Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Progettazione e implementazione di un'app Android per la gestione di attività di baratto tra utenti

Design and implementation of an Android app for the management of bartering activities between users

Relatore

Prof. Ursino Domenico

Candidato

Sospetti Mattia

Anno accademico 2019-2020

Indice

Introduzione 3

1 Il baratto 7
1.1 Concetto 7
1.2 Storia del baratto 8
1.2.1 Tempi antichi 8
1.2.2 Tempi moderni 10
1.3 Baratto online 11

2 Android 13
2.1 Introduzione ad Android 13
2.1.1 Storia 14
2.1.2 Diffusione e frammentazione 15
2.2 Sviluppo 16
2.2.1 Architettura 17
2.2.2 Java e macchina virtuale 18
2.2.3 Android Studio 20

3 Analisi dei requisiti e progettazione 23
3.1 Definizione dei requisiti 23
3.1.1 Descrizione del progetto 24
3.1.2 Definizione dei requisiti funzionali 24
3.1.3 Definizione dei requisiti non funzionali 30
3.2 Mappa del sito 31
3.3 Diagramma dei casi d'uso 32
3.3.1 Descrizione delle attività 34
3.4 Struttura del database 42
3.4.1 Oggetti 43
3.4.2 Utenti 46
3.4.3 Scambi 47
3.4.4 Riepilogo scambi 48

4 Implementazione dell'app e manuale utente 51
4.1 Implementazione dell'app 51
4.1.1 Registrazione e Login 51
4.1.2 Catalogo 53
4.1.3 Profilo 57
4.1.4 Aggiunta di un prodotto 60
4.1.5 Offerte inviate e ricevute 61
4.1.6 Riepilogo Scambi 64
4.1.7 Contattaci e Logout 66
4.2 Manuale utente 67
4.2.1 Primo avvio 67
4.2.2 Login e Catalogo 68
4.2.3 Visualizzazione prodotto 70
4.2.4 Formulare un'offerta 70
4.2.5 Profilo 71
4.2.6 Inserisci un prodotto 73
4.2.7 Miei oggetti 75
4.2.8 Offerte inviate e ricevute 75
4.2.9 Riepilogo scambi 76
4.2.10 Contattaci e Logout 77

5 Discussione in merito al lavoro svolto 79
5.1 Lezioni apprese 79
5.1.1 Android e programmazione mobile 79
5.1.2 Android Studio 80
5.1.3 Firebase 81
5.2 Revisione critica 82
5.2.1 Criticità, limiti e miglioramenti apportabili 82

6 Conclusioni 85

Riferimenti bibliografici 87

Ringraziamenti 89

Elenco delle figure

1.1 Rappresentazione del baratto 7
1.2 Copertina dell'Iliade, opera di Omero 9
1.3 Tavola di terracotta, una prima forma di conto corrente 10
1.4 Il baratto nell'Alto Medioevo tornò ad essere una pratica comune 10
1.5 Un esempio di shopping club 11
1.6 Il baratto online 12

2.1 Icona di Android 13
2.2 Android contro Apple nel 2011 15
2.3 Fenomeno della frammentazione di Android negli anni 16
2.4 Struttura di Android 17
2.5 Struttura di un'applicazione Android 19
2.6 Compilazione di un'applicazione Android 19
2.7 Icona di Android Studio 20
2.8 Una tipica schermata di Android Studio 21

3.1 Mappa del sito di BartApp 33
3.2 Diagramma dei casi d'uso 34
3.3 Template per la descrizione dei casi d'uso 35
3.4 Descrizione del caso d'uso CU1 (Registrazione) e diagramma UML associato 36
3.5 Descrizione del caso d'uso CU2 (Login) e diagramma UML associato . 37
3.6 Descrizione del caso d'uso CU3 (Visualizzare i dettagli del proprio profilo) e diagramma UML associato 38
3.7 Descrizione del caso d'uso CU4 (Visualizzare i dettagli di un prodotto) e diagramma UML associato 39
3.8 Descrizione del caso d'uso CU5 (Formulare un'offerta) e diagramma UML associato 40
3.9 Descrizione del caso d'uso CU6 (Inserire un prodotto) e diagramma UML associato 41
3.10 Descrizione del caso d'uso CU7 (Visualizzare la lista dei Miei oggetti) e diagramma UML associato 42

3.11 Descrizione del caso d'uso CU8 (Ritirare un'offerta precedentemente inviata) e diagramma UML associato 43
3.12 Descrizione del caso d'uso CU9 (Concludere uno scambio) e diagramma UML associato 44
3.13 Descrizione del caso d'uso CU10 (Contattare il Servizio Clienti) e diagramma UML associato 45
3.14 Descrizione del caso d'uso CU11 (Logout) e diagramma UML associato 46
3.15 La schermata di Firebase relativa alla struttura dati "oggetti" 47
3.16 La schermata di Firebase relativa alla struttura dati "utenti" 48
3.17 La schermata di Firebase relativa alla struttura dati "scambi" 49
3.18 La schermata di Firebase relativa alla struttura dati "riepilogoscambi" 50

