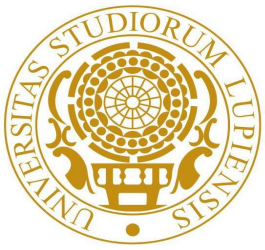

UNIVERSITA' DEL SALENTO

Corso di Laurea in Ingegneria dell'Informazione

Tesi di Laurea in Principi di Progettazione del Software

Migrazione di una architettura stand-alone verso una Cloud-based per dispositivi Android nel settore dell'e-commerce
(Migration from stand-alone to Cloud-based architecture for Android devices in the e-commerce field)

RELATORE:
Ch.mo Prof. Luca Mainetti

CORRELATORE:
Ch.mo Ing. Roberto Vergallo

STUDENTE:
Simone Russo
Matricola n°
20028586

Anno Accademico 2018 – 2019

Prima di procedere con la trattazione, vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questi anni di studio che hanno contribuito alla mia crescita personale e professionale. Il mio percorso è stato fortemente segnato dal periodo in cui ho redatto la tesi di laurea, ci siamo tutti ritrovati in una situazione surreale, prigionieri nelle nostre case. Convivere per due mesi a stretto contatto non è per nulla facile.
Ringrazio quindi mia madre per il supporto che mi fornisce giornalmente sopportando i miei cambi d'umore e i miei improvvisi scleri e per quelli che, purtroppo, potrebbe dover sopportare.
Ringrazio mio padre di aver insistito per farmi continuare gli studi quando ho avuto dei dubbi e per avermi permesso di dedicarmi completamente allo studio senza far cadere su di me ulteriori responsabilità.
Ringrazio mia sorella che ha dimostrato di sapermi dare supporto a suo modo e non esita ad appoggiarmi.
Ringrazio il prof. Mainetti e il prof. Vergallo che mi hanno guidato nella stesura di questa tesi di laurea, fornendo tutto l'aiuto possibile anche nella difficile situazione che abbiamo vissuto.
Ringrazio in fine tutti gli amici che c'erano da prima e che ho conosciuto durante questo magnifico percorso con cui condivido le piccole gioie della vita, la felicità per i successi gli uni degli altri, e la rabbia e la frustrazione degli insuccessi, che con il loro affetto sono meno difficili da affrontare.

Sommario

Capitolo 1 Introduzione

1.1 Introduzione

L'evoluzione tecnologica degli ultimi anni ha fatto si che il bisogno di possedere un dispositivo mobile Android o IOS, sia pari a quello di possedere un PC.
Con l'aumento dei lavori interattivi, digitali e da remoto, è ampiamente cresciuto l'utilizzo di applicazioni che danno la possibilità, a chi le utilizza, di diminuire i tempi di collegamento al servizio di cui si ha bisogno.
Proprio per questo motivo, con la stessa frequenza, è cresciuto il numero di applicazioni correlate tra PC e smartphone, per consentire un collegamento rapido e diretto in qualsiasi contesto ci si trovi ad operare.
Le informazioni vengono immagazzinate in un'unica memoria virtuale, ovvero il Cloud, valida per entrambe le piattaforme, così da poter usufruire delle informazioni necessarie da qualsiasi dispositivo. Questi processi vengono svolti autonomamente e favoriscono una fluidità nell'uso della stessa applicazione, ma in momenti e/o luoghi di interesse diversi, continuando a svolgere il lavoro iniziato, senza dover necessariamente attendere di trovarsi nella situazione inziale.
I vantaggi di questa coesistenza sono svariati ed oltre ad ottimizzare il lavoro in circostanze diverse, rendono inoltre più difficile la perdita di dati. Ad esempio, prendendo in considerazione smartphone o PC, entrambi possono presentare problemi temporanei e/o permanenti di qualsiasi tipo, proprio per l'uso costante e potrebbero aver bisogno di periodi d'assistenza più o meno lunghi, a seconda della gravità del problema che si presenta. Data quest'eventualità, la possibilità di interagire con l'applicazione e poter recuperare le informazioni da più piattaforme rende più efficace lo svolgimento e la continuità di un lavoro o il semplice utilizzo intrattenitivo della stessa applicazione, ma su dispositivi diversi.
La connessione immediata di un'app tra due dispositivi distinti è un fattore che può solo migliorare e velocizzare un lavoro, è un'importante risorsa, a prescindere dall'utilizzo del consumatore.
Questo lavoro di tesi si inserisce proprio in questo settore. L'obiettivo di questo lavoro di tesi infatti è migrare un'architettura stand-alone verso una Cloud-based per dispositivi mobili nel settore dell'e-commerce. In particolare, si è dovuto riproporre l'architettura del software stand-alone su una nuova piattaforma, ovvero Android, sfruttando anche la possibilità di utilizzare il Cloud, cercando di riutilizzare quanto più possibile il codice sorgente già a disposizione per quelle che sarebbero state le nuove esigenze.

1.2 Scenario

L'applicazione deve offrire a diversi utenti un servizio di e-commerce fruibile su più dispositivi contemporaneamente e tale che ogni utente possa avere la sua esperienza personalizzata.

Gli utenti sono identificati da quattro tipologie di attori: l'ospite, il cliente, l'amministratore e il gestore. Per ognuno di questi attori sono previste funzionalità differenti.
L'ospite può solo visualizzare il catalogo dei prodotti.
Il cliente, ovvero l'utente che ha effettuato una richiesta di registrazione, dopo essersi loggato può usufruire del carrello ed effettuare acquisti.
L'amministratore del sistema si occupa di gestire le registrazioni e gli ordini, è suo compito autorizzare i clienti che effettuano una richiesta di registrazione all'accesso al sistema e quindi a poter usufruire delle funzionalità dedicate ai clienti e modificare lo stato degli ordini ogni volta che uno di questi viene spedito, evaso o rifiutato.
Il gestore ha il compito di curare il catalogo, a lui spetta il compito di aggiungere prodotti o modificare i dettagli di quelli già presenti.

1.3 Obiettivi

Gli obiettivi principali di questo lavoro di tesi sono:

- effettuare una migrazione di un sistema stand-alone ad un sistema cloud-based per dispositivi mobili
- sviluppare le seguenti funzionalità: profilazione degli utenti; catalogo disponibile a chiunque visiti il servizio; la possibilità di effettuare acquisti; un sistema di gestione delle richieste di registrazione degli utenti e degli ordini per l'amministratore; un sistema di gestione dei prodotti per il gestore del sistema.
- definire una strategia di migrazione secondo la quale, partendo dal software esistente, si cerca di riciclare il più possibile il codice, mantenendo la stessa struttura ma utilizzando tecnologie diverse, quali Android e Firebase.
- studiare e comprendere al meglio la tecnologia Android nel suo insieme, come interagire con l'utente e come questa integra la Firebase sviluppata da Google.
- effettuare dei test funzionali che contribuiscono a verificare la corrispondenza dell'applicativo rispetto ai requisiti richiesti e il suo corretto funzionamento.

Capitolo 2 Strumenti utilizzati

Gli strumenti che sono stati utilizzati per lo sviluppo dell'applicazione sono: Java, Android, Android Studio, Firebase di Google.

2.1 Java

In informatica, Java [1] è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione, specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione

Java venne creato per soddisfare cinque obiettivi primari:

- essere "semplice, orientato agli oggetti e familiare";
- essere "robusto e sicuro";
- essere indipendente dalla piattaforma;
- contenere strumenti e librerie per il networking;
- essere progettato per eseguire codice da sorgenti remote in modo sicuro.

Gli sviluppatori si avvalgono del pacchetto JDK [2] (Java Development Kit) per sviluppare software e applicazioni con Java. Esso è l'insieme degli strumenti per sviluppare programmi da parte dei programmatori Java. È un prodotto della Oracle Corporation, e fin dall'introduzione di Java è sempre stato l'ambiente di sviluppo più utilizzato dai programmatori Java soprattutto per applicazioni desktop.

Nello sviluppo di questa applicazione ci si è serviti della versione Jdk SE 13.0.2.

2.2 Android

Android [3] è un sistema operativo per dispositivi mobili sviluppato da Google LLC e basato sul kernel Linux, segue il modello di sviluppo open source.

Le applicazioni (o app) sono la forma più generica per indicare i software applicativi installabili su Android. Per motivi di sicurezza informatica, le applicazioni sono fornite di un sistema di certificazione che verifica l'integrità del pacchetto da installare e la paternità rispetto a un distributore di software accreditato presso Google.

Il suo sviluppo prosegue attraverso l'Android Open Source Project, il quale è software libero con l'esclusione di diversi firmware non-liberi inclusi per i produttori di dispositivi e delle cosiddette "Google Apps", come ad esempio Google Play.

L'interfaccia utente si basa sul concetto di "direct manipulation", utilizzando sia ingressi mono-touch che multi-touch, come swiping, tapping e pinching per interagire con gli oggetti.

Un componente classico di questo sistema operativo è il launcher, ovvero l'applicazione di sistema che gestisce la Homescreen, le schermate secondarie, gli shortcut, l'app drawer, la dock bar, la status bar e il menu delle notifiche

2.3 Android Studio

Android studio [4] è un ambiente di sviluppo integrato adibito per la creazione di applicazioni Android (IDE, integrated development environment nel gergo anglosassone). L'ultima release rilasciata da Google è la 3.5.2, disponibile senza costi su licenza Apache 2.0 e basato su linguaggio Java. Si può scaricare su dispositivi Windows, Linux, Mac OS X. E' considerato l'IDE primario di Google per lo sviluppo nativo di applicazioni Android. L'IDE in questione contiene tutto il necessario per creare un emulare il sistema operativo Android. Ecco le principali peculiarità dell'applicativo:

- sistema di compilazione flessibile articolato su Gradle, per la building automation;
- facoltà di creare APK varianti e multipli;
- elementi per la cattura di prestazioni, compatibilità di versione, usabilità' e problematiche di vario genere;
- ampio supporto ai servizi Google e ad ulteriori tipologie di dispositivi;
- ambiente di layout con editing a tema abbastanza produttivo;
- supporto per Google Cloud Platform, con maggiore facilità nell'integrare Google Cloud Messaging e App Engine;
- software ProGuard e app signing.

