe public ed estende la classe `SqliteOpenHelper`. Oltre al nome del database (`DATABASE_NAME`) e al nome della tabella (`TABLE_NAME`), sono stati dichiarati tutti i nomi delle colonne presenti nella tabella (`COL1, COL2, COL3, COL4, COL5, COL6`), tutti aventi come tipo di dato `String`. La classe contiene, inoltre, un costruttore, al cui interno si invoca quello della classe base.

```
1   public class DatabaseHelper extends SQLiteOpenHelper {
2
3           public static final String DATABASE_NAME = "mylist.db";
4           public static final String TABLE_NAME = "mylist_data";
5           public static final String COL1 = "ID";
6           public static final String COL2 = "ITEM1";
7           public static final String COL3 = "COGNOME";
8           public static final String COL4 = "NUMERODITELEFONO";
9           public static final String COL5 = "DATADINASCITA";
10          public static final String COL6 = "ANNIVERSARIO";
```

```
11
12          public DatabaseHelper(Context Context) {
13          super(Context, DATABASE_NAME, null, 1);
14          }...}
```

Listato 4.1. Definizione della classe `DatabaseHelper`

Il metodo `onCreate` consente di specificare cosa deve fare l'Activity nel momento in cui viene creata. L'evento si verifica nel momento in cui l'Activity viene arrestata e riavviata.

In questo caso (Listato 4.2) il metodo sopracitato serve per creare la tabella, completa di tutte le informazioni necessarie, tramite il comando `CREATE TABLE`.

```
1   public void onCreate(SQLiteDatabase db) {
2
3           String createTable = "CREATE TABLE " + TABLE_NAME + " (ID INTEGER PRIMARY KEY AUTOINCREMENT,
4           ITEM1 TEXT, COGNOME TEXT, NUMERODITELEFONO TEXT, DATADINASCITA TEXT, ANNIVERSARIO TEXT) ";
5           db.execSQL(createTable);
6   }
```

Listato 4.2. Definizione del metodo `onCreate`

Dopo aver creato la tabella si potrebbe avere la necessità di eliminarla. È possibile farlo tramite il comando `DROP TABLE` integrato nel metodo `onUpgrade` (Listato 4.3).

```
1   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
2
3           db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
4           onCreate(db);
5   }
```

Listato 4.3. Definizione del metodo `onUpgrade`

Il metodo `addData` (Listato 4.4) aggiunge nel database, utilizzando la classe `ContentValues`, tutti i dati inseriti al runtime nell'apposita schermata dell'applicazione. Se l'inserimento va a buon fine, restituirà `true`, altrimenti `false`.

```
1   public boolean addData(String item1, String item2, String item3, String item4, String item5) {
2
3           SQLiteDatabase db = this.getWritableDatabase();
4           ContentValues contentValues = new ContentValues();
5           contentValues.put(COL2, item1);
6           contentValues.put(COL3, item2);
7           contentValues.put(COL4, item3);
8           contentValues.put(COL5, item4);
9           contentValues.put(COL6, item5);
10
11          long result = db.insert(TABLE_NAME, null, contentValues);
12
13          if (result == -1) {
14                  return false;
15                  } else {
16                  return true;
17                  }
18          }
```

Listato 4.4. Definizione del metodo `addData`

Se si vogliono visualizzare in ordine tutti gli elementi presenti in una tabella, è possibile utilizzare il metodo `getListContests` (Listato 4.5).

```
1   public Cursor getListContests() {
2
3           SQLiteDatabase db = this.getWritableDatabase();
4           Cursor data = db.rawQuery("SELECT * FROM " + TABLE_NAME + " ORDER BY ITEM1", null);
```

```
5           return data;
6   }
```

Listato 4.5. Definizione del metodo `getListContests`

L'ultimo metodo della classe `DatabaseHelper` è `deleteData` (Listato 4.6). Tale metodo serve per eliminare una specifica riga della tabella del database. Anche in questo caso, se l'operazione va a buon fine, il metodo restituirà `true`, altrimenti `false`.

```
1   public boolean deleteData(String id1, String id2, String id3) {
2
3           SQLiteDatabase db = this.getWritableDatabase();
4           int result = db.delete(TABLE_NAME, "ITEM1 = ? AND COGNOME = ? AND NUMERODITELEFONO = ?",
5           new String[] {id1,id2,id3});
6
7           if (result == -1) {
8                   return false;
9           } else {
10                  return true;
11          }
12  }
```

Listato 4.6. Definizione del metodo `deleteData`

4.2 Implementazione della lista di contatti

All'apertuta dell'applicazione *Eventi* la prima Activity che verrà visualizzata è `MainActivity`. In questa schermata sarà presentata la lista scorrevole formata da tutti gli elementi presenti nella tabella del database. L'implementazione è mostrata nella prossima sottosezione.

