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 dei clienti di uno studio professionale

Design and implementation of an Android app for managing customers of a professional studio

Relatore

Prof. Domenico Ursino

Candidato

Francesco Talento

Anno Accademico 2019-2020

Indice

Elenco delle figure

Elenco dei listati

Introduzione

Gli smartphone sono, ad oggi, i dispositivi mobile più diffusi. Essi permettono di compiere, dal palmo della propria mano, le più svariate funzioni, oltre a quelle proprie di un telefono cellulare. Queste vanno dalle ricerche e dagli acquisti online, alla visualizzazione e al download di contenuti multimediali, all'utilizzo di Social Network, e tanto altro ancora.

A confermare il ruolo che gli smartphone hanno raggiunto, vi sono le parole di Amit Singhal, vicepresidente senior della divisione Search di Google. Durante la conferenza annuale "Code Mobile" ha dichiarato che, per la prima volta, sulle oltre cento miliardi di ricerche su Google, effettuate in tutto il pianeta, oltre il 50% sono state effettuate da smartphone.

L'importanza assunta dal mondo mobile negli ultimi anni ha rivoluzionato la vita quotidiana di milioni di persone in tutto il mondo, nonché la loro esperienza in rete. Sempre più utenti stanno migrando dal mondo desktop a quello mobile; in questo contesto le app sono diventate per le aziende uno degli strumenti più efficaci per raggiungere i propri clienti.

Le app possono essere di tre tipologie diverse:

- App Native;
- Web App;
- App Ibride (o cross-platform).

Le App Native sono applicazioni sviluppate specificamente per un sistema operativo. In questo caso, possono essere sviluppate per Android o per iOS. Il primo è gestito da Google e rappresenta circa l'80% del mercato in Italia; il secondo, invece, è gestito da Apple e costituisce circa il 16%.

Ovviamente, due sistemi operativi diversi rappresentano due attitudini diverse dei consumatori. Per esempio, gli utenti di iOS sono, generalmente, maggiormente disposti a spendere denaro nelle app e sono anche più esigenti. Per questa ragione, è comune trovare un'app a pagamento sull'App Store che, invece, è gratuita sul Play Store di Google. I linguaggi di programmazione sono differenti; iOS sfrutta Swift, mentre Android utilizza Java o Kotlin. Essi hanno regole diverse, così come diverse sono le possibilità da essi offerte.

Detto ciò, un'applicazione Android non funzionerà mai sul sistema operativo iOS, e viceversa; perciò, prima di sviluppare un'applicazione, bisogna scegliere anche

a quale clientela ci si vuole rivolgere. È possibile, anche, sviluppare un'applicazione per entrambi i sistemi operativi; tuttavia in questo caso, ci sarà tanto lavoro da fare.

Le Web App sono app che funzionano come dei siti web, senza nessuna differenza tra piattaforma, sistema di sviluppo e codice. Ciò significa che gli utenti non dovranno installare l'applicazione sui dispositivi e, di conseguenza, esse non potranno essere pubblicate negli Store e, quindi, non usufruiranno dell'enorme visibilità offerta dai Market digitali.

Le App Ibride, o cross-platform, si inseriscono tra le due tipologie appena descritte. Esse, rispetto alle App Native, sono più rapide da sviluppare e meno dispendiose. Una caratteristica fondamentale delle delle App Ibride è quella di essere cross-platform, ovvero multipiattaforma. Ciò significa che viene generata una sola versione, indipendentemente dal numero di piattaforme sulle quali si desidera essere presenti. Tuttavia, questi benefici hanno un costo, ovvero le performance dell'app sono inferiori, risultando queste meno stabili, dal momento che il sistema è meno adattabile a ciascuna piattaforma.

Quale tecnologia preferire, quindi? Non esiste un tipo migliore dell'altro; tutto dipende da ciò che l'app deve svolgere, a quale clientela ci si vuole rivolgere e alle risorse che si intendono investire.

Il progetto della presente tesi nasce per permettere allo studio commercialistico Co.Fi.T. di gestire i propri clienti attraverso un'applicazione mobile. Per raggiungere l'obiettivo è stato scelto di sviluppare l'applicazione in Android, perché questo è il sistema operativo più diffuso. Si sarebbe potuto anche sviluppare un'app ibrida; tuttavia l'app nativa dà più garanzie di stabilità.

Per sviluppare l'applicazione è necessario effettuare un'attenta analisi dei requisiti, con lo scopo di aver chiaro quali funzionalità devono essere offerte dall'app e come implementarle.

La fase successiva è quella di progettazione. Durante questa fase vedremo, soprattutto, come sono strutturati i dati che l'app CoFiT dovrà manipolare e gestire. Inoltre, si progetteranno le interfacce utente dell'applicazione attraverso wireframe e mockup.

Dopo la fase di progettazione si passerà a quella di implementazione, in cui tutto ciò che è stato progettato e pensato durante le prime fasi verrà realizzato.

Una volta ottenuto il prodotto finale, esso verrà testato attraverso dispositivi reali ed emulatori. Infine, verrà steso un manuale utente.

Il lavoro si concluderà con la SWOT Analysis, che consente di individurare punti di forza e di debolezza, opportunità e minacce dell'app realizzata.

In particolare, la tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 verranno analizzati alcuni software gestionali, in modo tale da avere chiaro i servizi che dobbiamo garantire.
- Nel Capitolo 2 verrà descritto il mondo Android. Vedremo la sua nascita e, soprattutto, come si è sviluppato. In particolare, vedremo quali parti costituiscono un'applicazione.
- Nel Capitolo 3 riporteremo l'analisi dei requisiti e, successivamente, illustreremo la fase di progettazione. Durante questa fase si decideranno il lavoro da svolgere e i software da utilizzare per lo sviluppo dell'app.

- Nel Capitolo 4 verrà realizzato tutto ciò che è stato deciso durante la fase di progettazione, allo scopo di ottenere il prodotto finale. Inoltre, sarà effettuato il testing dell'applicazione e verrà presentato il manuale utente.
- Nel Capitolo 5 verranno analizzate le lezioni apprese durante il lavoro di tesi. Inoltre, verrà valutato il lavoro svolto attraverso la SWOT Analysis.
- Nel Capitolo 6 verranno riportate le conclusioni in merito al lavoro svolto e quali sono, o potrebbero essere, gli obiettivi da raggiungere in futuro.

1

Software a supporto di studi professionali

In questo capitolo verranno analizzate le caratteristiche generali che dovrebbe avere un software gestionale[1] *per soddisfare le esigenze di un professionista. In particolare saranno descritti alcuni software disponibili in commercio per tale attività.*

1.1 Caratteristiche generali

La professione di commercialista è in continua evoluzione e, per tale ragione, il commercialista non può che richiedere il supporto di un software per facilitare il suo operato, per non perdere di vista le scadenze e, soprattutto, cercare di soddisfare tutte le esigenze dei suoi clienti.

In tale ambito, la scelta di un buon software è, certamente, quel che ci vuole per poter incrementare il lavoro e ottenere un cospicuo risparmio di tempi e di costi. *Ma quali attività deve svolgere il software per poter essere effettivamente d'aiuto per lo studio?*

Partiamo dal presupposto che un commercialista deve adempiere efficacemente a numerosi aspetti della propria attività, i quali vanno dalla contabilità al bilancio, dagli adempimenti fiscali all'archiviazione elettronica dei documenti, dalla conservazione legale sostitutiva agli obblighi normativi antiriciclaggio. Con così tanto lavoro da svolgere, lo studio non potrà che attrezzarsi di software che possano supportare e coadiuvare tutti questi ambiti nel raggiungimento degli obiettivi preposti.

Diventa, altresì, importante che il software gestionale per lo studio di commercialisti possa favorire una congrua collaborazione tra lo studio e l'azienda, determinando dei vantaggi per entrambi. Non dovrebbero, poi, mancare la gestione contabile in senso stretto, con la predisposizione di bilanci e liquidazioni IVA, così come la gestione dello studio, in maniera tale che ogni aspetto possa essere efficacemente realizzato in modo snello e soddisfacente. Tra gli altri aspetti gestionali più ricorrenti troviamo poi la gestione documentale, quella delle forniture e quella delle pratiche.

Con un simile carico di lavoro la scelta del miglior software gestionale potrebbe divenire particolarmente complessa. Ad ogni modo bisogna comprendere quali siano

[1] Un software gestionale rappresenta l'insieme dei software che automatizzano i processi di gestione all'interno delle aziende.

le effettive esigenze del proprio studio e, di conseguenza, ponderare quali sono le proposte dei singoli software gestionali.

Tutto ciò senza dimenticare la necessità che il proprio software gestionale sia dotato di un'interfaccia intuitiva e piacevole, che possa rendere più semplice ed efficace l'inserimento dei dati e possa consentire l'automazione delle attività che ne derivano. Il software deve essere, altresì, scalabile, permettendo, quindi, al commercialista di acquistare i vari moduli che lo compongono, in relazione al carico di lavoro.

Il professionista può, inoltre, dotare i propri clienti di soluzioni digitali adatte alle loro esigenze, cosicché questi ultimi sono più soddisfatti e autonomi, lo studio è più efficiente nell'elaborazione dei dati e si concentra sulle attività di consulenza.

In questo capitolo esaminiamo alcuni dei software commerciali che meglio soddisfano le caratteristiche espresse in precedenza. In particolare, illustriamo:

- Passcom;
- Integrato GB;
- Ago Infinity;
- AppStudio;
- FiscalGregan.

Alcuni di essi sono disponibili in commercio nella sola versione desktop o in tecnologia web; altri, invece, come AppStudio e FiscalGregan, sono stati sviluppati per il mercato mobile, e quindi sono disponibili in Android e iOS.

Questi software hanno tutti il medesimo scopo: supportare e coadiuvare il commercialista nella gestione di tutti i suoi clienti.

Un'altra importante funzione che mettono a disposizione è la condivisione documentale; lo studio dispone di uno spazio di archiviazione documentale completo e condivisibile con il cliente. È, perciò, possibile visualizzare, archiviare, ricercare e trasmettere documenti. In questo modo studio e cliente dispongono di dati e documenti sempre allineati e aggiornati in tempo reale.

Riassumendo, i software che esamineremo permettono di:

- essere più efficiente nel lavoro "tradizionale" dello studio, automatizzando molte attività;
- sfruttare al massimo le potenzialità dei nuovi strumenti digitali;
- rafforzare la relazione con i propri clienti, offrendo un servizio a 360 gradi.

1.2 Passcom

Passcom(Figura 1.1.) è un software per commercialisti creato dalla software house Passpartout, usufruibile anche tramite tablet o smartphone, che consente di gestire in modo integrato l'intero processo amministrativo e gestionale dello studio.

Figura 1.1. Logo di Passcom

Esso fornisce un sistema innovativo per una reale gestione collaborativa tra commercialista e cliente, poiché integra alle applicazioni dedicate allo studio, anche soluzioni per piccole e medie imprese.

Ogni azienda gestita dallo studio dispone di una stazione di lavoro per gestire il proprio flusso di dati, i quali vengono registrati automaticamente in Prima Nota. Le operazioni contabili inserite dall'azienda sono immediatamente visibili dalla postazione del commercialista, mentre, per le realtà più grandi o che necessitano di funzionalità gestionali più evolute, sono previsti livelli superiori per l'accesso al sistema con diversi tipi di estensione. Il software, inoltre, permette allo studio di comunicare in tempo reale con i propri clienti e di reperire direttamente le informazioni senza bisogno di trasferimenti di dati fisici o digitali, essendo un ambiente operativo unico e condiviso.

Passcom è un software che opera in cloud: commercialista e aziende si collegano al programma installato presso la Server Farm Passepartout tramite internet e lo utilizzano in base alle loro necessità.

Quest'ultima caratteristica riduce i costi delle infrastrutture, elimina i problemi di spazio per la memorizzazione dei dati, permette di automatizzare le procedure di backup e la gestione della sicurezza; inoltre, gli aggiornamenti normativi e le implementazioni funzionali sono immediatamente disponibili a tutti.

Privacy e sicurezza sono tutelate da efficaci strumenti di criptazione delle transazioni e da una gestione dei dati estremamente scrupolosa che rappresentano la soluzione ideale per la gestione di documenti e informazioni.

1.3 Integrato GB

Il software *Integrato GB*, ideato da un commercialista negli anni 2000, è una piattaforma completa per la gestione contabile, fiscale e di lavoro.

Il nome deriva dal fatto che i dati integrati sono condivisi dai vari moduli che compongono il programma (Figura 1.2.) e dai processi che lo governano.

Figura 1.2. Pacchetto di Integrato GB

La condivisione dello stesso dato da parte dei moduli che compongono il sofware, crea un collegamento logico e strutturale e quindi procedure automatiche e intuitive.

Ne consegue che i moduli sono perfettamente interconnessi tra loro e il flusso di dati avviene in tempo reale, senza richiedere caricamenti manuali o passaggi intermedi di import/export.

I moduli sono disponibili su un'unica interfaccia, quindi è semplice e veloce passare da un'applicazione ad un'altra. Mediante questo software il cliente inserirà i propri dati contabili all'interno di un'area riservata dedicata, abilitata dallo studio stesso e quest'ultimo potrà accedervi dalla propria postazione.

I documenti prodotti potranno essere facilmente condivisi tra lo studio e il cliente e quindi saranno sempre aggiornati; vengono inoltre predisposti backup periodici per garantire la massima sicurezza.

Questo software, inoltre, mette a disposizione un dominio dove lo studio può inserire i servizi offerti; successivamente tutto ciò verrà pubblicizzato su Facebook e Google, per attirare nuovi clienti.

Una delle funzioni più utili di Integrato GB è la possibilità di poter importare dal vecchio software gestionale tutta la vecchia documentazione, così da non perdere nulla e risparmiare tempo nel reinserire tutti i dati.

1.4 Ago Infinity

Ago Infinity(Figura 1.3.), ideato dalla Zucchetti, è un software completamente sviluppato in tecnologia Web, che fornisce strumenti per una gestione contabile e fiscale.

Figura 1.3. Logo di Ago Infinity

Grazie allo sviluppo in tecnologia Web, permette di condividere dati e documenti in tempo reale e accedervi da qualsiasi dispositivo, sia fisso che mobile.

Tutti i documenti sono memorizzati in automatico nell'archivio e facilmente consultabili da qualsiasi contesto del software. Essi sono, altresì, condivisibili sia tramite Web che per posta elettronica. Ciò deriva dal fatto che aziende e clienti operano all'interno dello stesso sistema, cosicché i dati vengano inseriti una sola volta e risultino essere sempre aggiornati.

Poiché il software lavora su un unico ambiente, la soluzione garantisce un accesso in sicurezza e permette di assegnare le attività ai diversi utenti dello studio e delle aziende, a seconda delle diverse competenze ed esigenze.

Un'altra conseguenza dell'ambiente unico è quello di permettere allo studio di affidare al cliente parte dell'attività amministrativa.

Ottima è l'assistenza fornita da Ago Infinity, poiché il gruppo ha previsto una doppia forma di supporto; la prima forma è rivolta alle richieste base, e sfrutta

l'intelligenza artificiale per fornire risposte basate sull'auto-apprendimento, a partire dalle esperienze precedenti; l'altra forma è utilizzata per richieste più complesse ed è basata su un servizio tradizionale.

1.5 AppStudio

App studio(Figura 1.4.), ideata dalla Leonida Consulting srl, permette di creare un'applicazione Android o IOS personalizzata per il proprio studio professionale.

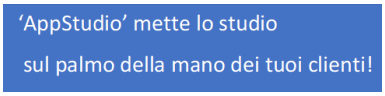

Figura 1.4. Slogan di AppStudio

Dopo aver acquistato l'applicazione arriverà un template da compilare, in cui verranno inseriti tutti i dettagli dello studio; sulla base di questi verrà sviluppata l'applicazione personalizzata. Una volta che l'applicazione sarà pronta verrà inserita nei market di Android e iOS permettendo ai clienti dello studio di procedere al download e all'installazione della stessa.

Per la parte amministrativa si potranno scegliere due soluzioni:

- se lo studio dispone di un dominio Web l'applicazione sarà installata all'interno del proprio Web server, lasciando tutti i dati nel server dello studio;
- se lo studio non dispone di un dominio Web, o se questo non è compatibile con l'applicazione, quest'ultima verrà installata all'interno del Web server dell'azienda venditrice del software.

Una volta sviluppate tutte le parti del software, lo studio potrà caricare tutti i dati dei clienti, i documenti e le comunicazioni all'interno del dominio web.

La sicurezza farà in modo che ogni cliente potrà vedere soltanto i dati collegati al proprio account.

La parte amministrativa, installata sul dominio Web, consta di un pannello di amministrazione, tramite al quale si potrà:

- gestire il database dei clienti;
- assegnare al cliente un nome utente ed una password per autorizzare l'ingresso nell'app;
- impostare una cartella per ogni cliente;
- caricare tutti i documenti;
- inviare circolari e newsletter;
- mandare comunicazioni di qualsiasi tipo.

Dall'applicazione il cliente potrà effettuare il login con nome utente e password ad esso riservatogli e avere accesso a tutti i documenti e alle comunicazioni inserite nella sua area(Figura 1.5.).

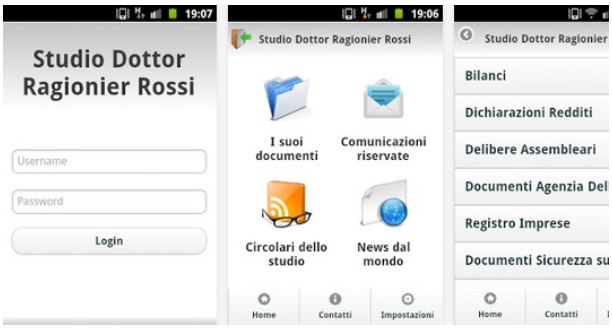

Figura 1.5. Screenshot AppStudio

1.6 FiscalGregan

FiscalGregan, sviluppata da Rainbow Srls e distribuita gratuitamente da AIACE (Associazione Italiana Assistenza Consumatore Europeo), è un'app scadenzario fiscale per i professionisti e per i loro clienti.

Essa garantisce al commercialista e ai suoi clienti la lista aggiornata delle principali scadenze fiscali, previdenziali e relative ai più importanti finanziamenti. Le scadenze potranno essere visualizzate tramite una vista ad agenda o da un comodo calendario (figura 1.6).

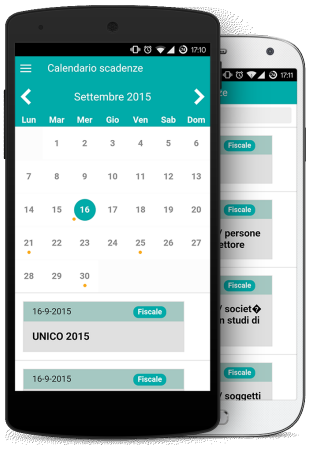

Figura 1.6. Emulatore con l'app FiscalGregan

La parte amministrativa (Figura 1.7.) è molto simile a quella vista nel paragrafo precedente, per AppStudio. Ne consegue che il commercialista avrà a disposizione una versione web per interfacciarsi con i clienti, inviare file, predisporre le proprie scadenze e chattare. In particolare, quest'ultima funzione è parte integrante dell'applicazione e avviene grazie ad una messaggistica privata per comunicare con tutti i propri clienti.

Per aggiungere i propri clienti basta inserire il loro indirizzo email.
In definitiva, quest'app permette:

- di avere scadenze sempre aggiornate;
- di organizzare le proprie attività in un unico ambiente;
- di gestire i propri clienti;
- di condividere file e documenti con i clienti;
- di usufruire di un sistema di messaggistica istantanea.

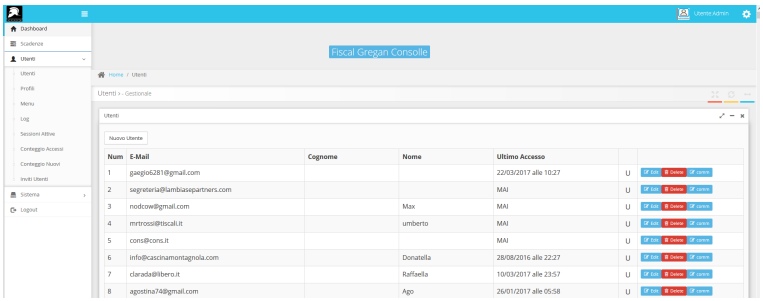

Figura 1.7. Screenshot della parte amministrativa di FiscalGregan

2

Android

In questo capitolo (ma sarebbe meglio chiamarlo viaggio) esploreremo il mondo Android, dalla sua nascita alla sua consacrazione nel mondo del mobile (e non solo), diventando quasi una costante di tutti i giorni. Vedremo da quali parti è costituita un'applicazione e cosa installiamo quando ne scarichiamo una nel nostro smartphone. In particolare, questo capitolo ci aiuterà ad apprendere la terminologia necessaria per capire al meglio i prossimi capitoli.

2.1 Cenni storici

Android nasce nel 2003, quando Andy Rubin, Rich Miner, Nick Sears e Chris White fondarono Android Inc., il cui scopo era quello di sviluppare, come specificò Rubin, *"dispositivi cellulari più consapevoli della posizione e delle preferenze del loro proprietario"*. Rubin decise di sfruttare un dominio acquistato alcuni anni prima, android.com, creando un software disegnato per dispositivi mobili e aperto a qualsiasi designer software.

Dopo aver investito tutti i propri averi, la società attirò l'interesse di alcuni investitori, in particolare Craig McCaw, uno dei pionieri della telefonia mobile. Durante le trattative, Rubin informò Larry Page (cofondatore di Google) della nuova potenziale partnership e, nel giro di poche settimane, Google completò l'acquisizione di Android per circa 50 milioni di dollari. La cifra rappresenta una stima, visto che quella ufficiale non è mai stata resa nota.

Dall'acquisizione, da parte di Google, ci vollero ancora oltre due anni di sviluppo, partendo da un kernel Linux, da parte del team di Rubin, rimasto nel progetto insieme a Miner e White.

Il 5 novembre 2007 avvenne la prima presentazione ufficiale, da parte dell'OHA[2].

Il debutto del nuovo sistema operativo avvenne più tardi, il 22 ottobre 2008, quando venne lanciato sul mercato HTC Dream (Figura 2.1). Da allora è iniziato un lunghissimo processo di aggiornamenti e miglioramenti, che hanno portato Android a essere il sistema operativo più diffuso al mondo, con aggiornamenti costanti che,

[2] L'OHA (Open Handset Alliance) è un consorzio di aziende tecnologiche come Google, HTC e Samsung, alcuni operatori di telefonia mobile e produttori di chip.

negli ultimi anni, hanno raggiunto una cadenza annuale, con release minori tra una versione e l'altra.

Figura 2.1. HTC Dream: il primo telefono Android

2.1.1 Versioni di Android

In maniera simile a quanto accade per Linux, anche Android ha adottato una convenzione alfabetica per identificare le varie versioni. Per le versioni precedenti alla 1.0 vennero utilizzate soltanto delle sigle. La prima release ad avere un nome in codice ufficiale è stata la 1.5, denominata Cupcake. Le Versioni 1.0 e 1.1 sono state, in seguito, definite Alpha e Beta, o Apple Pie e Banana Bread, ma, in realtà, non hanno mai avuto una denominazione ufficiale.

A partire dalla Versione 1.5, Google ha deciso di abbinare il nome di un dolce ad ogni versione, anche se nessuno è certo dell'origine di questa decisione, nata, a quanto pare, come un gioco interno al team di sviluppo.

Dopo un "calo" di fantasia con la Versione 9.0 di Android, chiamata, semplicemente, Android 9 Pie, il colosso californiano ha deciso, nell'agosto del 2019, di cambiare drasticamente rotta. Addio ai nomi dei dolcetti, difficili da comprendere per alcuni Paesi e culture. Al loro posto una soluzione molto più semplice, con il solo numero di versione. Il successore di Android 9 Pie è, dunque, conosciuto come Android 10, e le versioni successive saranno contrassegnate esclusivamente dal numero.

Nel seguito diamo uno sguardo alle versioni del sistema operativo Android (Figura 2.2) rilasciate nel corso degli anni, in particolare, fino ad oggi abbiamo:

- Android 1.5 "*Cupcake*": è la prima versione dotata di un nome ufficiale, rilasciata il 13 aprile 2009. Essa porta le API[3] 3, introducendo widget, la predizione del testo e il supporto per le tastiere personalizzate.
- Android 1.6 "*Donut*": è stata pubblicata 5 mesi dopo Cupcake. Non si tratta di un aggiornamento rivoluzionario, visto che le novità riguardano il supporto alle reti CDMA, la ricerca all'interno dello smartphone e nuove funzioni per la galleria.

[3] Le API sono un insieme di procedure atte all'espletamento di un dato compito. Spesso tale termine si riferisce alle librerie software di un linguaggio di programmazione.

Figura 2.2. I loghi delle diverse versioni Android, dalla 1.5 alla 9.0

Android Market, il precursore di Google Play Store, viene rinnovato, mettendo in evidenza le migliori applicazioni gratuite e quelle a pagamento, approfittando della rapida crescita del catalogo di prodotti di terze parti.

- Android 2.0 "*Eclair*": è il primo major update, rilasciato il 27 ottobre 2009. Android 2.0 Eclair, seguito il 12 gennaio 2010 da Android 2.1 Eclair (che propone solo alcuni bugfix) rappresenta un importante miglioramento nella storia del robottino verde.
  Vengono introdotte il supporto a Microsoft Exchange, la possibilità di gestire diversi account di posta elettronica, un restyling delle principali applicazioni, l'immissione vocale con il tasto del microfono, gli sfondi animati nella Home Page e il navigatore di Google Maps.
- Android 2.2 "*Froyo*": è stata rilasciata il 20 maggio 2010. Vengono introdotte le notifiche push, l'hotspot WiFi e il compilatore Dalvik JIT[4].
  Vengono implementate anche le azioni vocali con cui è possibile effettuare diverse operazioni. È, inoltre, possibile installare le applicazioni sulle memory card.
- Android 2.3 "*Gingerbread*": è stata rilasciata prima della fine del 2010. Viene introdotta la gestione della batteria, che permette di ottenere informazioni precise sul consumo di ogni singola funzione e applicazione. È con Android 2.3 Gingerbread che Google introduce gli *Easter Egg*[5], che possono essere rivelati cliccando ripetutamente la voce dedicata alla versione Android nelle impostazioni del dispositivo.
  La storia del primo Easter Egg di Android merita di essere raccontata: Dianne Hackborn, responsabile del framework team di Android, diventa amica dell'artista Jack Larson, che dipinge zombie di tutti i tipi. Hackborn pensa che sia divertente avere un quadro con un omino di panpepato zombie. il quadro fu talmente apprezzato da essere nascosto all'interno del sistema operativo, diventando una tradizione che prosegue tuttora.
- Android 3.0 "*Honeycomb*": è stata rilasciata il 27 gennaio 2011. È l'unica release dedicata esclusivamente ai tablet. L'interfaccia utente viene completamente

[4] Dalvik è una macchina virtuale ed è uno dei componenti di Android. È ottimizzata per sfruttare la poca memoria presente nei dispositivi mobili, consente di far girare diverse istanze della macchina virtuale contemporaneamente e nasconde al sistema operativo la gestione della memoria e dei thread.

[5] Un Easter egg (in italiano, letteralmente, uovo di Pasqua) in informatica è un contenuto, di solito, di natura faceta o bizzarra e innocuo, che i progettisti o gli sviluppatori di un prodotto, specialmente software, nascondono nel prodotto stesso.

ridisegnata e introduce il primo linguaggio di design di Google, chiamato Holo. Vengono introdotti i controlli per la navigazione sullo schermo, evitando di dover utilizzare i tasti fisici anche sui tablet. Arrivano le nuove Impostazioni rapide, che consentono di visualizzare l'ora, lo stato della batteria e quello della connettività in un'unica sezione.
Viene aggiunto il supporto ai processori multi core e viene migliorata la gestione del multitasking. Android 3.0 Honeycomb viene considerato da molti un flop per i tanti problemi presenti nella prima versione, marginalmente corretti con gli aggiornamenti 3.1 e 3.2.

- Android 4.0 "*Ice Cream Sandwich*": è stata rilasciata il 19 ottobre 2011. La grande novità è rappresentata dall'utilizzo del kernel 3.0 di Linux, ma arrivano anche le cartelle nella Home, i widget ridimensionabili, la possibilità di monitorare il consumo dei dati mobili, e Android Beam, che consente di condividere contenuti tra due smartphone utilizzando NFC.
  L'interfaccia è realizzata, ancora una volta, con il design Holo e vede, per la prima volta, l'utilizzo del font Roboto, sviluppato internamente e ottimizzato per gli schermi ad alta risoluzione. Il tasto "Menu" viene sostituito dal tasto "Recenti", che consente di visualizzare le applicazioni utilizzate di recente.
  La sezione Contatti viene spostata dall'applicazione "Telefono" in un'applicazione a sé stante. Arriva, anche, la possibilità di sbloccare lo smartphone inquadrando il proprio volto. Android 4.0 introduce, infine, la possibilità di acquisire gli screenshot.
- Android 4.1 "*Jelly Bean*": è stata rilasciata il 27 giugno 2012. Vengono introdotte le notifiche interattive, che permettono di rispondere subito a un messaggio o di svolgere altre operazioni. Nei mesi successivi verranno rilasciate, anche, le Versioni 4.2 e 4.3.
- Android 4.4 "*KitKat*": è stata rilasciata il 3 settembre 2013. Si tratta della prima versione di Android abbinata a un prodotto commerciale, i biscotti KitKat di Nestlé. Si completa lo sviluppo dell'interfaccia Holo, che raggiunge il suo massimo punto di successo prima di lasciare il campo a un nuovo design, che sarà introdotto con la versione successiva. Con questa versione arriva il comando vocale "Ok Google". Viene, inoltre, introdotto un nuovo design, che consente alle applicazioni di nascondere la barra di stato e quella di navigazione per sfruttare tutto lo schermo.
- Android 5.0 "*Lollipop*": è stato presentato ufficialmente il 25 giugno 2014 col nome di Android 5.0 L.
  La più grande novità è rappresentata da una nuova interfaccia utente, realizzata con uno stile piatto e semplice: denominata Material Design. Android 5.0 Lollipop rappresenta, indubbiamente, il più grande cambiamento da quando è stato rilasciato Android.
  Con Lollipop viene sostituito il compilatore Dalvik con il nuovo runtime ART; esso rappresenta, anche, la prima versione a supportare l'architettura a 64 bit.
  È, inoltre, possibile sbloccare lo smartphone con un dispositivo Bluetooth, come uno smartwatch, arrivano le notifiche Heads Up e viene modificata la schermata di blocco. Anche la schermata "Recenti" viene ridisegnata con una pila tridimensionale che mostra le applicazioni aperte.