Gli elementi inclusi nel download sono:

- software Android Studio;
- strumenti SDK Android;
- versione della piattaforma Android con la quale compilare l'app;
- versione dell'immagine del sistema Android per far avviare l'app nell'emulatore;

2.4 Firebase di Google

Firebase [5][6] è una piattaforma di sviluppo web e mobile. Si tratta di un database NoSQL dalle grandissime risorse, ad alta disponibilità ed integrabile in tempi rapidissimi in altri progetti software, offre numerosi servizi semplicemente con l'implementazione di poche righe di codice all'interno del proprio progetto. È possibile integrarlo in siti web, in app Android o iOS e in giochi.

Firebase consente agli sviluppatori di concentrarsi sulla creazione delle applicazioni Front-End, tralasciando la parte di gestione Back-End. Fornisce, infatti, un servizio di gestione di server, API e database.

La Firebase di Google offre anche un servizio di analisi per le applicazioni che la integrano, chiamato "Google Analytics". In questa sezione sono compresi i rapporti che contengono un'analisi dei dati di marketing, del comportamento dell'applicazione e di come gli utenti interagiscono con questa In particolare, ci si è serviti del servizio di Authentication e Realtime Database.

2.4.1 Authentication

L'autenticazione di Firebase [7][8] permette di creare in poche righe di codice un sistema di registrazione e login. Fornisce servizi di Back-End, quali SDK e librerie per l'autenticazione degli utenti. Supporta l'autenticazione attraverso password, numero di telefono e provider di identità federali popolari come Google, Facebook e Twitter.
Dopo aver eseguito correttamente l'accesso, è possibile accedere alle informazioni di base sul profilo dell'utente e controllare l'accesso dell'utente ai dati archiviati in altri prodotti Firebase. È inoltre possibile utilizzare il token di autenticazione fornito per verificare l'identità degli utenti nei propri servizi di back-end.

2.4.2 Realtime Database

Il Realtime Database [9][10] usufruisce della sincronizzazione dati ogni millisecondo e garantisce così assoluta velocità ed altissime performance. Firebase Realtime Database è un database NoSQL ospitato su Cloud. I dati vengono archiviati come JSON e sincronizzati in tempo reale su ogni client connesso. Il servizio fornisce agli sviluppatori un'API che consente di sincronizzare i dati dell'applicazione tra i client e archiviarli sul Cloud di Firebase, tutti i client condividono un'istanza di database in tempo reale e ricevono automaticamente gli aggiornamenti con i dati più recenti.
Ciò permette di utilizzare un data modeling semplice e flessibile. Il database è schemaless, ossia non richiede che si determini una struttura fissa nella fase iniziale. Persino la convalida dei dati non rappresenta un problema poiché il Realtime Database è dotato di regole per il linguaggio applicate dal server che consentono di convalidare la struttura dei dati di ogni scrittura nel database.
Il database in tempo reale fornisce un linguaggio di regole flessibile di tipo expression-based, chiamato Firebase Realtime Database Security Rules, per definire il modo in cui i dati devono essere strutturati e quando possono essere letti o scritti. Inoltre, gli sviluppatori possono definire chi ha accesso a quali dati e come possono accedervi.

Capitolo 3 Studio di fattibilità

La progettazione di un software passa attraverso vari stati. Si procede dapprima allo studio di fattibilità che studia la concreta possibilità di realizzare il software.
Trattandosi della migrazione di un software stand-alone verso una Cloud-based per dispositivi mobili Android, si analizzano le differenze dell'architettura delle due versioni del software.

3.1 Architettura AS-IS

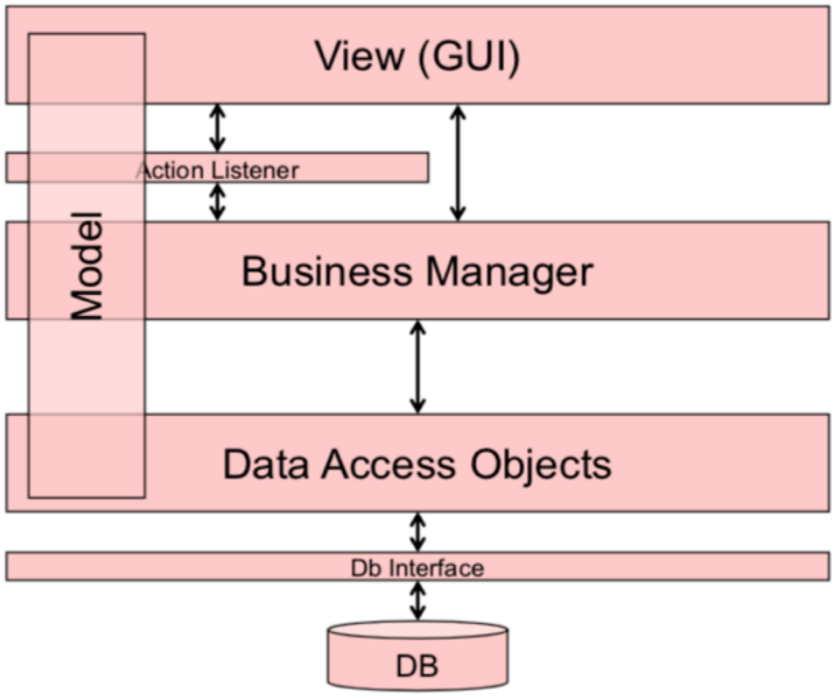

Immagine 1: Architettura AS-IS

Nello sviluppo di un software, si segue la struttura descritta nell'immagine 1: il database viene interrogato attraverso delle interfacce (Db Interface) e delle classi di DAO (Data Access Object) che si occupano di passare i dati contenuti nel database alle classi di Business Manager che li useranno per eseguire determinate operazioni.
La Business Manager si riferisce a tutta quella logica di elaborazione (sotto forma di codice sorgente) che rende operativa un'applicazione. La Business Manager non contiene dati, ma metodi che gestiscono le interazioni tra le entità business.
Il Model indica un modello per trattare un problema attraverso la rappresentazione tipica del Object-Oriented. Tipicamente un object model definisce un set di classi per rappresentare i le entità che vengono individuate e contengono gli attributi delle stesse.
Gli Action Listener si occupano di catturare le interazioni tra utente e interfaccia grafica.
La View o GUI (Graphical User Interface) è un tipo di interfaccia utente che consente l'interazione uomo-macchina in modo visuale utilizzando rappresentazioni grafiche. La GUI è implementata attraverso l'uso di librerie Swing.

3.2 Architettura TO-BE

Come si può vedere dall'immagine 2, anche l'architettura subirà importanti modifiche, di fatto la View sarà sostituita dalle Activity i cui layout saranno disegnati tramite l'XML. Non sarà più previsto il livello di Db Interface, il livello di DAO è sostituito dal Service e il database non sarà più locale ma sarà su un Cloud. Il livello di Service e di Business Manager comunicheranno direttamente con la Firebase. Il livello di Service avrà accesso al Realtime Database mentre il livello di Business Manager avrà accesso all'Authentication. A fare da tramite tra il livello di Business Manager e il livello di Service ci sarà il livello di Interface. Le classi di Model, Business Manager e di Action Listener rimarranno come nell'architettura del software di partenza.
Trovandosi su un Cloud la Firebase avrà un'architettura a sé stante che si divide in moduli. Quelli di interesse sono il Realtime e l'Authentication.
La connessione alla Firebase [11] avviene tramite un WebSocket. I WebSocket sono più veloci del protocollo http e non è necessario effettuare singole chiamate WebSocket, una connessione socket è sufficiente, infatti, tutti i dati si sincronizzano automaticamente alla velocità consentita dal proprio provider telefonico

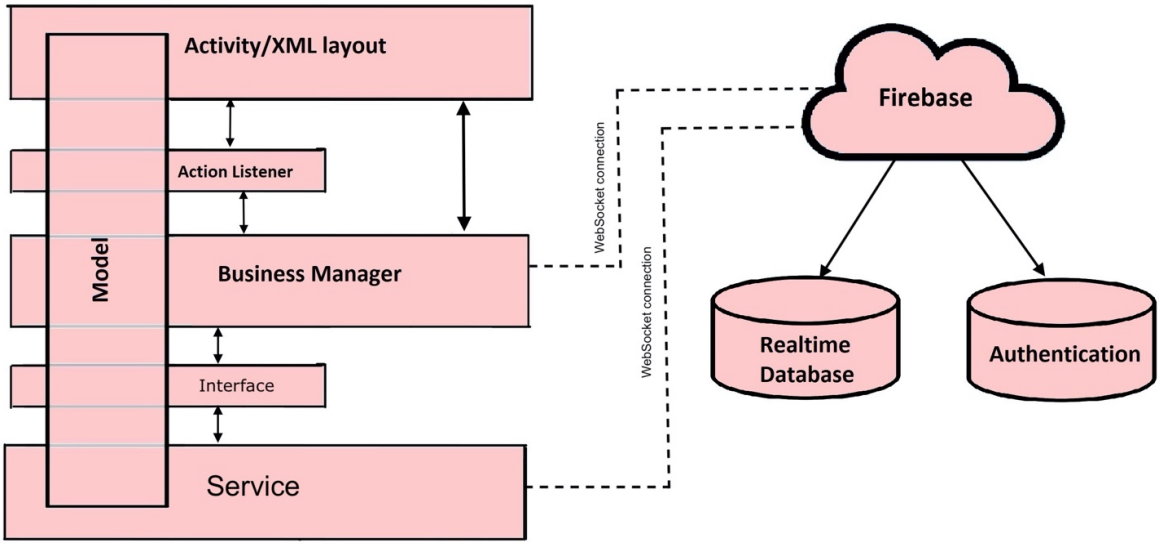

Immagine 2: Architettura TO-BE

3.3 Strategia di migrazione

Nella migrazione si dovranno compiere delle scelte. E' il caso del database che oltre, ad essere su un Cloud, avendo utilizzato il Realtime Database offerto dalla Firebase di Google, non è più relazionale e i dati sono immagazzinati come albero JSON e sincronizzati in tempo reale per ogni client connesso. Importanti cambiamenti si sono otterranno anche nel livello che si occupa di comunicare con il database, che verrà sostituito dal livello di Service. Infatti, la comunicazione con il Realtime Database non avverrà attraverso query SQL ma

si utilizzerà l'apposita libreria messa a disposizione da Google per la gestione delle interazioni con la Firebase. Le query verranno sostituite dal meccanismo del Data Snapshot ovvero un'interfaccia che rappresenta un'istantanea dei dati del Realtime Database di Firebase.