4.2.1 Classe `MainActivity`

La classe `MainActivity` (Listato 4.7) estende la classe `AppCompatActivity` utile per le attività che utilizzano le funzionalità della barra superiore. Inoltre, implementa la classe `View.OnClickListener`, fondamentale per gestire le interazioni con l'utente.

Dopo la creazione della `ListView` e di un `ArrrayList`, tramite un `if` ed un `while`, la lista scorrevole si popolerà degli elementi presenti nel database. Se il database è vuoto comparirà, in primo piano, un "Toast", cioè un messaggio a scomparsa, con su scritto "Il database è vuoto".

```
1   public class MainActivity extends AppCompatActivity implements View.OnClickListener {
2           DatabaseHelper myDB;
3           ListView listView;
4
5           protected void onCreate(Bundle savedInstanceState) {
6                   super.onCreate(savedInstanceState);
7                   setContentView(R.layout.activity_main);
8                   Toolbar toolbar = findViewById(R.id.toolbar);
9                   setSupportActionBar(toolbar);
10                  FloatingActionButton fab = findViewById(R.id.fab);
11                  fab.setOnClickListener(this);
12                  listView = (ListView) findViewById(R.id.listView);
13                  myDB = new DatabaseHelper(this);
14                  ArrayList<String> theList = new ArrayList<>();
15                  Cursor data = myDB.getListContests();
16
17                  if (data.getCount() == 0) {
18                          Toast.makeText(MainActivity.this, "Il database è vuoto", Toast.LENGTH_LONG).show();
19                  } else {
```

```
20                      while (data.moveToNext()) {
21                              theList.add(data.getString(1) + " " + data.getString(2));
22                              ListAdapter listAdapter = new ArrayAdapter<>(this,
23                              android.R.layout.simple_list_item_1, theList);
24                              listView.setAdapter(listAdapter);
25                      }
26              }
27              listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
28              public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
29                              Intent intent = new Intent(getApplicationContext(), Main4Activity.class);
30                              intent.putExtra("numero riga", position);
31                              startActivity(intent);
32              } }); }
```

Listato 4.7. Definizione della classe `MainActivity`

Nel Listato 4.8 vengono implementati i due metodi che gestiscono l'apertura delle opzioni (`onCreateOptionsMenu`), e l'apertura dell'Activity delle impostazioni (`onOptionsItemSelected`).

```
1   public boolean onCreateOptionsMenu(Menu menu) {
2           getMenuInflater().inflate(R.menu.menu_main, menu);
3           return true;
4   }
5   public boolean onOptionsItemSelected(MenuItem item) {
6           int id = item.getItemId();
7           if (id == R.id.action_settings) {
8                   Intent intent = new Intent (getApplicationContext(),Main3Activity.class);
9                   startActivity(intent);
10          }
11          return super.onOptionsItemSelected(item);
12  }
```

Listato 4.8. Definizione dei metodi `onCreateOptionsMenu` e `onOptionsItemSelected`

In questo caso il metodo `onClick` (Listato 4.9) serve, semplicemente, ad aprire l'Activity `Main2Activity` dopo aver premuto il `FloatingActionButton`, cioè il pulsante "+" posizionato in basso a destra.

```
1   public void onClick(View v) {
2           if (v.getId() == R.id.fab) {
3                   Intent intent = new Intent (getApplicationContext(),Main2Activity.class);
4                   startActivity(intent);
5           }
6   }
```

Listato 4.9. Definizione del metodo `onClick`

4.3 Implementazione della creazione dei contatti

Per popolare la `ListView` vista in precedenza, c'è bisogno di una schermata in cui inserire dei dati; è proprio questo il compito della classe `Main2Activity`.

4.3.1 Classe `Main2Activity`

La classe `Main2Activity` (Listato 4.10) si servirà di cinque `EditText` che l'utente avrà dovuto compilare con tutti i dati richiesti. Quando i campi saranno completati, l'utente premerà sul pulsante "CREA RICORRENZA" per aggiungere l'elemento alla lista scorrevole della classe `MainActivity`. Per fare ciò, ovviamente, l'Activity dovrà caricare i dati sul database; tramite l'istruzione `getText().toString()`