4.1 Icona di Bartapp 67
4.2 Schermata di Login 68
4.3 Schermata di Registrazione 69
4.4 Catalogo dei prodotti 69
4.5 Filtro delle categorie 70
4.6 Schermata dei dettagli del prodotto 71
4.7 Schermata di scelta del prodotto da scambiare 71
4.8 Accesso al menù laterale 72
4.9 Menù laterale 72
4.10 Area riservata dell'utente 73
4.11 Schermata per la modifica del nome utente 73
4.12 Schermata per la modifica della password 74
4.13 Schermata per l'inserimento di un oggetto nel catalogo 74
4.14 Lista degli oggetti dell'utente 75
4.15 Schermata delle offerte inviate dall'utente 76
4.16 Schermata delle offerte che sono state ricevute 77
4.17 Focus sul cestino per eliminare il riepilogo 77
4.18 Schermata per il riepilogo degli scambi effettuati in precedenza 78
4.19 Focus sulle ultime due funzionalità offerte dalla nostra applicazione . . 78

5.1 Icona di Android 80
5.2 Icona di Material Design 80
5.3 Icona di Android Studio 81
5.4 Icona di Firebase 82

Elenco dei listati

4.1 Codice per la registrazione 51
4.2 Codice per inserire un nuovo utente nel database 52
4.3 Codice per il login 52
4.4 Codice per la schermata principale dell'applicazione 53
4.5 Codice per la vista dei prodotti 54
4.6 Codice per la ricerca per nome 54
4.7 Codice per la ricerca per categoria 55
4.8 Codice per l'Adapter 56
4.9 Codice per l'immagine del profilo 57
4.10 Codice per la richiesta di permessi e per la modifica dell'immagine del profilo 58
4.11 Codice per la visualizzazione dell'immagine sullo schermo 60
4.12 Codice per la cattura della posizione del dispositivo 61
4.13 Codice relativo al metodo `onCreateView` 62
4.14 Codice per la gestione delle offerte ricevute 62
4.15 Codice per la gestione dei layout del riepilogo delle offerte 65
4.16 Codice per la gestione della cronologia delle offerte 66
4.17 Codice per l'invio della mail agli sviluppatori 66
4.18 Codice per il logout 66

Introduzione

Fin dagli albori della civiltà, l'uomo ha sempre avuto la necessità di scambiare oggetti. Sia per bisogno, sia per una volontà di disfarsi di un certo oggetto, lo scambio era la primitiva forma per ottenere quanto desiderato. Pertanto, molto rapidamente si è diffuso il concetto di "baratto"; esso ha rappresentato quindi la prima forma di commercio della storia dell'uomo e consisteva nello scambio di oggetti tra i rispettivi possessori, a seguito di una valutazione puramente qualitativa, tale da stimare i prodotti coinvolti con un valore pressoché identico.

Tramandata nel corso degli anni, tale pratica è valida tutt'oggi, con un aspetto diverso, sicuramente più moderno. Oggi, ad esempio, la valutazione è di stampo quantitativo, ed associa ad ogni prodotto un valore monetario, ovvero quanto richiesto dal possessore per ottenere quel particolare prodotto.

Ma la modernizzazione più importante che ha ricevuto tale pratica è la sua digitalizzazione. Del resto, con l'avvento delle piattaforme di e-commerce, il mondo del mercato è stato di certo rivoluzionato; si sta assistendo ad una graduale erosione di quelle che sono usanze, come il recarsi nel negozio per cercare, toccare e vedere con i propri occhi ciò che si vuole comprare, a favore di un modus operandi sicuramente più rapido e oltremodo comodo.

Infatti, l'e-commerce permette l'acquisto di un prodotto in ogni posto in cui ci si trovi, ed in qualunque momento della giornata, senza che ci sia la necessità di andare in negozio e comprare: semplicemente lo si ordina e lo si riceve direttamente a casa propria.

Per quanto una piattaforma di e-commerce possa essere considerata moderna e comoda, semplicemente una fetta della popolazione si oppone a tale rivoluzione, difendendo quelle che sono delle tradizioni definite come intramontabili, adducendo a tale opposizione anche la sincera necessità di barattare, operazione non realizzabile in tali piattaforme.

Ed è proprio in questo contesto che si colloca il nostro lavoro, che ha come obiettivo quello di realizzare ad un'applicazione che tenti di mediare, nonché trovare un punto d'incontro, tra chi è a favore di una modernizzazione estrema e i difensori della tradizione. Tale operazione di mediazione viene realizzata prendendo il lato moderno e digitale di una piattaforma di e-commerce (il che giustifica la nostra scelta di realizzare un'applicazione mobile), ed il lato più tradizionale, dando la possibilità di realizzare una pratica, affermatasi nel corso dei secoli come il baratto,

attraverso cui disfarsi di oggetti in disuso o di valore, per ottenerne di diversi e più utili.