Per integrare le funzionalità offerte dalla Firebase occorre creare un progetto sulla console Firebase registrando il nome del pacchetto Android. Verrà generato un file di configurazione (google-services.json) che deve essere inserito nella directory del modulo (a livello di app) dell'app. Infine, bisogna abilitare i prodotti Firebase integrando i plug-in Google Service e gli SDK Firebase nei file Gradle.

La GUI non sarà più basata sulle librerie Swing ma verrà utilizzato un apposito linguaggio di markup, ovvero l'XML, [12] che è basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

Nel livello di Model si manterranno le entità esistenti ma verranno incluse nuove entità che nascono dal passaggio da un database relazionale ad uno non relazionale.

Il livello di Action Listener subirà modifiche dovute alla nuova interfaccia grafica.

Il livello di Business Manager invece manterrà buona parte della sua implementazione.

Capitolo 4 Progettazione

4.1 Use Case

4.1.1 Login

Attori:

- Guest

1. il sistema stampa il form di accesso e pulsanti “sign in” e “recupera password”;
2. l’utente inserisce le proprie credenziali;
3. Il cliente viene reindirizzato alla pagina di pertinenza in base al tipo di utente, ovvero al catalogo se è un cliente, alla pagina dell’amministratore se è un amministratore o alla pagina del gestore se è un gestore.

Estensioni:

1. L'utente inserisce le credenziali sbagliate;
   1.1. Il sistema notifica l’errore;
2. il cliente clicca il pulsante “sign in”;
   2.1. il sistema rimanda l'utente la pagina di registrazione;
3. il cliente clicca su “recupera password”;
   3.1. il sistema invia un’e-mail contenente un link per reimpostare la password;

4.1.2 Registrazione

Attori:

- Guest

1. Il sistema stampa il form di registrazione e il pulsante “registrati”;
2. l'utente compila il form e clicca su “registrati”;
3. il sistema notifica all'amministratore il nuovo cliente.

Estensioni:

1. l'utente dimentica di compilare uno o più campi o inserisci una mail già in uso;
   1.1. Il sistema blocca la registrazione e notifica errore all’utente.

4.1.3 Consultazione del catalogo

Attori:

- Guest;
- Cliente.

1. il sistema stampa i prodotti presenti nel catalogo e i menu “accedi”, “registrati”, “filtra per”;

2. il cliente sfoglia il catalogo.

Estensioni:

1. l'utente clicca su "registrati";
   1.1. utente viene mandato alla pagina di registrazione;
2. l'utente clicca su "filtra per";
   2.1. il sistema mostra le opzioni "elimina filtri" e "prezzo", contenente le opzioni "meno di 50 E.", "tra 50 e 100 E.", "oltre 100 E.";
   2.2. l'utente sceglie il filtro e il sistema filtra i prodotti;
   2.3. l'utente clicca su "elimina filtri" e il sistema mostra nuovamente tutti i prodotti;
3. l'utente clicca su login;
   3.1. l'utente viene mandato alla pagina di login.

Attori:

- Cliente

Requisiti:

- utente deve essere loggato.

4. Il cliente vuole aggiungere prodotti al carrello;
   4.1. il sistema presenta la nuova interfaccia contenente prodotti disponibili e i menu "log out", "filtra per", "carrello";
   4.2. il cliente clicca sul tasto "aggiungi al carrello";
   4.3. il sistema aggiungi il prodotto al carrello;
5. il cliente clicca su "log out";
   5.1. il sistema effettua il log out del cliente;
6. il cliente clicca su "carrello";
   6.1. il sistema manda il cliente alla pagina del carrello;
7. l'utente clicca su "filtra per";
   7.1. il sistema mostra le opzioni "elimina filtri" e "prezzo", contenente le opzioni "meno di 50 E.", "tra 50 e 100 E.", "oltre 100 E.";
   7.2. il cliente sceglie il filtro e il sistema filtra i prodotti;
   7.3. il cliente clicca su "elimina filtri" e il sistema mostra nuovamente tutti i prodotti.

4.1.4 Visualizza carrello

Attori:

- cliente

Requisiti:

- ha effettuato il login.

1. il sistema stampa la lista dei prodotti che sono stati aggiunti al carrello dopo l'ultimo ordine effettuato, la quantità richiesta (può essere modificata), il momentaneo costo

totale, i pulsanti "rimuovi dal carrello", "procedi all'acquisto", "svuota carrello" e "logout";

2. il cliente clicca su "procedi all'acquisto";
3. il sistema rimanda l'utente alla pagina per completare l'acquisto.

Estensioni

1. Il cliente clicca su "svuota carrello";
   1.1. i prodotti vengono eliminati dal carrello;
   1.2. il costo totale viene calcolato;
2. il cliente clicca su "log out";
   2.1. il sistema effettua il log out del cliente e lo rimanda al catalogo;
3. il cliente clicca su "rimuovi dal carrello";
   3.1. il sistema rimuove il prodotto dal carrello;
   3.2. il costo totale viene ricalcolato.

4.1.5 Acquista prodotti

Attori:

- cliente

Requisiti:

- ha effettuato l'accesso;
- ha almeno un prodotto nel carrello.

1. Il sistema stampa le generalità del cliente, le preferenze della consegna della merce e i pulsanti "conferma" e "annulla";
2. il cliente (dopo aver modificato i campi se necessario) conferma cliccando sul pulsante "conferma";
3. il sistema crea l'ordine e lo notifica l'amministratore;
4. il carrello viene svuotato;
5. l'utente viene rimandato al catalogo.

Estensione

1. il cliente clicca su "annulla";
   1.1. il sistema rimanda il cliente al carrello.

4.1.6 Gestione registrazione

Attori:

- Amministratore.

Requisiti:

- ha effettuato l'accesso;
- ha almeno un cliente da verificare.

1. il sistema stampa la lista dei clienti non ancora verificati e i pulsanti “accetta” e “rifiuta”;
2. l'amministratore clicca su “accetta” per verificare un cliente e consentirli così di effettuare l'accesso al portale o su “rifiuta” per eliminare l'utente

4.1.7 Gestione ordini

Attori:

- Amministratore.

Requisiti:

- ha effettuato l'accesso;
- ha almeno un ordine non spedito.

1. Il sistema stampa la lista degli ordini non spediti e i pulsanti “evadi”, “spedisci” e “rifiuta”;
2. l'amministratore decide se spedire, evadere o rifiutare l'ordine cliccando sul corrispondente pulsante;
3. il sistema aggiorna lo stato dell'ordine.

4.1.8 Gestione catalogo

Attori:

- Gestore.

Requisiti:

- ha effettuato l'accesso.

1. il sistema stampa tutti i prodotti nel catalogo e il pulsante “aggiungi prodotto”;
2. Il gestore clicca su “aggiungi prodotto”;
3. il sistema stampa il form per aggiungere un nuovo prodotto;
4. il gestore salva il nuovo prodotto il sistema aggiorna il catalogo;

Estensioni:

1. Il gestore vuole modificare un prodotto;
   1.1. clicca sul prodotto e il sistema stampa l’opzione “modifica”;
   1.2. il gestore clicca su “modifica”;
   1.3. il sistema stampa il form con le informazioni del prodotto;
   1.4. Il gestore modifica le informazioni e clicca sul pulsante “salva”;
   1.5. il sistema aggiorna il catalogo.