preleva i dati immessi e, tramite il metodo `AddData` (Listato 4.11), li carica nella tabella del database. Se il primo campo, cioè quello del nome, non è stato compilato, l'operazione non andrà a buon fine e verrà visualizzato un "Toast" con un messaggio di errore; se, invece, l'inserimento va a buon fine, verrà presentato un messaggio di conferma.

```
1   public class Main2Activity extends AppCompatActivity {
2
3           DatabaseHelper myDB;
4           EditText editText1, editText2, editText3, editText4, editText5;
5
6           protected void onCreate(Bundle savedInstanceState) {
7                   super.onCreate(savedInstanceState);
8                   setContentView(R.layout.activity_main2);
9                   editText1 = (EditText) findViewById(R.id.editText1);
10                  editText2 = (EditText) findViewById(R.id.editText2);
11                  editText3 = (EditText) findViewById(R.id.editText3);
12                  editText4 = (EditText) findViewById(R.id.editText4);
13                  editText5 = (EditText) findViewById(R.id.editText5);
14                  myDB = new DatabaseHelper(this);
15
16                  Button button1 = (Button) findViewById(R.id.button1);
17                  button1.setOnClickListener(new View.OnClickListener() {
18                          public void onClick(View view) {
19                                  Intent intent = new Intent(getApplicationContext(), MainActivity.class);
20                                  startActivity(intent);
21                          }
22                  });
23                  Button button2 = (Button) findViewById(R.id.button2);
24                  button2.setOnClickListener(new View.OnClickListener() {
25                          public void onClick(View v) {
26                                  Vibrator V = (Vibrator) getBaseContext().getSystemService(getBaseContext().
27                                  VIBRATOR_SERVICE); V.vibrate(60);
28                                  if (editText1.length() != 0) {
29                                          AddData(editText1.getText().toString(), editText2.getText().toString(),
30                                          editText3.getText().toString(), editText4.getText().toString(),
31                                     editText5.getText().toString());
32                                          editText1.setText("");
33                                          editText2.setText("");
34                                          editText3.setText("");
35                                          editText4.setText("");
36                                          editText5.setText("");
37                                          Intent intent = new Intent(getApplicationContext(), MainActivity.class);
38                                          startActivity(intent);
39                                  } else {
40                                          Toast.makeText(Main2Activity.this,"Il nome è vuoto", Toast.LENGTH_LONG).show();
41                                  }
42                          }
43                  });
44          }
```

Listato 4.10. Definizione della classe `Main2Activity`

```
1   public void AddData(String newEntry1, String newEntry2, String newEntry3, String newentry4, String newentry5) {
2
3           boolean insertData = myDB.addData(newEntry1, newEntry2, newEntry3, newentry4, newentry5);
4           if (insertData == true) {
5                   Toast.makeText(Main2Activity.this, "Il contatto è stato aggiunto", Toast.LENGTH_LONG).show();
6           } else {
7                   Toast.makeText(Main2Activity.this, "Il contatto non è stato aggiunto!", Toast.LENGTH_LONG).show();
8           }
9   }
```

Listato 4.11. Definizione del metodo `AddData`

4.4 Implementazione delle notifiche

Riguardo l'implementazione delle notifiche c'è da fare subito una distinzione. Infatti, sono stati implementati due blocchi di codice diversi per versioni di Android differenti. Il primo blocco è per le versioni di Android uguali o superiori ad Android 8 (Oreo), mentre il secondo blocco è per tutte le versioni inferiori.

4.4.1 Classe `Main3Activity`

La classe `Main3Activity` (Listato 4.12) è semplice; essa è costituita da un `CheckBox` per attivare o disattivare le notifiche, e da un `EditText` nel quale l'utente dovrà immettere un orario per la ricezione delle notifiche; tramite il pulsante "INDIETRO" si potrà tornare alla schermata principale. All'interno della classe è presente la funzione `calendar.get()` per ricavare la data attuale.

```
1   public class Main3Activity extends AppCompatActivity implements View.OnClickListener {
2           TextView textView;
3           private int notificationId=1;
4           DatabaseHelper myDB;
5           CheckBox checkBox;
6           EditText editText;
7           Calendar calendar = Calendar.getInstance(TimeZone.getDefault());
8           int day = calendar.get(Calendar.DATE);
9           int month = calendar.get(Calendar.MONTH) + 1;
10          int year = calendar.get(Calendar.YEAR);
11          protected void onCreate(Bundle savedInstanceState) {
12                  super.onCreate(savedInstanceState);
13                  setContentView(R.layout.activity_main3);
14                  editText = (EditText) findViewById(R.id.editText1);
15                  textView = (TextView) findViewById(R.id.textView);
16                  myDB = new DatabaseHelper(this);
17                  checkBox = (CheckBox) findViewById(R.id.checkBox);
18                  checkBox.setOnClickListener(this);
19                  Button button = (Button) findViewById(R.id.button);
20                  button.setOnClickListener(this);
21          }
22          public void onClick(View v) {
23                  if (v.getId() == R.id.button) {
24                  Intent intent = new Intent(getApplicationContext(), MainActivity.class);
25                  startActivity(intent);
26          }
27  }
```