L'applicazione oggetto della tesi è nata per il sistema operativo mobile Android, scelto principalmente per la sua enorme diffusione nel mercato odierno dei dispositivi mobili, con l'obiettivo di raggiungere un bacino d'utenza quanto più ampio possibile.

La scelta è ricaduta su questo sistema operativo anche per una questione di sviluppo; infatti, data la sua diffusione, la quantità di materiale di supporto reperibile online è immensa ed, inoltre, l'ambiente di sviluppo integrato messo a disposizione, che prende il nome di Android Studio, rende estremamente coinvolgente e stimolante tutta la fase implementativa.

Ad anticipare la fase di sviluppo c'è quella progettuale, in cui sono stati definiti tutti i requisiti che la nostra applicazione deve soddisfare, intesi come un insieme di funzionalità offerte, associando a ciascuna di esse una descrizione ed una serie di condizioni che devono essere verificate.

Durante questa fase, inoltre, si struttura tutta l'applicazione, modellata da un punto di vista concettuale come un susseguirsi di schermate, a seconda degli input dell'utente, che possono essere dei click, degli swipe o degli inserimenti di caratteri dalla tastiera, dando vita alla mappa del sito.

Infine, si progetta tutta la struttura dati, definendo le diverse tabelle che contengono i dati di diverso genere, ed il modo in cui essi vengono immagazzinati all'interno di ciascuna tabella. Per la struttura dati ci siamo serviti della piattaforma Firebase, che ha reso più agevole tutta la fase implementativa; inoltre, abbiamo ritenuto che la capacità di gestire dati in cloud fosse un requisito necessario per uno sviluppatore software, data la sempre maggiore rilevanza che stanno avendo i cloud, ed i servizi ad essi associati.

Una volta terminata la fase progettuale, si è entrati nel vivo dell'implementazione, scrivendo quel codice che realizza le funzionalità descritte nella fase precedente.

Durante l'implementazione ci siamo sforzati di garantire che la nostra applicazione fornisse un'esperienza utente il più soddisfacente possibile. Del resto, numerosi studi sottolineano come l'utilizzatore medio di un software mobile si aspetti di realizzare qualunque azione nel minor tempo possibile, ed in modo quanto più agevole possibile. Ed è proprio su questo concetto che ci siamo basati nella realizzazione di ciascuna funzionalità, a discapito, eventualmente, di funzioni più articolate o di una grafica particolarmente pesante, preferendo, infatti, ad essa una veste minimale. La grafica, pertanto, è stata curata seguendo i dettami del Material Design, concentrandoci cioè su un maggiore uso di layout basati su una griglia, animazioni e transizioni ed effetti di profondità, come l'illuminazione e le ombre.

A questa fase è seguita un'analisi del lavoro svolto, dove abbiamo studiato esplicitamente tutti i limiti e le criticità dell'applicazione. Durante questa analisi abbiamo proposto degli spunti di sviluppo al fine di evidenziare come l'app sia da considerarsi non come un prodotto concluso, bensì come un punto di partenza per un eventuale progetto futuro, in cui migliorare tutti i punti critici presentati in precedenza, ed aprire allo sviluppo di ulteriori funzionalità, per garantire il successo nello store delle applicazioni mobili.

La presente tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 verrà presentata la pratica del baratto, definendola in modo formale e descrivendo la sua storia, per illustrare in che modo essa si sia conservata

nel corso degli anni. Successivamente, viene mostrata la sua forma più moderna, ovvero il baratto online.

- Nel Capitolo 2 viene illustrato Android, il sistema operativo su cui viene eseguita la nostra applicazione. Inizialmente viene descritta la sua storia, ponendo l'accento sulla rapida diffusione di questo sistema e sulle controversie legali che lo hanno caratterizzato. In seguito, vengono descritti la struttura stessa di Android nonché l'ambiente di sviluppo utilizzato, ovvero Android Studio.
- Nel Capitolo 3 si entra nella fase progettuale, illustrando i vari requisiti soddisfatti dalla nostra applicazione e descrivendo le singole attività realizzabili. In seguito viene analizzata la struttura del database in cui sono immagazzinati i dati.
- Nel Capitolo 4 viene descritta in dettaglio tutta la fase implementativa, soffermandoci sulle modalità con cui sono state realizzate a livello di codice le singole funzionalità. Successivamente la descrizione viene corredata da un manuale utente, che guida il lettore all'uso dell'app.
- Nel Capitolo 5 si discute di quanto svolto, presentando le lezioni apprese e tutte le conoscenze che sono state acquisite. In seguito si revisiona in modo critico l'app, definendo alcuni spunti per dei miglioramenti futuri.
- Nel Capitolo 6 vengono tratte le conclusioni di tutto il lavoro svolto.