Di seguito il diagramma dei casi d'uso

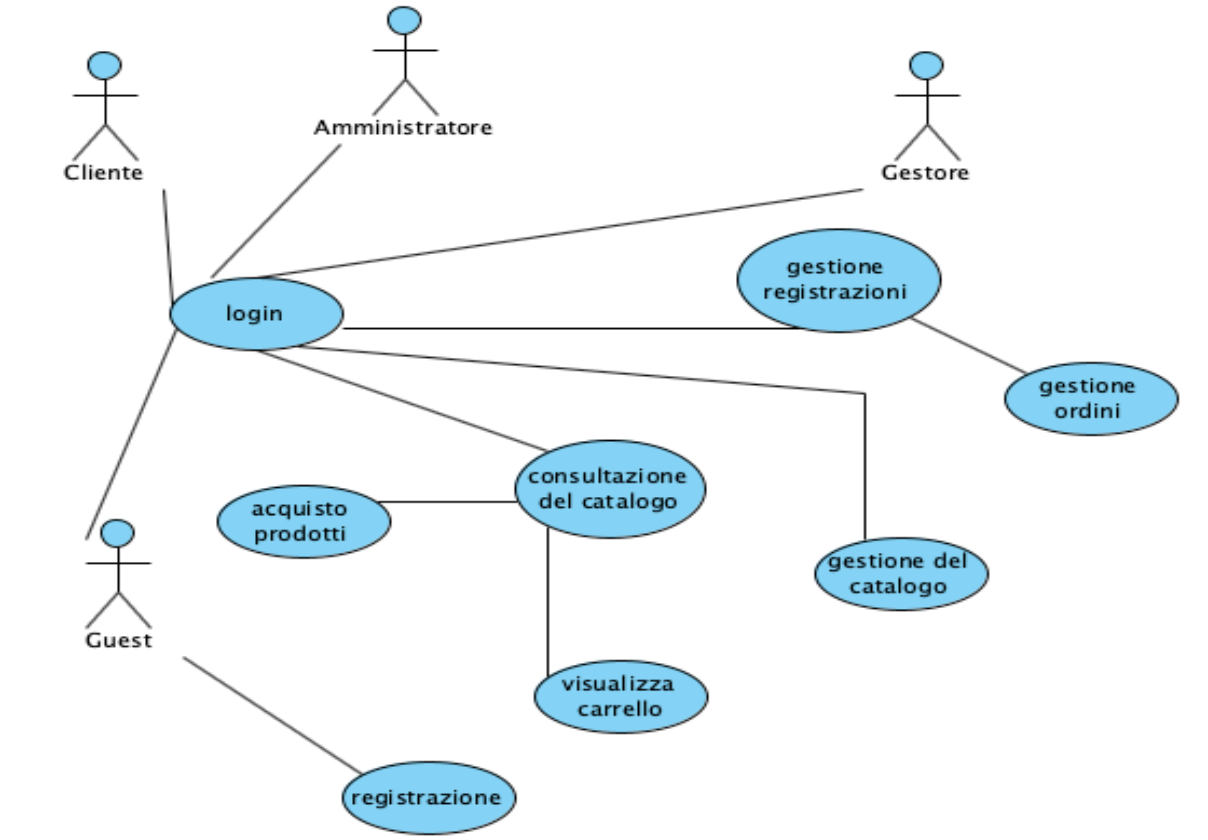

Immagine 3: Diagramma dei casi d'uso

4.2 Class diagram

In questo progetto si vuole rappresentare la struttura di un'attività di e-commerce. Sono presenti più zone a seconda della funzione che si svolge e dell'utente che la visita.
I tipi di utenti previsti sono: cliente, gestore e amministratore, per ognuno di essi si vuole tener traccia di: nome, cognome, e-mail, numero di telefono e indirizzo, in particolare per il cliente si vuole tener traccia anche del numero, intestatario e scadenza della carta di credito e dello stato di verifica di cui si occupa l'amministratore.
Per il cliente è presente un catalogo da sfogliare contenente tutti i prodotti, un carrello dove i clienti inseriscono i prodotti che desiderano acquistare e una "cassa" dove si richiede all'utente di inserire i dati per la spedizione per concludere l'acquisto dei prodotti.
Per il gestore è presente una sezione dove poter gestire il catalogo con la possibilità di aggiungere e modificare prodotti.
Per l'amministratore sono previste due pagine. La prima pagina contiene tutti gli utenti che hanno effettuato richiesta di registrazione e l'amministratore può accettare o rifiutare tale richiesta. La seconda pagina contiene tutti gli ordini che i clienti hanno effettuato e l'amministratore può evadere, spedire o rifiutare gli ordini.
Altre entità di cui si vuole tener traccia sono prodotto, ordine, prodotto presente nel carrello e prodotto presente nell'ordine.
Del prodotto si tiene traccia di: nome, prezzo, descrizione, disponibilità, sconti.

Dell'ordine si vuole tener traccia di: utente che ha effettuato l'ordine, data in questo effettuato l'ordine, lo stato, l'indirizzo e il nominativo di consegna.
Sia per il prodotto nell'ordine che è nel carrello si vuole tener traccia dei prodotti e dell'ordine (o del carrello) a cui appartiene e in che quantità e presente

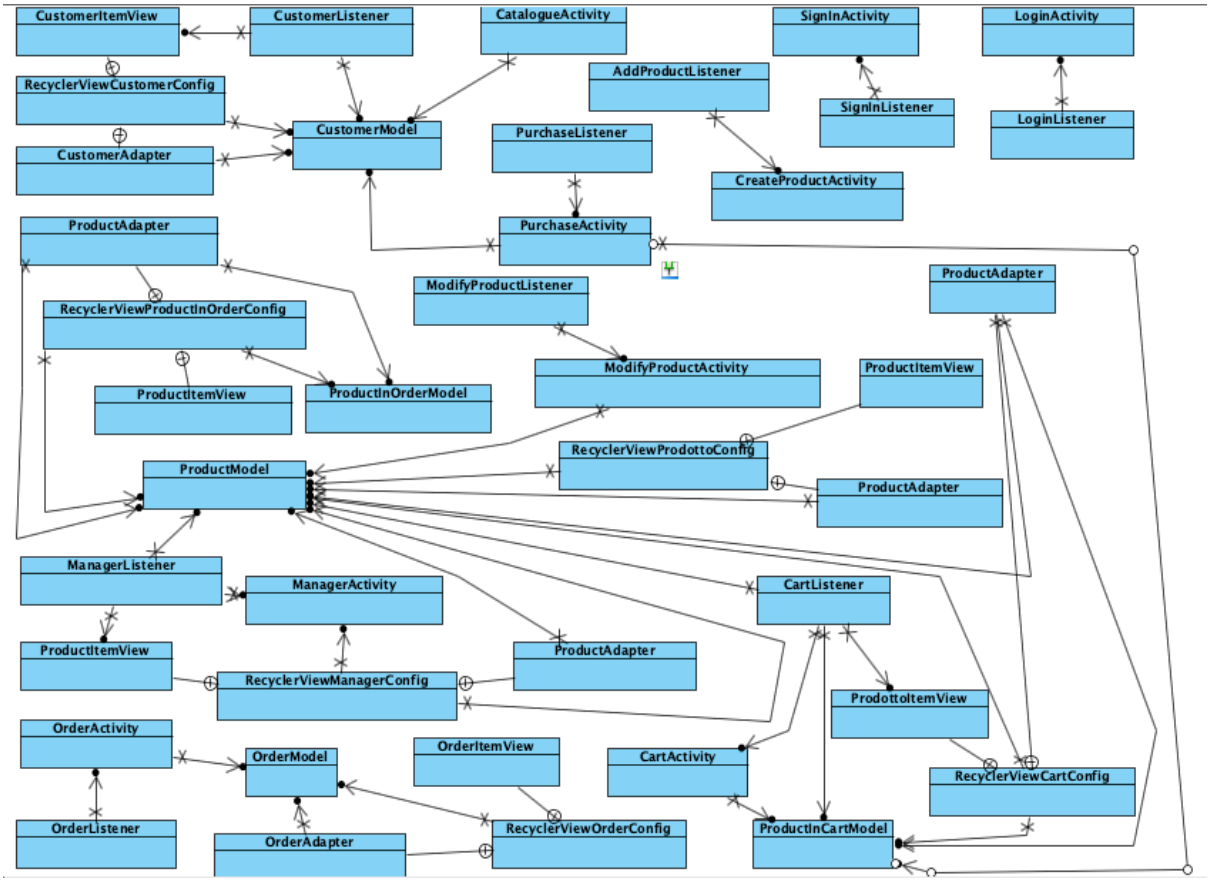

Immagine 4: Class diagram

4.3 Diagrammi di sequenza

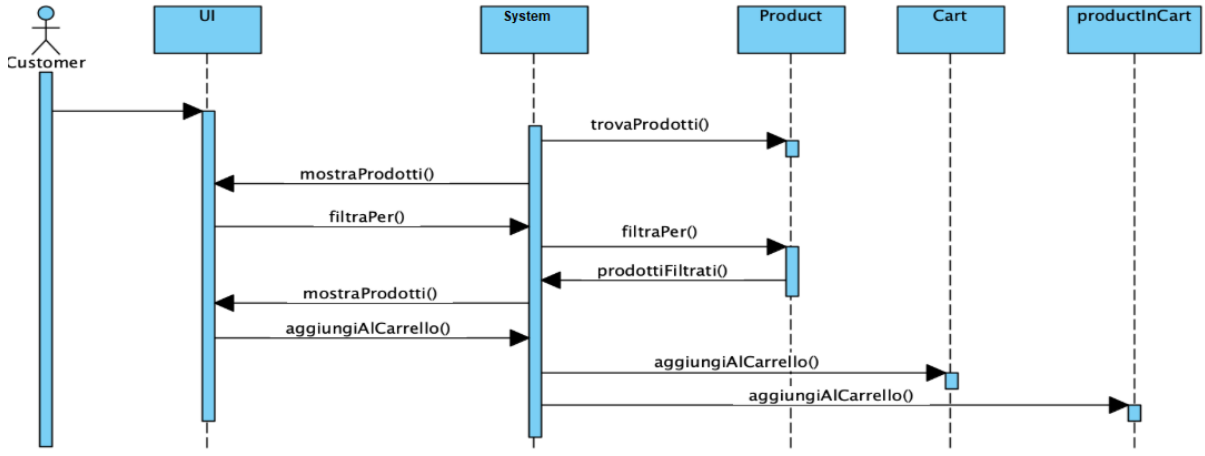

Immagine 5: Diagramma di sequenza "Consultazione catalogo"

Nel caso d'uso "Consultazione catalogo", il sistema istanzia i prodotti presenti nel catalogo e li mostra all'utente tramite l'UI, tramite la quale il cliente, dopo aver effettuato il login, ha la possibilità di filtrare i prodotti o di aggiungerli al carrello.

Nel primo caso l'UI chiede al sistema di filtrare i prodotti, il sistema istanzia solo i prodotti che rispettano il vincolo e li mostra al cliente. Nel secondo caso il sistema riceve le informazioni del prodotto dall'UI e crea il carrello.

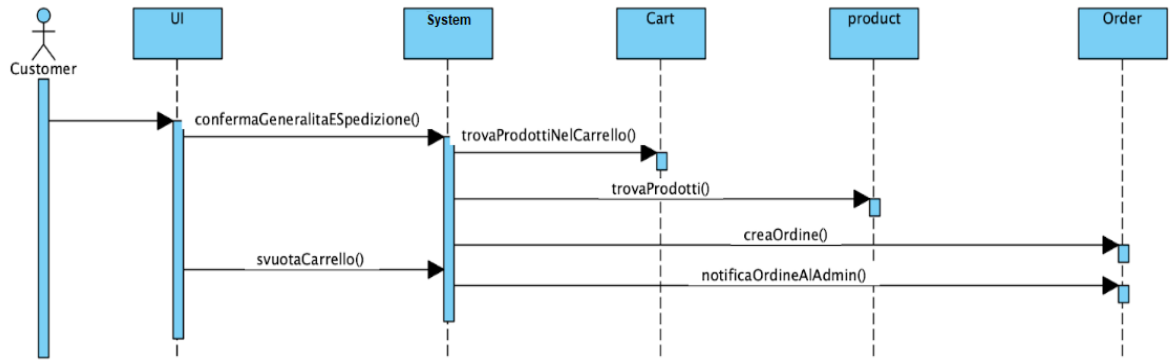

Immagine 6: Diagramma di sequenza "Acquisto prodotto"

Nel caso d'uso "Acquisto prodotto", il cliente tramite l'UI conferma le proprie generalità e le preferenze per la spedizione.