Listato 4.12. Definizione della classe `Main3Activity`

Il metodo `Notifiche` (Listato 4.13) è formato, come detto in precedenza, da due blocchi che saranno compilati a seconda del sistema operativo che monta il dispositivo in cui è installata l'app. Infatti, è presente una clausola `if else` che, grazie al codice `android.os.Build.VERSION.SDK_INT`, individua la versione di Android utilizzata e manda in esecuzione il blocco giusto.

```
1   public void Notifiche(String calendar) {
2           if (v.getId() == R.id.checkBox) {
3                   if (((CheckBox) v).isChecked()) {
4                           Toast.makeText(Main3Activity.this, "Notifiche Attive", Toast.LENGTH_LONG).show();
5                           if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.O) {
6                                   Cursor data = myDB.getListContests();
7                                   data.moveToFirst();
8                                   while (data.moveToNext()) {
9                                           if (data.getString(4).compareTo(day + "/" + month) == 0) {
10                                                  String CHANNEL_ID = "my_channel_01";
11                                                  CharSequence name = "Eventi";
12                                                  String Description = "È il compleanno di: " + data.getString(1) + " "
13                                                   + data.getString(2);
14                                                  int importance = NotificationManager.IMPORTANCE_HIGH;
15                                                  NotificationChannel mChannel = new NotificationChannel(CHANNEL_ID,
16                                                   name, importance);
17                                                  mChannel.setDescription(Description);
18                                                  NotificationManager notificationManager = (NotificationManager)
19                                                  this.getSystemService(Context.NOTIFICATION_SERVICE);
20                                                  notificationManager.createNotificationChannel(mChannel);
21                                                  NotificationCompat.Builder builder = new NotificationCompat
22                                                  .Builder(this,CHANNEL_ID).setSmallIcon(R.drawable.eventi)
23                                                  .setContentTitle(name).setContentText(Description)
24                                                  .setPriority(NotificationCompat.PRIORITY_DEFAULT);
25                                                  notificationManager.notify(notificationId, builder.build());
26                                          }
27                                  }
28                          } else {
29                                  Cursor data = myDB.getListContests();
30                                  data.moveToFirst();
31                                  while (data.moveToNext()) {
32                                          if (data.getString(4).compareTo(day + "/" + month) == 0) {
33                                                  String CHANNEL_ID = "my_channel_01";
34                                                  CharSequence name = "Eventi";
```

```
35                                          String Description = "È il compleanno di: " + data.getString(1) + " "
36                                           + data.getString(2);
37                                          NotificationCompat.Builder builder = new NotificationCompat
38                                          .Builder(this,CHANNEL_ID).setSmallIcon(R.drawable.eventi)
39                                          .setContentTitle(name).setContentText(Description)
40                                          .setPriority(NotificationCompat.PRIORITY_DEFAULT);
41                                          NotificationManagerCompat notificationManager =
42                                           NotificationManagerCompat.from(this);
43                                          notificationManager.notify(notificationId, builder.build());
44                                      } else {
45                                          Toast.makeText(Main3Activity.this, "Notifiche non attive", Toast,
46                                          LENGTH_LONG).show();
47                                      }
48                                  }
49                              }
50                          }
51                      }
52  }
```

Listato 4.13. Definizione del metodo `Notifiche`

4.5 Implementazione della visualizzazione dei contatti

Mediante il metodo `onItemClick` della classe `MainActivity`, nel momento in cui si preme su un elemento della `ListView`, si apre l'Activity di visualizzazione del contatto, implementata nella classe `Main4Activity`.

4.5.1 Classe `Main4Activity`

Questa classe (Listato 4.14) riceve, grazie alla funzione `getIntExtra`, la posizione dell'oggetto premuto dall'utente nella lista. Così, scorrendo il database fino alla riga giusta tramite `moveToPosition()`, si trovano tutte le informazioni necessarie da far visualizzare in questa schermata.