1

Il baratto

In questo capitolo verrà illustrata la pratica del baratto. Si partirà da una breve introduzione, alla quale seguirà la presentazione di un adeguato quadro storico. Infine verrà illustrato il modo in cui tale pratica si realizza al giorno d'oggi, garantendo al lettore una corretta contestualizzazione.

1.1 Concetto

Prima di entrare nel merito del baratto (Figura 1.1), è bene definirlo in modo formale. A tal proposito l'Enciclopedia Treccani alla voce "baratto" riporta la seguente definizione: "Scambio diretto di beni contro beni, senza uso della moneta".

Figura 1.1. Rappresentazione del baratto

Già a partire dalle prime civiltà, è nata molto rapidamente la necessità di praticare scambi, legata principalmente al desiderio o bisogno di ottenere un particolare oggetto, al punto che si è disposti a privarsi di un proprio bene, stimandone un valore pressochè equiparabile. Il baratto viene, dunque, comunemente considerato come la prima forma di scambio commerciale di beni, ognuno dei quali caratterizzato da un particolare valore. Quest'ultimo dipende da vari fattori, quali estetica, affettività, utilità, ma si esclude il valore monetario e, pertanto, si fa riferimento ad una valutazione quantitativa e qualitativa di quanto proposto.

Se viene raggiunto un accordo tra le parti coinvolte (domanda ed offerta) si può procedere con lo scambio, garantendo un vantaggio economico reciproco.

Com'è facile da prevedere, le principali difficoltà che si riscontrano nello scambio sono due:

- stabilire il più possibile in modo oggettivo il valore dei beni, senza il quale non si può raggiungere l'accordo;
- trovare un acquirente che desidera avere ciò che si è disposti a cedere, e, viceversa, che possa fornire in cambio quello di cui si ha bisogno.

Date tali difficoltà, fu naturale l' evoluzione del baratto in una nuova forma di scambio che prevede l'introduzione di una moneta, ottenibile tramite la vendita e utilizzabile all'acquisto. Il vantaggio risiede nel fatto che la moneta è una valuta accettata ovunque e da tutti, al punto che la si considera l'unità di misura del valore di un bene.

1.2 Storia del baratto

1.2.1 Tempi antichi

Fin dai tempi più antichi il commercio si basava sul baratto, ossia la possibilità di scambiare merci tra le persone. Tale sistema richiedeva un' abile capacità di negoziazione unita alla ben più rara capacità di definire un valore tra merci appartenenti a generi differenti. Chiaramente questa seconda capacità poteva venire meno, o essere superflua, quando le parti interessate appartenevano alla stessa classe economica, il che supponeva che i prodotti fossero generalmente simili. Un chiaro esempio di baratto ci viene offerto dai cartaginesi, popolo di commercianti per eccellenza che si munì tardi della moneta mantenendo il baratto a lungo.

Uso del bestiame

Una prima evoluzione di tale meccanismo ebbe luogo nell'area mediterranea, e consistette nell'utilizzo del bestiame come strumento di equiparazione.

Varie testimonianze storiche ricordano come venissero equiparati tra loro bestiame, schiavi ed altri oggetti. Un esempio letterario lo possiamo trovare nell'*Iliade* (XXIII, 703-705, 884-885) (Figura 1.2) dove in occasione di una gara di tiro con l'arco, che vedeva Achille partecipante, spiccava una schiava che, in quanto ottima lavoratrice, era valutata quattro buoi, o un grande tripode di bronzo del valore di dodici buoi, o, infine, una lancia che valeva un bue.

L'uso del bestiame come unità di misura aveva le sue controindicazioni. Del resto con l'aumentare degli spostamenti, la possibilità di avere con sè del bestiame si ridusse considerevolmente, e soprattutto si evidenziavano dei grossi limiti nel caso di commercio di piccola portata, dove venivano coinvolti oggetti dal valore modesto, difficilmente comparabili con degli animali. Sviluppandosi il commercio, parallelamente alle maggiori esigenze e ai maggiori spostamenti delle popolazioni, questo strumento iniziò ad apparire inadeguato e laborioso. Nelle società in evoluzione i governanti che chiedevano al popolo i loro tributi in peso, o sotto forma di prestazioni di manodopera, si posero il problema di intervenire nello scambio tra beni

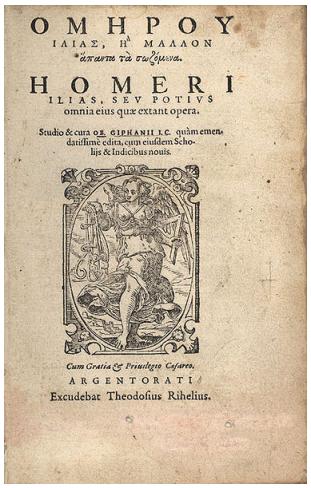

Figura 1.2. Copertina dell'Iliade, opera di Omero

contro beni e beni contro servizi, introducendo un equivalente che incorporasse un valore certo. Chiaramente la merce di riferimento era differente da popolo a popolo e da epoca ad epoca: a tal proposito si ricorda l'uso di sale, pelli di elefante, cacao, arachidi ed anche tabacco.