Il sistema comunica con il carrello per sapere quali prodotti il cliente desidera acquistare, in fine crea l'ordine e lo notifica all'amministratore.

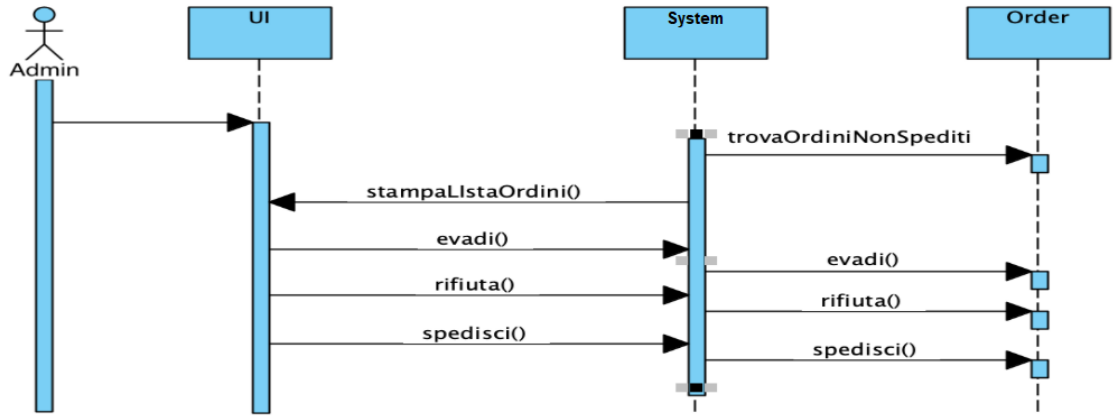

Immagine 7: Diagramma di sequenza "Gestione ordini"

Nel caso d'uso "Gestione ordini", il sistema mostra all'amministratore, che ha effettuato l'accesso, la lista degli ordini non spediti, istanziando gli ordini. L'amministratore eseguirà, tramite la UI, una operazione tra evadi, rifiuta o spedisci e il sistema modificherà l'ordine istanziato.

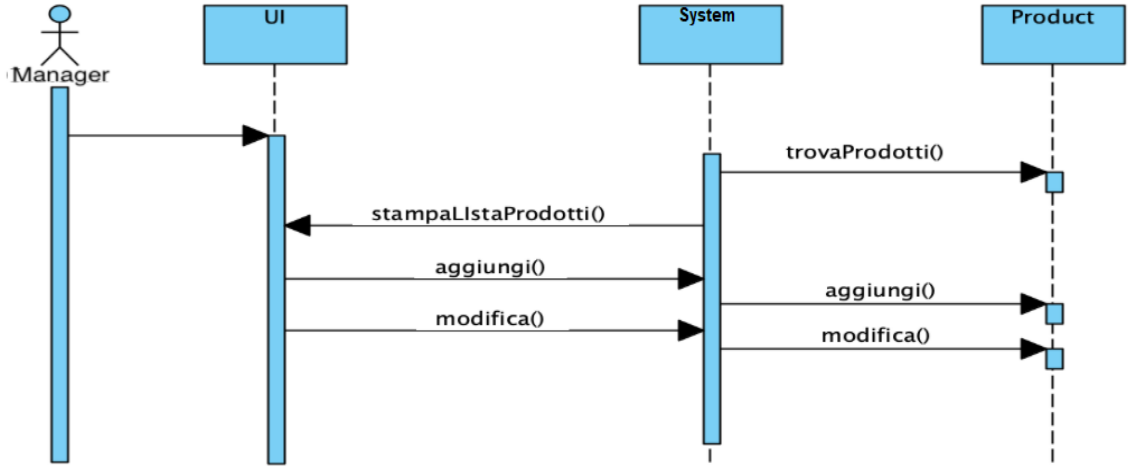

Immagine 8: Diagramma di sequenza "Gestione catalogo"

Nel caso d'uso "Gestione catalogo". Il sistema istanzia i prodotti e li mostra al gestore tramite l'UI. Il gestore può modificare o aggiungere prodotti tramite l'UI e il sistema aggiorna o crea prodotti istanziandoli.

4.4 Modello E/R e schema relazionale

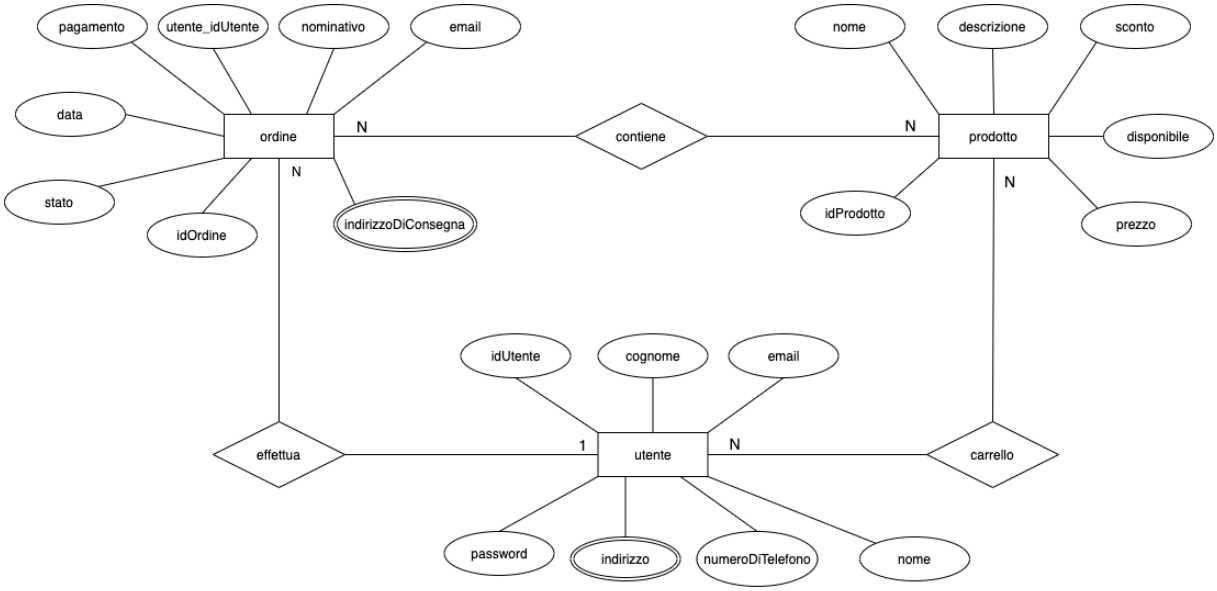

Immagine 9: Diagramma E/R

Nel progetto. Originale il database, come già specificato, era un database relazionale, quindi era stato progettato anche un modello E/R e uno schema relazionale.
Le entità individuate sono: prodotto, ordine e utente (che si divide nelle entità cliente, amministratore e gestore). Gli attributi candidati a chiave primaria sono rispettivamente idProdotto, idOrdine e idUtente, non sono previste chiavi esterne per le entità prodotto e utente, mentre per l'entità ordine è chiave esterna utente_idUtente che lega un ordine ad uno specifico utente.

Le relazioni identificate sono “contiene”, “carrello”, “effettua”.

Ogni ordine contiene uno o più prodotti e viceversa e questo si traduce con una relazione molti a molti.

Ogni cliente inserisce nel carrello uno o più prodotti e ogni prodotto si può trovare nel carrello di uno o più utenti, quindi, anche questa relazione si traduce con relazione del tipo una molti a molti.

Ogni cliente può effettuare uno o più ordine e ciò corrisponde ad una relazione del tipo uno a molti.

Ogni utente appartiene a una e una sola tipologia (cliente, gestore o amministratore).

Questo diagramma E/R si traduce nel seguente schema relazionale:

ordine(idOrdine, stato, data, pagamento, nominativo, email, cittaDiConsegna, IndirizzoDiConsegna, nCivicoDiConsegna, utente_idUtente)

utente(idUtente, nome, cognome, email, numeroDiTelefono, password, citta, indirizzo, nCivico)

prodotto(idProdotto, nome, descrizione, sconto, disponibile, prezzo)

carrello(utente_IdUtente, prodotto_idProdotto, quantita)

contiene(ordine_idOrdine, prodotto_idProdotto, quantita)

Capitolo 5 Implementazione

L'interazione con il database è molto differente da quella che è stata utilizzato nel progetto di partenza, in quanto si è passati a un database non relazionale.
Il risultato è stato una completa reimplementazione delle classi di DAO e delle interfacce e parte delle classi di Model. Si sono, perciò, dovute adattare anche le classi di Business Manager.
Avendo a che fare con un ambiente diverso, la view è stata riscritta completamente da zero e questo ha portato a dover adattare anche le classi di Listener.

```
private FirebaseDatabase mDatabase;
private DatabaseReference mReference;

private ProductDao() {
   mDatabase = FirebaseDatabase.getInstance();
   mReference = mDatabase.getReference("product");
}
```

5.1 Inserire dati nel database

Il metodo usato per aggiungere elementi al database, nell'esempio seguente, per aggiungere prodotti al catalogo, è addProduct():

```
public void addProduct(ProductModel productModel, final DataStatusProduct dataStatusProduct)
{
   String key = mReference.push().getKey();
   mReference.child(key).setValue(productModel).addOnSuccessListener(new
OnSuccessListener<Void>() {
       @Override
       public void onSuccess(Void aVoid) {
           dataStatusProduct.DataIsInserted();
       }
   });
}
```

Si è utilizzato un oggetto Java, i cui contenuti vengono automaticamente mappati sulle posizioni figlio in modo nidificato. L'uso di un oggetto Java inoltre rende il codice più leggibile e più facile da gestire.
Tramite il metodo push() si aggiunge un oggetto al nodo "product" e con il metodo getKey() si ottiene la chiave dell'oggetto. Dopo aver creato l'oggetto si crea un riferimento al nodo con il metodo child(key).
Il metodo setValue(productModel) sovrascrive i dati nella posizione specificata, inclusi eventuali nodi figlio. Tuttavia, è ancora possibile aggiornare un figlio senza riscrivere l'intero oggetto. Alla fine l'operazione è stata legata al Listener OnSuccessListener che, se l'operazione è andata a buon fine, lo comunicherà alla classe di Business tramite il metodo DataIsInserted().