- Android 6.0 "*Marshmallow*": è stata annunciata il 17 agosto 2015. Con Marshmallow viene aggiunta la gestione delle autorizzazioni, che consente di conoscere quali informazioni condividere con le applicazioni installate nel proprio dispositivo.
  La novità più significativa è Doze, che gestisce il risparmio energetico in standby; le connessioni in background sono limitate alle applicazioni ad alta priorità, ma è possibile escludere manualmente le applicazioni dalla gestione energetica.
  Ci sono novità anche per la parte hardware; arrivano il supporto nativo ai lettori di impronte digitali e quello allo standard USB Type-C per la ricarica e il trasferimento di dati.
- Android 7.0 "*Nougat*": è stata rilasciata ufficialmente il 22 agosto 2016. Viene introdotta la modalità Split Screen per gli smartphone, che consente di visualizzare due applicazioni nello schermo, mentre viene nascosta una funzione che consente di visualizzare alcune applicazioni in finestre galleggianti. Arriva una nuova tendina delle notifiche, che consente un raggruppamento per singola app e viene ulteriormente evoluto il sistema di gestione energetica Doze.
  È, inoltre, possibile impostare il consumo del traffico dati in background da parte delle singole applicazioni e viene introdotto il tasto "Hamburger", che permette di muoversi più rapidamente tra i sottomenù.
  Altre novità sono Project Svelte, che riduce il consumo di RAM, un nuovo compilatore JIT e Android For Work, per gestire lo smartphone durante l'orario di lavoro.
- Android 8.0 "*Oreo*": è stata annunciata il 21 agosto 2017. Ci sono molte novità della nuova versione, a partire da Project Treble, che consente di separare gli aggiornamenti del sistema operativo da quelli dell'hardware.
  Arrivano i canali nelle notifiche, che possono essere rimandate a un secondo momento con la funzione Snooze, la modalità picture-in-picture per eseguire alcune applicazioni in una finestra galleggiante, il supporto al Bluetooth 5, nuove limitazioni alle applicazioni in background, per ottimizzare i consumi della batteria, e le icone adattive.
- Android 9.0 "*Pie*": è stata annunciata il 6 agosto 2018. Vengono aggiornate la tendina delle notifiche, i quick toggle e le finestre pop up di sistema con un look più vivace e angoli arrotondati. È, inoltre, possibile scegliere come deve comportarsi il sistema all'inserimento del cavo USB e modificare gli screenshot.
- Android 10: è stata distribuita dal 3 settembre 2019 ed è la prima versione che abbandona il nome del dolcetto, passando, dunque, a una numerazione più semplice da ricordare. La svolta è in un certo senso epocale, e contribuisce a dare un'immagine più seria al sistema operativo. Non ci sono tantissimi stravolgimenti rispetto alle versioni precedenti quanto, piuttosto, una serie di miglioramenti.
- Android 11: è stata distribuita dall'8 settembre 2020. Questa versione presenta diverse novità, alcune più a livello di grafica, altre a carattere funzionale, quindi non immediatamente visibili.
  Tra le cose più evidenti abbiamo: notifiche separate per le app di messaggistica, denominate notifiche a bolle, che possiamo spostare dove vogliamo, controlli multimediali sotto ai quick toggle, i permessi "solo una volta" e auto-resettanti, nel caso in cui un'app non venga aperta per molto tempo.

2.2 Android OS

Prima di definire nel dettaglio l'architettura Android, è utile dare la definizione di sistema operativo. Un sistema operativo è un software di base, che gestisce le risorse hardware e software della macchina, fornendo servizi di base ai software applicativi.

Per rendere possibile il lavoro di gestione, il sistema operativo ha una struttura diversa rispetto a quella della maggior parte degli altri programmi. Esso è un programma strutturato in diversi livelli. In quello inferiore, ossia quello più lontano dall'interfaccia utente, si trova il kernel, ed è il primo a caricarsi in memoria.

Sul kernel si basano altri livelli, che si allontanano sempre di più dall'interazione con l'hardware. Ogni livello comunica, quindi, solo con quello che si trova a livello superiore o inferiore. Infine, nella parte superiore, si trova l'interfaccia utente, ossia l'interfaccia presente tra utente e software. Se un utente svolge un'attività, questa istruzione viene inoltrata ai vari livelli, fino al raggiungimento della posizione corretta, per esempio nel processore.

2.2.1 Architettura di Android

Android è un sistema operativo e, come tale, mantiene la struttura tipica di questi sistemi, ovvero la suddivisione in livelli (Figura 2.3)

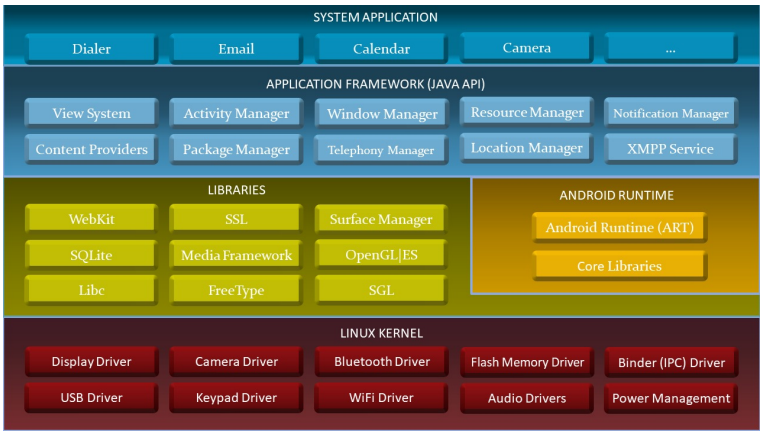

Figura 2.3. Suddivisione in livelli di Android OS

Com'è possibile vedere dalla figura 2.3, al livello più basso vi è il kernel Linux.

La necessità era 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. Esso fornisce, quindi, tutte le feature di sicurezza, la gestione della memoria, la gestione dei processi, la power management e la sicurezza di avere alla base un sistema affidabile e testato.

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

Di notevole importanza è 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.

Nel livello superiore ci sono un insieme di librerie native, realizzate in C e C++, che rappresentano il core vero e proprio di Android; tra queste abbiamo:

- *Surface Manager*: è un componente fondamentale in quanto ha la responsabilità di gestire le View, ovvero ciò di cui un'interfaccia grafica è composta. Il compito del Surface Manager è di coordinare le diverse finestre che le applicazioni vogliono visualizzare sullo schermo.
- *Open GL ES*: essa permette di utilizzare la grafica 2D e 3D all'interno della stessa applicazione.
- *Scalable Graphics Library (SGL)*: è una libreria 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.
- *Media Framework*: è un componente in grado di gestire i diversi CODEC per i vari formati di acquisizione e riproduzione audio e video.
- *FreeType*: è utilizzato per la gestione del font. 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, di essere transazionale.
- *WebKit*: è un motore di rendering[6] per browser web utilizzato per il rendering delle pagine web.
- *Secure Socket Layer (SSL)*: è la libreria che si occupa della sicurezza, attraverso la gestione dei Secure Socket Layer[7].
- *Libc*: si tratta di un'implementazione della libreria standard `C libc`, ottimizzata per i dispositivi basati su Linux embedded, come Android.

Sullo stesso livello, oltre alle librerie, è presente l'Android Runtime.

Quando un'app Android viene compilata sull'IDE, essa viene convertita in un formato bytecode intermedio (denominato formato DEX). Quando l'applicazione viene, successivamente, caricata sul dispositivo, l'ambiente di runtime converte il bytecode in linguaggio macchina, così da essere comprensibile per il processore del dispositivo.

Fino alla versione Android 4.4 Kitkat, l'ambiente di runtime utilizzava il compilatore Dalvik, basato sulla compilazione Just-In-Time (JIT). Quando un utente apriva un'app, Dalvik la compilava in tempo reale e questa veniva eseguita. La compilazione fatta dallo smartphone, comportava un leggero ritardo nell'apertura. Per far fronte a ciò, Google ha deciso, con Android 5.0 Lollipop, di introdurre quello che viene chiamato ART, Android Runtime.

[6] Un motore di rendering, in informatica, ed in particolare nella computer grafica, è un componente hardware o software che interpreta delle informazioni in ingresso, codificate secondo uno specifico formato, e le elabora creandone una rappresentazione grafica.

[7] I Secure Socket Layer sono dei protocolli crittografici che permettono una comunicazione sicura, dalla sorgente al destinatario, su reti TCP/IP.

Il compilatore ART, a differenza di Dalvik, utilizza una compilazione Ahead Of Time (AOT), il quale compila le applicazioni durante la prima installazione.

Nonostante fossero ridotti i tempi di caricamento, Google non era soddisfatta a pieno delle prestazioni. Ha così introdotto, con Android 7.0 Nougat, un'evoluzione di ART, che unisce i benefici di ART a quelli di Dalvik. Con questa tecnologia, le app non vengono più compilate interamente durante l'installazione, ma vengono compilate dinamicamente solo le componenti utilizzate quando si usa l'app. Al successivo avvio dell'app, verrà eseguito il codice già compilato e, se sarà necessario, verranno compilate nuove parti.

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

- *Activity Manager*: è lo strumento fondamentale che gestisce il ciclo di vita delle singole schermate dell'applicazione. Questo componente ha la responsabilità di organizzare le varie schermate di un'applicazione in un unico stack, a seconda dell'ordine di visualizzazione delle stesse sullo schermo.
- *Package Manager*: gestisce il ciclo di vita delle applicazioni nei dispositivi.
- *Window Manager*: è un servizio responsabile della gestione delle finestre; esso stabilisce quali finestre sono visibili e come sono disposte sullo schermo. Inoltre, esegue automaticamente transizioni di finestra e animazioni quando si apre o si chiude un'app o si ruota lo schermo.
- *Telephony Manager*: è responsabile delle funzionalità caratteristiche di un telefono, come la semplice possibilità di iniziare una chiamata o di verificare lo stato della chiamata stessa.
- *Content Provider*: è un componente che gestisce la condivisione di informazioni tra i vari processi.
- *Resource Manager*: è un componente che ha la responsabilità di gestire le risorse che, oltre al codice, compongono un'applicazione. Per le risorse, come avviene già per il codice, esiste un processo di trasformazione delle stesse in contenuti binari, ottimizzati per il loro utilizzo all'interno di un dispositivo.
- *View System*: è il gestore del rendering dei componenti dell'interfaccia grafica, nonché della gestione degli eventi associati.
- *Location Manager*: è un componente che mette a disposizione le API necessarie per creare applicazioni che gestiscono la localizzazione del dispositivo.
- *Notification Manager*: è un componente che fornisce diversi strumenti alle app per inviare notifiche al dispositivo.
- *XMPP Service* (Extensible Messaging and Presence Protocol): è un insieme di protocolli di messaggistica istantanea basato su XML.

L'ultimo livello, il System Application, è rappresentato dalle applicazioni presenti sul telefono. A questo livello verranno installate le app create dall'utente o scaricate dal Play Store.

2.3 File APK

Quando scarichiamo un'app Android dallo store di Google, questa si presenta come un file con estensione `.apk`. Ciò significa che noi scarichiamo un Android Package (APK) (Figura 2.4), ovvero un archivio, simile al file ZIP, che contiene al proprio interno tutto il codice e tutte le risorse utilizzate nello sviluppo dell'app.

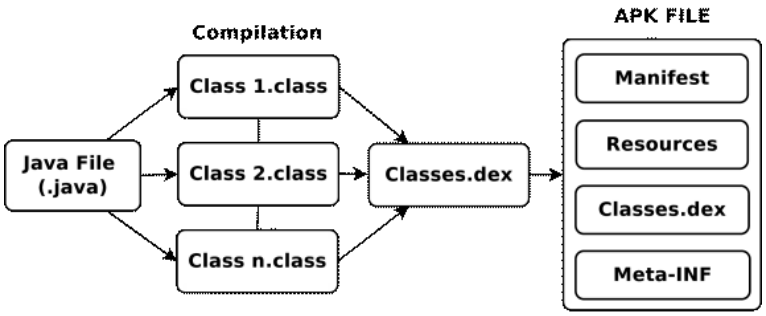

Figura 2.4. Dal file Java al file .apk

Dalla Figura 2.4 possiamo osservare che il file `.apk` è composto dai seguenti file:

- *Android Manifest* (Figura 2.5): è un file XML che raccoglie informazioni basilari sull'app; tali informazioni necessarie al sistema per far girare qualsiasi porzione di codice della stessa.

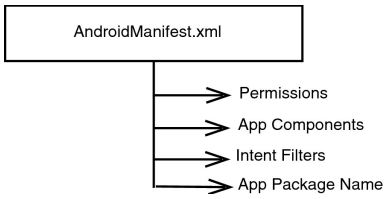

Figura 2.5. File Android Manifest

Tra le informazioni contenute nel manifest ci sono: il nome del package dell'app, la versione minima dell'API Android richiesta, le componenti utilizzate e tutti i permessi richiesti dall'app.

- *Resources*: sono i file aggiuntivi di un'app, come immagini, stringhe dell'interfaccia utente e layout. In altre parole, rappresentano il contenuto statico utilizzato nel codice.
- *Meta-INF*: è una directory che contiene le informazioni sul manifest e altri metadati relativi al pacchetto java trasportati dall'apk.
- *Classes.dex*: contiene il codice che viene, infine, eseguito dal runtime Android. Ogni apk dispone di un singolo file, che fa riferimento a qualsiasi classe o metodo usato all'interno di un'app. Essenzialmente, tutti i component (Figura 2.6)

utilizzati all'interno del codice verranno trasformati in byte all'interno di un file `.dex`, che può essere eseguito come app Android.

2.3.1 I building block di un'applicazione

Come detto nella sezione precedente, all'interno del file `.apk` ci sono i classes.dex. Essi sono una traduzione, in formato `.dex`, dei component utilizzati all'interno dell'app.

I component (Figura 2.6.) rappresentano i building block di un'applicazione Android; in altre parole, essi rappresentano gli elementi base che costituiscono un'applicazione.

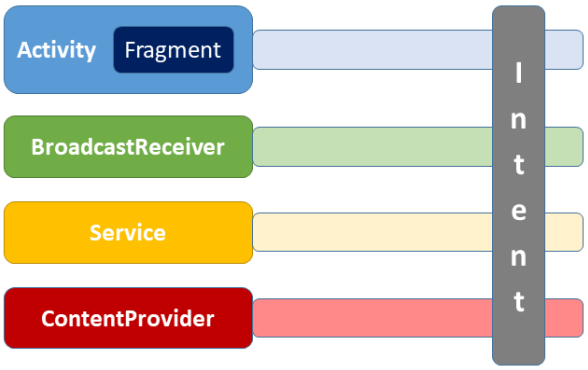

Figura 2.6. Component di un'app Android

Un'app è costituita da diversi tipi di component, ciascuno specializzato in una particolare tipologia di attività. Abbiamo le Activity e i Fragment per le interfacce utente, i Content Provider per lo scambio di informazioni, i Service per le attività in background, ed i Broadcast Receiver per la reazione agli eventi. I Fragment non fanno parte ufficialmente dei component, in quanto devono essere ospitati all'interno di un'Activity.

Anche gli Intent non fanno parte dei component, ma servono per far comunicare questi ultimi tra di loro.

Quasi tutti i component necessari ad un'applicazione vanno menzionati nel file Android Manifest.

Ogni component segue un ciclo di vita che è sempre controllato dal sistema; in base alla configurazione ricevuta, alcuni component dell'app possono entrare in azione in momenti diversi. Lo stato dell'applicazione può essere distrutto, in quanto Android è un sistema che nasce per lavorare con poche risorse e, in caso di necessità, ogni component può essere distrutto con conseguente perdita di informazioni. Questa è una tematica molto importante e determina il ruolo, particolarmente critico, che la persistenza dei dati assume in Android.

Activity

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

Un'app è, in genere, costituita da più schermate, che sono legate tra loro in modo lasco. L'Activity principale di un'app, ovvero la prima schermata visualizzata all'avvio della stessa, prende il nome di Main 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 (Figura 2.7.).

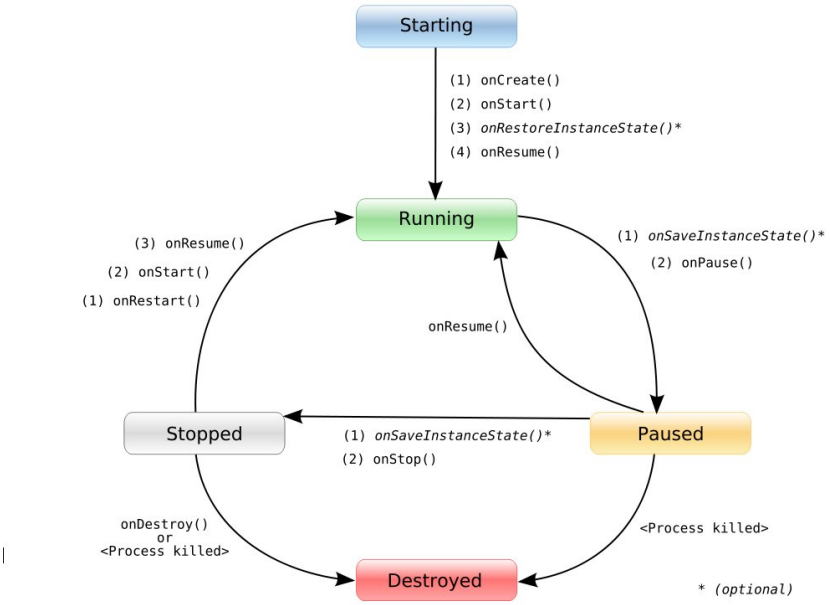

Figura 2.7. Ciclo di vita delle Activity

Il ciclo di vita è gestito da Android stesso, utilizzando diversi metodi. Questi possono subire un override[8] per permettere allo sviluppatore di decidere quali azioni intraprendere a seguito di un particolare stato.

Ecco una breve descrizione dei metodi:

- `onCreate(Bundle)`: è richiamato quando l'Activity viene avviata per la prima volta. Il metodo accetta un parametro, che può essere nullo, oppure può ritornare le informazioni salvate precedentemente tramite metodo `onSaveInstanceState()`.
- `onSaveInstanceState(Bundle)`: Android invoca questo metodo per salvare alcune informazioni di stato dell'Activity. Normalmente, non serve eseguire l'override di questo metodo, perché Android salva per noi tali informazioni.

[8] Un override è la riscrittura di un metodo.

- `onRestoreInstanceState(Bundle)`: richiamato solo se alcuni stati dell'activity sono precedentemente stati salvati tramite `onSaveInstanceState()`.
- `onStart()`: indica che l'Activity sta per essere visualizzata.
- `onResume()`: è richiamato quando l'Activity è pronta ad interagire con l'utente; è qui che conviene gestire gli elementi multimediali inseriti nell'app.
- `onPause()`: viene richiamato quando l'Activity sta per andare in background; ciò succede, normalmente, perché è stata avviata un'altra Activity che si prepara a prendere il foreground. In questo metodo conviene salvare tutti i dati persistenti, per poi ripristinarli sull'`onResume()`.
- `onStop()`: richiamato quando l'Activity non è più visibile all'utente.
- `onRestart()`: se è richiamato, vuol dire che l'Activity sta per essere riavviata.
- `onDestroy()`: richiamato prima che l'Activity venga distrutta.

Il sistema Android mantiene in primo piano (foreground) una sola schermata alla volta, portando in secondo piano (background) le altre.

Ogni volta che viene avviata una nuova Activity, quella precedente viene interrotta; tuttavia, essa viene mantenuta in memoria dal sistema. La nuova Activity assume lo stato di foreground, quella precedente passa in background. Si può dire che le Activity sono impilate come un mazzo di carte: il tasto "Back" permette di passare da una schermata all'altra, portando in background quella attuale e assegnando il foreground alla precedente. Questa sorta di history delle Activity viene chiamata "Activity stack" e viene gestita dall'Activity Manager. L'activity stack segue la logica LIFO (Last In First Out); perciò, premendo il tasto "Back", l'Activity corrente viene eliminata dalla memoria e l'Activity precedente viene richiamata (Figura 2.8.).

Fragment

I Fragment costituiscono uno dei più importanti elementi per la creazione di una interfaccia utente Android.

Un Fragment è una porzione di Activity. Esso, però, non deve essere visto come una semplice sezione del layout. Al contrario, può essere definito più come una specie di sub-Activity, con un suo ruolo funzionale ed un suo ciclo di vita.

Un Fragment non può vivere senza un'Actvity; tipicamente nelle app ci sono più Fragment che si alternano nel layout, mentre di Activity ne sarà sufficiente una.

Siccome il Fragment dipende dall'Activity che lo ospita, anche il suo ciclo di vita dipenderà da quello dell'Activity (Figura 2.9.).

Dalla Figura 2.9 possiamo distinguere i metodi con cui gestire il ciclo di vita di un Fragment:

- `onAttach()`: segnala il momento in cui il Fragment scopre l'Activity di appartenenza. È opportuno evidenziare che, a quel punto, l'Activity non è stata ancora creata; quindi si può solo conservare un riferimento ad essa, ma non interagirvi.
- `onCreate()`: si occupa della creazione del Fragment in quanto componente.
- `onCreateView()`: è il metodo che si occupa della creazione del Fragment.
- `onActivityCreated()`: segnala che la creazione dell'Activity è stata completata; da quel momento in poi, si può interagire con essa in tutto e per tutto.
- `onDestroy()`: è chiamato quando il Fragment non è più in uso.

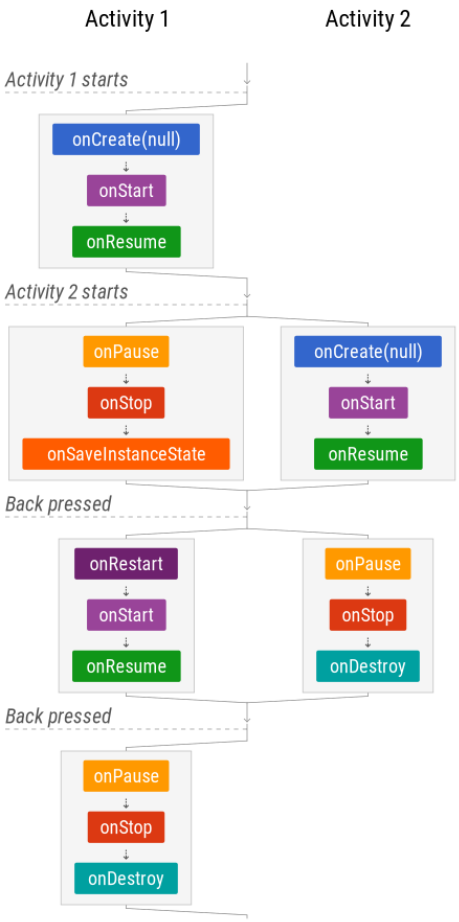

Figura 2.8. Esempio di interazione tra due Activity

- `onDestroyView()`: è chiamato quando il layout, precedentemente creato da `onCreateView()`, è stato scollegato dal Fragment.
- `onDetach()`: è chiamato quando il Fragment non è più associato alla sua Activity.

Gli altri metodi sono analoghi a quelli dell'Activity.

In un'applicazione possiamo trovare due tipi di Fragment; ovvero statico e dinamico. I primi non possono essere gestiti dinamicamente in fase di esecuzione, a differenza di quelli dinamici; ciò significa che, una volta che il Fragment ha preso posto nell'Activity, esso non può cambiare dinamicamente.

I Fragment dinamici permettono di dare all'utente un'esperienza fluida e veloce all'interno di una stessa Activity. Questo è impossibile da ottenere se si salta da un'Activity all'altra.

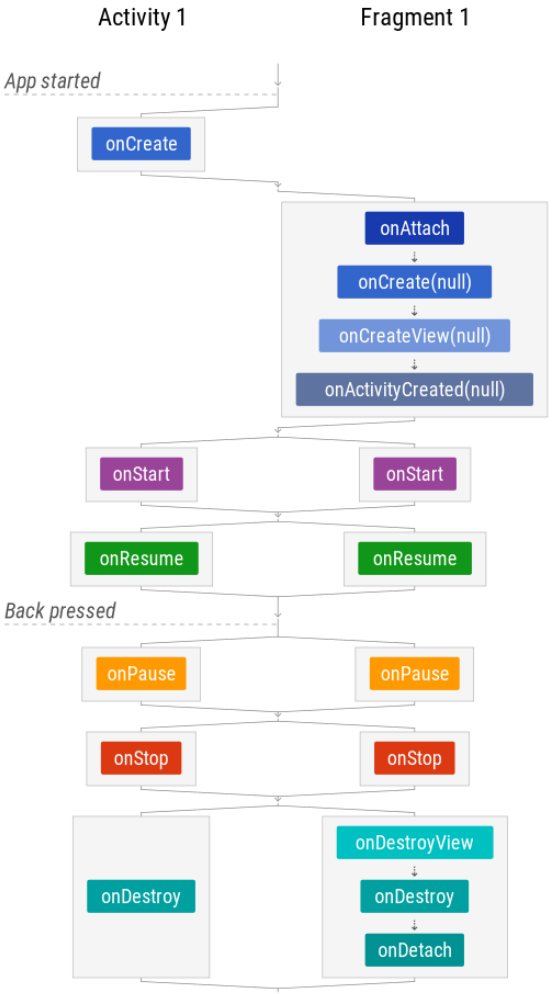

Figura 2.9. Ciclo di vita di un Fragment

Broadcast Receiver

Un Broadcast Receiver (Figura 2.10) è un componente che reagisce ad un invio di messaggi a livello di sistema, appunto in broadcast, con cui Android notifica l'avvenimento di un determinato evento, ad esempio l'arrivo di un SMS o di una chiamata.

Questi componenti, come si può immaginare, sono particolarmente utili per la gestione istantanea di determinate circostanze speciali. I Broadcast Receiver non utilizzano interfaccia grafica, sebbene possano inoltrare notifiche alla barra di stato per avvisare l'utente dell'avvenimento.

Ci sono due tipi di Broadcast Receiver; ovvero ordinati e normali. I Broadcast ordinati, denominati anche sincroni, sono inviati secondo un ordine corretto. Questo ordine è dettato in base alla priorità attribuita ad un evento.

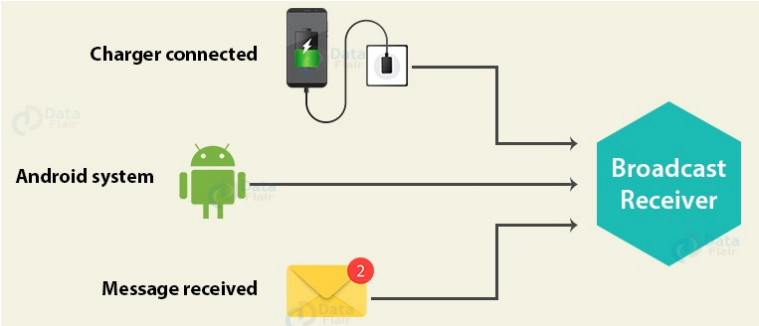

Figura 2.10. Esempio di Broadcast Receiver

I Broadcast normali, viceversa, sono asincroni e sono inviati secondo un ordine casuale. Essi possono essere inviati in modo ordinato, oppure tutti insieme; il sistema, però, per evitare sovraccarichi, può scegliere di mandarli uno alla volta.

Un Broadcast Receiver per funzionare ha bisogno di essere registrato. Android mette a disposizione due metodi per registrare un Broadcast Receiver; ovvero all'interno del file Android Manifest oppure definendolo all'interno del codice.

Service

Un Service svolge un ruolo, se vogliamo, opposto all'Activity. Infatti, rappresenta un'attività generalmente lunga e continuata, che viene svolta interamente in background, senza bisogno di interazione diretta con l'utente. Ad esempio, un'app che permette di avviare un audio, che non si interrompa alla chiusura della sua interfaccia, con tutta probabilità basa il suo funzionamento su un Service.

I loro usi sono i più disparati e, a seconda dei casi, l'utente potrebbe anche non notare il loro avvio, ottenendone però i benefici.

Da una punto di vista strutturale, i Service sono di due tipologie; ovvero started e bounded.

- Un Service è started quando un'app ha bisogno di svolgere un'attività in background, mirata ad uno scopo specifico, fino al suo completamento. I Service started continuano a essere in esecuzione anche quando l'utente non interagisce con l'app. Quando si utilizza questa tipologia di Service, è necessario visualizzare una notifica, in modo che gli utenti siano consapevoli che il Service è in esecuzione. Questa notifica non può essere chiusa a meno che il Service non venga arrestato o rimosso. La riproduzione di un audio in background rientra in questa casistica.
- I Service bounded vivono in una modalità client-server e vengono attivati solo nel caso in cui un'altra app abbia bisogno di connettersi ad essi. Tali Service vengono interrotti nel momento in cui non vi sono più client ad essi collegati. I Service bounded svolgono il ruolo di supporto ad altre applicazioni; essi non rischiano, pertanto, di essere "dimenticati" in background come potrebbe succedere per i Service started.