Avvento del metallo

Da un certo punto in poi, grazie anche all'avvento della metallurgia, con la nascita di officine attrezzate si diffuse rapidamente per il baratto l'utilizzo di oggetti in metallo, che vantava una deperibilità limitata (caratteristica fondamentale nello scambio di oggetti), un'alta mobilità e frazionabilità. Ritrovamenti archeologici confermano che, a partire dal III millennio a.C., nel vicino Oriente ed in Egitto, piccoli pezzi d'oro e d'argento furono utilizzati come valori di scambio ed elementi di contabilità economica.

Il passaggio dall' utilizzo del bestiame (o altre merci genericamente caratterizzate da deperibilità) al metallo non fu immediato e, per molto tempo, i due sistemi convissero.

I matematici sumeri, seguiti da quelli babilonesi, credevano che il pallido argento fosse sacro alla divinità lunare, mentre l'oro, con i suoi bagliori di fuoco, fosse sacro alla divinità solare; fu in questo periodo che cominciarono ad imporsi per il loro valore intrinseco questi due metalli preziosi, particolarmente apprezzati per l'inattaccabilità rispetto ai processi d'ossidazione, lucentezza e malleabilità.

A Babilonia troviamo il primo metallo considerabile come moneta, utilizzato come mezzo di pagamento, il cui valore era legato a quello dell'argento. Nella stessa regione videro la luce alcune istituzioni che possiamo paragonare alle moderne banche: i templi assunsero la funzione di depositi dove il popolo poteva portare l'eccedenza di prodotti e di metallo, con i sacerdoti contabili che aprivano un primitivo "conto corrente" e consegnavano delle tavole di terracotta (Figura 1.3), in cui veniva stabilita una quantità astratta di valore corrispondente alla merce depositata.

Successivamente, quando le persone volevano un altro tipo di prodotto depositato nel tempio, si seguiva il procedimento inverso.

E da lì all'attuale sistema economico il passo è davvero breve.

Figura 1.3. Tavola di terracotta, una prima forma di conto corrente

Parentesi medievale

Il baratto fu utilizzato diffusamente, oltre che nelle fasi primordiali della nostra civiltà, nella prima fase dell'Alto Medioevo (Figura 1.4), a causa di economie chiuse ad assetto agrario-feudale, con pochi scambi mercantili di tipo monetario. La moneta a circolazione locale veniva usata a quell'epoca ad integrazione del valore delle merci oggetto di baratto.

Figura 1.4. Il baratto nell'Alto Medioevo tornò ad essere una pratica comune

1.2.2 Tempi moderni

Una pratica antica come il baratto è sopravvissuta nel corso degli anni ed, in tempi di crisi come quelli che stiamo attraversando oggi, ci sono numerosi e validi motivi per far tornare in auge tale strumento; qualcuno lo sceglie per necessità economiche, chi per questioni ambientali e altri per semplice opposizione al consumismo. C'è, anche, chi preferisce ricorrere a tale pratica per allungare la vita dei propri prodotti o per interiorizzare una consapevolezza sempre maggiore sul valore dei beni in possesso. Pertanto la prima forma di scambio commerciale della storia è facilmente adattabile anche ai tempi moderni, seppur con una veste più moderna.

Un esempio è quello che hanno fatto le concessionarie di pubblicità, accettando in pagamento merci anziché denaro, dato che le aziende a cui si rivolgono spesso soffrono di mancanza di liquidità.

Tale pratica prende il nome di cambio merce pubblicitario, ed è riuscita ad ottimizzare la situazione critica nella quale si trovavano molte aziende che, potendo pagare con le propri merci le aziende, hanno la possibilità di beneficiare dei servizi pubblicitari senza dover effettivamente attingere ai fondi aziendali. A questo punto, però, la problematica si è spostata dal lato delle concessionarie di pubblicità, che si trovano a dover monetizzare quanto ricevuto. Ma per loro la strada è sicuramente in discesa, in quanto il valore economico delle merci sono ai minimi dei listini, per cui è sufficiente metterle sul mercato anche con un minimo rincaro, garantendo guadagno. Al fine di non creare una situazione di concorrenza sleale, i canali di vendita tradizionali non possono essere usati, poiché, a parità di merce, non ci sarebbe la possibilità di competere a livello di prezzi di vendita.

A tal fine sono nati gli "shopping club" (Figura 1.5), strumenti di vendita che possiamo considerare a tutti gli effetti dei negozi, sia sul territorio sia on line (per il baratto online si faccia riferimento alla Sezione 1.3), ma ai quali non può accedere chiunque, in quanto si deve sempre cercare di rendere tale canale di vendita accessibile a pochi utenti selezionati, sempre per non incorrere nella concorrenza sleale. Tale selezione si effettua sulla base del possesso di alcuni requisiti fondamentali, come, ad esempio, il possesso di una tessera, l'iscrizione ad un sito o l'aver ricevuto un invito.