5.2 Leggere dati dal database

Il metodo usato per leggere elementi dal database, nell'esempio seguente per trovare i prodotti in vendita nel catalogo, è `findAllProduct()`:

```
public void findAllProduct(final DataStatusProduct dataStatusProdotto) {

    mReference.addValueEventListener(new ValueEventListener() {
        ArrayList<ProductModel> productModelArrayList = new ArrayList<>();
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            productModelArrayList.clear();
            for(DataSnapshot keyNode : dataSnapshot.getChildren()){
                ProductModel productModel = keyNode.getValue(ProductModel.class);
                productModel.setId(keyNode.getKey());
                productModelArrayList.add(productModel);
            }
            dataStatusProdotto.DataIsLoaded(productModelArrayList);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });
}
```

Si è legato, al riferimento del nodo da cui si vuole leggere i dati, un listener che rimane costantemente in ascolto. Questo listener implementa il metodo `onDataChange()`, usato per leggere un'istantanea statica dei contenuti in un determinato percorso, così come esistevano al momento dell'evento. Questo metodo viene attivato una volta quando il listener è collegato e di nuovo ogni volta che i dati cambiano. Al callback dell'evento viene passata un'istantanea contenente tutti i dati in quella posizione, inclusi i dati child. Se non ci sono dati, lo snapshot restituirà "null" quando si chiama il metodo `getValue()` su di esso. Ogni oggetto che viene trovato all'interno dello snapshot viene aggiunto ad un array che viene restituito alla classe di Business che ha richiamato il metodo `findAllProduct()` tramite il metodo `DataIsLoaded(productModelArrayList)`.
In alcuni casi è necessario leggere i dati una sola volta e quindi chiudere immediatamente il collegamento con il database, in questo caso è possibile utilizzare il metodo `addListenerForSingleValueEvent()` per semplificare questo scenario: si innesca una volta e poi non si innesca più.

5.3 Aggiornare dati del database

In questo caso si procede in modo analogo alla scrittura, i dati presenti al nodo indicato verranno semplicemente sovrascritti. Nel seguente esempio si mostra come viene aggiornato un prodotto dopo che il gestore ha modificato i suoi attributi:

```
public void update(ProductModel productModel, final DataStatusProduct dataStatusProduct) {
   mReference.child(productModel.getId()).setValue(productModel).addOnSuccessListener(new
OnSuccessListener<Void>() {
       @Override
       public void onSuccess(Void aVoid) {
           dataStatusProduct.DataIsUpdated();
       }
   });
}
```

L'unica differenza è che se l'operazione va a buon fine sarà richiamato il metodo `DataIsUpdated()`.

5.4 Eliminare dati dal database

È inoltre possibile eliminare specificando null come valore per un'altra operazione di scrittura come `setValue()` , è il caso dell'amministratore che decide di rifiutare la richiesta di registrazione del cliente.
anche in questo se l'operazione va a buon fine, l'avvenuta eliminazione sarà notificata alla classe di Business tramite il metodo `DataIsDeleted()`.

```
private FirebaseDatabase mDatabase;
private DatabaseReference mReference;

private CustomerDao() {
   mDatabase = FirebaseDatabase.getInstance();
   mReference = mDatabase.getReference("customer");
}
…

public void refuse(String idCustomer, final DataStatusCustomer dataStatusCustomer) {
   mReference.child(idCustomer).setValue(null).addOnSuccessListener(new
OnSuccessListener<Void>() {
       @Override
       public void onSuccess(Void aVoid) {
           dataStatusCustomer.DataIsDeleted();
       }
   });
}
```

5.5 Interface

Ogni classe di DAO ha un'interfaccia contenente i metodi che occorrono per comunicare con le classi di Business che le richiamano, nel caso della classe ProducDao l'interfaccia è:

```
public interface DataStatusProduct {
   void DataIsLoaded(ArrayList<ProductModel> productModelArrayList);
   void DataIsInserted();
   void DataIsUpdated();
   void DataIsDeleted();
}
```

5.6 Business Manager

I metodi contenuti nelle classi di DAO sono richiamati nelle classi di Business per poter eseguire operazioni e i dati vengono passati tramite i metodi contenuti nelle interfacce, che sono implementati a seconda dell'uso che bisogna farne, nel seguito si mostra come viene utilizzato il metodo `findAllProduct()` per creare il catalogo dei prodotti:

```
public void createCatalague(final CatalogueActivity activity, final String orderBy) {
   ProductDao.getInstance().findAllProduct(new DataStatusProduct() {
       @Override
       public void DataIsLoaded(ArrayList<ProductModel> productModelArrayList) {
           activity.findViewById(R.id.progressBar).setVisibility(View.GONE);

           ArrayList<ProductModel> productModelArrayListForoRecyclerView = new
ArrayList<>();

           for (ProductModel productModel : productModelArrayList) {
               if(productModel.isAvailability())
                   productModelArrayListForoRecyclerView.add(productModel);
           }
           RecyclerView recyclerView = activity.findViewById(R.id.recycleview);

           new RecyclerViewProdottoConfig().setConfig(recyclerView, activity,
                   filtraPer(productModelArrayListForoRecyclerView, orderBy));


       }

       @Override
       public void DataIsInserted() {

       }

       @Override
       public void DataIsUpdated() {

       }

       @Override
       public void DataIsDeleted() {

       }
   });
}
```

Si è creata un'istanza dell'interfaccia per ottenere i dati dalla classe di DAO all'interno della quale si è implementato il metodo `DataIsLoaded()` per creare il catalogo.

5.7 Authentication

Nelle app moderne è importante conoscere l'identità di un utente, ciò consente di: salvare in modo sicuro i dati relativi a ogni utente, fornire loro un'esperienza personalizzata, consentire l'accesso da diversi dispositivi.

Firebase Authentication [8] fornisce servizi di back-end, SDK di facile utilizzo e librerie dell'interfaccia utente pronte all'uso per autenticare gli utenti.
Supporta l'autenticazione tramite password, numeri di telefono, Google, Facebook e Twitter.
In questo elaborato si è implementato solo l'autenticazione tramite e-mail e password.

5.7.1 Sign in

Si crea un'istanza di FirebaseAuth e si usa il metodo `createUserWithEmailAndPassword` che prende come parametri l'e-mail e la password inseriti dall'utente. Questo metodo crea un utente che viene salvato nella Firebase.

```
FirebaseAuth auth = FirebaseAuth.getInstance();
auth.createUserWithEmailAndPassword(activity.getEmailEditText().getText().toString(),
activity.getPasswordEditText().getText().toString()).
           addOnCompleteListener(activity, new OnCompleteListener<AuthResult>() {
       @Override
       public void onComplete(@NonNull Task<AuthResult> task) {
           if(task.isSuccessful()){
               ...
           }
           else{
               ...
           }
       }
   });
```

5.7.2 Login

Si crea un'istanza di FirebaseAuth e si usa il metodo `signInWithEmailAndPassword()` che prende come parametri l'email e la password dell'utente. Se l'utente esiste si può accedere ai dati tramite il metodo `getCurrentUser()`

```
FirebaseAuth auth = FirebaseAuth.getInstance();
auth.signInWithEmailAndPassword(activity.getEmailEditText().getText().toString(),
       activity.getPasswordEditText().getText().toString()).addOnCompleteListener(activity,
new OnCompleteListener<AuthResult>() {

   @Override
   public void onComplete(@NonNull Task<AuthResult> task) {
       if (task.isSuccessful()) {
           UserBusiness.getInstance().findUser(activity, auth.getCurrentUser());

       } else {
           ...
       }
   }
});
```

Capitolo 6 Testing

6.1 Testing funzionale

Il Test Funzionale [13] è una tecnica di tipo black-box, ovvero il comportamento dell'applicazione è descrivibile solo tramite il suo comportamento esterno in risposta a una sollecitazione.

Il Test Funzionale rappresenta il primo passo per la garanzia della qualità dell'applicazione. Esso permette di verificare sia che l'applicazione funzioni in modo corretto dal lato tecnologico sia che effettivamente le funzionalità offerte corrispondano a quelle richieste dall'utilizzatore finale.

6.1.1 Consultazione catalogo

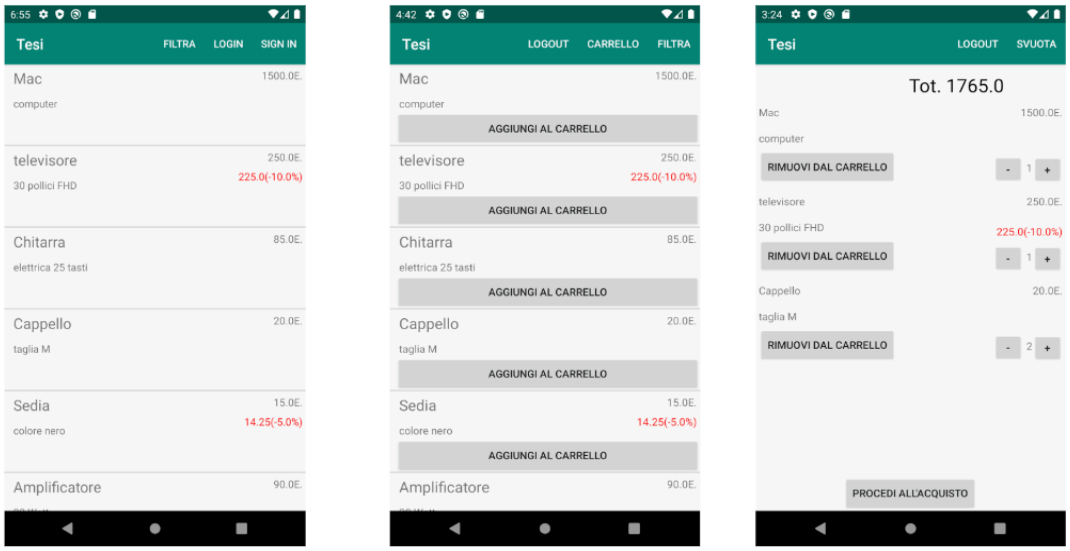

Immagine 10: A sinistra l'utente non ha ancora effettuato l'accesso, al centro il cliente ha effettuato l'accesso, a destra il cliente visualizza il carrello.