Successivamente alla visualizzazione delle informazioni, l'utente può decidere di accedere alla schermata di creazione dell'evento. Infatti, tramite il pulsante "COMPLEANNO" si accederà alla schermata di creazione dell'evento del compleanno, tramite il pulsante "ONOMASTICO" si accederà alla schermata di crazione dell'evento dell'onomastico, e tramite "ANNIVERSARIO" si accederà a quella di creazione dell'anniversario. Ovviamente, se qualche dato, ad esempio la data di nascita, non fosse stato inserito, la schermata di creazione del compleanno non si aprirà, ma sarà visualizzato il solito "Toast" di errore.

```
1   public class Main4Activity extends AppCompatActivity implements View.OnClickListener {
2           TextView textView1,textView2,textView3,textView4,textView5,textView6;
3           EditText editText6,editText7,editText8;
4           DatabaseHelper myDB;
5           protected void onCreate(Bundle savedInstanceState) {
6                   super.onCreate(savedInstanceState);
7                   setContentView(R.layout.activity_main4);
8                   textView1 = (TextView) findViewById(R.id.textView1);
9                   textView2 = (TextView) findViewById(R.id.textView2);
10                  editText6 = (EditText) findViewById(R.id.editText6);
11                  editText7 = (EditText) findViewById(R.id.editText7);
12                  editText8 = (EditText) findViewById(R.id.editText8);
13                  myDB = new DatabaseHelper(this);
14                  Cursor data = myDB.getListContests();
15                  Intent i = getIntent();
16                  int n = i.getIntExtra("numero riga", 10);
17                  data.moveToPosition(n);
18                  final String nome = data.getString(1);
19                  final String cognome = data.getString(2);
20                  final String telefono = data.getString(3);
```

```
21              final String datadinascita = data.getString(4);
22              String onomastico = null;
23              final String anniversario = data.getString(5);
24              if (nome.compareTo("Alessio") == 0) onomastico = "17/07";
25              {...}
26              if (nome.compareTo("Piero") == 0)  onomastico = "29/06";
27              textView1.setText(nome + " " + cognome);
28              textView2.setText(telefono);
29              editText6.setText(datadinascita);
30              editText7.setText(onomastico);
31              editText8.setText(anniversario);
32              Button button3 = (Button) findViewById(R.id.button3);
33              button3.setOnClickListener(new View.OnClickListener() {
34                      public void onClick(View v) {
35                              Intent intent = new Intent (getApplicationContext(),MainActivity.class);
36                              startActivity(intent);
37                      }
38              });
39              Button button4 = (Button) findViewById(R.id.button4);
40              button4.setOnClickListener(new View.OnClickListener() {
41                      public void onClick(View v) {
42                              Vibrator V = (Vibrator) getBaseContext().getSystemService(getBaseContext().
43                              VIBRATOR_SERVICE); V.vibrate(60);
44                              deleteData(nome, cognome, telefono);
45                              Intent intent = new Intent(getApplicationContext(), MainActivity.class);
46                              startActivity(intent);
47                      }
48              });
49              Button button5 = (Button) findViewById(R.id.button5);
50              button5.setOnClickListener(new View.OnClickListener() {
51                      public void onClick(View v) {
52                              if (datadinascita.length() != 0) {
53                                      Intent intent = new Intent(getApplicationContext(), Main5Activity.class);
54                                      intent.putExtra("evento", "Compleanno di " + nome + " " + cognome);
55                                      startActivity(intent);
56                              }
57                              else {
58                                      Toast.makeText(Main4Activity.this, "La data di nascita non è
59                                      stata inserita!", Toast.LENGTH_LONG).show();
60                              }
61                      }
62              });
63              Button button6 = (Button) findViewById(R.id.button6);
64              button6.setOnClickListener(new View.OnClickListener() {
65                      public void onClick(View v) {
66                              if (editText7.length() != 0) {
67                                      Intent intent = new Intent(getApplicationContext(), Main5Activity.class);
68                                      intent.putExtra("evento", "Onomastico di " + nome + " " + cognome);
69                                      startActivity(intent);
70                              }
71                              else {
72                                      Toast.makeText(Main4Activity.this, "L'onomastico non esiste
73                                      per questo nome!", Toast.LENGTH_LONG).show();
74                              }
75                      }
76              });
77              Button button7 = (Button) findViewById(R.id.button7);
78              button7.setOnClickListener(new View.OnClickListener() {
79                      public void onClick(View v) {
80                              if (anniversario.length() != 0) {
81                                      Intent intent = new Intent(getApplicationContext(), Main5Activity.class);
82                                      intent.putExtra("evento", "Anniversario di " + nome + " " + cognome);
83                                      startActivity(intent);
84                              }
85                              else {
86                                      Toast.makeText(Main4Activity.this, "L'anniversario non è stato inserito!",
87                                      Toast.LENGTH_LONG).show();
88                              }
89                      }
90              });
91      }
```

Listato 4.14. Definizione della classe `Main4Activity`

In quest'Activity (`Main4Activity`) è possibile, inoltre, eliminare un contatto tramite il pulsante "ELIMINA CONTATTO", posizionato in basso a destra. In questo caso ci verrà in aiuto il metodo `deleteData` (Listato 4.15), che richiama il metodo omonimo già visto nella classe `DatabaseHelper`.

```
1   public void deleteData (String deleteItem1, String deleteItem2, String deleteItem3) {
2           boolean result = myDB.deleteData(deleteItem1, deleteItem2, deleteItem3);
3
4           if (result == true) {
5                   Toast.makeText(Main4Activity.this, "Il contatto è stato eliminato", Toast.LENGTH_LONG).show();
6           } else {
7                   Toast.makeText(Main4Activity.this, "Il contatto non è stato eliminato!", Toast.LENGTH_LONG).show();
```

```
8         }
9   }
```