Figura 1.5. Un esempio di shopping club

1.3 Baratto online

Il commercio elettronico via Internet (Figura 1.6), abbattendo i costi di ricerca per la facilità dei collegamenti e l'accesso all'informazione condivisa, ha permesso la nascita di reti di baratto on-line tra consumatori o imprese.

Una forma di baratto sempre più popolare sul web, è detto "swapping", un sistema di scambi molto informale, in cui gli utenti possono scambiarsi oggetti a seguito di un accordo sulla base dei valori degli oggetti proposti e su base fiduciaria.

Per quanto riguarda i beni scambiati si va dagli indumenti, ai CD musicali, ad ogni tipo di oggetto e, persino, ai gadget. Esistono anche "e-mail swap", nei quali, in

genere, le merci scambiate sono informazioni, opinioni o anche foto. Data la rapida diffusione di tali canali di vendita online, al loro supporto sono nate applicazioni ad hoc per barattare anche in modalità mobile.

Tali applicazioni hanno un funzionamento tipicamente molto basilare ed una struttura simile; esse presentano un catalogo, nel quale sono esposti tutti (o una parte) degli oggetti proposti da altri utenti; e quindi l'utente può decidere di contattare il possessore di un particolare oggetto al fine di accordarsi per un baratto, nel caso in cui si fosse in possesso di un oggetto di pari valore e che desta l'interesse del venditore.

Figura 1.6. Il baratto online

2

Android

In questa sezione verrà proposta un'introduzione ad Android, alla sua storia e alle motivazioni che hanno fatto ricadere lo sviluppo della nostra applicazione lungo i binari di questa piattaforma.

2.1 Introduzione ad Android

Android è un sistema operativo sviluppato da Google, nato principalmente per smartphone e tablet (Figura 3.1). Nel corso degli anni abbiamo visto questo sistema operativo evolversi, supportando anche altri dispositivi, tra i quali ricordiamo i seguenti:

- Android Wear (successivamente Wear OS) per dispositivi wearable, come gli orologi da polso;
- Android TV, per i televisori;
- Android Auto, specifico per l'infotainment delle automobili;
- Android Things, per dispositivi intelligenti e Internet delle cose.

Per ciascuna di queste varianti, Android presenta una interfaccia specializzata, che garantisce una esperienza utente quanto più efficiente possibile.

Figura 2.1. Icona di Android

2.1.1 Storia

Android è nato ufficialmente nel mese di Ottobre del 2003, quando Andy Rubin, Rich Miner, Nick Sears e Chris White fondarono una società, l'Android Inc. L'obiettivo iniziale era, come riportato da Rubin stesso, quello di sviluppare "dispositivi cellulari più consapevoli della posizione e delle preferenze del loro proprietario".

Purtroppo il budget si esaurì molto rapidamente, entro il primo anno di vita dell'azienda, al punto che i fondatori furono costretti a chiedere un finanziamento ad un amico stretto di Rubin, che, con una donazione di circa 10.000 dollari, aiutò Rubin e i suoi collaboratori a continuare il progetto. Nel 2005 Google acquisì per ben 50 milioni di dollari la società, in quanto vedeva in essa il mezzo con cui competere nel mondo della telefonia mobile, ambiente in cui essa voleva entrare a far parte.

La prima presentazione ufficiale del robottino verde risale solo al 2007; una presentazione che, tra l'altro, riscosse molte perplessità circa la bontà di tale progetto. Del resto gli analisti erano dubbiosi sulla possibilità, da parte dei produttori di telefoni cellulari, di sostituire il loro sistema operativo corrente con Android. Inoltre, ulteriori preoccupazioni nacquero di fronte ad una forte concorrenza da parte di aziende affermate nel mercato degli smartphone, come Nokia e Microsoft, ed altri sistemi operativi mobili Linux, che erano ancora in fase di sviluppo.

La Versione 1.0 del sistema operativo venne rilasciata solo nel 2008, e da quel momento il team di sviluppo lavorò molto intensamente, con 5 major release nel giro di 2 anni. Cosìmolti detrattori dovettero ricredersi di fronte all'espansione a macchia d'olio di questo nuovo sistema operativo, fino ai giorni odierni, dove la stragrande maggioranza dei dispositivi mobili è dotata Android.

Cause legali

Android, dato il suo enorme successo, si è visto coinvolto anche in varie questioni legali: numerose sono, infatti, le cause in materia di brevetti ed altre sfide legali. Nel 2010 Oracle fece causa a Google per una violazione dei diritti d'autore ed altri brevetti di Java, chiedendo danni pari a 6 miliardi di dollari. Nonostante il ribasso della stima dei danni, Oracle perse la causa, con il tribunale che difese Google, sostenendo che non c'era stata alcuna violazione dei diritti d'autore. Nel 2014 la sentenza venne ribaltata, con il Circuito Federale a favore di Oracle, e solo nel 2015 Google decise di passare ad OpenJDK, l'implementazione ufficiale open source della piattaforma Java, mostrando una prima forma di conciliazione. La storia si conclude l'anno successivo, con un ultimo tribunale federale che si pronunciò a favore di Google, sostenendo come corretta la sua interpretazione dell'utilizzo delle API.