I due tipi di Service non sono molto diversi. Ciò che li contraddistingue è il metodo con cui vengono avviati e i metodi presenti al loro interno (Figura 2.11); ne consegue, che uno stesso Service può essere utilizzato sia in modalità started che bounded.

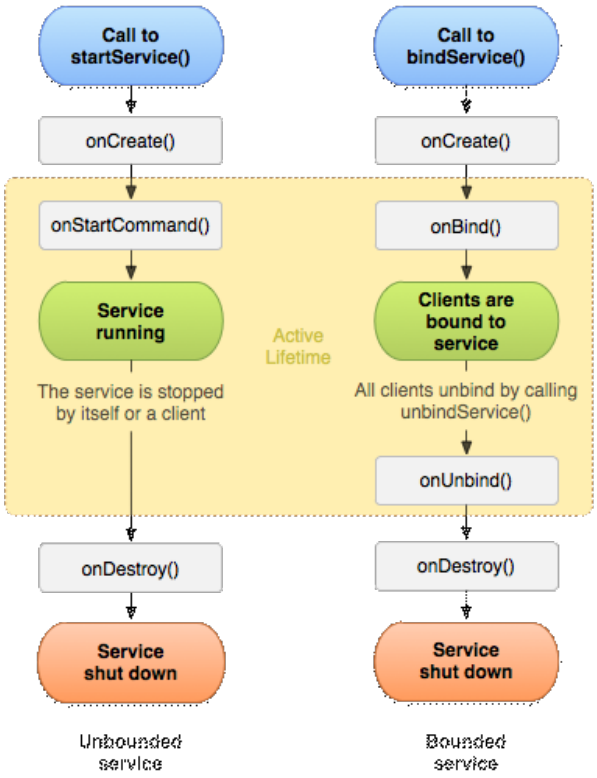

Figura 2.11. Ciclo di vita dei Service

Come si evince dalla Figura 2.11, entrambi i cicli di vita iniziano e terminano con i metodi `onCreate()` e `onDestroy()`. Le differenze si concentrano nella fase in cui il Service viene attivato; mentre l'avvio di un Service started viene notificato per mezzo di `onStartCommand()`, l'inizio e la fine della connessione con un Service bound viene segnalato dai metodi `onBind()` e `onUnbind()`.

Content Provider

Un Content Provider nasce con lo scopo di condividere i dati tra le applicazioni.

Questi componenti permettono di condividere, nell'ambito del sistema, contenuti custoditi in un database, su file o reperibili mediante accessi in Rete. Tali contenuti potranno essere usati da altre applicazioni senza invadere lo spazio di memoria. L'idea principale è quella di incapsulare i dati, che si desidera condividere, e fornire meccanismi per accedervi in modo sicuro, tramite un'interfaccia Content Resolver

(Figura 2.12). Prima il client effettua una richiesta al Content Resolver, poi quest'ultimo comunica con il Content Provider responsabile, esegue l'azione richiesta e restituisce i risultati.

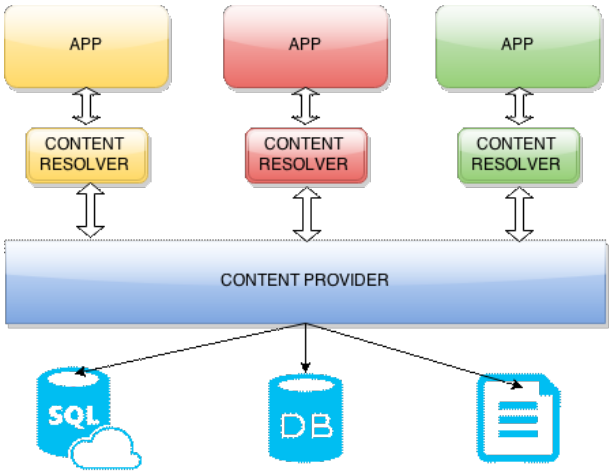

Figura 2.12. Funzionamento dei Content provider

È possibile scegliere diversi livelli di sicurezza del Content Provider; in particolare, è possibile:

- limitare l'accesso a un Content Provider esclusivamente all'interno della stessa applicazione;
- concedere l'autorizzazione ad altre applicazioni;
- configurare autorizzazioni diverse per la lettura e la scrittura dei dati.

Intent

I component dell'applicazione Android possono connettersi tra di loro, o ad altre applicazioni. Tale connessione avviene grazie alla presenza degli Intent.

Gli Intent (Figura 2.13) sono messaggi asincroni, che consentono di interagire con i component della stessa applicazione e con quelli forniti da altre applicazioni. Ad esempio, un'Activity può avviare un'Activity esterna per scattare una foto.

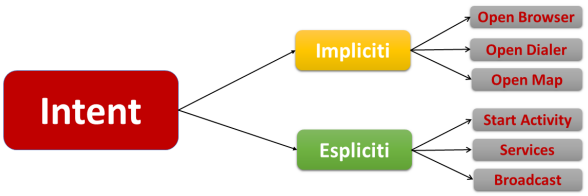

Figura 2.13. Classificazione degli Intent

Essi sono uno strumento molto duttile (gli utilizzi più comuni possono riguardare l'avvio un'Activity (Figura 2.14), o di un Service, oppure l'invio di un messaggio in broadcast.

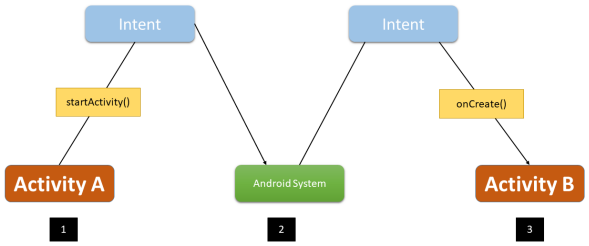

Figura 2.14. Avvio di un'Activity tramite l'Intent

Inoltre, com'è possibile vedere dalla Figura 2.13, gli Intent possono essere:

- Impliciti: in questo caso essi non specificano quale component deve essere attivato, ma si limitano ad indicare quale azione deve essere svolta. La loro invocazione si estrinseca spesso nell'apertura di una finestra di dialogo, che chiede all'utente quale app, installata nel dispositivo, vuole si apra per completare l'azione richiesta.
- Espliciti: viene dichiarato esplicitamente quale component dovrà essere attivato. Solitamente, gli Intent espliciti vengono utilizzati nell'apertura di una nuova Activity.

Un altro aspetto molto utile degli Intent è che essi, nel recapitare il messaggio al component destinatario, hanno a disposizione una specie di "bagagliaio". In questo bagagliaio, essi possono custodire diversi dati, chiamati Extras, che possono essere recuperati dal component ricevente.

La gestione degli Extras negli Intent funziona in maniera simile ad una struttura dati a mappa; con dei metodi `put` viene inserito un valore etichettato con una chiave e con i corrispondenti metodi `get` viene prelevato il valore, richiedendolo mediante la chiave di riconoscimento.

2.4 User Interface Android

Un'Activity ha bisogno di un volto, di un suo aspetto grafico; questo volto prende il nome di User Interface (UI).

Esistono tre modi per creare una User Interface in Android; ovvero procedurale, dichiarativo e in design mode.

Nel primo caso, si effettua l'implementazione dell'interfaccia grafica nel codice Java; col metodo dichiarativo, invece, possiamo creare un'interfaccia utente utilizzando il linguaggio XML. Entrambi questi metodi sono validi, ma il più indicato è, sicuramente, quello dichiarativo; in quanto è più leggibile e più strutturato.

Il linguaggio XML è un linguaggio di Markup[9], quindi della stessa tipologia di HTML.

Android, inoltre, permette anche un approccio ibrido, in cui si crea un'interfaccia in modo dichiarativo e la si controlla in modo procedurale.

L'approccio in design mode è totalmente affidato alla potenza degli ambienti di sviluppo. Questo approccio ha ormai raggiunto livelli encomiabili, mettendo a disposizione del programmatore un ambiente completo per creare le interfacce utente. Tuttavia, la conoscenza di XML e delle classi Java resta un elemento imprescindibile per padroneggiare bene lo sviluppo dell'app.

2.4.1 Componenti di una User Interface

L'interfaccia utente per un'app Android viene costruita utilizzando una gerarchia di layout e widget. I Layout sono oggetti ViewGroup che, a loro volta, rappresentano una sottoclasse delle View (Figura 2.16). Essi sono contenitori e, al loro interno, possono ospitare sia altri Layout sia Widget (Figura 2.15).

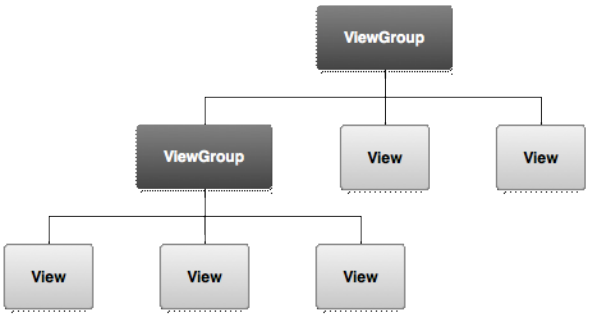

Figura 2.15. Gerarchia di layout e widget

Gli oggetti di tipo ViewGroup controllano il modo in cui le componenti figlio sono posizionate sullo schermo.

I widget sono oggetti di tipo View. Essi rappresentano i componenti dell'interfaccia utente, come pulsanti e caselle di testo, con cui l'utente può interagire.

Layout

Android mette a disposizione diversi tipi di layout (Figura 2.17). Essi sono:

- *Absolute Layout*: dà la possibilità di specificare le posizioni esatte, utilizzando le coordinate X e Y dei relativi elementi figlio. Essi sono poco usati a causa della poca flessibilità.
- *Relative Layout*: consente di specificare la modalità di posizione delle View figlio. La posizione di ogni View può essere specificata in relazione agli elementi di pari livello o rispetto all'elemento padre.

[9] Il Markup è il processo di utilizzo di codici, denominati tag, per definire la struttura e l'aspetto visivo dei dati.

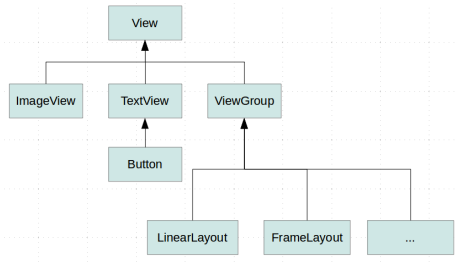

Figura 2.16. Gerarchia della classe View

Figura 2.17. Esempi di Layout

- *Linear Layout*: allinea le View figlio orizzontalmente o verticalmente.
- *Grid Layout*: visualizza gli elementi in una griglia scorrevole bidimensionale.
- *Scroll View*: viene utilizzato quando il contenuto non si adatta allo schermo.
- *Table Layout*: raggruppa le View in righe e colonne.
- *Frame Layout*: viene utilizzato per bloccare un'area sullo schermo e visualizzare solo un singolo elemento figlio. Di solito, è utilizzato per visualizzare singoli Fragment sullo schermo.
- *Constraint Layout*: consente di posizionare e ridimensionare le View in modo flessibile. Esso è simile al RelativeLayout, in quanto tutte le visualizzazioni sono disposte in base alle relazioni tra le View di pari livello e il layout principale, ma è più flessibile e più facile da utilizzare. Il Constraint Layout è molto utile perché permette di adattare l'interfaccia utente dell'app alle diverse dimensioni degli schermi degli smartphone.

Widget

La classe View mette a disposizione diversi Widget (Figura 2.18), tra cui abbiamo:

- *TextView*: visualizza un testo all'utente.

- *EditText*: è una casella in cui è possibile inserire e modificare un testo.
- *CheckBox*: è una casella di controllo formata da un pulsante a due stati, che può essere selezionato o deselezionato.
- *RadioButton*: è un pulsante a due stati: quando è deselezionato, l'utente può fare clic su di esso per selezionarlo. Tuttavia, contrariamente al CheckBox, un RadioButton non può essere deselezionato dall'utente una volta selezionato.
- *Spinner*: fornisce un set di valori contenuti in un menu a discesa, che si apre quando si fa click su di esso. Nello stato di default, uno Spinner mostra sempre un valore selezionato, che cambia quando si fa click sulle opzioni disponibili all'interno del menu.
- *ImageButton*: è un pulsante con un'immagine che può essere cliccato dall'utente.
- *ImageView*: visualizza un'immagine; quest'ultima si trova, solitamente, nella cartella Resources dell'app.
- *Button*: è un semplice pulsante che permette di avviare un evento.
- *ProgressBar*: indica lo stato di avanzamento di un'operazione.
- *Switch*: è un widget di commutazione a due stati. L'utente può semplicemente toccare il widget per attivarlo o disattivarlo, come se fosse un interruttore On/Off.

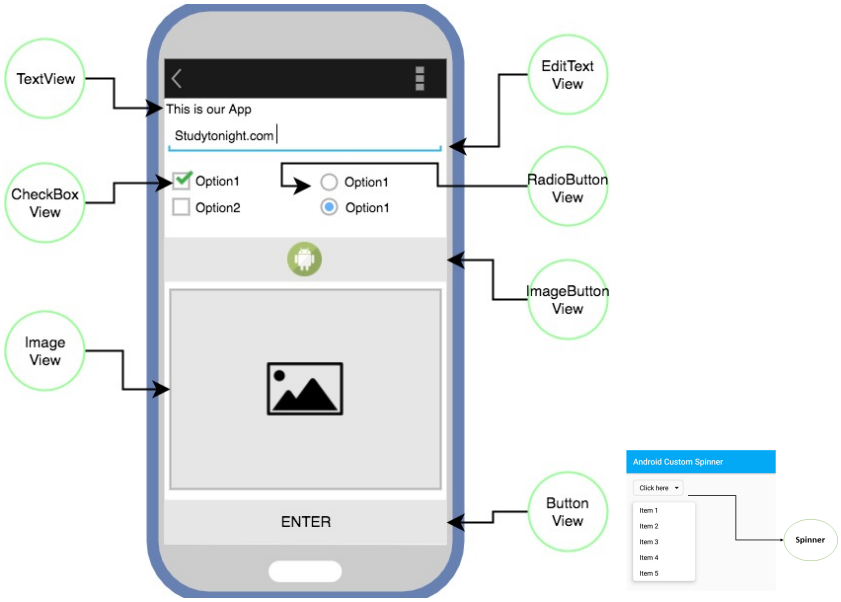

Figura 2.18. Esempi di Widget

2.5 Android Studio

Android Studio (AS) (Figura 2.19) è l'IDE ufficiale per il sistema operativo Android, basato sul software IntelliJ IDEA, progettato specificamente per lo sviluppo Android.

Figura 2.19. Logo di Android Studio

Android Studio è stato annunciato il 16 maggio 2013 e la prima versione stabile è stata rilasciata nel dicembre 2014.

AS permette di utilizzare due differenti linguaggi di programmazione: Java e Kotlin. Il 7 maggio 2019, Kotlin ha sostituito Java come linguaggio preferito da Google per lo sviluppo di app Android. Nonostante ciò, Java è ancora supportato.

Android Studio sfrutta come compilatore Gradle, il quale, consente una compilazione personalizzata per generare più varianti di compilazione in base al dispositivo Android.

Inoltre, AS dà la possibilità di definire il contesto applicativo su cui vogliamo sviluppare la nostra app; infatti, possiamo scegliere tra Phone e Tablet, Wear OS, TV, Android Auto e Android Things.

Un'altra caratteristica di Android Studio è la suddivisione in moduli, ciascuno dei quali può svolgere un ruolo diverso.

Come anticipato nel paragrafo precedente, l'interfaccia utente può essere sviluppata anche in design mode. Tale modalità permette di trascinare gli elementi dell'interfaccia utente in un editor di progettazione visiva. Quest'ultimo può visualizzare in anteprima il layout su diversi dispositivi e versioni Android, ed è possibile, altresì, ridimensionare dinamicamente il layout per assicurarsi che funzioni bene su diverse dimensioni dello schermo.

Infine, Android Studio permette di utilizzare un emulatore Android, qualora non fosse disponibile provare la propria app sul dispositivo reale. È possibile installare differenti emulatori a seconda della versione del sistema operativo Android e della dimensione dello schermo.

Quest'ultima caratteristica è molto utile perché permette di provare l'app su diversi dispositivi senza la necessità di possederli.

2.5.1 Struttura di un progetto in Android Studio

In Android Studio, il progetto di un'app è contenuto nel modulo app. Quest'ultimo rappresenta il modulo di default di un'app Android. Esso è suddiviso in tre parti principali (Figura 2.20); ovvero la cartella con il codice java, la cartella `res`, contenente risorse per lo più realizzate in XML, e la cartella `manifest`, contenente, appunto, il file `Manifest.xml`.

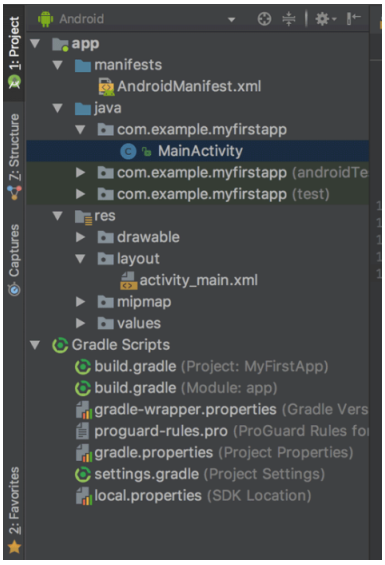

Figura 2.20. Suddivisione di un progetto in AS

Dopo il modulo app, troviamo la sezione Gradle Scripts. Qui ci sono i file di build, che useranno Gradle per trasformare il nostro progetto in un'app funzionante. In particolare, i file di build sono due; ovvero uno per tutto il progetto ed uno per il solo modulo app; quest'ultimo è quello tipicamente modificato dal programmatore (Figura 2.21).

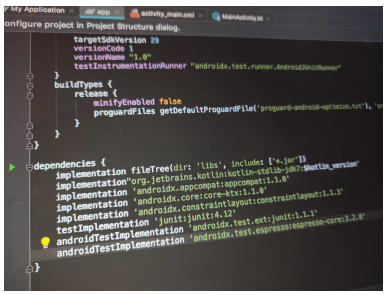

Figura 2.21. File Gradle di un progetto di Android Studio

Nel file Gradle del modulo app vengono impostati alcuni fattori, tra cui, la versione minima dell'Android OS cui l'app è destinata, e la versione di sviluppo dell'applicazione.

Inoltre, nel Gradle, è presente una sezione, denominata dependencies, molto importante per l'espansione delle funzionalità del progetto. Infatti, in questa sezione, possiamo aggiungere diverse librerie con cui arricchire il progetto.

Come detto precedentemente, un aspetto significativo di Android Studio consiste nell'anteprima di layout, praticamente istantanea. Se si apre un file contenente la struttura del layout, reperibile dalla cartella layout, si vede che il suo contenuto può essere mostrato in modalità Design (Figura 2.22) oppure Text; con quest'ultima modalità viene mostrato il tipico formato XML.

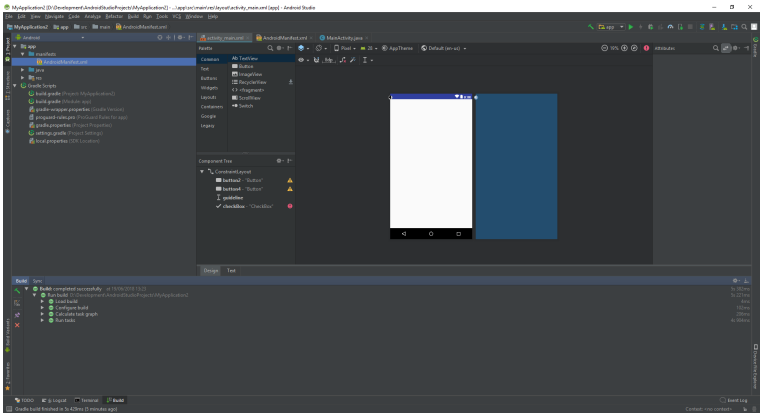

Figura 2.22. Il design mode di Android Studio

Il pannello Design può essere utile per visualizzare rapidamente le modifiche apportate al sorgente XML, oppure per disegnare l'interfaccia trascinando i diversi Widget direttamente sul display.

Inoltre, il pannello Design presenta alcuni menù a tendina, che permettono di modificare le condizioni dell'anteprima in termini di modello, orientamento e versione Android disponibile.

Il codice Java, che realizza l'Activity (Figura 2.23.), risiede nella cartella `java`.

All'interno del file `java` verrà inserita tutta la logica dell'Activity relativa. In esso verrà, inoltre, sovrascritto il metodo `onCreate()`. All'interno del metodo `onCreate()`, verranno inizializzati tutti gli oggetti di tipo View inseriti nell'interfaccia utente. Ciò è possibile grazie al metodo `setContentView()`, il quale collega il codice Java dell'Activity al file XML presente nella cartella `layout`.

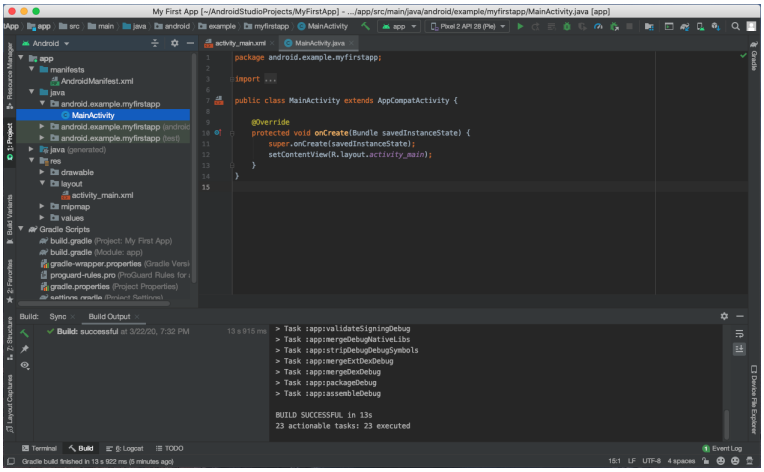

Figura 2.23. Codice Java di un'Activity

3

Analisi dei requisiti e progettazione

In questo capitolo verrà eseguita la prima fase di un progetto di un software. In particolare, ci sarà una breve descrizione dei servizi che l'app, oggetto della tesi, dovrà garantire. Da tale descrizione verranno estratti i requisiti funzionali e non che dovranno caratterizzare l'app. Inoltre, dopo questa prima analisi, vedremo la fase di progettazione dell'app.

3.1 Descrizione dell'app CoFiT

L'app nasce dall'esigenza dello studio commercialistico Co.Fi.T. (Consulenza Finanziaria e tributario) (Figura 3.1) di offrire ai propri clienti dei servizi aggiuntivi, ovvero quello di mantenere un database di tutti i pagamenti delle tasse, da effettuare o già effettuati, e di facilitare lo scambio di documenti tra commercialista e cliente, facendo quindi diminuire al minimo i contatti diretti tra i due soggetti, offrendo al contempo un servizio a 360 gradi.

Figura 3.1. Logo dello studio Co.Fi.T.

Uno degli obiettivi è quello di ottenere un'app semplice da utilizzare, permettendo all'utente medio di poter eseguire le operazioni desiderate effettuando pochi click.

Il prodotto rilasciato sarà lo stesso sia per il commercialista che per il cliente, ma ciascuno avrà accesso ad aree diverse per eseguire le proprie operazioni, nascoste all'altra tipologia di utenza.

Le principali funzionalità offerte dall'app sono le seguenti:

- L'utente anonimo potrà:

  - registrarsi all'app CoFiT;
  - effettuare il login, se già registrato;
- Il commercialista potrà:
  - visualizzare la dei lista clienti registrati all'app;
  - visualizzare l'anagrafica di ciascun cliente registrato;
  - inserire le tasse, specifiche per ogni cliente registrato;
  - condividere i documenti con ciascun cliente registrato.
- Il cliente potrà:
  - visualizzare le proprie tasse;
  - gestire un'agenda di crediti e debiti, indipendentemente dal commercialista;
  - condividere i propri documenti col commercialista.

In particolare, la gestione dei documenti avverrà per mezzo di una piattaforma cloud. Il cliente avrà a disposizione una propria area in cui potrà caricare e scaricare i documenti condivisi col commercialista.

3.2 Analisi dei requisiti

L'analisi dei requisiti è un'attività preliminare alla fase di progettazione, il cui scopo è quello di definire le funzionalità che il nuovo prodotto deve offrire, ovvero i requisiti che devono essere soddisfatti dal software sviluppato.

L'analisi dei requisiti si compone di due fasi:

1. *Analisi dei requisiti funzionali*: è un elenco di funzionalità o servizi che il sistema deve fornire.
2. *Analisi dei requisiti non funzionali*: rappresentano i vincoli e le proprietà o caratteristiche relative al sistema.

3.2.1 Requisiti funzionali

I requisiti funzionali descrivono le funzionalità del software in termini di:

a) servizi che il software stesso deve fornire;
b) risposte che l'utente aspetta dal software in determinate condizioni;
c) risultati che il software deve produrre in risposta a specifici input.

I requisiti funzionali vengono descritti in linguaggio naturale; per capire come è possibile collegarli tra di loro è necessario creare dei modelli. Questi ultimi prendono il nome di diagramma dei casi d'uso.

Successivamente, bisogna costruire tutti i possibili scenari dell'applicazione. Uno scenario è una sequenza di eventi che si verifica in una particolare esecuzione del caso d'uso.

All'interno di un singolo caso d'uso vi possono essere diversi scenari possibili.

I requisiti funzionali che l'app COFIT deve soddisfare sono i seguenti:

- un nuovo utente si deve poter registrare;
- un utente registrato deve poter effettuare il login;

- il cliente deve inserire e modificare il proprio profilo;
- i debiti e i crediti del cliente devono poter essere inseriti all'interno del database locale;
- il cliente deve poter modificare i propri crediti e debiti;
- il cliente deve poter cancellare i propri crediti e debiti;
- il commercialista deve poter inserire, visionare, modificare ed eliminare gli F24 per ciascun cliente.
- il cliente deve poter prendere visione di tutte gli F24 inseriti dal commercialista;
- il commercialista deve poter registrare il pagamento per ciascuna tassa.
- il cliente deve poter inserire i documenti da condividere col commercialista;
- il cliente e il commercialista devono poter cancellare i documenti condivisi;
- il commercialista e il cliente devono poter scaricare i file condivisi;
- il commercialista deve poter inserire il documenti da condividere con uno specifico cliente;
- il commercialista e il cliente devono poter eliminare i file condivisi;
- il commercialista deve poter visionare l'anagrafica di ciascun cliente;
- il commercialista deve poter vedere la lista dei clienti del proprio studio;

Siccome i requisiti espressi in linguaggio naturale non sono privi di ambiguità, può risultare utile costruire un glossario dei termini (Tabella 3.1), il cui scopo è quello di fornire, per ogni concetto rilevante, una breve descrizione ed eventuali sinonimi.

Termine	Descrizione	Sinonimi	Collegamenti
Utente anonimo	Persona che ancora non ha effettuato la registrazione o il login.	//	Cliente
Commercialista	Persona che gestisce lo studio professionale.	//	Cliente, Anagrafica, F24, Documenti
Cliente	Persona che usufruisce delle prestazioni del commercialista.	//	Utente anonimo, F24, Commercialista, Crediti, Debiti, Anagrafica, Documenti
F24	Modello utilizzato dai contribuenti per il pagamento delle tasse.	Tassa	Cliente, Commercialista
Anagrafica	Dati anagrafici dei clienti dello studio	Profilo	Cliente, Commercialista
Crediti	Entrate economiche che il cliente può registrare nell'app.	//	Cliente
Debiti	Uscite economiche che il cliente può registrare nell'app.	//	Cliente
Documenti	File condivisi tra commercialista e cliente.	File	Cliente, Commercialista

Tabella 3.1. Glossario dei termini

Diagramma dei casi d'uso

Il diagramma dei casi d'uso è composto, prevalentemente, da attori e casi d'uso. Un attore specifica un ruolo assunto da un utente o da un'altra entità che interagisce con il sistema nell'ambito di un'unità di funzionamento. Nel diagramma dei casi d'uso l'attore è rappresentato con un omino stilizzato (Figura 3.2).

Figura 3.2. Rappresentazione dell'attore nel diagramma dei casi d'uso

Un caso d'uso è uno specifico modo di utilizzare il sistema da parte di un attore per eseguire una certa funzionalità del sistema stesso. Inoltre, ad ogni caso d'uso possono essere associati più requisiti funzionali.

Il diagramma dei casi d'uso è, quindi, una rappresentazione schematica delle interazioni tra gli attori e le funzioni offerte dal sistema.

Nel diagramma dei casi d'uso un caso d'uso è rappresentato con un ovale (Figura 3.3).

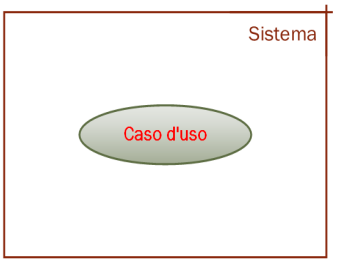

Figura 3.3. Rappresentazione del caso d'uso nel diagramma dei casi d'uso

Attori e casi d'uso vengono collegati mediante le associazioni. I casi d'uso non si possono collegare tra di loro, tuttavia si possono creare delle dipendenze. Ci possono essere due tipi di dipendenze, ovvero inclusione ed estensione, entrambe rappresentate da una linea tratteggiata.

L'inclusione indica che un caso d'uso principale viene sempre eseguito durante l'esecuzione di un caso d'uso subordinato.