Listato 4.15. Definizione del metodo `deleteData`

4.6 Implementazione della creazione degli eventi

Dopo aver selezionato l'evento del contatto scelto nelle precedenti Activity, si visualizzerà la corrispettiva schermata di creazione. L'utente, per creare l'evento, dovrà semplicemente inserire negli appositi spazi il luogo, la data e l'ora dell'evento e premere sul pulsante "SPEDISCI INVITI"; si aprirà, così, in automatico, l'applicazione di messaggistica predefinita sul proprio smartphone.

4.6.1 Classe `Main5Activity`

La classe `Main5Activity` (Listato 4.16) riceve dalla classe `Main4Activity` le informazioni richieste per la creazione dell'evento, quali il tipo di evento e la persona festeggiata. Questo è possibile grazie alla funzione `getStringExtra`, che, integrata nell'`Intent`, comunica stringhe di caratteri tra le Activity. Invece, la gestione della data, dell'ora e del luogo avviene tramite `EditText`; infatti, alla pressione del pulsante "SPEDISCI INVITI", le variabili immesse dall'utente saranno salvate e inserite nel messaggio da spedire a tutti gli invitati.

Inoltre, è presente il pulsante "INDIETRO", nel caso in cui l'utente avesse sbagliato evento o volesse modificare qualcosa.

```
1   public class Main5Activity extends AppCompatActivity {
2           TextView textView9;
3           Button button8, button9;
4           ListView listView1;
5           DatabaseHelper myDB;
6           CheckBox checkBox;
7           EditText editText8, editText9, editText11;
8           protected void onCreate(Bundle savedInstanceState) {
9                   super.onCreate(savedInstanceState);
10                  setContentView(R.layout.activity_main5);
11                  textView9 = (TextView) findViewById(R.id.textView9);
12                  Intent i = getIntent();
13                  String n = i.getStringExtra("evento");
14                  textView9.setText(n);
15                  listView1 = (ListView) findViewById(R.id.listView1);
16                  myDB = new DatabaseHelper(this);
17                  listView1.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
18                  ArrayList<String> theList1 = new ArrayList<>();
19                  Cursor data = myDB.getListContests();
20                  if (data.getCount() == 0) {
21                          Toast.makeText(Main5Activity.this, "Il database è vuoto", Toast.LENGTH_LONG).show();
22                  } else {
23                          while (data.moveToNext()) {
24                          theList1.add(data.getString(1) + " " + data.getString(2));
25                          ListAdapter listAdapter = new ArrayAdapter<>(this, android.R.layout.
26                          simple_list_item_multiple_choice, theList1);
27                          listView1.setAdapter(listAdapter);
28                  }
29          }
30          button8 = (Button) findViewById(R.id.button8);
31          button8.setOnClickListener(new View.OnClickListener() {
32                  public void onClick(View v) {
33                          finish();
34                  }
35          });
36          button9 = (Button) findViewById(R.id.button9);
37          button9.setOnClickListener(new View.OnClickListener() {
38                  public void onClick(View v) {
39                          Intent intent = new Intent(Intent.ACTION_SEND);
40                          intent.setData(Uri.parse("smsto:"));
```

```
41                      intent.putExtra("sms_body", "Ciao, sei invitato/a all'evento " + textView9+ " il giorno "
42                      + editText9 + " alle ore " + editText11 + " presso " + editText8);
43                      if (intent.resolveActivity(getPackageManager()) != null) {
44                            startActivity(intent);
45                      }
46                }
47          });
48    }
```

Listato 4.16. Definizione della classe `Main5Activity`

5

Discussione, conclusioni e sfide future

Questo capitolo inizia con una discussione del lavoro svolto, con l'obiettivo di delineare i punti di forza e di debolezza e le lezioni apprese. Successivamente verranno tratte alcune conclusioni in merito al lavoro svolto e verranno proposti alcuni possibili sviluppi futuri.

5.1 Discussione

È possibile individuare una serie di punti di forza e di debolezza che caratterizzano l'applicazione *Eventi*.

5.1.1 Punti di forza

Sicuramente, il primo lato positivo di questa applicazione è il fatto che sia progettata per Android; di conseguenza, può essere potenzialmente scaricata ed usata da milioni di utenti su miliardi di dispositivi. Anche la piattaforma con cui è stata progettata, cioè Android Studio, è un punto a favore, poichè questa è la piattaforma più utilizzata per realizzare app Android; quindi, la gestione e la modifica del codice da parte di altri programmatori non sarà complicata.