Inoltre, nel corso degli anni, Google si è trovata ad affrontare altre cause anticoncorrenziali. Nel 2013 FairSearch citò in giudizio Android presso la Commissione Europea, sostentendo che la distribuzione gratuita del sistema operativo costituiva un prezzo decisamente anticoncorrenziale; le accuse, però furono contestate rapidamente. Solo 3 anni dopo, nel 2016, l'Unione Europea, basandosi sulle precedenti accuse di FairSearch, presentò un reclamo antitrust formale, nel quale si diceva che i prodotti Google ostacolavano la diffusione di altri motori di ricerca in quanto erano presenti quelli preinstallati nativamente da Google, e impedivano la produzione di device con all'interno delle fork di Android, e questa situazione era da definirsi anticoncorrenziale. Nell'agosto 2016, Google è stato multato di 6.75 milioni di dollari

dal servizio antimonopolio federale russo (FAS) a seguito di accuse simili. Ma la più celebre tra tutte queste questioni legali, sicuramente, è quella che vide fronteggiarsi Apple e Samsung, due titani tecnologici, con accuse a vicenda, in una guerra per i brevetti.

Nella prima metà del 2011 Apple accusa Samsung per violazione di brevetti (Figura 2.2), e nell'Agosto dello stesso anno la causa raggiunse dimensioni enormi, con 9 paesi coinvolti per un totale di 19 processi in corso, al punto che gli esecutivi delle due società furono costretti da un tribunale statunitense a cercare conciliazioni, per ridurre il numero di processi, ma senza successo.

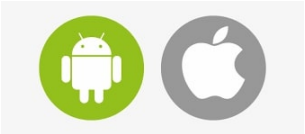

Figura 2.2. Android contro Apple nel 2011

Nel Luglio del 2012 un tribunale americano si pronuncia a favore della Mela, imponendo a Samsung un risarcimento di un miliardo di dollari, cifra che non fu sborsata immediatamente in quanto la società sudcoreana voleva prima vedere terminare tutti gli altri processi, vedendosi del resto vincitrice in Regno Unito, Corea del Sud e Giappone. Ma la storia continua nel 2014, con una nuova citazione in giudizio da parte di Apple contro Samsung, sostenendo una violazione di altri 5 brevetti. La difesa di Samsung si basò sulla presenza tra i propri sviluppatori di un team di Google che sviluppò in modo indipendente quelle funzionalità che, agli occhi di Apple, erano viste come dei furti.

Nonostante tutte queste controversie, il sistema operativo di Google ha continuato a crescere ed espandersi sempre più, ottenendo delle fette di mercato sempre maggiori.

2.1.2 Diffusione e frammentazione

Tra i principali motivi che ci hanno spinto ad utilizzare questa piattaforma per sviluppare l'applicazione mobile oggetto della presente tesi c'è, sicuramente, l'elevata diffusione di Android nel mercato mobile di oggi, con più di 2 miliardi di utenti attivi nel Maggio del 2017.

Ripercorrendo la storia, si parte dal 2009, con una quota inferiore al 3% delle spedizioni mondiali di smartphone. Nel giro di un anno Android raggiunse il 10% delle spedizioni a livello globale, e, solo negli Stati Uniti, il 28%, superando anche iPhone ed il suo sistema operativo.

Nel 2011 la metà del mercato globale apparteneva ad Android, raggiungendo l'anno seguente il 75%; Android diventava, in questo modo, il sistema operativo più utilizzato al mondo, con più di un miliardo di attivazioni da parte dei dispositivi mobili.

Nel 2015 Android riesce ad avere una fetta di mercato comparabile addirittura con Windows, il sistema operativo per PC Desktop più diffuso, con una base instal-

lata di 1.8 miliardi di unità, superandolo anche in popolarità per l'utilizzo totale di Internet.

Questa enorme diffusione a livello globale però, ha portato anche ad un fenomeno irreparabile: la frammentazione (Figura 2.3). Del resto, data l'enorme mole di dispositivi e di produttori, è stato impossibile per Google distribuire aggiornamenti per tutti i dispositivi in modo omogeneo, così come imporre ai produttori di mantenere aggiornati tutti i dispositivi. Pertanto, i dispositivi nel mondo sono muniti di diverse versioni dello stesso sistema operativo, non essendo in grado di aggiornarsi, sia per questioni hardware, sia per volontà dei produttori, che spingono all'acquisto di nuovi prodotti. Una situazione di questo tipo può portare a incompatibilità, mancanza di supporto alle funzionalità più recenti o, semplicemente, delle interfacce utente differenti che possono disorientare gli utilizzatori meno esperti. Degno di nota negli anni è stato il problema degli aggiornamenti di sicurezza, che ha esposto i dispositivi più datati a vulnerabilità durante la navigazione in rete, non potendo accedere agli ultimi aggiornamenti in termini di sicurezza ed autodifesa. Ciononostante, Google si è affaccendata e, nel corso degli anni, è riuscita a porre rimedio a questa annosa problematica [1].