L'estensione indica che un caso d'uso può essere eseguito in determinate circostanze di un altro caso d'uso.

Dalla stesura dei requisiti funzionali, effettuata nella sezione precedente, possiamo individuare tre diversi attori:

- utente "generico";
- cliente;
- commercialista.

L'utente generico dovrà, innanzitutto, poter creare un nuovo account, se ancora non l'ha fatto, oppure effettuare il login (Figura 3.4).

Successivamente, una volta che l'utente generico ha eseguito una delle operazioni a sua disposizione, sarà riconosciuto dal sistema e avrà libero accesso a tutte le funzionalità dell'app (Figura 3.5).

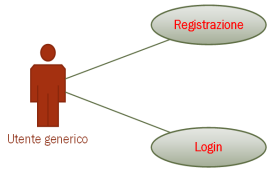

Figura 3.4. Diagrammi dei casi d'uso dell'utente generico

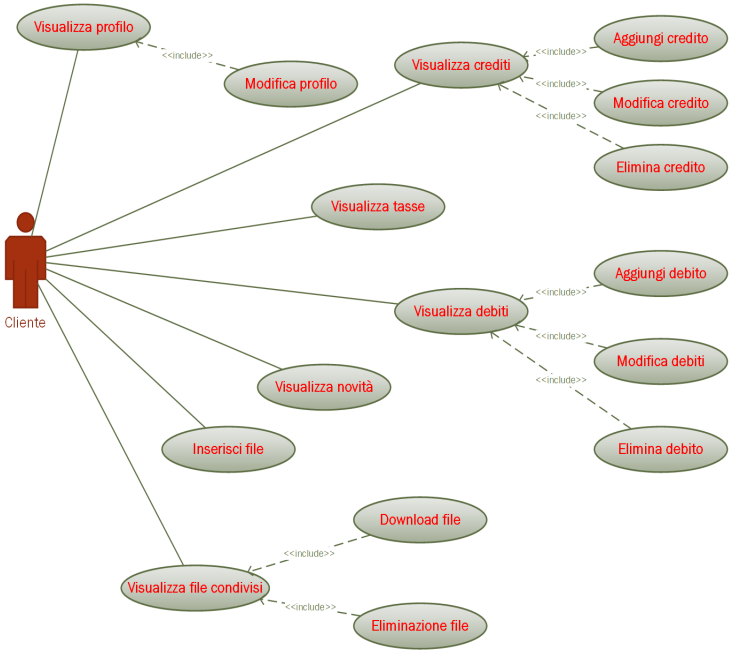

Figura 3.5. Diagrammi dei casi d'uso del cliente

Viceversa, se l'utente generico che effettua il login è in realtà il commercialista, cambiano i casi d'uso e, di conseguenza, il suo diagramma (Figura 3.6).

Scenari

Uno scenario descrive il comportamento del sistema quando tutto funziona correttamente, oppure il comportamento in situazioni di errore corredato dalla descrizione del comportamento richiesto al sistema. I primi sono definiti scenari principali; gli ultimi sono scenari di eccezione.

La definizione degli scenari principali permette di realizzare sistemi corretti; ovvero sistemi che effettivamente realizzano i servizi richiesti dall'utente, mentre la definizione degli scenari di eccezione permette di realizzare sistemi robusti, cioè sistemi in grado di intercettare situazioni anomale e di avviare, come risposta, procedure prestabilite.

Vediamo, quindi, quali sono i principali scenari dell'app CoFiT:

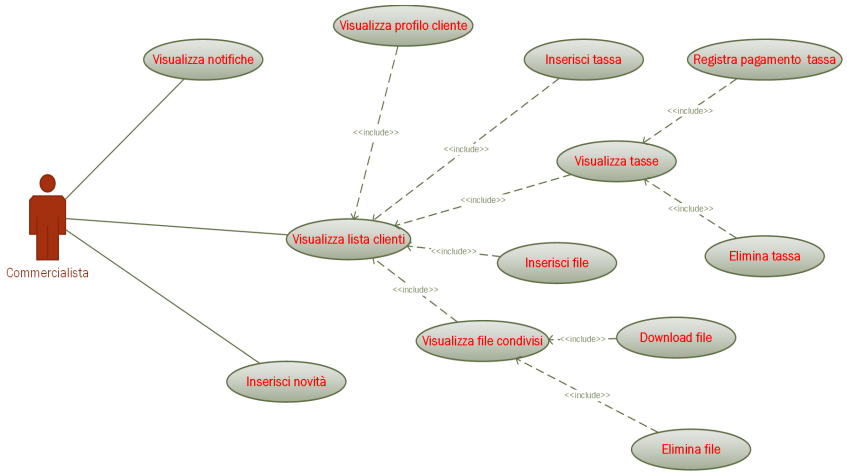

Figura 3.6. Diagrammi dei casi d'uso del commercialista

- *Registrazione di un nuovo utente*: il nuovo utente potrà essere creato specificando l'e-mail e la password; per motivi di sicurezza, quest'ultima verrà richiesta due volte. Tutti i passaggi dello scenario sono indicati nella Tabella 3.2.

Nome	Registrazione
Attore	Utente generico
Scenario	1. L'utente avvia l'applicazione. 2. Il sistema presenta i campi per creare il nuovo utente. 3. L'utente inserisce e-mail, password e conferma password. 4. L'utente preme "Inserisci". 5. Il sistema registra il nuovo utente. Fine
Scenario di eccezione	3a. L'utente non compila tutti i campi. 3b. Il sistema avverte che non tutti i campi sono completi. Torna al punto 2.
Scenario di eccezione	3a. L'e-mail non è del formato corretto. 3b. Il sistema avverte che l'email non è nel formato corretto. Torna al punto 2.
Scenario di eccezione	3a. La password non è del formato corretto. 3b. Il sistema avverte che la password non è nel formato corretto. Torna al punto 2.
Scenario di eccezione	3a. Password e conferma password non sono uguali. 5b. Il sistema avverte che la password non corrisponde al testo inserito in conferma password. Torna al punto 2.

Tabella 3.2. Scenario "Registrazione di un nuovo utente"

- *Inserimento di nuova tassa da parte del commercialista*: il commercialista dovrà inserire tutti i dati della nuova tassa. Non sarà possibile lasciare nessun campo vuoto, altrimenti si ricade nello scenario di eccezione. Tutti i passaggi sono indicati nella Tabella 3.3.
- *Inserimento di un file*: il commercialista potrà condividere i file con i propri clienti. I file di un cliente non potranno confondersi con quelli di un altro cliente; perciò il commercialista dovrà selezionare prima il cliente con cui condividere quel file. Successivamente, potrà decidere se scattare una nuova foto o prendere un file dalla memoria del dispositivo e condividerlo. I passaggi completi sono descritti nella Tabella 3.4.

Nome	Aggiungi tassa
Attore	Commercialista
Scenario principale	1. Il sistema presenta la lista clienti. 2. Il commercialista seleziona il cliente. 3. Il commercialista seleziona "Inserisci tassa". 4. Il sistema presenta i campi per l'aggiunta di una tassa. 5. Il commercialista inserisce i dati in tali campi. 6. Il commercialista conferma l'inserimento. 7. Il sistema aggiunge la nuova tassa e mostra la lista di tutti le tasse del cliente. Fine
Scenario di eccezione	5a. Il commercialista non inserisce tutti i dati. 5b. Il sistema avverte che non tutti i campi sono completi. Torna al punto 4.

Tabella 3.3. Scenario "Inserimento di una tassa"

Nome	Upload file
Attore	Commercialista
Scenario principale	1. Il sistema presenta la lista dei clienti. 2. Il commercialista seleziona il cliente. 3. Il commercialista seleziona "Carica documenti". 4. Il sistema presenta i campi per l'aggiunta di un documento o una foto. 5. Il commercialista carica un file dalla memoria del dispositivo o scatta una foto. 6. Il commercialista completa tutti i campi. 7. Il commercialista conferma l'inserimento. 8. Il sistema aggiunge il nuovo documento o la foto e mostra la lista di tutti i documenti condivisi con il cliente. Fine
Scenario di eccezione	5a. Il commercialista non carica la foto o il file. 5b. Il sistema avverte che non è stato trovato nessun file da caricare. Torna al punto 4.
Scenario di eccezione	6a. Il commercialista non inserisce tutti i dati. 6b. Il sistema avverte che non tutti i campi sono completi. Torna al punto 4.

Tabella 3.4. Scenario "Inserimento di un file"

- *Download o eliminazione di un file condiviso*: questo scenario è pressoché identico sia per il cliente che per il commercialista. Entrambi devono prima visualizzare la lista dei file condivisi e, successivamente, selezionare quello desiderato. Infine, essi devono decidere se eliminare o scaricare il file dalla rete. Tutti i passaggi sono descritti nella Tabella 3.5.

Nome	Download o eliminazione file
Attore	Commercialista
Scenario principale	1. Il sistema presenta la lista dei clienti. 2. Il commercialista seleziona il cliente. 3. Il commercialista seleziona "Visualizza documenti". 4. «include» Il sistema presenta la lista di tutti i documenti condivisi col cliente. 5. Il commercialista seleziona un file dalla lista e procede al suo download o alla sua eliminazione. 6. Il sistema scarica o elimina il file selezionato. Fine

Tabella 3.5. Scenario "Download o eliminazione di un file"

- *Eliminazione o registrazione del pagamento di una tassa*: una volta inserita una tassa, il commercialista dovrà registrarne l'eventuale pagamento. Se, invece, ci sono stati errori di inserimento della tassa, il commercialista potrà eliminarla. Tutti i passaggi dei due casi d'uso sono descritti nella Tabella 3.6.

Nome	Eliminazione o registrazione del pagamento di una tassa
Attore	Commercialista
Scenario principale	1. Il sistema presenta la lista dei clienti. 2. Il commercialista seleziona il cliente. 3. Il commercialista seleziona "Visualizza F24". 4. «include» Il sistema presenta la lista di tutte le tasse del cliente. 5. Il commercialista seleziona una tassa dalla lista e decide cosa fare. 6. Il sistema registra il pagamento o elimina la tassa. Fine

Tabella 3.6. Scenario "Registrazione pagamento o eliminazione di una tassa"

- *Modifica del profilo*: l'app salva tutti i dati anagrafici del cliente. Questi dati potranno essere visibili anche dal commercialista.
  Alcuni dati anagrafici dei clienti potrebbero cambiare nel tempo; quindi bisogna consentire al cliente di poterli modificare. Tutti i passaggi sono visibili nella Tabella 3.7.

Nome	Modifica del profilo
Attore	Cliente
Scenario principale	1. Il sistema presenta la "Home" dell'applicazione. 2. Il cliente seleziona la voce "Profilo" sul menù dell'applicazione. 3. «include» Il sistema visualizza l'anagrafica corrente del cliente". 4. Il cliente seleziona "Modifica profilo". 5. Il sistema presenta i campi per la modifica dell'anagrafica del cliente. 6. Il cliente modifica i campi dell'anagrafica. 7. Il cliente preme su "Inserisci". 8. Il sistema salva l'anagrafica del cliente. Fine
Scenario di eccezione	6a. Il cliente lascia un campo vuoto. 6b. Il sistema avverte che non tutti i campi sono completi. Torna al punto 5.

Tabella 3.7. Scenario "Modifica dell'anagrafica"

- *Aggiunta di un credito o di un debito*: i clienti possono usufruire di un database locale per per inserire i propri crediti e debiti. Questi non sono gestiti dal commercialista; il cliente potrà inserirli, modificarli ed eliminarli in modo autonomo. Tutti i passaggi sono descritti nella Tabella 3.8.

Nome	Aggiunta credito (debito)
Attore	Cliente
Scenario principale	1. Il sistema presenta la "Home" dell'applicazione. 2. Il cliente preme su "Visualizza crediti (debiti)". 3. «include» Il sistema visualizza la lista dei crediti (debiti) ". 4. Il cliente seleziona "Aggiungi credito (debito)". 5. Il sistema presenta i campi per l'inserimento di un credito (debito). 6. Il cliente compila i campi per l'inserimento di un credito (debito). 7. Il cliente preme su "Inserisci". 8. Il sistema aggiunge il credito (debito). Fine
Scenario di eccezione	6a. Il cliente lascia un campo vuoto. 6b. Il sistema avverte che non tutti i campi sono completi. Torna al punto 5.

Tabella 3.8. Scenario "Inserimento di un credito (debito)"

3.2.2 Requisiti non funzionali

I requisiti non funzionali riguardano tutti gli altri requisiti dell'applicazione che non sono di per sé essenziali alle funzionalità implementate dal sistema, ma piuttosto alle modalità operative, come usabilità, prestazioni, scalabilità, e robustezza.

I requisti non funzionali definiscono, quindi, i vincoli sullo sviluppo del sistema.

Rientrano tra i requisiti non funzionali i requisiti di prodotto, di processo ed esterni.

Requisiti di prodotto

- l'applicazione deve essere fluida ed avere tempi di risposta brevi;
- l'applicazione non deve andare in crash quando carica i dati dal database;
- l'applicazione deve essere semplice da utilizzare;
- il layout dell'applicazione si deve adattare, il più possibile, al display di diversi dispositivi;
- un cliente non deve poter accedere all'area riservata al commercialista.

Requisiti di processo

- l'applicazione verrà sviluppata in linguaggio Java orientato per applicazioni Android;
- l'ambiente di sviluppo scelto per l'applicazione è Android Studio;
- per testare l'applicazione verranno utilizzati gli emulatori messi a disposizione da Android Studio;
- l'applicazione potrà essere installata in qualsiasi smartphone dotato di Android 6.0 o successive;
- la parte back-end sarà gestita con Firebase;
- per la persistenza dei dati, in locale, verrà utilizzato il DBMS MySQL.

Requisiti esterni

- è necessario garantire una connessione ad internet tramite Wi-Fi, oppure tramite connessione dati;
- è necessario garantire l'accesso alla memoria interna ed esterna del dispositivo.

3.3 Progettazione

La progettazione è la seconda fase del ciclo di vita del software. Essa avviene sulla base della specifica dei requisiti, prodotta durante la fase precedente. Il progetto definisce come tali requisiti saranno soddisfatti, entrando nel merito della struttura che dovrà essere data al software.

La prima fase di progettazione riguarda la componente dati. Successivamente, ci occuperemo della progettazione del Front-end e del Back-end dell'applicazione.

3.3.1 Progettazione della componente dati

La progettazione della componente dati è utile per capire come organizzare tutti i dati all'interno del software. Essa richiede la progettazione di un diagramma Entità - Relazione (Schema E-R), un dizionario delle entità e la definizione dei vincoli di integrità.

Diagramma E-R

Lo schema E-R (Figura 3.7) mostra come le entità, ovvero gli oggetti che compongono l'app, si relazionano tra di loro all'interno del sistema.

I modelli E-R utilizzano un set definito di simboli, come rettangoli, rombi, ovali e linee di collegamento per rappresentare l'interconnessione tra entità, relazioni e i loro attributi. Infine, tra entità e relazioni vi sono i vincoli di cardinalità, i quali definiscono il massimo o minimo numero di istanze delle relazioni a cui partecipa un'istanza dell'entità.

Dall'analisi dei requisiti sappiamo che le entità principali che compongono l'app CoFiT sono gli F24, i Documenti, i Clienti dello studio, l'Anagrafica, il Commercialista, i Crediti e i Debiti.

Dal diagramma E-R possiamo individuare i diversi attributi che identificano un'entità.

Le entità Cliente e Commercialista hanno tre importanti attributi, ovvero e-mail, password e UID. Questi attributi sono necessari per gestire l'autenticazione degli utenti registrati. In particolare, l'UID rappresenta la loro chiave primaria.

L'entità Cliente può essere sia un'azienda che un privato cittadino. e ciò è specificato dall'attributo "tipo cliente", presente nell'entità Anagrafica. Ne consegue che l'attributo Denominazione può indicare il nome dell'azienda o il nome e cognome della persona. Nel caso di un'azienda è presente il solo attributo "nome azienda"; infatti, l'attributo "cognome" ha una cardinalità (0,1), in quanto può anche non esserci.

Per quanto riguarda i Documenti, essi sono identificati da una Key. Gli altri attributi dei documenti sono il nome e l'url, ovvero il percorso dello storage in cui viene salvato il file.

L'entità F24 è identificata dal nome, dall'importo, dalla data di scadenza e dallo stato. Il primo fa da chiave primaria, poiché non ci potranno essere per lo stesso cliente due F24 con lo stesso nome. Esso è formato dal modello (IVA, CCIA, INAIL, IMU ecc.), dall'anno e dal periodo a cui la tassa si riferisce. L'attributo "Stato" indica se la tassa è solo emessa, pagata o scaduta.

Infine, l'entità Cliente può gestire dei propri Crediti e Debiti all'interno del software. Essi sono generalizzazioni dell'entità generale Registro finanziario. Ogni Credito e ogni Debito avrà un proprio ID per essere identificato. Gli altri attributi sono molto simili a quelli visti per l'entità F24, tranne che per l'aggiunta dell'attributo Descrizione.

Per motivi di praticità, il diagramma E-R non è stato progettato ad hoc; tuttavia è utile per analizzare quali attributi debbano avere le diverse entità. Inoltre, sono state anche tralasciate diverse operazioni, come l'eliminazione di un F24 o il download di un documento.

Dizionario delle entità

Per inquadrare meglio gli attributi delle entità presenti nel progetto possiamo utilizzare un dizionario delle entità (Tabella 3.9), in cui, per ciascuna di esse, verranno elencati tutti gli attributi e la chiave primaria.

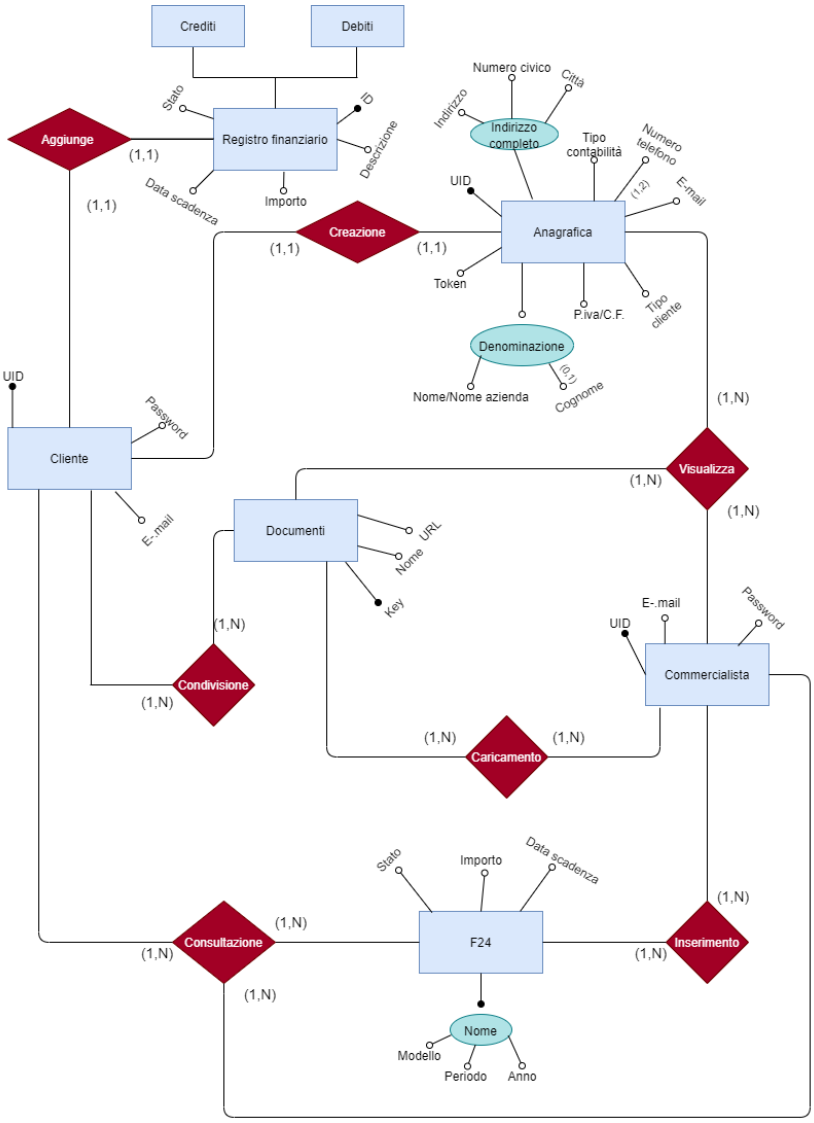

Figura 3.7. Il diagramma E-R del progetto

Vincoli di integrità

L'integrità dei dati serve a garantire che i dati memorizzati siano corretti; se i dati sono imprecisi o incoerenti, l'integrità viene violata.

L'integrità dei dati, praticamente, impone delle restrizioni sui valori assunti dagli attributi delle entità.

I vincoli di integrità individuati all'interno del progetto sono i seguenti:

- V1: nella registrazione la password deve essere di almeno sei caratteri;
-]V2: il numero di telefono deve essere formato da almeno 7 cifre;

Entità	Attributi	Chiave primaria
Cliente	ID, E-mail, password	ID
Commercialista	ID, E-mail, Password	ID
Anagrafica	ID, Indirizzo, Numero civico, Città, Tipo contabilità, Numero di telefono, E-mail, Nome, Cognome, C.F, P.Iva, Nome azienda, Tipo cliente, Token	ID
F24	Modello, Periodo, Anno, Stato, Importo, Data di scadenza	Nome(Modello, Periodo, Anno)
Documenti	Key, Url, Nome	Key
Registro finanziario	ID, Tipo, Stato, Descrizione, Data di scadenza	ID

Tabella 3.9. Dizionario delle entità

- V3: il Codice Fiscale deve essere formato da sedici caratteri alfanumerici;
-]V4: la partita IVA deve essere formata da 11 caratteri;
- V5: nell'Anagrafica deve essere inserito almeno un numero di telefono (cellulare o fisso);
- V6: nell'Anagrafica deve essere inserito almeno uno tra Codice Fiscale e Partita IVA;
- V7: nell'inserimento di un modello F24 non deve essere lasciato vuoto nessun attributo;
- V8: l'importo di un F24 deve essere un numero decimale;
- V9: la data di scadenza di un F24 deve essere il formato: gg/mm/aaaa;
- V10: nell'inserimento di un Documento non deve essere lasciato vuoto nessun attributo;
- V11: nessun attributo del registro finanziario deve essere lasciato vuoto;
- V12: l'importo del Credito o del Debito deve essere un numero decimale;
- V13: la data di scadenza di un Credito o di un Debito deve essere del formato: gg/mm/aaaa.

3.3.2 Progettazione del Front-end

La progettazione del Front-end riguarda la progettazione della parte visibile all'utente, con cui egli può interagire; ovvero la progettazione dell'interfaccia utente.

In particolare, progetteremo la mappa dell'app, i wireframe, i mockup e la storyboard. Successivamente, con i wireframe daremo una prima forma grezza alle interfacce utente.

Dopo aver disegnato i wireframe potremo aggiungere dettagli dell'interfaccia utente con i Mockup; questi ultimi rappresentano ciò che si avvicinerà al risultato finale. In questa fase potremo, inoltre, decidere quali saranno i colori principali del software.

Mappa dell'app

La mappa dell'app (Figura 3.8) ci permette di capire da quante interfacce utente è costituita l'applicazione e di come esse devono essere collegate.

Dalla mappa rileviamo che la prima interfaccia è la registrazione o, se questa è già avvenuta, il login. Dalla pagina di login possiamo anche procedere al cambio password, qualora si volesse cambiarla o se è stata dimenticata.

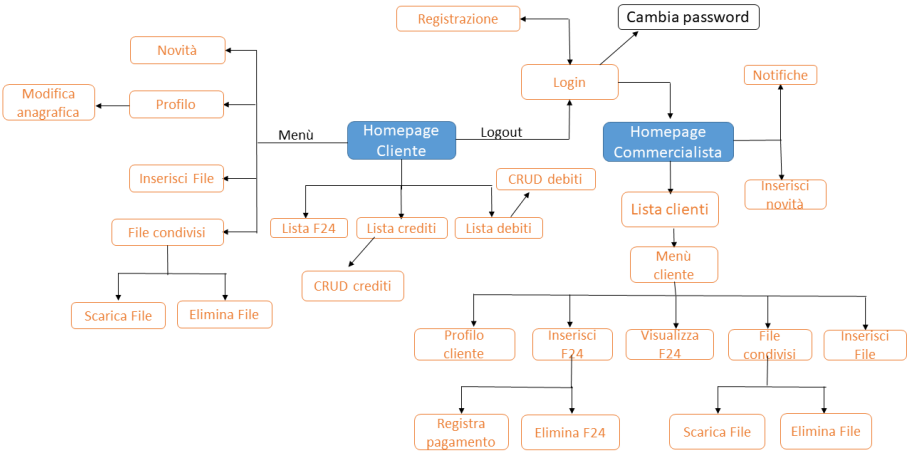

Figura 3.8. Mappa dell'applicazione CoFiT

Una volta effettuato il login, possiamo entrare nella Homepage dell'applicazione.

Come è possibile notare dalla mappa, ci sono due home page, ovvero una riservata ai clienti e una per il commercialista. Le due home page non possono essere collegate; quindi, il cliente non può entrare nella sezione del commercialista, e viceversa.

Dalla home page riservata ai clienti, questi ultimi possono visionare la lista degli F24, dei crediti e dei debiti. Dalla stessa pagina è possibile anche inserire, modificare ed eliminare i crediti e i debiti.

Per passare alle interfacce successive possiamo implementare la home page con un menù a comparsa, in cui saranno presenti i collegamenti per usufruire degli altri servizi.

Per quanto riguarda la home page riservata al commercialista, si è pensato di visualizzare immediatamente la lista dei clienti. Da questa lista è possibile entrare nella sezione di ogni cliente, chiamata, per comodità, "Menù cliente". Da quest'ultima è possibile accedere alle aree in cui si può visualizzare l'anagrafica del cliente, visualizzare e inserire gli F24 e visualizzare e condividere i documenti.

Inoltre, all'interno dell'applicazione si possono inserire servizi accessori. Un esempio di questi servizi è la sezione per inserire le novità dello studio professionale Co.Fi.T, le quali vengono caricate dal commercialista e visualizzate dai clienti, oppure un secondo un secondo esempio riguarda la sezione delle notifiche, utili per informare il commercialista che sono stati inseriti dei documenti da un cliente specifico.

Wireframe

Un wireframe è un'illustrazione bidimensionale dell'interfaccia utente di un software che si concentra, in particolare, sull'allocazione dello spazio e la prioritizzazione del contenuto, delle funzionalità disponibili e dei comportamenti previsti.

I wireframe sono, in genere, in scala di grigi in stile e colore. Essi sono utili nelle seguenti attività:

- collegare l'architettura delle informazioni dell'app al suo design visivo;
- chiarire modi coerenti per la visualizzazione delle informazioni sull'interfaccia utente;
- determinare la funzionalità prevista nell'interfaccia;
- assegnare la priorità al contenuto determinando la quantità di spazio da allocare a un determinato elemento e il punto dello schermo in cui posizionarlo.

Attraverso i wireframe ci concentriamo sulla struttura dell'applicazione, perciò non useremo dei veri elementi grafici e neanche i colori. Il motivo per il quale i wireframe sono così basici è perché devono far conoscere il funzionamento base dell'interfaccia e il flusso operativo dell'app, perciò non ci devono essere elementi estetici che possano distrarre da questi aspetti.

Nel seguito diamo uno sguardo ai diversi wireframe disegnati per l'app CoFiT.

I wireframe delle interfacce utente responsabili della registrazione e del login (Figura 3.9) sono costituite da un'immagine centrale, alcune EditText per inserire e-mail e Password, e un Button per proseguire.

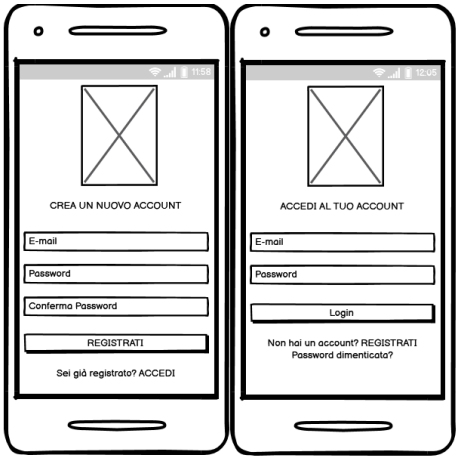

Figura 3.9. Wireframe relativo alla Registrazione e al Login

La home page del cliente (Figura 3.10) si articola in un TabLayout e in una Navigation Drawer. La TabLayout ci permette di scorrere tra le tabelle, dandoci l'impressione di cambiare pagina.

Nella sezione crediti e debiti compare un Floating Button per inserire facilmente un nuovo credito (Figura 3.11); nel caso di un debito, il wireframe è identico.

Mediante la Navigation Drawer sarà possibile navigare facilmente con pochissimi click all'interno dell'app, rendendola, perciò, veloce e semplice da utilizzare.

Per completare la registrazione del cliente dovremo far inserire i dati anagrafici, utili al commercialista per avere tutte le informazioni dei propri clienti (Figura 3.12).