Il sistema sia nel caso in cui il cliente abbia effettuato l'accesso, sia nel caso non l'abbia fatto, mostra tutti i prodotti disponibili presenti nel catalogo e può filtrarli per prezzo. Il cliente che non ha effettuato l'accesso può accedere alla pagina di registrazione con il pulsante "sign in" oppure alla pagina di login con il pulsante "login". Il cliente che ha effettuato l'accesso visualizzerà invece che i pulsanti "sign in" e "login" i pulsanti "logout" e "carrello", con quest'ultimo può visualizzare il suo carrello. Ogni prodotto presenta il pulsante "aggiungi al carrello" con il quale può aggiungere i prodotti al carrello. Quando il cliente clicca su "carrello", il sistema mostra tutti i prodotti contenuti nel carrello del cliente. Quest'ultimo può modificare la quantità dei prodotti premendo sui pulsanti "+" e "-", rimuovere un singolo prodotto con il pulsante "rimuovi dal carrello", svuotare il carrello cliccando su "svuota" o procedere all'acquisto cliccando su "procedi all'acquisto".

6.1.2 Acquista prodotto

Dopo che il cliente ha cliccato su "procedi all'acquisto", pulsante presente nella pagina del carrello, il sistema chiede all'utente di confermare i suoi dati personali e le preferenze per la consegna. Infine, clicca sul pulsante "conferma" per effettuare l'ordine. Il sistema si occuperà di notificare l'ordine all'amministratore.

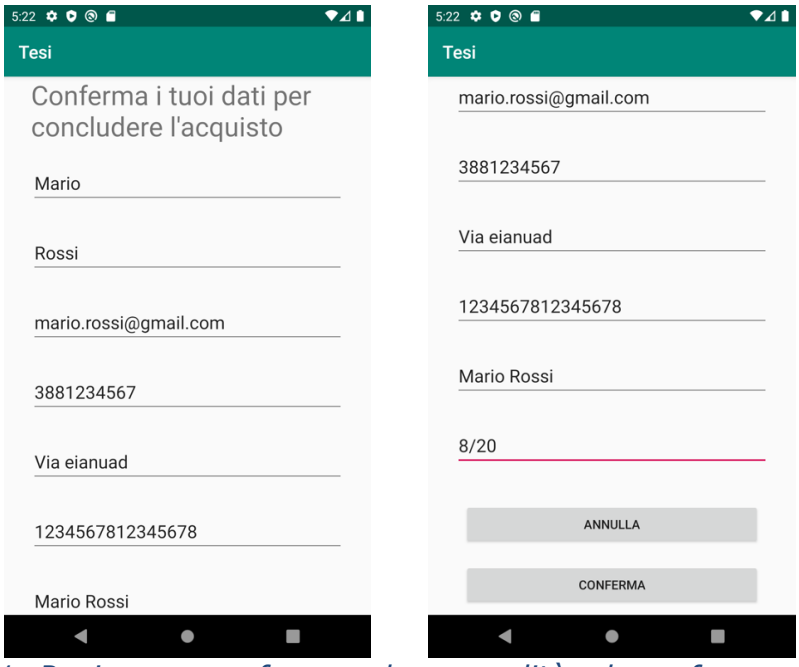

Immagine 11: Pagina per confermare le generalità e le preferenze di consegna

6.1.3 Gestione catalogo

Il sistema mostra al gestore tutti i prodotti presenti nel catalogo, sia quelli disponibili che quelli non disponibili. Il gestore può scegliere quale operazione effettuare cliccando sul prodotto oppure sul pulsante "aggiungi prodotto". Nel primo caso il sistema apre la pagina modifica prodotto, dove vengono mostrate le attuali informazioni del prodotto. Dopo aver modificato le informazioni, il gestore clicca sul pulsante "salva" per rendere definitive le modifiche.

Nel secondo caso il sistema apre la pagina aggiungi prodotto, che mostra un form vuoto dove il gestore può inserire le informazioni del nuovo prodotto. Cliccando su "salva", il gestore inserisce definitivamente il prodotto nel catalogo.

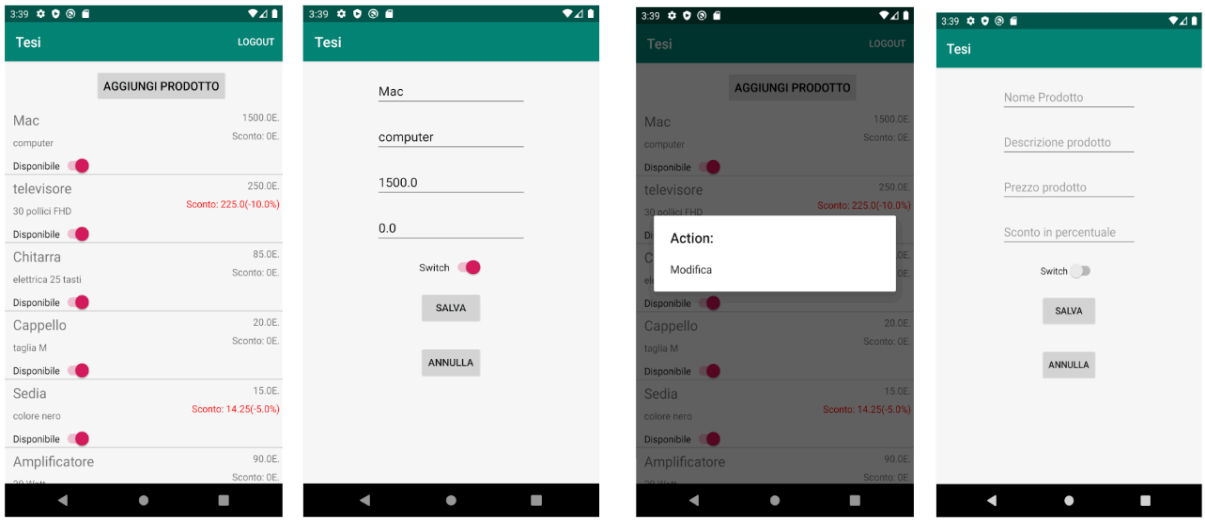

Immagine 12: Da sinistra, la pagina principale di gestione dei prodotti, il menu "modifica", la pagina "modifica prodotto", la pagina "aggiungi prodotto"

6.1.4 Gestione registrazioni

Il sistema mostra all'amministratore i clienti che hanno richiesto di effettuare la registrazione ed accanto ad ognuno di essi il pulsante "rifiuta" o "conferma". Cliccando su "rifiuta" il sistema elimina l'utente, cliccando su "conferma" il sistema abilita il cliente all'accesso.

Inoltre, vengono mostrati anche i pulsanti "logout" e "gestione ordini". Il secondo pulsante porta l'amministratore alla pagina di gestione degli ordini.

Immagine 13: "Gestione delle registrazioni"

6.1.5 Gestione ordini

L'amministratore accede alla pagina di gestione degli ordini cliccando su "gestisci ordini" presente nella pagina principale dell'amministratore. Qui il sistema mostra tutti gli ordini effettuati dagli utenti che non sono ancora stati spediti, dando informazioni sul cliente che ha effettuato l'ordine, lo stato dello stesso e la data in cui è stato effettuato.
Cliccando su un ordine l'amministratore può visualizzare i prodotti contenuti nell'ordine e in che quantità sono stati ordinati. Cliccando sugli appositi pulsanti "evadi", "rifiuta", "spedisci" l'amministratore può evadere, rifiutare o spedire l'ordine e il sistema aggiorna lo stato dell'ordine.

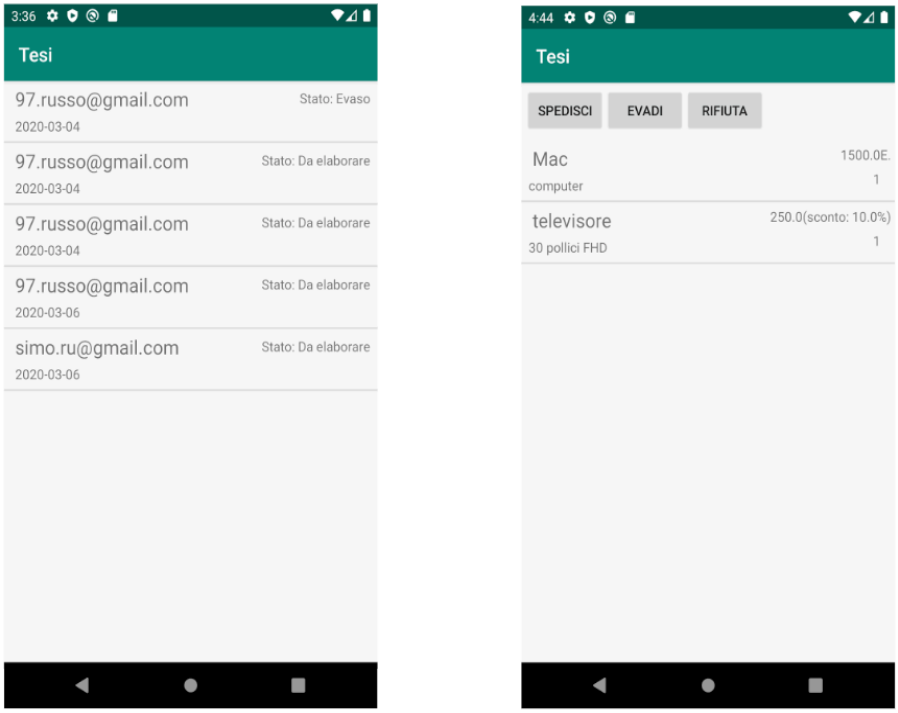

Immagine 14: A sinistra la pagina contenente tutti gli ordini, a destra la pagina di un determinato ordine.