Un altro punto di forza può essere l'idea dell'applicazione *Eventi*, poichè, come detto in precedenza, non ha un numero molto elevato di competitor ed offre funzionalità quasi uniche nel suo genere. Infatti, oltre alla gestione delle notifiche, non è banale il fatto di poter mandare messaggi precompilati agli invitati.

L'ultimo fattore positivo sta nel fatto che l'app progettata non ha bisogno di Internet per funzionare, a differenza della maggior parte delle applicazioni in commercio oggi.

5.1.2 Punti di debolezza

Ovviamente non esistono solo punti di forza, ma bisogna anche fare autocritica e dare spazio all'esame dei lati negativi dell'applicazione *Eventi*.

Il primo aspetto negativo riguarda il fatto di non aver bisogno di Internet; questo, infatti, ha, dall'altra faccia della medaglia, la necessità di dover compilare la lista di contatti ex novo, senza poter magari attingere a liste predefinite presenti nei social network, ad esempio Facebook.

Altro punto a sfavore è che, nella lista scorrevole iniziale, oltre al nome e cognome del contatto, non sono presenti altre informazioni che potrebbero essere utili, quali la vicinanza temporale ad un evento.

Inoltre, l'invio degli inviti avviene mediante l'ormai quasi inutilizzata app di messaggistica di default del dispositivo; in molti casi, poi, l'invio è anche a pagamento.

Un ultimo punto negativo potrebbe risiedere nel fatto che il codice non è stato mai commentato; di conseguenza, nel caso in cui un altro programmatore voglia metterci le mani, potrebbe avere delle difficoltà nel farlo.

5.1.3 Lezioni apprese

I progetti possono insegnare molte lezioni preziose su come vengono gestiti i processi di lavoro nel tempo.

Avendo acquisito una certa familiarità con Android Studio, non si può che rimanere impressionati dalla potenza e dalle possibilità che offre uno strumento del genere. Lo sviluppo dell'applicazione, tramite i linguaggi Java e XML, non è stato banale, ma un programma così ben fatto ci ha sicuramente dato una mano. Infatti, esistono due tipi di errori di programmazione, l'errore di compilazione e l'errore di esecuzione. Se l'errore di esecuzione è più complicato da trovare, quello di compilazione ci verrà sicuramente segnalato da Android Studio.

Ma non è possibile costruire un'app senza una solida base. Queste fondamenta sono composte dall'analisi dei requisiti. Infatti, è stato fondamentale stilare una lista di requisiti da cui, poi, partire per implementare le varie funzionalità. Anche il diagramma dei casi d'uso e il diagramma E/R svolgono un ruolo basilare poichè, senza di essi, il progetto sarebbe probabilmente andato a rilento. Ultima nota di merito va ai mockup, che ci hanno dato una riproduzione, molto vicina alla rappresentazione finale, di come si sarebbero presentate tutte le schermate dell'applicazione *Eventi*, così da semplificare la fase di realizzazione.

L'ultima lezione appresa riguarda il progetto in generale. Abbiamo notato che non è possibile lasciare nulla al caso, ma ogni passo e ogni sfida vanno affrontati nel migliore dei modi. Abbiamo osservato che l'apprendimento che si verifica durante ed al termine di ogni progetto, breve o complesso che sia, avviene anche grazie agli errori commessi. Infatti, l'insuccesso non è qualcosa da temere; d'altronde, non prendere in considerazione ciò che è andato male durante un progetto è stupido, poichè significa mettersi nelle condizioni di ripetere gli stessi errori. Il progetto è stato affrontato con questa mentalità e, a nostro avviso, è stato un metodo efficace.

5.2 Conclusioni

La tesi tratta gli aspetti fondamentali della programmazione di App per Android mediante lo studio di una soluzione per la gestione degli eventi.

In particolare, dapprima viene proposta una panoramica sulla storia della tecnologia, introducendo i sistemi operativi per dispositivi fissi e mobili. Questo poichè è di particolare rilievo fare un discorso sul sistema operativo Android. Infatti, l'applicazione che abbiamo realizzato è proprio finalizzata a dispositivi che montano Android.

Successivamente viene descritta l'organizzazione iniziale del progetto con tutte le sue sfaccettature. Si inizia con la raccolta delle informazioni, passando per la descrizione della componente dati ed arrivando ai requisiti, funzionali e non, grazie ai quali si delinea la nostra app.