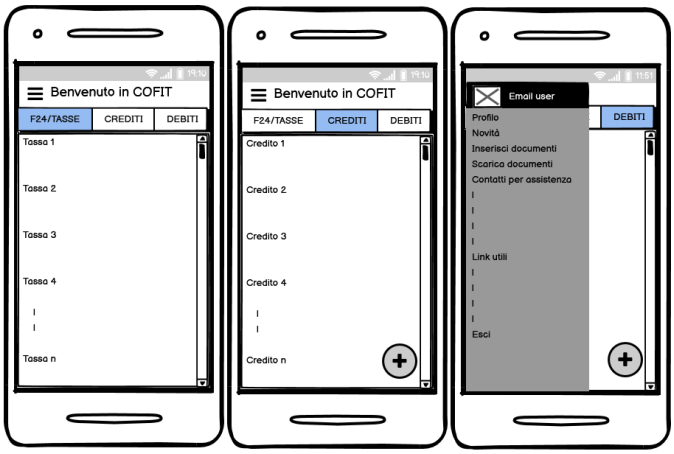

Figura 3.10. Wireframe relativo alla home page del cliente

Figura 3.11. Wireframe relativo all'inserimento di un credito

Per questo motivo, all'interno di questa interfaccia, abbiamo diverse EditText e altri oggetti di tipo View per poter inserire i dati.

L'interfaccia utente relativa alla home page del commercialista (Figura 3.13) sarà molto snella. L'idea è di avere, immediatamente, la lista dei clienti registrati. Inoltre, sarà presente una Search bar che permetterà di cercare facilmente un cliente desiderato. Infine, nella Toolbar ci sarà anche una sezione per le notifiche, cosicché il commercialista saprà quando uno dei clienti ha caricato un file all'interno dello storage.

Una volta che il commercialista avrà selezionato una voce dalla lista dei clienti, egli potrà entrare nel menù cliente (Figura 3.14) e selezionare una delle voci disponibili.

Tra le voci disponibili vi è la visualizzazione dell'anagrafica del cliente. Il wi-

Figura 3.12. Wireframe relativo alla Visualizzazione e all'inserimento dei dati anagrafici

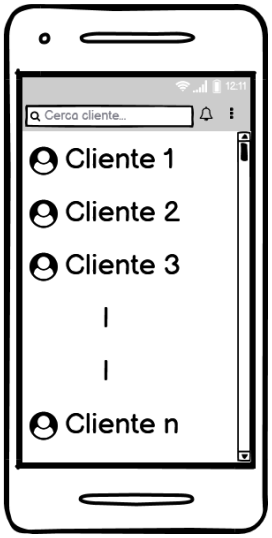

Figura 3.13. Wireframe relativo alla home page del commercialista

reframe di questo servizio è identico a quello del cliente; tuttavia, in questo caso, il commercialista potrà solo visualizzare i dati e non avrà modo di modificarli, mancherà, perciò, l'ImageButton nella parte superiore destra.

Tornando nel menù del cliente e cliccando su "Inserisci F24", il commercialista avrà la possibilità di aggiungere un F24 (Figura 3.15). Questa interfaccia è costituita da uno Spinner, per identificare il Modello F24, e da diverse EditText, per inserire gli altri dati.

L'interfaccia utente per caricare il file (Figura 3.15) presenterà un Button per selezionare il file o scattare una foto. Mediante uno Spinner potremo selezionare la categoria nella quale rientrerà il documento. Una volta selezionato il file, nel caso esso sia una foto, verrà visualizzata un'anteprima al centro dello schermo.

Figura 3.14. Wireframe relativo al menù per gestire i clienti

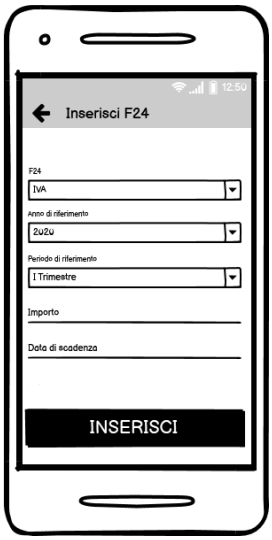

Figura 3.15. Wireframe relativo all'inserimento di un F24

Infine, dopo aver completato tutti i campi potremo cliccare sul Button "Inserisci" e comparirà una ProgressBar nella parte inferiore dello schermo. Questo Layout, con alcune modifiche, sarà utilizzato anche dal cliente.

Infine, nella Figura 3.16 abbiamo l'interfaccia utente per poter visualizzare tutti i file condivisi (Figura 3.16). Cliccando sull'icona presente alla sinistra del nome potremo procedere al loro download o alla loro eliminazione; anche in questo caso il layout è il medesimo sia nel caso del cliente che del commercialista.

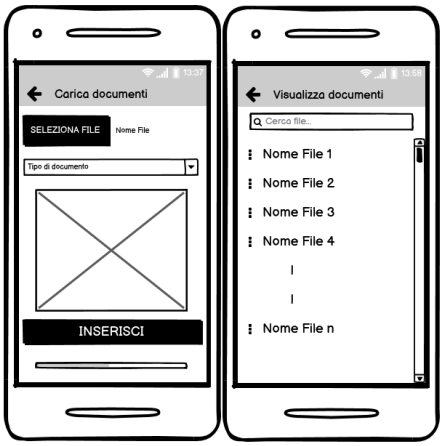

Figura 3.16. Wireframe relativo all'inserimento e alla lista dei documenti

I mockup

Terminati i wireframe possiamo concentrarci sulla creazione dei mockup, i quali, come i primi, sono una rappresentazione statica, con la differenza che essi presentano una grafica molto più dettagliata e vicina a quella reale. Perciò, data la struttura definitiva del wireframe, vengono inseriti colori, stili, font e altri elementi visivi, dando, quindi, un'idea più realistica di come sarà il prodotto finale.

Il primo mockup disegnato riguarda le interfacce utente per la registrazione e il Login (Figura 3.17). La scelta dei colori principali è dettata dal fatto che il logo dello studio commerciale CoFiT presenta il viola e il prugna; quindi l'intera app sarà "colorata" in questo modo.

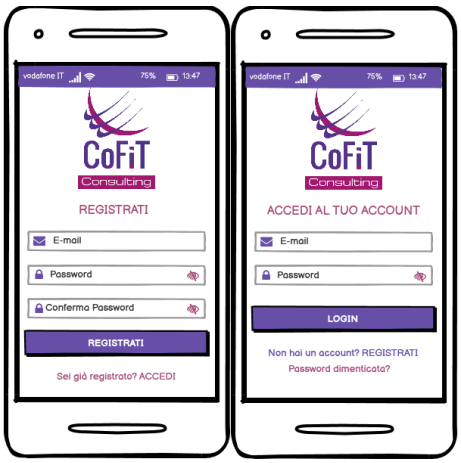

Figura 3.17. Mockup relativo alla registrazione e del login

Il mockup della home page del cliente (Figura 3.18) ha gli stessi elementi inseriti

nel wireframe; tuttavia è possibile vedere da quali parti è costituito un elemento della lista degli F24. Ogni elemento sarà costituito dal nome del modello F24, dalla data di scadenza, dall'importo e da una TextView che avviserà se questo è scaduto o è stato pagato.

Il floating button per l'inserimento dei crediti e dei debiti sarà di colore verde per i primi e rosso per i secondi. Com'è possibile notare dalla Figura 3.18, quando la lista dei crediti (debiti) è vuota, viene mostrata la scritta "Non ci sono crediti (debiti)"; questa sparirà non appena verrà inserito almeno un elemento nella lista.

Il menù di navigazione, presente nella home page è formato da un'icona (in bianco e nero), da un testo e da una scrollbar che ci permette di scorrere tutte le voci del menù.

Esso permetterà di passare nelle varie sezioni dell'app, di richiedere assistenza allo studio del commercialista, di leggere i termini e le condizioni della privacy e di aprire diversi siti Web utili, come il sito dell'INPS, dell'INAIL o della Camera di Commercio di Pescara. L'ultima voce del menù permetterà di effettuare il logout, riportandoci alla pagina del login.

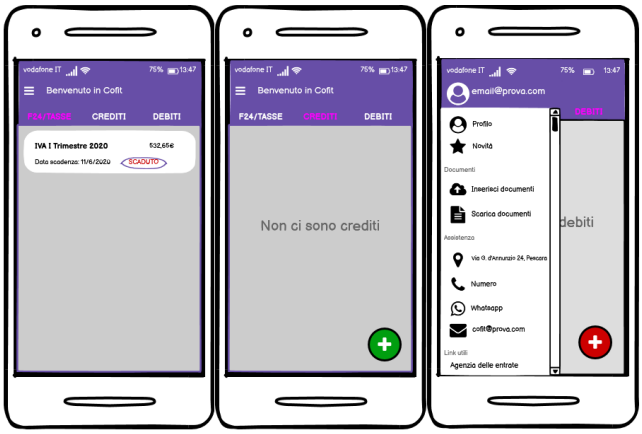

Figura 3.18. Mockup relativo alla home page per il cliente

I due mockup dell'interfaccia utente della gestione dell'anagrafica del cliente (Figura 3.19) riguardano l'inserimento dell'anagrafica e la visualizzazione del profilo. Nell'interfaccia della visualizzazione del profilo verrà dato modo di inserire un'immagine del profilo, così da arricchire l'esperienza utente. Successivamente, verranno mostrati si mostrano i recapiti più importanti; nella parte inferiore, infine, verranno visualizzati gli altri dati.

Il mockup per la home page del commercialista (Figura 3.20) presenta, come descritto nel wireframe, la semplice lista dei clienti. In ogni elemento della lista figurano la foto del profilo, il nome e l'indirizzo e-mail dell'utente.

Come spiegato nella sezione precedente nel caso del wireframe, anche il mockup per la visualizzazione del profilo è uguale a quello visto nella Figura 3.19. Tuttavia, nel caso del commercialista, quest'ultimo potrà avviare l'app del dialer, premendo

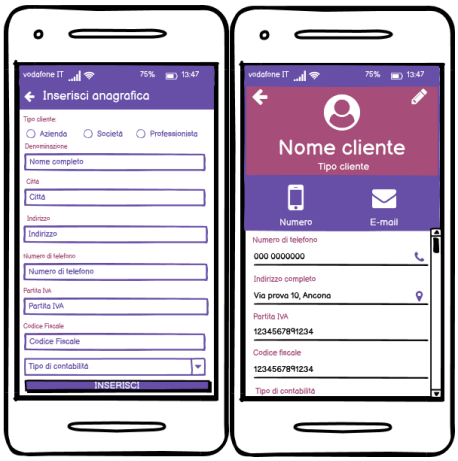

Figura 3.19. Mockup relativo alla visualizzazione e all'inserimento dei dati anagrafici

sull'immagine dello smartphone, e l'app per il servizio di posta elettronica, premendo sull'immagine relativa.

Figura 3.20. Mockup relativo alla home page del commercialista e il menù relativo alla gestione del cliente

Gli ultimi mockup (Figura 3.21) sono relativi all'inserimento di un F24, di un documento e alla visualizzazione dei file condivisi.

Tra i mockup disegnati manca quello dell'inserimento di un credito o di un debito. L'interfaccia utente di questo servizio è praticamente identica a quella per l'inserimento di un F24; cambiano solo alcuni campi.

Dopo aver terminato tutti i mockup, possiamo costruire una storyboard (Figura 3.22). La funzione della storyboard è molto simile a quella della mappa del software;

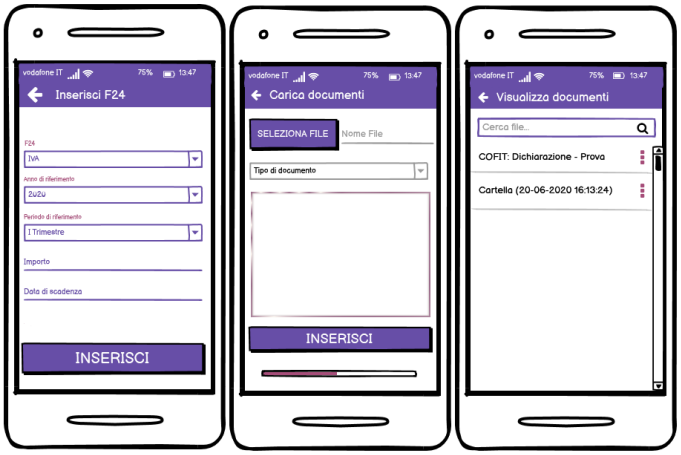

Figura 3.21. Mockup relativi alla'inserimento di un F24, di un documento e della visualizzazione dei file condivisi

essa, infatti, serve per illustrare come saranno collegate tutte le interfacce utente dell'app.

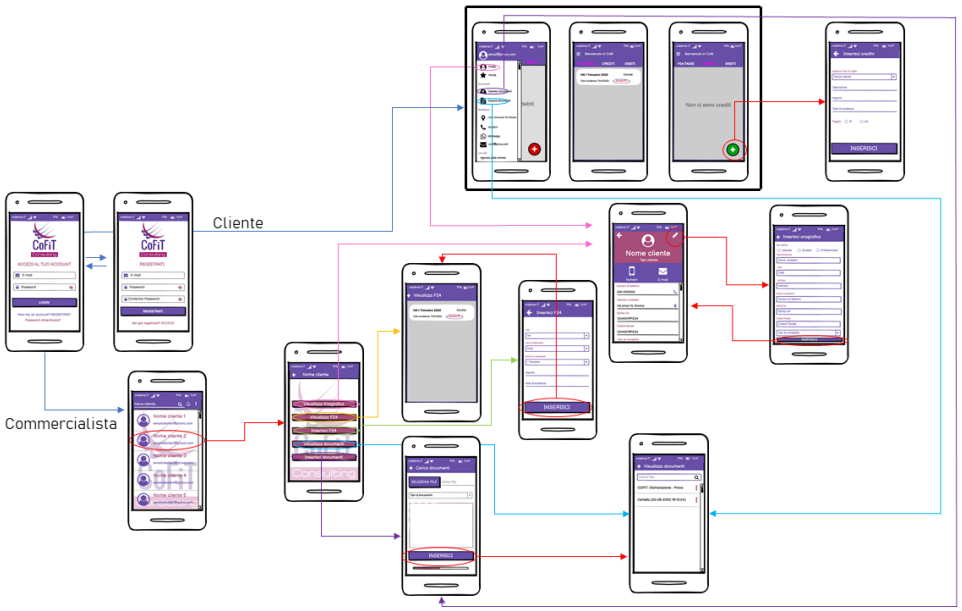

Figura 3.22. La storyboard dell'app CoFiT

3.3.3 Progettazione Back-end

Il protagonista della parte di Back-end del progetto CoFiT è, sicuramente, Firebase (Figura 3.23).

Figura 3.23. Il logo di Firebase

Firebase è un DBMS per il Back-end di piattaforme realizzato da Google, il quale offre diversi servizi utili alle applicazioni (Figura 3.24).

I motivi per i quali si è scelto di utilizzare Firebase sono la sua facilità d'uso e la perfetta compatibilità con Android Studio.

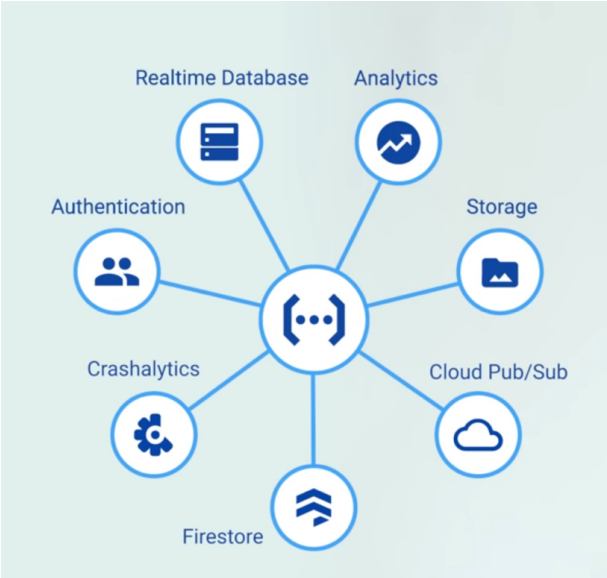

Figura 3.24. I servizi offerti da Firebase

Nel progetto si prevede di utilizzare i seguenti servizi di Firebase:

- autenticazione;
- database (cloud e real time);
- storage;
- messaging.

Questi verranno illustrati in dettaglio nella prossime sottosezioni.

Autenticazione

Firebase permette diversi tipi di autenticazione (Figura 3.25), anche più di uno contemporaneamente, tuttavia quella che verrà implementata all'interno del progetto sarà una semplice autenticazione attraverso e-mail e password.

Firebase gestirà anche il reset della password. Infatti, se questo servizio verrà richiesto, Firebase invierà una e-mail all'indirizzo di posta elettronica dell'utente che ha richiesto il reset. Il testo di questa e-mail conterrà tutta la guida per generare una nuova password.

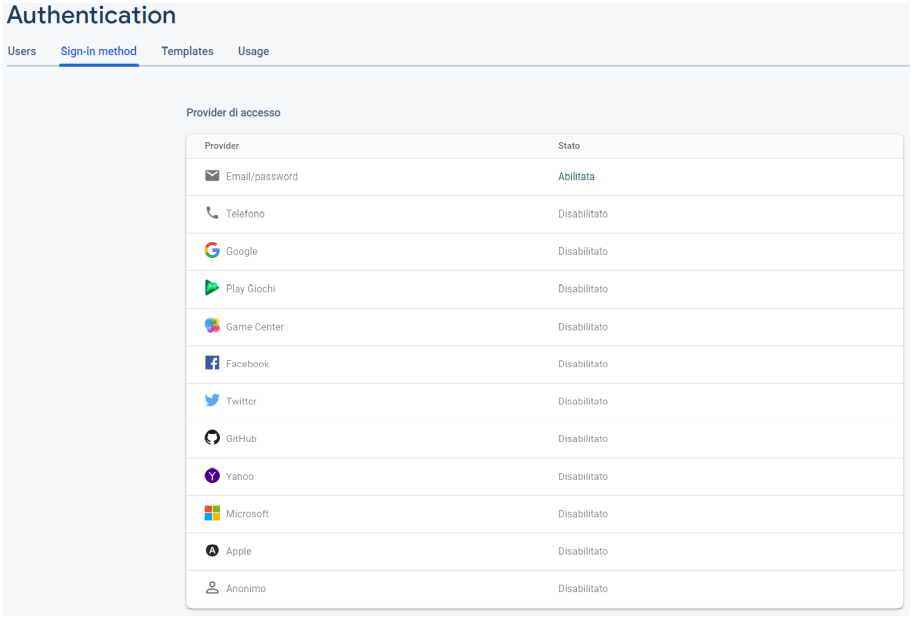

Figura 3.25. Metodi di autenticazione di Firebase

Dopo che un utente ha eseguito la registrazione Firebase associa ad esso un UID, oltre, ovviamente, all'indirizzo e-mail e alla password (Figura 3.26). È possibile, inoltre, visualizzare la data di creazione dell'utente, l'ultimo accesso e il provider utilizzato per la registrazione.

Database

Firebase ha due opzioni di database NoSQL: Cloud Firestore (Figura 3.27) e Realtime Database (Figura 3.28). Il progetto CoFiT sfrutterà entrambe le opzioni. Nel Cloud Firestore verranno salvati i dati relativi agli utenti e i loro F24, invece, il Realtime Database verrà utilizzato per salvare le informazioni relative ai file, come il nome e l'URL. Si è scelto di dividere i compiti tra i due database in questo modo, perché il database in realtime misura la memoria utilizzata; viceversa, il cloud misura il numero di operazioni eseguite (lettura e scrittura).

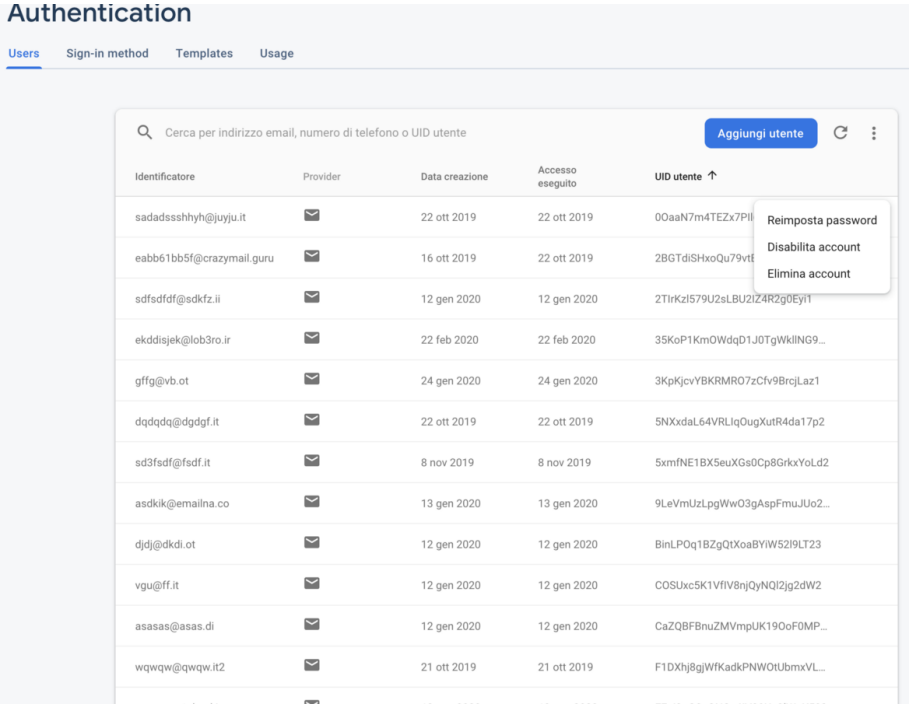

Figura 3.26. Le informazioni degli utenti registrati

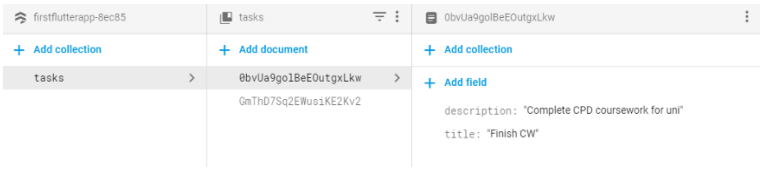

Figura 3.27. Esempio del servizio di Cloud in Firebase

Oltre al servizio offerto da Firebase, nell'app viene utilizzato un database locale SQLite, per la gestione dei crediti e dei debiti di ciascun utente.

Dalla Figura 3.27 è possibile notare che il Cloud è suddiviso in collezioni, mentre queste ultime sono suddivise in documenti, all'interno dei quali è possibile salvare una coppia di oggetti, dove il primo è il nome dell'attributo e il secondo è il suo valore.

Applicando tutti questi concetti al progetto CoFit, si è pensato di avere una collezione denominata "Users" e identificare il documento con l'UID di ciascun cliente. Successivamente, all'interno del documento, vengono salvati tutti i dati anagrafici del cliente.

Per quanto riguarda gli F24, possiamo pensare all'UID per denominare la collezione e al nome per identificare ciascun documento. Infine, all'interno di quest'ultimo salvare tutti gli attributi di un F24 con i rispettivi valori.

Il Database in Realtime di Firebase è, invece, suddiviso in nodi.

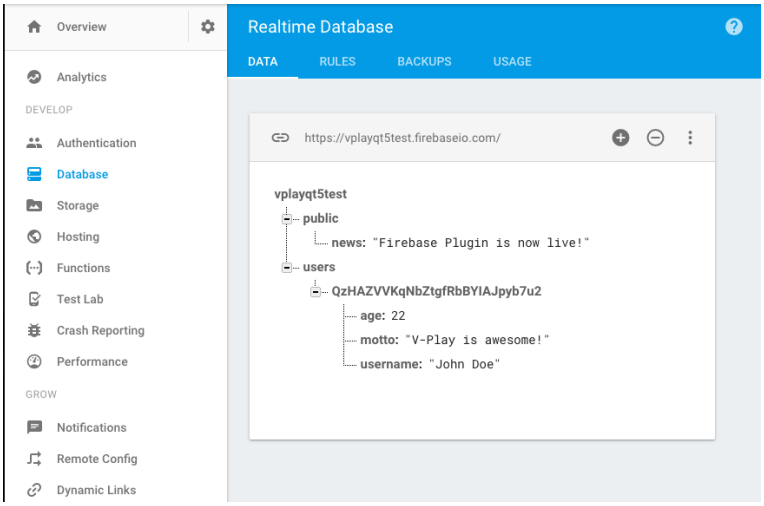

Figura 3.28. Esempio del servizio di Database in Realtime di Firebase

Il nodo radice corrisponde alle collezioni del Cloud; tutti i nodi figli corrispondono ai documenti. Le foglie di questo albero binario corrispondono ai diversi valori di ciascun attributo.

Applicando, ancora una volta, questi concetti al progetto CoFiT potremmo pensare di identificare il nodo radice con l'UID dell'utente e i nodi figli con l'attributo key del documento caricato. Gli altri attributi dell'istanza Documento saranno le nostre foglie.

Storage

Lo storage di Firebase è stato utilizzato per memorizzare tutti i file condivisi tra il cliente e il commercialista.

Nello storage di Firebase (Figura 3.29), oltre al file, viene salvato il nome, lo spazio di memoria occupato e il tipo di file.

All'interno dello Storage vengono, inoltre, archiviate le immagini del profilo degli utenti.

Dallo storage l'utente può scaricare un file attraverso il suo URL, facilmente reperibile dal Database in Realtime.

Messaging

Firebase Cloud Messaging (FCM) è una soluzione cloud multipiattaforma per messaggi e notifiche per applicazioni Android, iOS e Web.

FCM ha tre funzionalità principali. La prima funzionalità consente all'utente di ricevere messaggi di notifica o messaggi di dati che possono essere decifrati dal codice dell'applicazione. La seconda funzionalità è la destinazione dei messaggi. I messaggi possono essere inviati all'applicazione client attraverso diversi metodi; dalla piattaforma FCM ai singoli dispositivi, ai gruppi di dispositivi specificati o ai

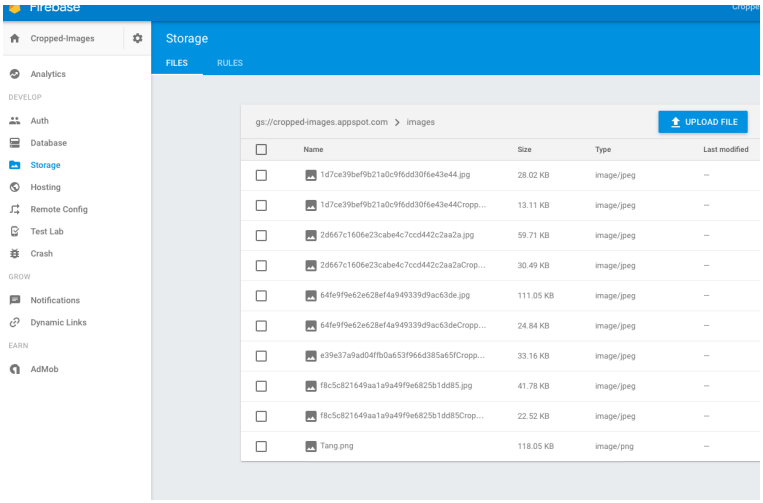

Figura 3.29. Esempio del servizio di Storage di Firebase

dispositivi sottoscritti a particolari domini tematici. La terza funzionalità chiave è il canale di connessione dalle applicazioni client al server. FCM consente l'invio di messaggi di vario tipo da dispositivi o app client selezionati tramite il canale FCM.

Nel progetto CoFiT utilizzeremo FCM per l'invio e la ricezione delle notifiche push.

Per sfruttare questa funzionalità è necessario salvare il token del dispositivo dell'utente.

Il token è una chiave, in formato alfanumerico, che identifica il dispositivo su cui è installato il software.

Il token del dispositivo di ciascun utente verrà salvato all'interno del Cloud di Firebase, tra i dati anagrafici dei clienti.

La notifica push verrà inviata al commercialista ogniqualvolta il cliente inserirà un nuovo documento, e viceversa.

Il commercialista farà recapitare al cliente una notifica anche quando inserirà un nuovo F24.

4

Implementazione dell'app e manuale utente

In questo capitolo verrà descritta la fase di implementazione dell'app e verrà proposto il manuale utente. In particolare, vedremo l'implementazione del Front-end e del Back-end, con uno sguardo alle tecniche e alle librerie utilizzate per arrivare al prodotto finale.

4.1 Implementazione dell'app

Nel capitolo precedente abbiamo individuato la possibile struttura dell'app, ora non resta che renderla reale attraverso l'implementazione.

L'implementazione, detta anche sviluppo o codifica del prodotto, è la fase che concretizza la soluzione software attraverso la programmazione, ovvero la stesura del codice.

In altre parole, in questa sezione vedremo la codifica dell'app CoFiT mediante il linguaggio di programmazione e markup scelti. In particolare, codificheremo la logica del Front-end utilizzando i linguaggi Java ed XML, e la logica del Back-end per collegare il nostro progetto ai servizi offerti da Firebase.

Per sfruttare diversi servizi è necessario, inoltre, installare nel progetto diverse librerie di terze parti, le quali saranno utilizzate sia per il Front-end che per il Back-end.

4.1.1 Implementazione della parte Front-end

Nell'implementazione del Front-end si lavorerà alla codifica del codice necessario per collegare le interfacce utente (in file `.xml`) alle classi Java che rappresentano la logica del software. In particolare, vedremo le differenze tra usare un Fragment o un'Activity.

Librerie utilizzate nel Front-end

Nel progetto CoFiT sono state utilizzate diverse librerie, alcune prodotte da Android e alcune da terze parti; queste, poi, dovranno essere importate nel file `.class` che le utilizza. Esse sono le seguenti:

- `androidx.navigation:navigation-fragment:2.3.1 (ui:2.3.1)`: essa è stata installata per ottimizzare la navigazione all'interno dell'app, per ridurre gli errori e semplificare il lavoro dei Fragment.
- `com.google.android.material:material:1.2.1`: essa è stata installata per utilizzare il Material design di Google.
- `com.getbase:floatingactionbutton:1.10.1`: essa è stata installata per poter utilizzare il Floating Button.
- `com.makeramen:roundedimageview:2.3.0`: essa è stata installata per poter dare un effetto arrotondato alle immagini.
- `com.squareup.picasso:picasso:2.71828`: essa è stata installata per visualizzare le immagini;
- `com.github.chrisbanes:PhotoView:2.3.0`: essa è stata utilizzata per poter effettuare lo zoom su alcune immagini.
- `de.hdodenhof:circleimageview:3.1.0`: essa è stata utilizzata per inserire l'immagine del profilo all'interno di un cerchio.