6.1.6 Login

L'utente può accedere alla pagina di login dalla pagina del catalogo cliccando su "login".
Il sistema stampa il form di login, dopo che l'utente lo compila e clicca su "login", il sistema procede all'identificazione dell'utente, aprendo la pagina del catalogo se l'utente è un cliente ed è stato autorizzato all'accesso da un amministratore, in caso contrario il sistema gli nega l'accesso segnalando il fatto che la sua richiesta di registrazione non è ancora stata approvata. L'utente viene reindirizzato alla pagina di gestione delle registrazioni se è un amministratore oppure alla pagina di gestione del catalogo se è un gestore.
Se l'utente non è registrato può accedere alla pagina di registrazione cliccando su "sign in".

Se un utente ha perso la password può cliccare su "recupera password" dopo aver inserito il suo indirizzo e-mail, il sistema invia una mail all'indirizzo specificato contenente un link per reimpostare la password.
Nel caso in cui le credenziali non siano corrette il sistema avvisa l'utente.

6.1.7 Registrazione

Un utente che vuole registrarsi può accedere alla pagina di registrazione dalla pagina del catalogo o dalla pagina di login, in entrambi i casi, cliccando su "sign in".
Il sistema mostra al cliente il form di registrazione il quale dopo averlo compilato clicca su "sign in". Il sistema notifica la richiesta all'amministratore.
Se uno dei campi è vuoto o se l'utente tenta di registrarsi con un'e-mail già in uso il sistema blocca la registrazione e lo notifica all'utente.

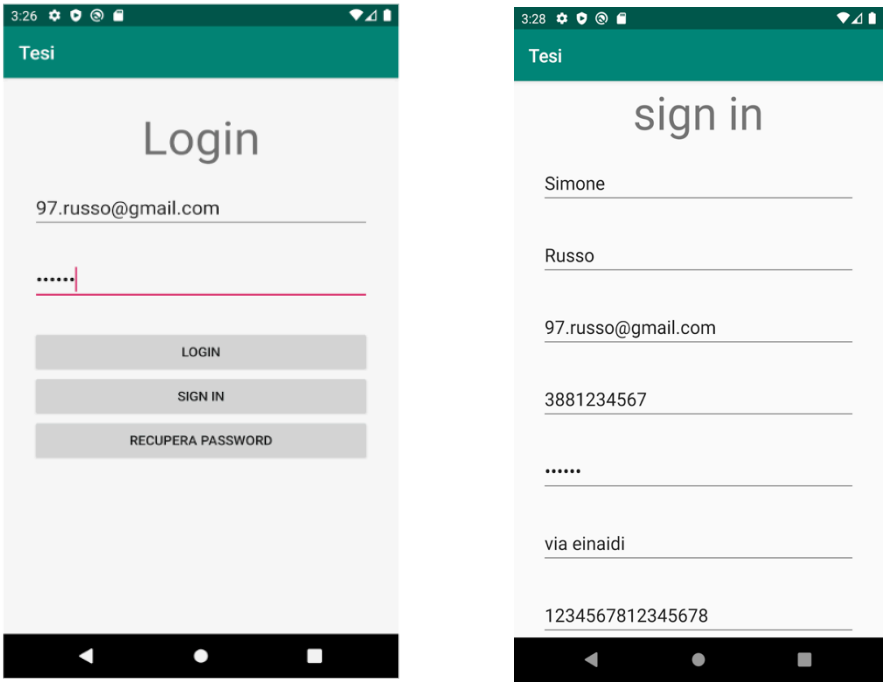

Immagine 15: A sinistra la pagina di Login, a destra la pagina di registrazione

Capitolo 7 Conclusioni

In questa tesi sono stati migrati alcuni casi d'uso di una applicazione preesistente in ambito e-commerce, a partire da un'architettura software stand-alone verso una nuova architettura software per dispositivi mobili Cloud-based.

Sono stati raggiunti i seguenti obiettivi:

- effettuare una migrazione di un sistema stand-alone ad un sistema cloud-based per dispositivi mobili
- sviluppare le seguenti funzionalità: profilazione degli utenti; catalogo disponibile a chiunque visiti il servizio; la possibilità di effettuare acquisti; un sistema di gestione delle richieste di registrazione degli utenti e degli ordini per l'amministratore; un sistema di gestione dei prodotti per il gestore del sistema.
- definire una strategia di migrazione secondo la quale, partendo dal software esistente, si cerca di riciclare il più possibile il codice, mantenendo la stessa struttura ma utilizzando tecnologie diverse, quali Android e Firebase.
- studiare e comprendere al meglio la tecnologia Android nel suo insieme, come interagire con l'utente e come questa integra la Firebase sviluppata da Google.
- effettuare dei test funzionali che contribuiscono a verificare la corrispondenza dell'applicativo rispetto ai requisiti richiesti e il suo corretto funzionamento.

In particolare, si evidenzia la possibilità di gestire diversi client connessi contemporaneamente, fornendo informazioni sempre aggiornate attraverso il servizio di Authentication e Real Database offerti dalla Firebase di Google.

Nel processo di migrazione non è stato possibile riutilizzare gran parte del codice sorgente implementato nel progetto originario, questo a causa dell'utilizzo di un nuovo database e della nuova User Interface, che è stata completamente ridisegnata per adattarsi alle esigenze di un dispositivo mobile. Durante l'implementazione le difficoltà riscontrate hanno riguardato principalmente questi due aspetti.

L'integrazione delle funzionalità offerte dalla Firebase, in particolare, con riferimento al Realtime Database, prevede un tipo implementazione del tutto nuovo, basato sul meccanismo del DataSnaphot, che non fornisce dati direttamente fruibili dopo aver interrogato il database ma bisogna ricorrere alle interfacce per far comunicare il livello di DAO e Business Manager. Esse devono essere implementate ogni volta in modo diverso, a seconda dello scenario da modellare.

L'implementazione dell'Authentication e la gestione degli utenti è stata più immediata.

Il nuovo linguaggio di markup ha richiesto uno studio specifico per poter sfruttare al meglio le funzionalità offerte.

7.1 Implementazioni future

L'implementazione del codice è robusta e rispetta i parametri di sviluppo previsti dal modello insegnato nel corso di Principi di Progettazione del Software, con la suddivisione del codice in classi di View, Listener, Business Manager, Model, Service e Interface.
Ulteriori implementazioni possono prevedere, ad esempio, l'aggiunta di panieri di spesa dove i clienti possono creare liste predefinite di prodotti che potrebbero voler acquistare periodicamente o il login via Social Media o Google. Un'ulteriore funzione che si potrebbe decidere di implementare è la possibilità di inviare push notification al gestore quando un utente effettua un ordine, all'utente quando lo stato di un suo ordine cambia o all'amministratore quando un utente effettua la registrazione.
Ognuna di queste implementazioni, per il tipo di architettura che si è seguita nello sviluppo, non necessita di grandi modifiche al codice già esistente e può essere sviluppato come una semplice aggiunta a questo, in particolare per implementare le push notification si può sfruttare ancora una volta la Firebase di Google che offre un servizio di Cloud Messaging.
Infine, per rendere completo il servizio bisognerebbe migrare l'app anche verso il sistema operativo iOS. Il linguaggio di programmazione che è necessario conoscere per la realizzazione di applicazioni iOS è l'Objective-C, quindi non si riuscirebbe a recuperare nessuna parte del codice. La Firebase di Google e tutte le sue funzionalità sono disponibili anche per questo nuovo linguaggio ma non sarà possibile riciclare alcuna parte di codice.

Bibliografia

[1] https://it.wikipedia.org/wiki/Java_(linguaggio_di_programmazione)

[2] https://it.wikipedia.org/wiki/Java_Development_Kit

[3] https ://it.wikipedia.ord/wiki/Android

[4] https://it.wikipedia.org/wiki/Android_Studio

[5] https://www.html.it/articoli/firebase/

[6] https://www.emanuelepaluzzi.it/2019/03/04/cose-firebase/

[7] https://www.chimerarevo.com/guide/internet/firebase-305791/#0

[8] https://firebase.google.com/docs/auth

[9] https://firebase.google.com/docs/database

[10] https://en.wikipedia.org/wiki/Firebase

[11]https://www.emanuelepaluzzi.it/2019/03/04/cose-firebase/

[12] https://it.wikipedia.org/wiki/XML

[13] https://www.slideshare.net/IxmaSoft/test-funzionale-8311138

UNIVERSITÀ DEL SALENTO
DIPARTIMENTO DI INGEGNERIA DELL'INNOVAZIONE
Biblioteca dip.le aggr.– sez. Ing. Innov.
Via per Monteroni
I 73100 Lecce
T +39 0832 297201/202/203
F +39 0832 325362
E ...@unisalento.it

AUTORIZZAZIONE ALLA CONSULTAZIONE DELLA TESI DI LAUREA

Il/La sottoscritto Simone Russo,
nato a Lecce il 18/03/97
e residente in via Luigi Einaudi, 24
Città Cavallino (L), laureando del Corso di Laurea

- (•) triennale
- ( ) magistrale
- ( ) quinquennale

in Ingegneria dell'Informazione,
matricola 20028586, e-mail Simone.Russo1@studenti.unisalento.it,
Titolo della tesi: Migrazione di un'architettura stand-alone verso una Cloud-based per dispositivi Android nel settore dell'e-commerce (Migration from stand-alone to Cloud-based architecture for Android devices in the e-commerce field)

	Data	**Firma**
(•) AUTORIZZA LA CONSULTAZIONE ( ) NON AUTORIZZA LA CONSULTAZIONE	Lecce, 5/04/2020	Simone Russo
( ) AUTORIZZA LA CONSULTAZIONE ( ) NON AUTORIZZA LA CONSULTAZIONE	Lecce, _____	_____
( ) AUTORIZZA LA CONSULTAZIONE ( ) NON AUTORIZZA LA CONSULTAZIONE	Lecce, _____	_____
( ) AUTORIZZA LA CONSULTAZIONE ( ) NON AUTORIZZA LA CONSULTAZIONE	Lecce, _____	_____

della propria tesi di laurea depositata presso la Biblioteca del Dipartimento di Ingegneria dell'Innovazione.[1]

L'autore della tesi di laurea mantiene su di essa tutti i diritti d'autore, morali ed economici, ai sensi della Legge Italiana sul Diritto d'Autore (legge 633/1941 e successive modificazioni).

[1] Il laureando ha facoltà di cambiare l'autorizzazione concessa o negata. È considerata correntemente valida l'autorizzazione con la data più recente.