Dopo di ciò, vengono illustrati i componenti logici che costituiscono un'applicazione, prestando particolare attenzione all'Activity e al ciclo di vita che ne governa il funzionamento. Una parte chiave della tesi è quella successiva, riguardante le basi per la programmazione e la grafica delle app. L'obiettivo è quello di capire i diversi ruoli di Java e XML nella costruzione del progetto e fornire gli strumenti necessari per la creazione della prima Activity. Successivamente, viene affrontato l'argomento dell'interfaccia, presentando i controlli grafici alla base di gran parte delle applicazioni, come TextView, EditText, Button e ListView. Inoltre, riprendendo in mano il progetto, vengono esposti il diagramma dei casi d'uso, la progettazione concettuale, completa di entità e relazioni, e la progettazione logica. La fine di questa sezione è dedicata a tutte le schermate presenti nell'applicazione, create tramite mockup.

Infine, è stata affrontata l'implementazione dell'app illustrando tutte le classi comprensive di tutti i metodi.

5.3 Sfide future

Come accennato precedentemente, un'interessante punto di partenza per uno sviluppo futuro potrebbe essere l'integrazione nel progetto di un collegamento ad Internet, non solo, come detto, per aggiungere liste precompilate, ma anche per far comunicare dispositivi differenti tramite l'applicazione. Un'idea sarebbe quella di spedire e ricevere gli inviti direttamente all'interno dell'applicazione *Eventi*.

Inoltre, si potrebbe aggiungere la possibilità di inserire immagini per ogni contatto, così da essere visualizzate come icone delle notifiche.

Un'altra idea interessante sarebbe quella di poter impostare notifiche mirate per ogni ricorrenza specifica. Infatti, l'utente potrebbe non desiderare che gli venga notificato nessun evento, ma soltanto quelli precedentemente selezionati all'interno dell'applicazione.

Ringraziamenti

Arrivato a questo punto molto importante della mia carriera universitaria, sento il desiderio di ringraziare tutti coloro che mi hanno aiutato e sostenuto durante questo percorso.

Desidero ringraziare coloro che hanno dato il loro contributo nella realizzazione di questa tesi di laurea.

Ringrazio, innanzitutto, il mio relatore, il Professore Domenico Ursino, che mi ha accompagnato in questo lungo percorso, facendosi trovare sempre disponibile ad incontrarmi e sciogliere i miei dubbi grazie alla sua esperienza e capacità.

Vorrei ringraziare la mia famiglia, che ha sempre creduto in me e mi ha aiutato a raggiungere questo traguardo, sostenendomi economicamente e moralmente in ogni mia scelta.

Inoltre, vorrei ringraziare tutti coloro che hanno intrapreso il percorso di laurea al mio fianco e, in generale, tutti gli amici con cui mi sono relazionato durante questo periodo.

Riferimenti bibliografici

1. V. Albertoni. *SQLite, il database per tutti*. Streetlib, 2018.
2. M. Bonifazi. *Sviluppare applicazioni per Android in 7 giorni*. Edizioni LSWR, 2016.
3. P. Camagni, R. Nikolassy, and E. Falzone. *Sviluppare App per Android*. Hoepli, 2017.
4. R. Campi. *Alle basi della programmazione*. Mondadori, 1985.
5. M. Carli. *Svilupare applicazioni per Android*. Apogeo Editore, 2011.
6. M. Carli. *Android 9. Guida completa per lo sviluppo di applicazioni mobile*. Apogeo Editore, 2019.
7. C. De Sio Cesari. *Manuale di Java 8*. Ulrico Hoepli Editore, 2014.
8. G. Cioffi and V. Falzone. *Manuale di informatica*. Calderini, 2002.
9. E. Cisotti and M. Giannino. *Android: Guida completa*. Edizioni LSWR, 2015.
10. F. Collini and M. Bonifazi. *Android. Programmazione Avanzata*. Edizioni LSWR, 2015.
11. L. Comi. *Java*. Apogeo Editore, 2004.
12. R. Crosato. *Android per esempi: guida allo sviluppo di applicazioni*. Edizioni LSWR, Pubblicazione indipendente, 2017.
13. M. Ferrero. *SQL*. Apogeo Editore, 2004.
14. M. Gargenta. *Sviluppare con Android: Realizzare le applicazioni mobili con Java ed Eclipse*. Hoepli, 2014.
15. G. Grandinetti. *Android studio 2*. `www.edizionifutura.it`, 2016.
16. A. Lorenzi and A. Rizzi. *Java. Programmazione ad oggetti e applicazioni Android*. Atlas, 2013.
17. G. Maggi. *Java 9: Guida completa*. Edizioni LSWR, 2018.
18. P. Morvan. *Dizionario di informatica*. Gremese Editore, 1989.
19. M. Owens. *The Definitive Guide to SQLite*. APress, 2014.
20. W. Savitch. *Programmazione di base e avanzata con Java*. Pubblicazione indipendente, 2018.
21. D. Sillars. *Sviluppare applicazioni Android ad alte prestazioni*. O'Reilly, 2017.