Implementazione della User Interface

In questa sezione vedremo nel dettaglio da quali oggetti `View` sono costituite le interfacce utente dell'app CoFiT.

La prima interfaccia utente implementata è quella di registrazione (Figura 4.1).

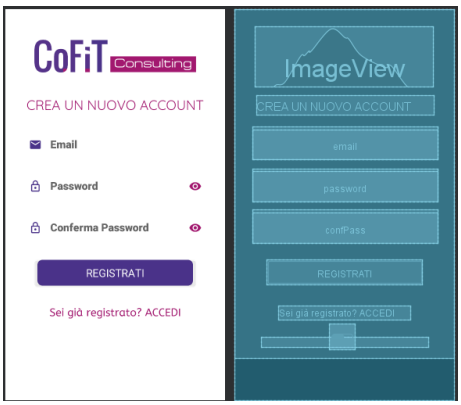

Figura 4.1. Interfaccia relativa all'Activity di registrazione

Il Listato 4.1 mostra tutto il codice XML necessario per creare l'interfaccia per la registrazione. Il codice è riportato interamente solo in questo caso, con lo scopo di rendere l'idea dei diversi tag che costituiscono ciascuna View di un'interfaccia utente.

```
        <?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white"
```

```
tools:context=".RegisterActivity">

<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/imageVIew"
        android:layout_width="333dp"
        android:layout_height="112dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"
        android:background="@drawable/background_white"
        android:padding="5dp"
        android:src="@drawable/nuovo_logo"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:fontFamily="@font/quicksand"
        android:text="@string/crea_un_nuovo_account"
        android:textColor="@color/prugna"
        android:textSize="25sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageVIew"
        app:layout_constraintVertical_bias="0.0" />



    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/etEmailLayout"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox.Dense"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="24dp"
        android:layout_marginEnd="32dp"
        app:boxStrokeColor="@color/viola"
        app:hintTextColor="@color/prugna"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2"
        app:layout_constraintVertical_bias="0.0"
        app:startIconDrawable="@drawable/ic_email_black_24dp"
        app:startIconTint="@color/viola">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/email"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="sans-serif-black"
            android:hint="@string/email"
            android:inputType="textEmailAddress"
            android:padding="15dp"
            android:textColor="@color/viola"
            android:textColorHint="@android:color/white"
            android:textSize="20sp" />

    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/etPasswordLayout"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox.Dense"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="32dp"
        app:hintTextColor="@color/prugna"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/etEmailLayout"
        app:layout_constraintVertical_bias="0.0"
        app:passwordToggleEnabled="true"
        app:passwordToggleTint="@color/prugna"
        app:startIconDrawable="@drawable/ic_lock_outline_black_24dp"
        app:startIconTint="@color/viola">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/password"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:drawableLeft="@drawable/ic_lock_outline_black_24dp"
```

```
        android:fontFamily="sans-serif-black"
        android:hint="@string/password"
        android:inputType="textPassword"
        android:padding="15dp"
        android:textColor="@color/viola"
        android:textSize="20sp" />

</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/etConfPasswordLayout"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox.Dense"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="32dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="32dp"
    app:hintTextColor="@color/prugna"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etPasswordLayout"
    app:layout_constraintVertical_bias="0.0"
    app:passwordToggleEnabled="true"
    app:passwordToggleTint="@color/prugna"
    app:startIconDrawable="@drawable/ic_lock_outline_black_24dp"
    app:startIconTint="@color/viola">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/confPass"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:drawableLeft="@drawable/ic_lock_outline_black_24dp"
        android:fontFamily="sans-serif-black"
        android:hint="@string/conferma_password"
        android:inputType="textPassword"
        android:padding="15dp"
        android:textColor="@color/viola"
        android:textSize="20sp" />

</com.google.android.material.textfield.TextInputLayout>

<Button
    android:id="@+id/btnRegistra"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="60dp"
    android:layout_marginTop="32dp"
    android:layout_marginEnd="60dp"
    android:layout_marginBottom="136dp"
    android:background="@drawable/et_style"
    android:backgroundTint="@color/prugna"
    android:fontFamily="sans-serif"
    android:text="@string/registrati"
    android:textColor="#FFFFFF"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etConfPasswordLayout"
    app:layout_constraintVertical_bias="0.0" />

<TextView
    android:id="@+id/btnViewLogin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:fontFamily="@font/abeezee"
    android:text="@string/sei_gi_registrato_accedi"
    android:textColor="@color/prugna"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/btnRegistra"
    app:layout_constraintVertical_bias="0.13999999" />

<TextView
    android:id="@+id/condizioni"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="50dp"
    android:layout_marginTop="32dp"
    android:layout_marginEnd="50dp"
    android:textSize="15sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/btnViewLogin"
    app:layout_constraintVertical_bias="0.0" />


<ProgressBar
```

```
            android:id="@+id/progressBar"
            style="?android:attr/progressBarStyle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="8dp"
            android:visibility="invisible"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.49"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/btnViewLogin"
            app:layout_constraintVertical_bias="0.0" />


    </androidx.constraintlayout.widget.ConstraintLayout>

</ScrollView>
```

Listato 4.1. Codice del file XML relativa all'interfaccia utente dell'Activity di registrazione all'app CoFiT

In particolare, il layout per la registrazione ha nella parte superiore una ImageView centrale, utilizzata per mostrare il logo dello studio Co.Fi.T., e una TextView, che indica il servizio offerto da questa Activity. Nella parte centrale ci sono le opportune caselle di testo per poter inserire i dati richiesti dalla registrazione. Queste caselle non sono semplici EditText, ma sono state realizzate sfruttando la libreria Material Design. Nella parte inferiore sono, invece, presenti una TextView che, se cliccata, permette di passare alla schermata di Login (Figura 4.2), e un Button che, una volta cliccato, attiva l'evento e invia tutti i dati inseriti a Firebase per completare la registrazione.

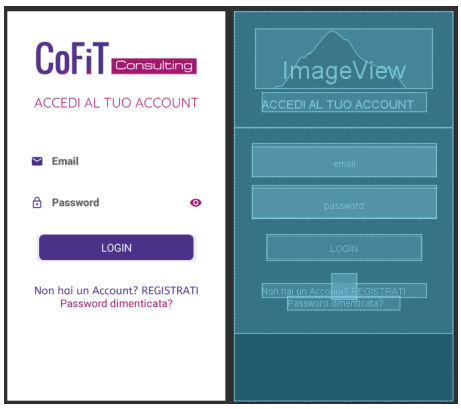

Figura 4.2. Interfaccia relativa all'Activity di login

Il layout presente in Figura 4.2 è molto simile al precedente. Infatti, gli oggetti di tipo View utilizzati sono gli stessi.

Particolarmente complicata è stata l'implementazione della home page del cliente. Questo perché quest'ultima doveva ospitare al proprio interno una NavigationView (Listato 4.2) in grado di far comparire il menù a comparsa, un Frame che ospitasse i corrispettivi Fragment (Listato 4.3), una TabLayout per inserire la lista delle tasse, dei crediti e dei debiti (Listato 4.4), e la ViewPager per ospitare i Fragment della TabLayout (Listato 4.5).

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/drawerLayout"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".MainActivity">

    <include
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        layout="@layout/frame_main"/>

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/navigationView"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:background="@android:color/white"
        app:menu="@menu/navigation_menu"
        app:headerLayout="@layout/layout_navigation_header"
        android:fitsSystemWindows="true"
        android:layout_gravity="start"/>


</androidx.drawerlayout.widget.DrawerLayout>
```

Listato 4.2. Codice del file XML utilizzato per ospitare la NavigationView nella home page del cliente

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">


    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/viola"
        android:theme="@style/ThemeOverlay.AppCompat.Dark"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0"
        app:title="Benvenuto␣in␣Cofit" />


    <FrameLayout
        android:id="@+id/fragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/toolbar" />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Listato 4.3. Codice del file XML utilizzato per ospitare i Fragment associati alla NavigationView

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">


    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:layout_marginStart="24dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="24dp"
```

```
        android:background="@android:color/transparent"
        android:foreground="@drawable/nuovo_logo_toolbar"
        android:foregroundGravity="center"
        android:theme="@style/ThemeOverlay.AppCompat.Dark"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tableLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:background="@color/viola"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/toolbar"
        app:tabSelectedTextColor="@color/prugna2"
        app:tabTextColor="@android:color/white">

        <com.google.android.material.tabs.TabItem
            android:id="@+id/crediti"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />


        <com.google.android.material.tabs.TabItem
            android:id="@+id/debiti"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />


        <com.google.android.material.tabs.TabItem
            android:id="@+id/tasse"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />


    </com.google.android.material.tabs.TabLayout>

    <androidx.viewpager.widget.ViewPager
        android:id="@+id/view_pager"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_marginTop="1dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tableLayout" />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Listato 4.4. Codice del file XML utilizzato per ospitare la TabLayout della home page del cliente

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".user.registro_finanziario.HomeFragment">

    <include
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        layout="@layout/tablayout_main"/>

</FrameLayout>
```

Listato 4.5. Codice del file XML utilizzato per ospitare la TabLayout della home page del cliente

La NavigationView, oltre ad ospitare l'elenco delle voci del menù, ha bisogno anche di un'intestazione (Figura 4.3). Questa è stata sviluppata, semplicemente, con una ImageView circolare e l'indirizzo e-mail del cliente.

Una volta completata la struttura fisica della NavigationView, bisogna creare un file `.xml` per implementare l'elenco delle voci del menù (Listato 4.6)

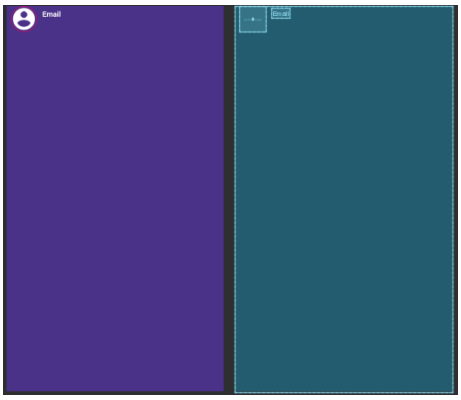

Figura 4.3. Intestazione della NavigationView

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">


    <item android:id="@+id/menuHome"
        android:title="@string/home"
        android:icon="@drawable/ic_home_black_24dp"/>

    <item android:id="@+id/menuProfile"
        android:title="@string/profilo"
        android:icon="@drawable/ic_account_circle_black_24dp" />

    <item android:id="@+id/menuNovita"
        android:title="@string/novit"
        android:icon="@drawable/ic_star_black_24dp"/>


    <item android:title="@string/documenti">


        <menu>

            <item android:id="@+id/menuInserisciDoc"
                android:title="@string/inserisci_documenti"
                android:icon="@drawable/ic_cloud_upload_black_24dp" />


            <item android:id="@+id/menuVisualizzaDoc"
                android:title="@string/scarica_documenti"
                android:icon="@drawable/ic_description_black_24dp" />


        </menu>

    </item>
```

Listato 4.6. Estratto del Codice del file XML per l'elenco del menù della NavigationView

Purtroppo, non è possibile vedere il design mode completo della home page, perché il codice XML presente nei listati precedenti verrà unito a runtime da Android.

All'interno del Fragment per le tasse (ma anche nel caso dei crediti e dei debiti) verrà implementata una RecyclerView (Listato 4.7). Quest'ultima rappresenta un oggetto di tipo View adatto per la creazione delle liste per la visualizzazione dei dati ottenuti da un database. Ogni elemento della lista è formato da più oggetti View, la RecyclerView permette di creare tanti elementi della lista quanti sono i dati ottenuti dal database.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@color/grigio">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerview_tasse"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:background="@color/grigio"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:divider="@android:color/transparent" />

</FrameLayout>
```

Listato 4.7. Codice del file XML del Fragment per la visualizzazione degli F24

Con la RecyclerView abbiamo bisogno di creare un nuovo oggetto (Figura 4.4) in grado di contenere tutti i dati di un F24; questo oggetto verrà ripetuto per ogni F24 trovato nel database.

Figura 4.4. Design mode relativo ad un F24

L'interfaccia utente per l'inserimento dell'anagrafica (Figura 4.5) è costituita da un RadioGroup con cui il cliente può dichiarare se è un'azienda, una società o un professionista, da diverse EditText e da uno Spinner che richiede il tipo di contabilità utilizzata dal cliente.

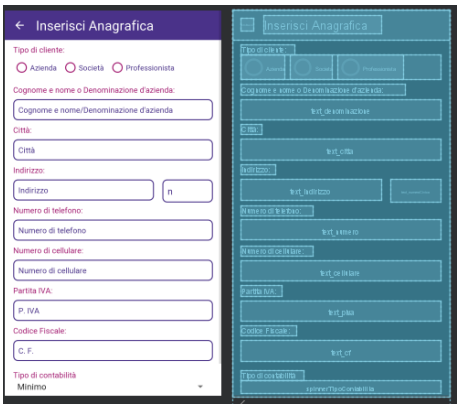

Figura 4.5. Design mode relativo al file XML per l'inserimento dell'anagrafica

Il layout principale dell'interfaccia per l'inserimento dell'anagrafica è uno ScrollView; perciò, in Figura 4.5 non è possibile vedere il Button per completare l'inseri-

mento dei dati anagrafici. Una volta cliccato il Button, il sistema ci farà vedere la schermata del nostro profilo modificato (Figura 4.6).

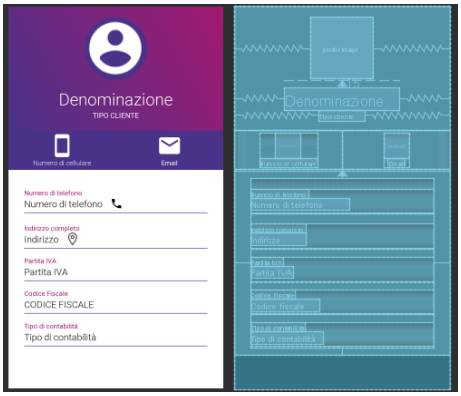

Figura 4.6. Design mode relativo al file XML del profilo del cliente

Il layout per visualizzare il profilo è costituito da un'intestazione sfumata con al centro una CircleView, che conterrà l'immagine del profilo e il nome del cliente. L'immagine del profilo può essere modificata premendo su di essa. Nella parte centrale ci sono due ImageView con il numero e l'indirizzo e-mail del cliente. Nella parte inferiore ci sono il resto dei dati anagrafici del cliente, visibili attraverso diverse TextView. La Toolbar verrà aggiunta a runtime; all'interno di essa ci sarà un piccolo menù per modificare il profilo, il quale ci porterà nell'area per l'inserimento dell'anagrafica.

Infine, per l'implementazione dell'area dedicata al cliente, abbiamo le interfacce utente per la gestione dei file.

L'interfaccia utente per il caricamento dei file (Figura 4.7) è costituita da una ImageView rappresentata da una cornice centrale, la quale mostrerà un'anteprima se il cliente caricherà un'immagine. Quest'ultimo potrà scegliere il file da caricare attraverso il Button presente nella parte superiore e, attraverso uno Spinner, potrà specificare in quale categoria rientra il documento che intende caricare. Nella parte inferiore c'è una ProgressBar che si caricherà non appena il cliente premerà sul Button "Inserisci". Una volta cliccato quest'ultima, il sistema ci porterà nell'area per la visualizzazione di tutti i file condivisi con il commercialista (Figura 4.8).

Quest'ultima è costituita da una SearchBar, che permette di cercare il file per nome, e da una RecyclerView. Anche in questo caso, la RecyclerView ha bisogno di un oggetto in grado di contenere altri oggetti View (Figura 4.9), ovvero il nome del file e un ImageButton che rappresenta un menù. Quest'ultimo permetterà di procedere al download del file o alla sua eliminazione.

Terminata l'implementazione della sezione del cliente, passiamo ora a quella del commercialista.

La home page del commercialista (Figura 4.10) è composta sostanzialmente dalla lista dei clienti registrati. Quest'ultima è stata implementata mediante una RecyclerView; perciò, anche in questo caso, è stato necessario creare un elemento che

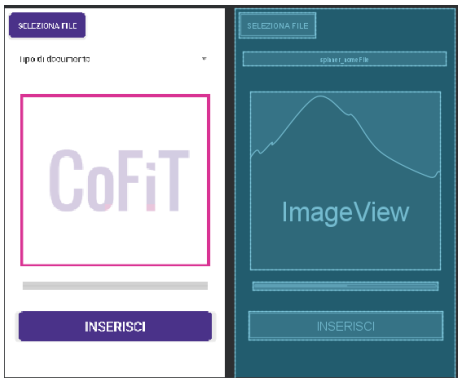

Figura 4.7. Design mode relativo al file XML dell'interfaccia per inserire un file

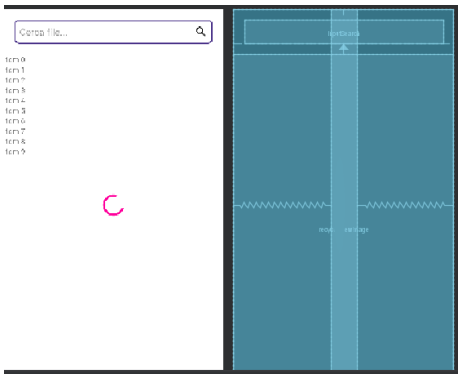

Figura 4.8. Design mode relativo al file XML della lista dei file condivisi

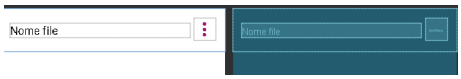

Figura 4.9. Design mode del file XML di un elemento della lista dei file condivisi

ospiti alcune informazioni del cliente (Figura 4.11). Tali informazioni comprendono il nome del cliente, l'indirizzo e-mail e la sua immagine del profilo. All'interno della Toolbar troviamo una comoda SearchBar che permette di cercare un cliente per nome. Inoltre, nella Toolbar è stato implementato un piccolo menù.

Cliccando su un elemento della RecyclerView possiamo scegliere una delle opzioni del menù del cliente (Figura 4.12).

Le opzioni in Figura 4.12 non sono tutte visibili; per questo motivo il layout principale è uno ScroolView. Questa interfaccia è stata implementata con una griglia composta da un ImageButton e da una TextView.

Nell'interfaccia mostrata in Figura 4.12, cliccando su "Anagrafica", è possibile vedere il profilo del cliente. L'interfaccia è praticamente uguale a quella della Figura 4.6. Analogamente, cliccando su "Visualizza F24", è possibile vedere la lista degli F24 del cliente. Tale lista è implementata con una RecyclerView e ciascun elemento è uguale a quello in Figura 4.4. Lo stesso discorso vale per le altre voci del menù,

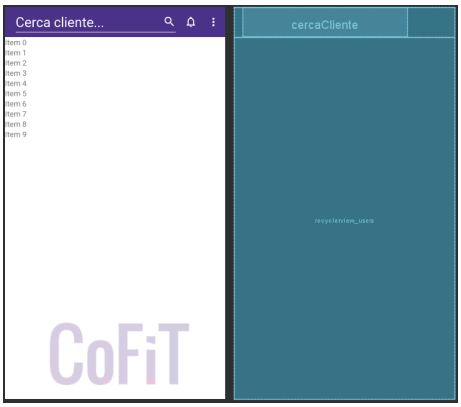

Figura 4.10. Design mode relativo al file XML della Homepage del commercialista

Figura 4.11. Design mode relativo al file XML di un elemento della lista clienti

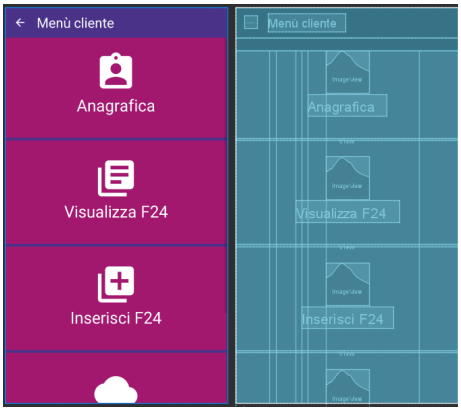

Figura 4.12. Design mode relativo al file XML del menù del cliente

ovvero "Visualizza documenti" e "Inserisci documenti", dove le loro interfacce sono, praticamente, identiche a quelle viste nel caso del cliente. L'unica modifica è nel caso dell'inserimento del file, in quanto è presente, oltre allo Spinner per selezione la categoria alla quale appartiene il file, anche una EditText per fornire un'ulteriore descrizione del file.

Infine, vediamo l'interfaccia per inserire un nuovo F24 (Figura 4.13).

L'interfaccia presente in Figura 4.13 è composta da tre Spinner (per scegliere il modulo al quale appartiene la tassa, l'anno di riferimento di quest'ultima e il suo periodo di riferimento) e da due EditText per l'importo e la data di scadenza. Infine, tramite il Button "Inserisci", potremo aggiungere la tassa.

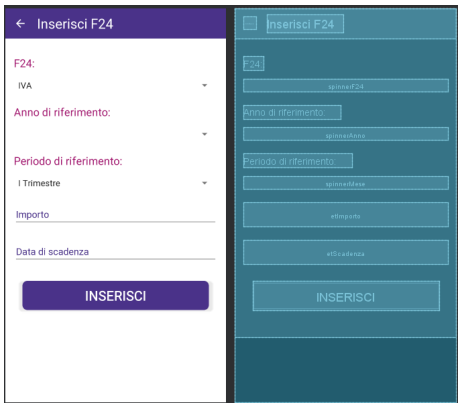

Figura 4.13. Design mode relativo al file XML per l'inserimento di un F24

Implementazione della logica dell'app

In questa sezione vedremo come è stata implementata la logica del Front-end dell'applicazione CoFiT. In particolare, vedremo come il file `.class` riesce a recuperare il file `.xml` e a trovare tutte gli oggetti di tipo View implementati.

Per capire come vengono collegati i file `.xml` e `.class` esaminiamo un estratto del codice dell'Activity relativa alla registrazione di un nuovo utente (Listato 4.8).

```
package com.cofitconsulting.cofit;

+ import...


public class RegisterActivity extends AppCompatActivity {

    private EditText mEmail, mPassword, mConfPass;
    private Button mBtnRegistra;
    private TextView linkLogin, condizioni;
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);

        mEmail = findViewById(R.id.email);
        mPassword = findViewById(R.id.password);
        mConfPass = findViewById(R.id.confPass);
        mBtnRegistra = findViewById(R.id.btnRegistra);
        linkLogin = findViewById(R.id.btnViewLogin);
        fAuth = FirebaseAuth.getInstance(); //crea un'istanza
        progressBar = findViewById(R.id.progressBar);
        condizioni = findViewById(R.id.condizioni);
        String htmlText = "Registrandoti␣dichiari␣di␣aver␣preso␣visione␣e␣accetti␣integralmente␣i␣nostri␣" +
                "<A␣HREF='https://www.cofitconsulting.com/termini-e-condizioni-duso/'>termini␣e␣condizioni</A>.␣" +
                "\nI␣tuoi␣dati␣personali␣saranno␣trattati␣in␣conformit␣con␣privacy␣policy.";
        condizioni.setText(Html.fromHtml(htmlText));
        condizioni.setMovementMethod(LinkMovementMethod.getInstance());
```

Listato 4.8. Codice Java relativo all'Activity per la registrazione di un utente

Dal Listato 4.8 osserviamo che il collegamento tra i due file avviene grazie al metodo `setContentView()`.

Per identificare tutte le View presenti all'interno del file `.xml` bisogna inizializzare degli oggetti dello stesso tipo di quelli presenti nell'interfaccia utente. Successivamente, grazie al metodo `findViewById()`, avverrà il collegamento con le View

presenti nell'interfaccia utente, in quanto esse hanno un tag `id`. La stessa tecnica è utilizzata anche per tutte le altre Activity.

Nel caso dei Fragment, la procedura cambia. Per capire meglio analizziamo un estratto del codice dal file `.class` del Fragment relativo alla visualizzazione del profilo utente (Listato 4.9).

```
package com.cofitconsulting.cofit.user.anagrafica;

+ import...


public class ProfiloFragment extends Fragment {

    private TextView nome, numero, numeroCell, email, indirizzo_completo, contabilita, pi,
    cf, tipo_azienda, viewNumero, viewIVA, viewCF;
    private CircleImageView profileImage;
    private static final int GALLERY_INTENT_CODE = 1023;
    private String userId;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        View v = inflater.inflate(R.layout.fragment_profilo, container, false);

        profileImage = v.findViewById(R.id.profileImage);
        tipo_azienda = v.findViewById(R.id.textTipoAzienda);
        nome = v.findViewById(R.id.textNome);
        numero = v.findViewById(R.id.textNumero);
        numeroCell = v.findViewById(R.id.textCellulare);
        email = v.findViewById(R.id.textEmail);
        indirizzo_completo = v.findViewById(R.id.textIndizzo);
        contabilita = v.findViewById(R.id.text_contabilita);
        pi = v.findViewById(R.id.text_pIva);
        cf = v.findViewById(R.id.text_cf);
        viewCF = v.findViewById(R.id.text_cf);
        viewIVA = v.findViewById(R.id.text_pIva);
        viewNumero = v.findViewById(R.id.text_numero);
```

Listato 4.9. Codice Java del Fragment relativo alla visualizzazione del profilo utente

Come si evince dal Listato 4.9, per collegare l'interfaccia al file `.class` occorre dapprima creare un oggetto di tipo View. Quest'ultimo verrà collegato all'interfaccia mediante il metodo `inflate()`. Successivamente, non cambia quasi nulla e tutte le View dell'interfaccia vengono recuperate con gli id.

Adesso, ricordiamo che un Fragment ha bisogno di un'Activity che lo ospiti. Siccome il Fragment per la visualizzazione del profilo è accessibile dalla Navigation-View, e quest'ultima è stata implementata nella home page del cliente, l'Activity che si occupa di ospitare il Fragment è proprio quella della home page (Listato 4.10).

```
@Override
  public boolean onNavigationItemSelected(@NonNull MenuItem item) {
      drawerLayout.closeDrawer(GravityCompat.START);

      FragmentManager fm = getSupportFragmentManager();
      FragmentTransaction ft = fm.beginTransaction();

      switch (item.getItemId()) {
          case R.id.menuHome: {
              Intent intent = new Intent(MainActivity.this, MainActivity.class);
              startActivity(intent);
              finish();
              break;
          }

          case R.id.menuProfile: {

              ProfiloFragment fragProfilo = new ProfiloFragment();
              ft.replace(R.id.fragment, fragProfilo);
              ft.commit();
              toolbar.setTitle("Il␣tuo␣profilo");
              toolbar.getMenu().clear();
              toolbar.inflateMenu(R.menu.menu_profilo);
              break;
          }
```

Listato 4.10. Frammento di Codice Java relativo al collegamento del Fragment del profilo nella NavigationView della home page del cliente

Il Listato 4.10 presenta un estratto, non completo, di codice relativo all'implementazione del menù della NavigationView. Al suo interno vi è il metodo `onNavigationItemSelected()` che recupera gli id dell'elenco del menù del Listato 4.6 e associa ad ogni voce l'apertura di un nuovo Fragment o Activity.

Una volta collegata l'interfaccia utente al file `.class`, sia che esso sia un Fragment o sia che esso sia un'Activity, è possibile recuperare tutto ciò che scriviamo all'interno delle EditText o ciò che selezioniamo negli Spinner; da lì poi non resta che salvarlo all'interno di Firebase, ma quest'ultimo aspetto verrà esaminato nel momento in cui discuterà l'implementazione della parte Back-end.

4.1.2 Implementazione della parte Back-end

Nell'implementazione della parte di Back-end si lavorerà alla codifica del codice necessario per collegare l'app CoFiT a Firebase. In particolare, vedremo come funzionano l'autenticazione, il database, lo storage e il messaging di Firebase. Inoltre, vedremo cosa succede quando clicchiamo su un oggetto Button dell'interfaccia utente, ovvero come gestire gli eventi.

Librerie utilizzate

Le librerie utilizzate per la parte di Back-end sono necessarie per usufruire dei servizi di Firebase. Esse sono le seguenti:

- `com.google.firebase:firebase-auth:20.0.1`: essa viene utilizzata per usufruire del servizio di autenticazione di Firebase.
- `com.google.firebase:firebase-analytics:18.0.0`: essa raccoglie dati sull'utilizzo e sul comportamento dell'app.
- `com.google.firebase:firebase-firestore:22.0.0`: essa viene utilizzata per usufruire del Cloud Firestore di Firebase.
- `com.google.firebase:firebase-database:19.5.1`: è utilizzata per usufruire del Realtime Database di Firebase.
- `com.google.firebase:firebase-storage:19.2.0`: essa viene utilizzata per usufruire dello Storage di Firebase.
- `com.google.firebase:firebase-messaging:21.0.0`: essa viene utilizzata per usufruire del servizio di Messaging di Firebase.
- `com.squareup.retrofit2:retrofit:2.3.0` e `com.squareup.retrofit2:converter-gson:2.3.0`: essa viene utilizzata per l'invio delle notifiche push.

Implementazione dell'autenticazione

Come detto in precedenza, Firebase permette diversi tipi di autenticazione, ovvero autenticazione tramite e-mail e password, con l'account di Facebook o con quello di Google.

Per quanto riguarda l'app CoFiT è stato scelto quello con e-mail e password.

Tornando alla nostra interfaccia di registrazione (Figura 4.1), una volta cliccato sull'oggetto Button, tutto ciò che è contenuto nelle caselle di testo viene inviato a Firebase per completare l'autenticazione (Listato 4.11).

```
public class RegisterActivity extends AppCompatActivity {

    private EditText mEmail, mPassword, mConfPass;
    private Button mBtnRegistra;
    private TextView linkLogin, condizioni;
    private ProgressBar progressBar;
    private FirebaseAuth fAuth; //crea un oggetto della classe FirebaseAuth per l'autenticazione

mBtnRegistra.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
             String email = mEmail.getText().toString().trim();
             String password = mPassword.getText().toString().trim();
             String confPass = mConfPass.getText().toString().trim();

             progressBar.setVisibility(View.VISIBLE);

             //Registra l'utente in Firebase
             fAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener
             (new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) { //bisogna gestire le attivit asincrone
                   if(task.isSuccessful())
                   {
                      progressBar.setVisibility(View.INVISIBLE);

                      Toast.makeText(RegisterActivity.this, "Utente creato correttamente",
                      Toast.LENGTH_LONG).show();
                      startActivity(new Intent(getApplicationContext(), MainActivity.class));
                      finish();
                   } else if(!task.isSuccessful())
                   {
                      progressBar.setVisibility(View.INVISIBLE);
                      try
                      {
                         throw task.getException();
                      }
                      // if user enters wrong email.
                      catch (FirebaseAuthInvalidCredentialsException malformedEmail)
                      {
                         Toast.makeText(RegisterActivity.this, "L'email  errata!", Toast.LENGTH_LONG).show();

                      }
                      catch (FirebaseAuthUserCollisionException existEmail)
                      {
                         Toast.makeText(RegisterActivity.this, "Email gi utilizzata!", Toast.LENGTH_LONG).show();
                      }
                      catch (Exception e)
                      {
                         Toast.makeText(RegisterActivity.this, "Errore!" , Toast.LENGTH_LONG).show();
                      }

                   }

                }
             });
          }
       });
```

Listato 4.11. Frammento di Codice Java relativo all'autenticazione

Come si evince dal Listato 4.11, l'utente clicca sul Button "Registrati", si attiva l'evento collegato e lo si gestisce attraverso il metodo `onClick()`. Da qui in poi viene recuperato il testo contenuto all'interno delle caselle di testo, e il metodo `fAuth.createUserWithEmailAndPassword()` si occupa di inviare tutto a Firebase per registrare l'utente. All'interno del metodo `fAuth.createUserWithEmailAndPassword()` è possibile gestire sia l'esito positivo che quello negativo. In quest'ultimo caso bisogna gestire le eccezioni.

Molto simile è il caso del login, l'unica differenza sostanziale è che useremo il metodo `fAuth.signInWithEmailAndPassword()` invece di `fAuth.createUserWithEmailAndPassword()`.

La procedura di autenticazione dev'essere completata in background; quindi, viene utilizzato un oggetto di tipo AsyncTask.

Implementazione del database

Nell'implementazione del database useremo i servizi Firestore Cloud e Realtime Database, messi entrambi a disposizione da Firebase. Il primo viene utilizzato per salvare e recuperare gli F24 e i dati anagrafici dei clienti. Il secondo viene utilizzato per salvare e recuperare le informazioni dei file condivisi.

Facendo riferimento alla Figura 4.14, quando l'utente clicca su "Inserisci", viene attivato l'evento per inviare tutti i dati al cloud di Firebase (Listato 4.12).

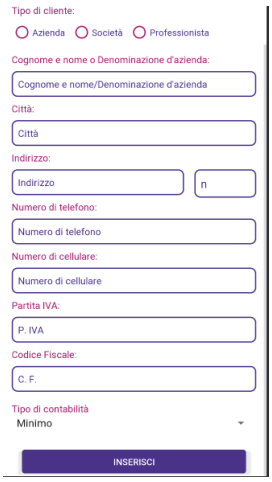

Figura 4.14. Interfaccia relativa all'inserimento dell'anagrafica

```
btnSalva.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
              final String nome = text_nome.getText().toString();
              final String indirizzo = text_indirizzo.getText().toString();
              final String numeroCivico = text_numeroCivico.getText().toString();
              final String citta = text_citta.getText().toString();
              final String numero = text_numero.getText().toString();
              final String cellulare = text_cellulare.getText().toString();
              final String iva = text_iva.getText().toString();
              final String cf = text_cf.getText().toString();
              final String contabilita = text_contabilita.getSelectedItem().toString();
              final String email = fAuth.getCurrentUser().getEmail();

String uid = FirebaseAuth.getInstance().getCurrentUser().getUid();
              String tokenUser = FirebaseInstanceId.getInstance().getToken();
              writeOnDatabaseUsers(nome, citta, indirizzo, numeroCivico, numero, cellulare, iva, cf, contabilita,
              tipo_cliente, email, uid, tokenUser);
              Toast.makeText(InserimentoAnagraficaActivity.this, "Inserimento avvenuto", Toast.LENGTH_SHORT).show();
              finish();
          }
      });
  }
```

Listato 4.12. Frammento di Codice Java utilizzato per gestire l'evento di inserimento dell'anagrafica

Come si evince dal Listato 4.12, il metodo `onClick()` recupera tutto ciò che si trova all'interno delle EditText e nello Spinner, mentre il metodo `writeOnDatabaseUsers()` (Listato 4.13) si occupa di inviare tutti i dati al cloud.Questi ultimi verranno inseriti all'interno del documento denominato con l'UID del cliente appartenente alla collezione "Users".

Infine, mediante il metodo `onFinish()`, l'Activity relativa all'inserimento dell'anagrafica viene rimossa dalla memoria e il focus torna al Fragment relativo alla visualizzazione del profilo utente.

```
private void writeOnDatabaseUsers(String nome, String citta, String indirizzo, String numeroCivico,
                                  String numero, String cellulare, String iva, String cf, String contabilita,
                                  String tipo_cliente, String email, String uid, String token){
      Map<String, Object> user = new HashMap<>();
      user.put("Id", uid);
      user.put("Token", token);
      user.put("Email", email);
      user.put("Denominazione", nome);
      user.put("Numero␣di␣telefono", numero);
      user.put("Numero␣di␣cellulare", cellulare);
      user.put("Citt", citta);
      user.put("Indirizzo", indirizzo);
      user.put("Numero␣civico", numeroCivico);
      user.put("Partita␣IVA", iva);
      user.put("Codice␣Fiscale", cf);
      user.put("Tipo␣di␣contabilit", contabilita);
      user.put("Tipo␣cliente", tipo_cliente);
      user.put("search", nome.toLowerCase());

      FirebaseFirestore db = FirebaseFirestore.getInstance();
      db.collection("Users").document(uid).set(user);
  }
```

Listato 4.13. Frammento di Codice Java utilizzato per inserire i dati anagrafici sul Cloud

Adesso vediamo il codice relativo al recupero dei dati anagrafici dal cloud per poterli poi visualizzare all'interno dell'interfaccia del profilo utente (Listato 4.14).

```
//recupero le informazioni dal database e setto il text dele textView

          final DocumentReference documentReference = fStore.collection("Users").document(userId);
          documentReference.addSnapshotListener(new EventListener<DocumentSnapshot>() {
              @Override
              public void onEvent(@Nullable DocumentSnapshot documentSnapshot, @Nullable FirebaseFirestoreException e) {

                  try {
                  tipo_azienda.setText("Tipo␣Cliente:␣" + documentSnapshot.getString("Tipo␣cliente"));
                  nome.setText(documentSnapshot.getString("Denominazione"));
                  email.setText(documentSnapshot.getString("Email"));
                  String citta = documentSnapshot.getString("Citt");
                  String indirizzo = documentSnapshot.getString("Indirizzo");
                  String numeroCivico = documentSnapshot.getString("Numero␣civico");
                  String nTelefono = documentSnapshot.getString("Numero␣di␣telefono");
                  String pIva = documentSnapshot.getString("Partita␣IVA");
                  String cFiscale = documentSnapshot.getString("Codice␣Fiscale");
                  String nCellulare = documentSnapshot.getString("Numero␣di␣cellulare");
                  indirizzo_completo.setText(indirizzo + "␣" + numeroCivico + ",␣" + citta);
                  numero.setText(nTelefono);
                  numeroCell.setText(nCellulare);
                  contabilita.setText(documentSnapshot.getString("Tipo␣di␣contabilit"));
                  pi.setText(pIva);
                  cf.setText(cFiscale);
                  if (nTelefono.isEmpty()) {
                      numero.setText("Non␣inserito");
                  }
                  if (pIva.isEmpty()) {
                      pi.setText("Non␣inserita");
                  }
                  if (cFiscale.isEmpty()) {
                      cf.setText("Non␣inserito");
                  }
                  if (nCellulare.isEmpty()) {
                      numeroCell.setText("Non␣inserito");
                  }
                  } catch (Exception E)
                  {

                  }
              }
```

```
        });
```

Listato 4.14. Frammento di Codice Java utilizzato per recuperare i dati anagrafici dal Cloud

Nel Listato 4.14 osserviamo che il metodo `addSnapshotListener()` permette di recuperare tutti i dati salvati all'interno del documento appartenente al cliente della collezione "Users". Successivamente, viene settato il testo di ogni TextView dell'interfaccia utente.

Nel caso relativo all'inserimento e al recupero degli F24 la procedura sarà analoga.

Per quanto riguarda l'utilizzo del Realtime Database, analizzeremo il suo funzionamento nella sezione successiva, in quanto è strettamente collegato con lo storage di Firebase.

Implementazione dello storage

Abbiamo utilizzato lo storage di Firebase per memorizzare i file condivisi tra cliente e commercialista, e le immagini del profilo degli utenti.

Per quanto riguarda il caricamento dei file analizziamo il codice Java contenuto nel file `.class` del Fragment relativo all'inserimento di un documento (Listato 4.15).

```
btnCaricaImag.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
              String default_ = "Tipo␣di␣documento";
              String spinner = spinner_NomeFile.getSelectedItem().toString().trim();
              if(default_.equals(spinner))
              {
                  Toast.makeText(getActivity(), "Seleziona␣il␣tipo␣di␣documento␣da␣inviare", Toast.LENGTH_SHORT).show();
                  return;
              }
              else {
                  uploadFile();
              }
          }
      });
```

Listato 4.15. Frammento di Codice Java utilizzato per gestire l'evento per inserire un file nello storage

L'utente, una volta che ha scelto il file da inserire e selezionato una voce dallo Spinner, può cliccare sull'oggetto Button per avviare l'evento. Il metodo `onClick()` verificherà che è stato selezionato un modello dallo Spinner e chiamerà il metodo `uploadFile()` (Listato 4.16).

```
private void uploadFile() {
      if (fileUri != null) {
          StorageReference fileReference = storageReference.child(System.currentTimeMillis()
                  + "." + utility.getFileExtension(fileUri, getContext()));

          uploadTask = fileReference.putFile(fileUri)
                  .addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
                      @Override
                      public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                          Handler handler = new Handler();
                          handler.postDelayed(new Runnable() {
                              @Override
                              public void run() {
                                  progressBar.setProgress(0);
                              }
```

```
        }, 500);

        Toast.makeText(getContext(), "File caricato", Toast.LENGTH_SHORT).show();
        Task<Uri> urlTask = taskSnapshot.getStorage().getDownloadUrl();
        while(! urlTask.isSuccessful());
        Uri downloadUrl = urlTask.getResult();
        String nome = spinner_NomeFile.getSelectedItem().toString().trim();
        Date currentTime = Calendar.getInstance().getTime();
        DateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
        String strDate = dateFormat.format(currentTime);
        String fileName = nome + " (" + strDate + ")";
        ModelFile modelFile = new ModelFile(fileName, downloadUrl.toString());
        String uploadId = databaseReference.push().getKey();
        databaseReference.child(uploadId).setValue(modelFile);
```

Listato 4.16. Frammento di Codice Java del metodo `uploadFile()`

Il metodo `updateFile()`, contenuto nel Listato 4.16, verifica, innanzitutto, che ci sia un file selezionato dall'utente. Successivamente, attraverso il metodo `putFile()`, il file viene caricato nello storage di Firebase.

Terminata la prima fase, viene recuperato il nome attribuito al file, la sua estensione e il percorso dello storage in cui esso viene memorizzato. Successivamente, tutte queste informazioni vengono salvate nel Realtime Database utilizzando l'oggetto `DatabaseReference`.

Per salvare le foto del profilo degli utenti il procedimento è simile, ma viene memorizzata solo la foto all'interno dello storage; tutte le altre informazioni non vengono mantenute.

Implementazione del messaging

Il messaging di Firebase è stato utilizzato per implementare le notifiche push all'interno dell'app CoFiT.

Per permettere a tali notifiche di giungere al destinatario è indispensabile che l'applicazione e la rete siano attive, eventualmente anche in background; occorre, inoltre, che l'utente abbia autorizzato l'applicazione a inviare le notifiche. Se l'account è disconnesso dalla rete o l'applicazione non è attiva, la notifica push non può giungere.

Per implementare le notifiche all'interno del progetto è necessario utilizzare un Service, estendere la classe `MyFirebaseMessagingService`(Listato 4.17) e implementare un metodo per l'invio della notifica (Listato 4.18).

Ricordiamo che per inviare una notifica è necessario conoscere il token del dispositivo del destinatario.

```
public class MyFirebaseMessagingService extends FirebaseMessagingService {

   String title,message;
   @Override
   public void onMessageReceived(@NonNull RemoteMessage remoteMessage) {
       super.onMessageReceived(remoteMessage);
       title = remoteMessage.getData().get("Title");
       message = remoteMessage.getData().get("Message");
       Intent intent = new Intent(this, MainActivity.class);
       intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
       PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

       String CHANNEL_ID="MESSAGE";
       String CHANNEL_NAME="MESSAGE";
       NotificationManagerCompat manager= NotificationManagerCompat.from(getApplicationContext());
       if(Build.VERSION.SDK_INT>= Build.VERSION_CODES.O){
           NotificationChannel channel=new NotificationChannel(CHANNEL_ID,CHANNEL_NAME,
           NotificationManager.IMPORTANCE_DEFAULT);
           manager.createNotificationChannel(channel);

       }
```

```
        Notification notification = new NotificationCompat.Builder(getApplicationContext(),CHANNEL_ID)
                .setSmallIcon(R.mipmap.ic_launcher_round)
                .setContentTitle(title)
                .setContentText(message)
                .setContentIntent(pendingIntent)
                .setAutoCancel(true)
                .build();
        manager.notify(getRandomNumber(),notification);
    }
```

Listato 4.17. Codice Java relativo alla classe `MyFirebaseMessagingService`

Nel Listato 4.17 viene aperto un canale di notifica con Firebase. Inoltre, tramite il metodo `onMessagereceived()`, si dice al sistema come comportarsi quando arriva una notifica push relativa all'app CoFiT. Nella fattispecie, quando arriva una notifica viene aperta la home page dell'applicazione.

```
public void sendNotifications(String usertoken, String title, String message) {
        Data data = new Data(title, message);
        NotificationSender sender = new NotificationSender(data, usertoken);
        apiService.sendNotifcation(sender).enqueue(new Callback<MyResponse>() {
            @Override
            public void onResponse(Call<MyResponse> call, Response<MyResponse> response) {
                if (response.code() == 200) {
                    if (response.body().success != 1) {
                        Toast.makeText(InserimentoTasseActivity.this, "Failed␣", Toast.LENGTH_LONG);
                    }
                }
            }
```

Listato 4.18. Codice Java relativo al metodo `sendNotification()`

Il metodo `sendNotification()` riceve in ingresso il token del destinatario, nonché il titolo e il messaggio della notifica. Successivamente, viene costruito un oggetto Data, che rappresenta la notifica, e viene inviato utilizzando il Service di Firebase.

Infine, richiamando il metodo `sendNotification`, possiamo inviare la notifica (Listato 4.19).

```
final DocumentReference documentReference = fStore.collection("Users").document(userID);
        documentReference.addSnapshotListener(this, new EventListener<DocumentSnapshot>() {
            @Override
            public void onEvent(@Nullable DocumentSnapshot documentSnapshot, @Nullable FirebaseFirestoreException e) {

                token = documentSnapshot.getString("Token").trim();
            }
        });

String title = f24 + "␣" + mese + "␣" + anno;

String message = title;

              sendNotifications(token, "Hai␣una␣nuova␣scadenza", message );
```

Listato 4.19. Codice Java necessario per l'invio di una notifica

Nel Listato 4.19 recuperiamo dapprima il token del destinatario. Successivamente, formiamo il messaggio che dovrà essere recapitato e, infine, chiamiamo il metodo `sendNotification()` per inviare la notifica.

4.2 Testing sull'emulatore dei risultati ottenuti

Dopo aver implementato il Front-end e il Back-end all'interno del progetto CoFiT, è il momento di testare e, soprattutto, vedere il risultato ottenuto.

Per la fase di testing sono stati utilizzati due emulatori Android, messi a disposizione da Android Studio. Un emulatore si occupa di riprodurre le attività di un cliente, l'altro quelle del commercialista.

Testing dell'autenticazione

Nel primo test è stata effettuata una registrazione di un nuovo cliente (Figura 4.15). Subito dopo aver cliccato su "Registrati", Firebase ha aggiunto un nuovo utente nella lista degli utenti autentificati (Figura 4.16).

Figura 4.15. Testing della registrazione

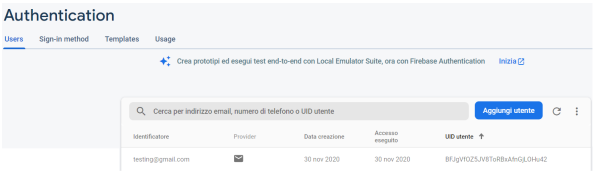

Figura 4.16. Screenshot relativo alla lista degli utenti autenticati di Firebase

I parametri richiesti per effettuare correttamente la registrazione sono un indirizzo e-mail valido e una password di almeno sei caratteri. Ovviamente, non è possibile utilizzare un indirizzo e-mail già registrato nell'app CoFiT.

Testing del salvataggio del profilo utente

Dopo aver registrato l'utente, proviamo ad inserire tutti i dati anagrafici (Figura 4.17).

Figura 4.17. Testing dell'inserimento dell'anagrafica

Successivamente, una volta cliccato su "Inserisci", il sistema ci mostrerà il nostro profilo utente modificato (Figura 4.18).

Figura 4.18. Testing della visualizzazione del profilo

Una volta che il cliente ha inserito tutti i suoi dati anagrafici, egli comparirà nella lista dei clienti del commercialista (Figura 4.19).

Analogamente, andiamo a controllare su Firebase se la struttura con cui esso ha memorizzato i dati è corretta (Figura 4.20).

Figura 4.19. Lista dei clienti del commercialista

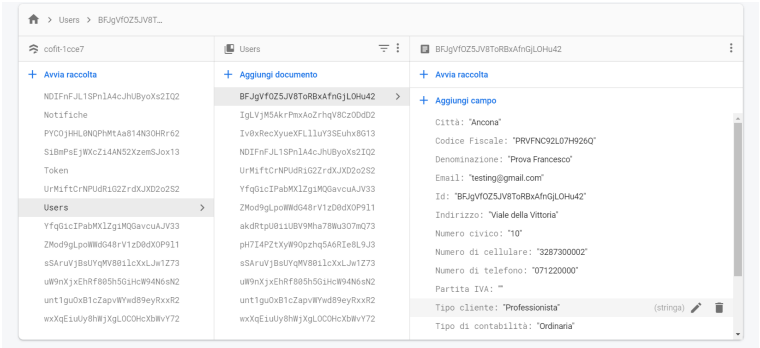

Figura 4.20. Struttura dei dati anagrafici salvati da Firebase

Testing dell'inserimento di un F24

Il Testing dell'inserimento di un F24 è necessario per verificare il funzionamento del Cloud di Firebase e dell'invio e ricezione delle notifiche.

Utilizzeremo i due emulatori sopracitati (Figura 4.21), ovvero quello del cliente (a sinistra) e quello del commercialista (a destra) .

Come si evince dalla Figura 4.21, il cliente non ha nessuna tassa e il commercialista ne sta inserendo una nuova. Non appena il commercialista clicca su "Inserisci", il cliente avrà un nuovo F24 (Figura 4.22) e gli verrà recapitata una notifica (Figura 4.23). Analogamente, il commercialista può visualizzare la lista degli F24 del cliente.

Sulla notifica ricevuta dagli emulatori non viene visualizzata l'icona dell'app CoFiT; tuttavia il testing effettuato sui dispositivi reali non ha evidenziato questo problema.

Controlliamo se Firebase ha salvato correttamente la tassa e verifichiamone la struttura (Figura 4.24).

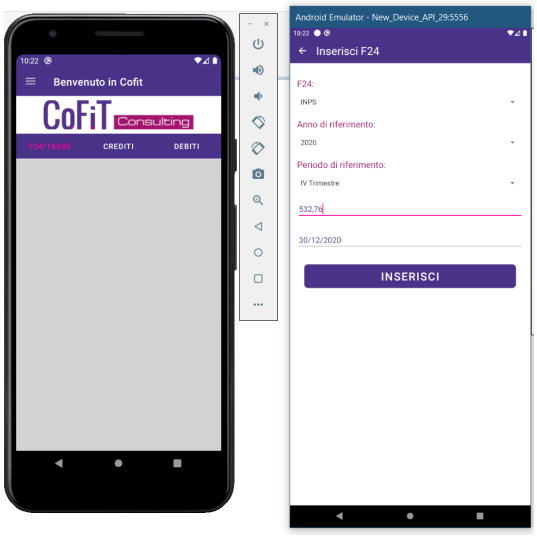

Figura 4.21. Testing relativo all'inserimento di un F24

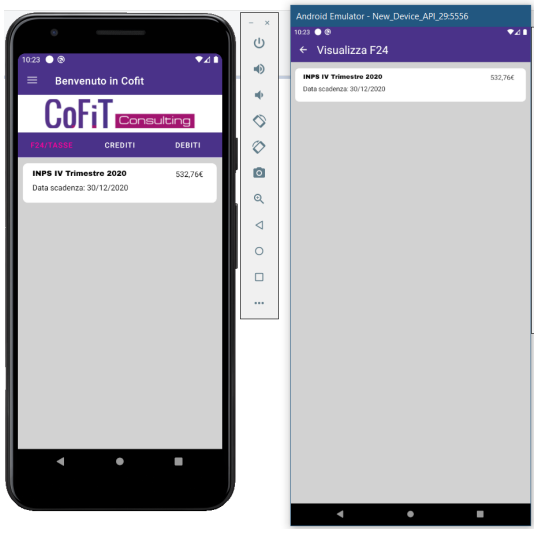

Figura 4.22. Testing relativo alla visualizzazione di un F24

Testing dell'invio dei file

Il testing relativo all'invio dei file permette di verificare il funzionamento dello storage, del Realtime Database nonché l'invio e la ricezione delle notifiche.

Ancora una volta useremo i due emulatori utilizzati in precedenza (Figura 4.25).

La Figura 4.25 mostra i due dispositivi intenti ad inviare un file. Non appena entrambi cliccano su "Inserisci", la lista dei file condivisi mostrerà i due file appena

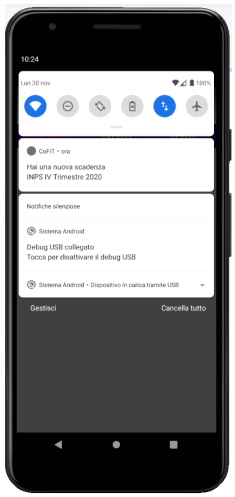

Figura 4.23. Testing relativo all'invio e alla ricezione di una notifica causata dall'inserimento di un F24

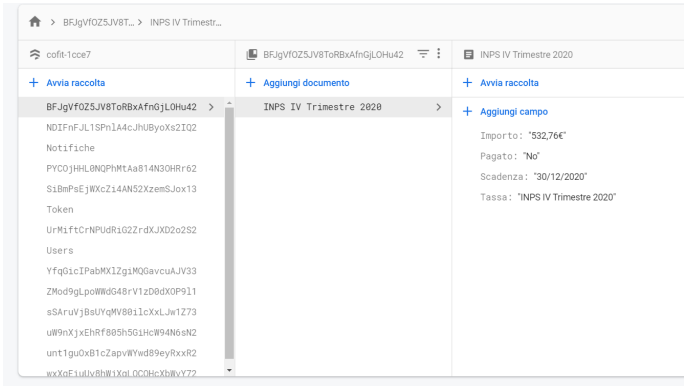

Figura 4.24. Struttura di un F24 salvato su Firebase

caricati (Figura 4.26) e ai due dispositivi verrà inviata una nuova notifica (Figura 4.27).

Dalla schermata della visualizzazione dei file condivisi, possiamo procedere al download del file inserito dal commercialista. Il testing non ha evidenziato alcun problema; il servizio Android per il download dei file ha funzionato correttamente (Figura 4.28).

L'ultima verifica riguarda ancora Firebase; verifichiamo, infatti, che sia stato salvato tutto correttamente. Controlliamo, quindi, prima il Realtime Database (Figura 4.29) e, successivamente, lo storage (Figura 4.30).

Come si evince dalle Figure 4.29 e 4.30, Firebase ha salvato correttamente tutte le informazioni del file e ha memorizzato la stessa immagine caricata dai due emulatori.

In conclusione, possiamo affermare che i diversi test effettuati non hanno evidenziato problemi. Gli stessi risultati sono stati ottenuti utilizzando dispositivi

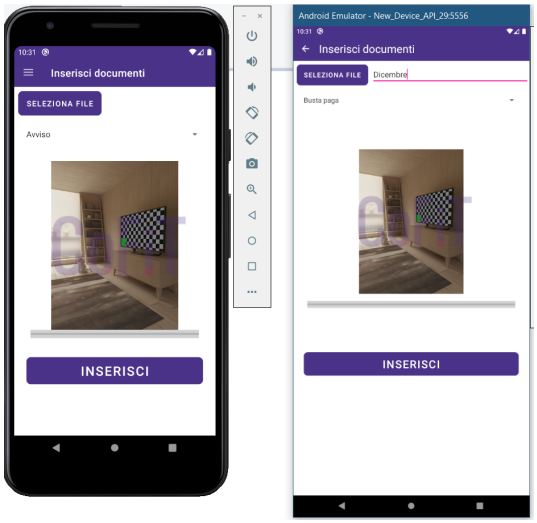

Figura 4.25. Testing dell'invio dei file

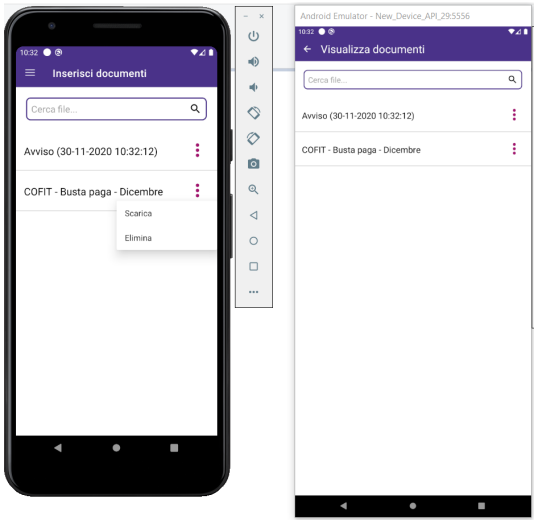

Figura 4.26. Testing della visualizzazione dei file condivisi

reali.

4.3 Manuale utente

In questa sezione verrà illustrato la documentazione necessaria per utilizzare l'app CoFiT. Verrà implementato solo il manuale utente per l'utilizzo da parte del cliente.

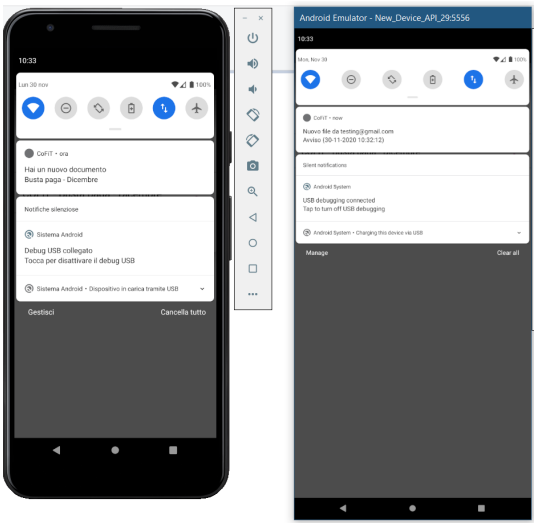

Figura 4.27. Testing relativo all'invio e alla ricezione della notifica causata dall'inserimento di un file

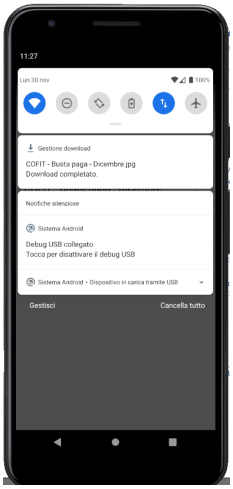

Figura 4.28. Testing del download di un file

Figura 4.29. Informazioni salvate sul Realtime Database

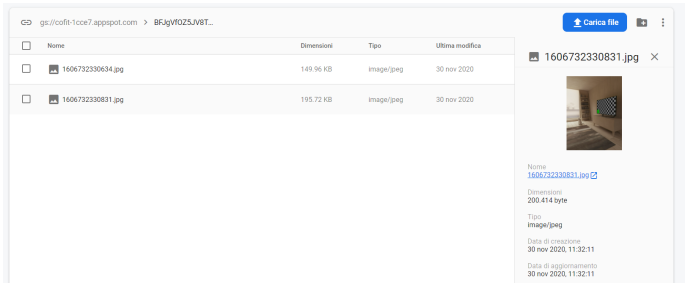

Figura 4.30. Immagini caricate nello storage di Firebase

Installazione dell'app CoFiT

Una volta ricevuto dal commercialista il file `.apk` dell'app CoFiT, possiamo procedere alla sua installazione (Figura 4.31).

Figura 4.31. Fase 1 dell'installazione dell'app CoFiT

Successivamente, è necessario premere su Installa e attendere il suo completamento (Figura 4.32).

Figura 4.32. Fase 2 dell'installazione dell'app CoFiT

Terminata l'installazione, clicchiamo su Apri (Figura 4.33).

Registrazione

Procediamo con la registrazione del nuovo utente (Figura 4.34). In particolare, sarà necessario effettuare le seguente attività:

- inserire l'indirizzo e-mail (l'email deve rispettare il formato corretto);
- inserire la password (la password deve contenere almeno sei caratteri);

Figura 4.33. Fase 3 dell'installazione dell'app CoFiT

- inserire nuovamente la password;
- Premere su "Registrati".

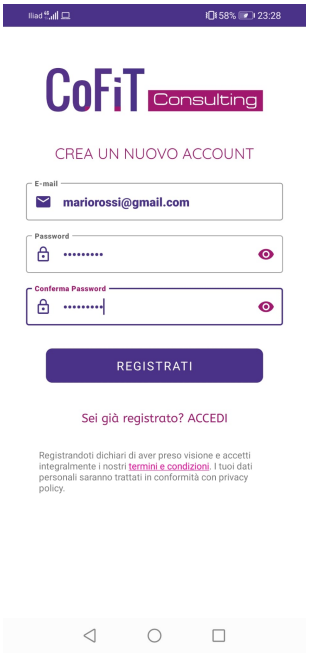

Figura 4.34. Registrazione all'app CoFiT

Login

Se l'utente è già registrato è possibile effettuare direttamente il login. A tal fine vengono effettuate le seguenti attività:

- inserire l'indirizzo e-mail con cui abbiamo creato l'account;
- inserire la password associata all'account;
- premere su "Login".

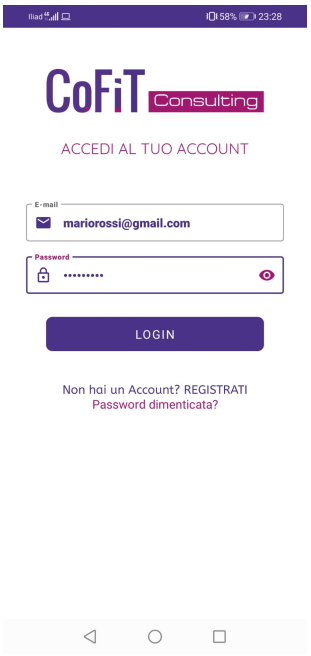

Figura 4.35. Login all'app CoFiT

Inserimento dei dati anagrafici

Per completare la registrazione si devono inserire tutti i dati anagrafici (Figura 4.35). Le regole per l'inserimento dei dati anagrafici sono le seguenti:

- selezionare il tipo di cliente;
- se abbiamo selezionato "Azienda" o "Società", inserire nel campo successivo la denominazione della vostra azienda. Se abbiamo selezionato "Professionista" inserire il cognome e il nome del professionista.
- Inserire almeno un nostro numero di telefono.
- inserire il Codice Fiscale o la Partita IVA;
- inserire città, indirizzo e numero civico;
- selezionare il tipo di contabilità utilizzata;
- premere su "Inserisci".

Descrizione della home page

La home page si presenta con tre tabelle, ovvero quella degli F24, dei crediti e dei debiti. Nella prima tabella (Figura 4.36) si possono visualizzare gli F24. Questi saranno caricati dallo studio CoFiT. Non sono permesse azioni di modifica o eliminazione. Nella seconda tabella è possibile visualizzare i crediti aggiunti (Figura 4.37). Nella terza tabella è possibile visualizzare i debiti (4.38). Cliccando su un elemento della lista dei crediti o dei debiti è possibile modificare o eliminare l'elemento.

Figura 4.36. Manuale per l'inserimento dell'anagrafica

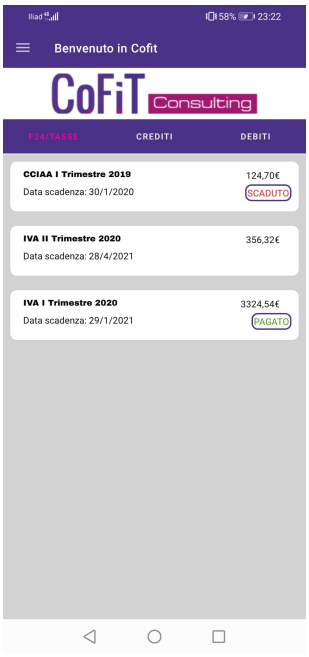

Figura 4.37. Home page dell'app CoFiT

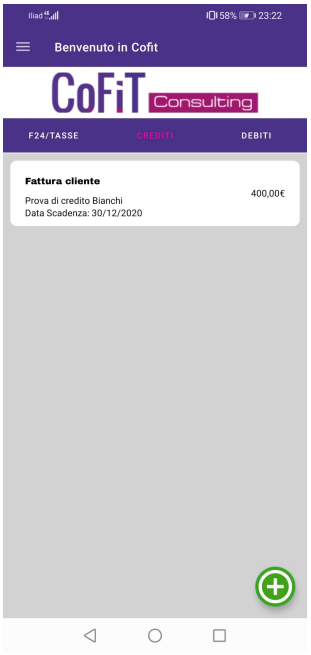

Figura 4.38. Tabella dei crediti

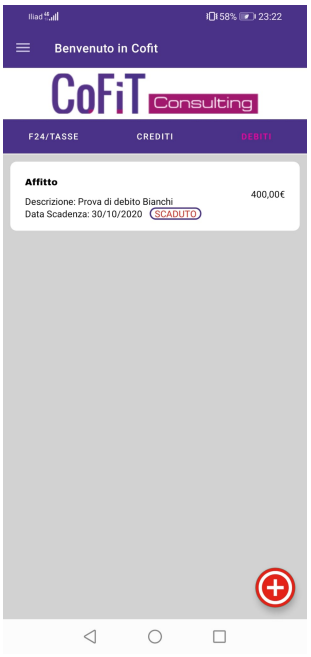

Figura 4.39. Tabella dei debiti

Nella parte superiore sinistra della Figura 4.36 c'è l'icona del menù. Premere l'icona per aprire il menù a comparsa (Figura 4.39). Da questo si può accedere a tutte le funzionalità dell'app CoFiT.

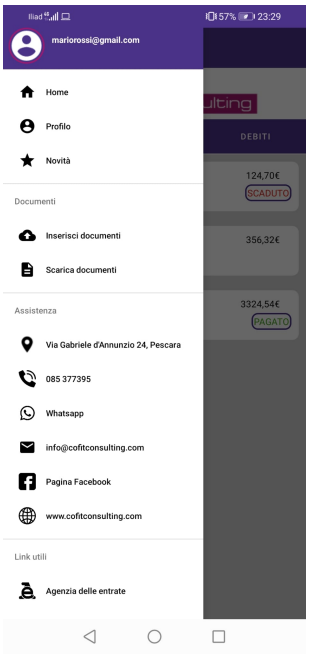

Figura 4.40. Menù a comparsa della home page

Inserimento di un credito o di un debito

Facendo riferimento alla Figura 4.36, per inserire un nuovo credito è necessario effettuare le seguenti attività:

- passare alla tabella dei crediti;
- premere il pulsente verde con il simbolo "+" nella parte inferiore destra dello schermo;
- compilare tutti i dati del nuovo credito richiesti (Figura 4.40);
- premere su "Inserisci".

Analogamente, per inserire un nuovo debito è necessario effettuare le seguenti attività:

- passare alla tabella dei debiti.
- premere il pulsante rosso con il simbolo "+" nella parte inferiore destra dello schermo;
- compilare tutti i dati richiesti del nuovo debito.
- premere su "Inserisci".

Figura 4.41. Inserimento di un credito

Visualizzare il profilo

Facendo riferimento alla Figura 4.39, per visualizzare il profilo, è necessario premere su "Profilo" sul menù di navigazione. Avremo la schermata presente nella Figura 4.42.

Premendo sull'icona dell'edit ritorniamo nella schermata presente in Figura 4.35 e possiamo modificare i dati anagrafici.

Inserire un file

Facendo riferimento alla Figura 4.39, per inserire un file, è necessario premere su "Inserisci documento" sul menù di navigazione. Avremo la schermata presente nella Figura 4.43. Per inserire un nuovo file è necessario effettuare le seguenti attività:

- premere su "Seleziona file";
- se non è stato ancora fatto, accettare i permessi richiesti (Figura 4.44);
- selezionare la sorgente da cui prendere il file (Figura 4.45);
- scegliere un file dalla memoria o scattare una foto;
- scegliere la categoria alla quale appartiene il documento da inviare (Figura 4.46);
- premere su "Inserisci".

Scarica un file

Facendo riferimento alla Figura 4.39, per scaricare un file è necessario premere su "Scarica documento" sul menù di navigazione. A seguito di ciò avremo la schermata presente nella Figura 4.47.

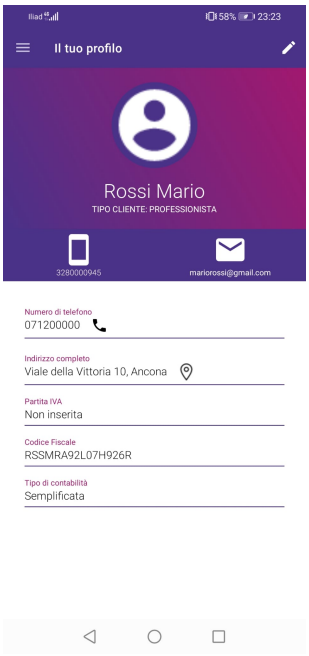

Figura 4.42. Visualizzazione del profilo

Figura 4.43. Inserimento di un file

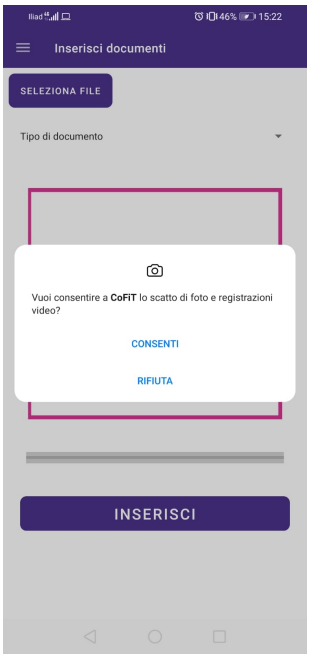

Figura 4.44. Fase 2: Accettare i permessi richiesti per inserire un file

Figura 4.45. Fase 3: Selezionare la sorgente

Figura 4.46. Fase 4: Caricare il file

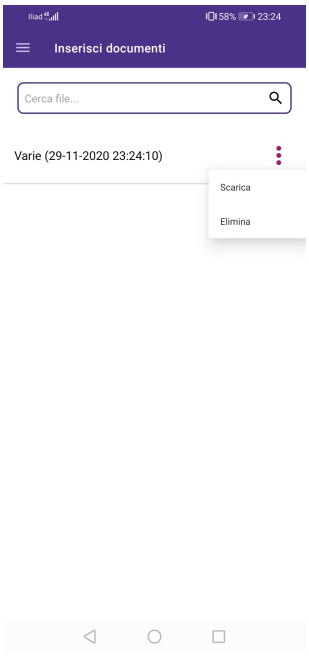

Figura 4.47. Guida per scaricare un file

Per scaricare un file è necessario effettuare le seguenti attività:

- aprire il l'icona del menù alla destra del nome del file da scaricare;
- premere "Scarica";
- attendere l'avvio del download;
- attendere che il sistema termini il download del file.

Il file verrà salvato nella directory dell'applicazione CoFiT, accessibile da: `gestione file/Memoria interna/Android/data/com.cofitconsulting.cofit/files/Download`

5

Discussioni in merito al lavoro svolto

In questo capitolo si parlerà delle conoscenze acquisite durante questo percorso sia per quanto riguarda Android, sia per tutte le fasi di progettazione e implementazione necessarie, indipendenti dal tipo di software che si intende sviluppare. Inoltre, verrà esaminata la SWOT Analysis dell'app CoFit.

5.1 Lezioni apprese

Al termine della fase di sviluppo possiamo tirare le somme sia per quanto riguarda il lavoro svolto e sia per ciò che abbiamo appreso; alla fine, ciò che resta di questo lavoro è un'app, che si spera possa essere d'aiuto per gli utenti che la utilizzeranno, e, soprattutto, ciò che abbiamo imparato, il quale è utile per lo sviluppo di tante altre applicazioni e per ampliare la nostra conoscenza, il che, nel settore ICT, non guasta mai.

Lo studio della storia di Android, dalla sua creazione ed evoluzione, non è stato particolarmente utile ai fini dello sviluppo dell'applicazione; tuttavia è stato molto interessante. Analizzare e leggere articoli sulle release del sistema operativo Android, associando tutti i servizi offerti da Android alle diverse versioni è stato "divertente", soprattutto perché lo smartphone, ormai, fa parte della vita di tutti i giorni e conoscere come e cosa lo ha reso così importante è il minimo che potessimo fare. In particolare, abbiamo appreso cos'è un file `.apk` e da cosa è composto.

Passando alla progettazione si è potuto osservare l'importanza notevole di questa fase, in quanto permette di organizzare tutto il lavoro, nonché di conoscere in anticipo tutti gli strumenti da utilizzare, così da non farsi trovare impreparati durante la fase di implementazione. Particolarmente appagante è stata la realizzazione delle interfacce utente durante l'implementazione effettuata partendo dai Mockup disegnati.

Durante la fase di implementazione è stato possibile apprendere l'utilizzo di Android Studio e di approfondire il linguaggio Java. In particolare, è stato possibile capire e approfondire tutto ciò che costituisce un'applicazione Android, di cui prima si conosceva la funzione solo a livello teorico.

Oltre ad Android Studio, abbiamo avuto modo di scoprire Firebase, con i tanti servizi messi a disposizione.

5.2 SWOT Analysis

La SWOT Analysis è uno strumento di pianificazione strategica usato per valutare i punti di Forza (Strenghts), di Debolezza (Weaknesses), le Opportunità (Opportunities) e le Minacce (Threats) di un progetto, o in ogni altra situazione in cui un'organizzazione o un individuo debbano prendere una decisione per il raggiungimento di un obiettivo.

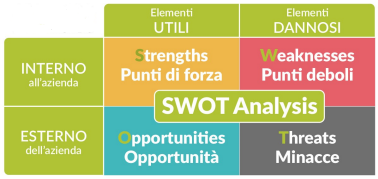

Figura 5.1. I punti chiave della SWOT Analysis

Ora vediamo nel dettaglio i quattro punti che compongono la SWOT Analysis; essi sono i seguenti:

- *Punti di Forza (Strengths)*: sono tutti quei punti positivi che devono essere valorizzati, divulgati ed enfatizzati. Questi rappresenteranno gli elementi positivi che porteranno il progetto al successo.
- *Punti deboli o Debolezze (Weaknesses)*: sono tutti gli elementi di debolezza o negativi che rischiano di far fallire il progetto; sulle debolezze si deve lavorare per annullarle o, se questo non è possibile, per limitare al massimo il loro effetto negativo.
- *Opportunità (Opportunities)*: sono tutti i coinvolgimenti esterni al progetto, nuove occasioni o condivisioni che si possono sviluppare attorno ad esso e che possono rappresentare ulteriori momenti di crescita e di successo. Prevedere in anticipo le opportunità consentirà di poterle sfruttare pienamente nei momenti in cui si presenteranno e, nel contempo, di non farsi sorprendere dagli eventi.
- *Minacce (Threats)*: sono tutti gli elementi dannosi per il progetto, i rischi o gli eventi che potrebbero danneggiarlo o ostacolarlo. È fondamentale individuarli accuratamente per poter elaborare strategie idonee alla loro neutralizzazione o trasformazione in ulteriori opportunità di successo.

La SWOT Analysis applicata al mercato delle app può servire ad un'analisi competitiva o benchmarking[10] con quelle dei potenziali concorrenti. Per utilizzarla in modo efficace è sufficiente porsi delle domande per ciascuno dei quattro componenti che la caratterizzano. Alcune di queste domande sono le seguenti:

- *Punti di forza*:
  - Quali sono i vantaggi della mia app?

[10] Il benchmarking è una metodologia basata sul confronto sistematico che permette alle aziende che lo applicano di confrontarsi con le migliori e, soprattutto, di apprendere da queste per migliorare.

  - Cosa offre che altri non hanno?
  - Quali altri punti di forza ha?
- *Punti di debolezza*:
  - Cosa è possibile migliorare?
  - Cosa andrebbe evitato?
  - Dispongo di un piano di marketing?
  - Su quali piattaforme sono assente?
  - La mia app ha problemi di scalabilità?
  - Quali altre debolezze esistono?
- *Opportunità*:
  - Quali e quante opportunità esistono? Come trarne vantaggio?
  - Quali sono i trend di mercato da cui beneficiare?
- *Minacce*:
  - Quali sono gli ostacoli possibili?
  - Cosa possono fare i concorrenti per creare difficoltà?
  - Cosa succede se cambiano le tecnologie?

Dopo aver introdotto la SWOT Analysis, e dopo aver dato una definizione dei punti che la caratterizzano, applichiamo nel seguito tecnica all'app CoFiT.

5.2.1 Punti di forza

Per trovare i punti di forza cerchiamo di rispondere alle domande poste nella sezione precedente.

Uno dei vantaggi dell'app CoFiT è, sicuramente, quello di avere a disposizione, ovvero a portata di un click, lo scadenzario di tutti gli F24.

In una delle interviste effettuate durante la fase dell'analisi dei requisiti, parlando con il responsabile dello studio CoFiT, quest'ultimo ci raccontava delle numerose chiamate ricevute, nei giorni prossimi alle scadenze, in cui gli veniva richiesta la data di scadenza di ciascun F24. Da ciò, possiamo dedurre che l'avere lo scadenzario sempre a portata di mano è un ottimo punto di forza.

Il problema è che, come è stato possibile osservare dal primo capitolo in cui si analizzavano altri software gestionali, anche questi ultimi godono di questa funzionalità. Perciò, se volessimo trovare una differenza con gli altri software gestionali, potremmo pensare che questi ultimi vengono utilizzati da chi ha un'impresa. Nel caso dell'app CoFiT, essa è utilizzata anche dai privati che non hanno alcuna necessità di acquistare un software gestionale, complesso, che debba gestire la contabilità.

Un ulteriore punto di forza è la condivisione di file, tramite la quale è possibile scattare una foto e inviarla. In questo modo un utente, anche se non ha il file digitale, può fotografare il cartaceo e inviarlo tranquillamente.

5.2.2 Punti di debolezza

Il maggiore punto di debolezza dell'app CoFiT è, senza dubbio, la mancanza di una versione per iOS. Sebbene il numero di dispositivi con Android siano di più rispetto a quelli con iOS, quest'ultimo rappresenta, comunque, una buona fetta di mercato.

Una volta testata l'app Android, e verificata la sua reale utilità, si potrebbe risolvere questo problema implementando una versione per iOS.

Inoltre, l'app CoFiT si appoggia su Firebase, perciò, nonostante quest'ultimo sia una piattaforma che ben si sposa con le applicazioni Android, si potrebbe progettare e implementare un proprio DBMS, che non abbia costi aggiuntivi. Firebase propone diversi piani tariffari (tra cui uno gratuito), i quali si differenziano per il numero di letture e scritture che si effettuano o per lo spazio di memoria utilizzato. Il piano gratuito potrebbe bastare per un progetto di piccole dimensioni, tuttavia se l'app CoFiT dovesse crescere nei servizi o negli utenti registrati questo piano potrebbe non bastare. In questo caso, ci sono due soluzioni, ovvero l'implementazione di un proprio DBMS o l'upgrade ad un piano enterprise di Firebase.

Infine, l'app CoFiT potrebbe aumentare i servizi offerti al commercialista o al cliente. In tal caso, si potrebbe pensare di implementare un servizio per gestire gli appuntamenti dello studio.

5.2.3 Opportunità

Per quanto riguarda le opportunità offerte dall'app CoFiT, se questa si dovesse affermare e venisse utilizzata da un gran numero di clienti, si potrebbe offrire la stessa, con le dovute modifiche, ad altri studi commercialistici affiliati allo studio Co.Fi.T. A quel punto si potrebbero sviluppare nuove app, le quali avrebbero la stessa ossatura dell'app CoFiT.

5.2.4 Minacce

Le minacce dell'app CoFiT sono, ovviamente, relative al gran numero di software gestionali già presenti sul mercato, i quali offrono più servizi rispetto alla neonata CoFiT; perciò, al momento, è possibile solo dare solidità ai servizi già offerti dall'applicazione, per poi pensare di espandere quest'ultima.

Inoltre, ricordiamo che il linguaggio di programmazione Kotlin è, al momento, il preferito di Android rispetto a Java. In futuro Java potrebbe essere completamente sostituito da Kotlin, perciò potremo essere costretti a "tradurre" l'intera app nel linguaggio Kotlin.

6

Conclusioni

L'obiettivo dell'elaborato di tesi è stato lo sviluppo di un'applicazione a supporto dello studio professionale Co.Fi.T. Tale applicazione doveva fornire una serie di servizi utili, che permettessero al commercialista di alleggerire il proprio carico di lavoro ma, allo stesso tempo, di assistere efficacemente il cliente.

L'applicazione è stata sviluppata per il sistema operativo Android, essendo, quest'ultimo, gratuito, oltre che semplice per lo sviluppo, grazie alla sua natura open source e all'ampia documentazione disponibile sulle API. I servizi disponibili in rete si sono resi fondamentali per la realizzazione delle funzionalità di base dell'applicazione.

In particolare, nel primo capitolo della tesi abbiamo esaminato le caratteristiche di diversi software gestionali. Tali caratteristiche hanno facilitato l'analisi dei requisiti funzionali e ci hanno consentito di svolgere al meglio la SWOT Analysis.

Successivamente, si è raccontato della storia di Android e di tutti i suoi componenti principali. Questo ha permesso ai lettori di prendere confidenza con alcuni termini tecnici e di conoscere gli strumenti utilizzati nelle fasi successive del progetto di tesi.

Dopo questa prima parte più teorica, è cominciato il lavoro vero e proprio della tesi. In particolare, si è potuto vedere quale può essere una delle tecniche di progettazione di un software. Nella fattispecie, si è scelto di analizzare al meglio la struttura dei dati, poiché questi ultimi, dovendo sviluppare un'applicazione di gestione, svolgono un ruolo centrale. Successivamente, è stata disegnata, in modo grezzo, l'interfaccia utente attraverso i wireframe e i mockup. Ciò ha permesso di studiare in anticipo come implementare i layout scelti, riducendo di molto i tempi di sviluppo dell'applicazione.

La parte cruciale del progetto di tesi è stata l'implementazione dell'app, la quale è stata alleggerita di molto dalla fase di progettazione e dalla presenza in rete di tutorial e guide per sviluppatori Android.

Dopo la fase di implementazione, l'applicazione è stata testata sugli emulatori e sui dispositivi in nostro possesso. Successivamente, l'applicazione è stata distribuita ad un numero ristretto di clienti dello studio Co.Fi.T. per poter ricevere delle recensioni da chi, realmente, la utilizzerà. Le recensioni sono state positive, l'applicazione è risultata molto utile, perciò l'obiettivo iniziale è stato pienamente soddisfatto.

Nel breve periodo l'applicazione sarà distribuita a tutti i clienti dello studio. Questa distribuzione avverrà per gradi, consegnando l'app prima ad un numero ristretto di clienti, per poi raggiungere la quasi totalità di essi. Sono, infatti, da escludere coloro che non posseggono un dispositivo Android o coloro che possiedono uno smartphone con una versione di Android non in grado di gestire le API utilizzate nell'applicazione. La distribuzione avverrà in questo modo per consentire al commercialista di controllare i nuovi utenti, per sorvegliare il flusso di dati in lettura e scrittura di Firebase e per verificare che l'applicazione sia ben supportata dalla maggior parte dei dispositivi Android. Una volta verificate tutte queste condizioni, potrà essere pubblicata sul Play Store di Android.

Inoltre, saranno limati alcuni dettagli dei servizi già implementati, e se ne penseranno altri per arricchire l'esperienza utente.

Nel lungo periodo si pensa di progettare e implementare una versione desktop dell'applicazione. Questa versione sarà utilizzata dal solo studio Co.Fi.T. per facilitare l'attività di gestione dei clienti. Inoltre, anche se l'utilizzo della piattaforma Firebase dovesse garantire affidabilità, si vorrebbe implementare un proprio database.

Infine, l'obiettivo finale è arrivare ad altri studi professionali, offrendo loro un software gestionale completo e personalizzato con il nome e i colori del proprio studio, ovvero ciò che è l'app CoFiT per lo studio Co.Fi.T.

Riferimenti bibliografici

[1] Capire e programmare le Activity. https://www.androidiani.com/applicazioni/sviluppo/capire-e-programmare-le-activity-869, 2009.

[2] Lavoriamo in background con i Service. https://www.html.it/pag/19502/lavoriamo-in-backgroud-con-i-service/, 2014.

[3] Introduction to Android Content Providers. https://devcfgc.com/introduction-to-android-content-providers-685ed2468935, 2015.

[4] Tutorial Com'è fatta l'architettura di Android? http://www.quickgo.it/tutorial-come-fatta-larchitettura-di-android/, 2015.

[5] Fragment in Android. https://www.html.it/pag/49108/fragment-in-android/, 2016.

[6] Android Application/Package APK Structure Part 1. http://www.ryantzj.com/android-applicationpackage-apk-structure-part-1.html, 2017.

[7] Android Architecture Components. https://www.html.it/pag/68724/android-architecture-components/, 2017.

[8] Gli elementi e il funzionamento di base di un'applicazione. https://www.html.it/pag/48525/gli-elementi-e-il-funzionamento-di-base-di-unapplicazione/, 2019.

[9] Android Broadcast Receiver Tutorial – A beginner-friendly guide. https://data-flair.training/blogs/android-broadcast-receiver//, 2020.

[10] Android fundamentals 02.1:Activities and intents. https://developer.android.com/codelabs/android-training-create-an-activity?hl=nb_NO#0, 2020.

[11] Android: Origini, storia, Versioni e varianti del robottino verde - TUTTOANDROID. https://www.tuttoandroid.net/android/, 2020.

[12] Android Studio: tutorial per creare un'app. https://www.solotablet.it/app/sviluppo-app-identifica-i-concorrenti-potenziali-e-esercitati-in-una-analisi-swot, 2020.

[13] Il layout di un'app Android. https://www.html.it/pag/48738/il-layout-di-unapp-android/, 2020.

[14] Intent e messaggi. https://www.html.it/pag/19500/le-azioni-intent/, 2020.

Ringraziamenti

Nonostante non sia finito, o almeno si spera, il mio percorso universitario, è tempo di ringraziare chi mi ha aiutato e supportato (leggere sopportato), permettendomi di tagliare questo mio primo traguardo.

In primis, voglio (non è un dovere ma un enorme piacere) ringraziare il Prof. Domenico Ursino per aver avuto tanta pazienza e per aver combattuto la sindrome dello studente (e pare averla vinta) svolgendo un lavoro impeccabile. È stato davvero un piacere averlo incontrato nel mio percorso di studi. Per me è motivo d'orgoglio essere stato un suo studente e che questa tesi porti anche la sua firma. È, davvero, un professore e una persona eccezionale. Di nuovo, grazie di cuore!

Subito dopo, ma, ovviamente, al primo posto anche loro, voglio ringraziare mio padre e mia madre per avermi concesso e permesso di studiare lontano da casa, sostenendomi anche quando gli esiti non erano positivi. Nonostante questo periodo non ci permetta di stare vicini, anche in questa giornata di festa, per me è come avervi qui. Vi ringrazio di tutto, per il passato, il presente ed il futuro.

Ringrazio Martina, probabilmente è la persona che ha dovuto sopportare di più, soprattutto in questo ultimo periodo. Nonostante fosse parecchio indaffarata, è sempre stata qui (non che avesse parecchia scelta) mettendo i suoi impegni sempre al secondo posto. Probabilmente dovrò farmi perdonare in qualche modo; ci proverò, promesso!

Ringrazio Antonio, purtroppo hai lasciato questa città e, a causa di ciò, probabilmente, ci vedremo molto meno spesso. Non dimenticherò mai tutte le giornate all'università a studiare e i nostri post-studio. Ti ringrazio per essere stato la mia ombra (per fortuna non la mia guida spirituale) per tanto tempo.

Ringrazio Federica che mi ha aiutato e sostenuto durante questa prima tappa universitaria, soprattutto nel primo periodo. Grazie per avermi fatto scoprire la possibilità di studiare ad Ancona e grazie per esserci sempre stata!

Ringrazio Miriam, per aver sempre risposto a tutte le mie mille domande sull'università. Probabilmente, senza di te ci sarebbero stati molti più ostacoli; per questo, grazie! Un pezzettino di questa laurea è anche tuo!

Ringrazio Bocci che, al pari di Miriam, ha dovuto rispondere a tante domande sull'università. Mi fa sempre sorridere quando chiedo ad entrambi (te e Miriam) informazioni su determinate materie e le risposte sono discordanti e, di conseguenza, non ci capisco mai nulla.

Ringrazio Roberto! Poverino, ha dovuto vivere per quattro anni nella stessa casa con me. Grazie per avermi spronato molte volte a provare un esame. Non dimenticherò mai tutte le volte che tentavi (a volte inutilmente) di farmi "ripetere" la sera prima di un esame.

Infine, vorrei ringraziare tutte le altre persone che mi sono state vicino in questo periodo, anche a distanza. Grazie anche a tutti voi!