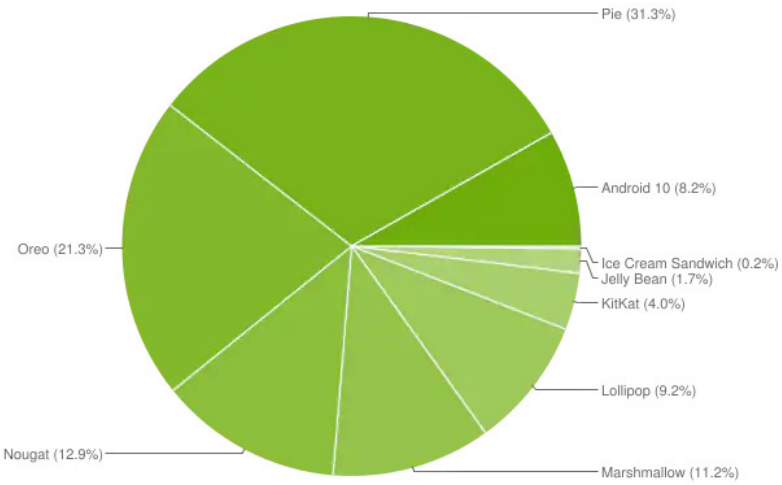

Figura 2.3. Fenomeno della frammentazione di Android negli anni

2.2 Sviluppo

In questa sezione entreremo nel merito di Android da un punto di vista dello sviluppatore, concentrandoci sulle caratteristiche tecniche e sulle componenti di tale sistema operativo. Inoltre presenteremo brevemente l'IDE che è stato utilizzato per dar luogo all'applicazione.

[1] Nello sviluppo della nostra applicazione abbiamo fatto riferimento alla Versione 6.0 di Android (API 23), che garantisce un buon equilibrio tra supporto e funzionalità offerte.

2.2.1 Architettura

La parte più visibile di Android per un utente qualunque comprende la sua interfaccia e le funzionalità che offre. Pertanto, è possibile ignorare tutto ciò che si frappone tra hardware e l'utente, cioè tutta la controparte software, nonché tutto il codice che c'è dietro la realizzazione di una particolare funzione. Ed è proprio qui che interviene lo sviluppatore, che realizza tutto il lato software del dispositivo e garantisce il buon funzionamento del sistema operativo.

Nel suo modo più semplice di operare, il sistema operativo si occupa di 3 cose:

- gestione dell'utilizzo dell'hardware da parte delle applicazioni che lo richiedono;
- gestione dell'esecuzione concorrente di più processi, garantendo il cosiddetto "multitasking";
- fornitura di servizi come la sicurezza, la gestione della memoria, il networking, etc.

Possiamo, a questo punto, introdurre una sommaria rappresentazione del sistema operativo, tramite il diagramma logico riportato in Figura 2.4.

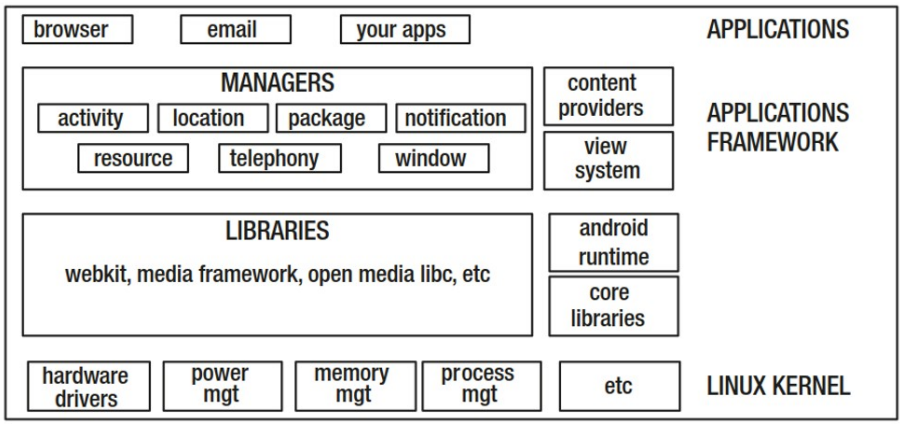

Figura 2.4. Struttura di Android

Nonostante non mostri tutte le componenti, tale diagramma può di certo aiutare nella comprensione di come Android sia strutturato secondo livelli e di quale sia lo scopo di ciascuno di essi.

Nello specifico, il livello più basso si occupa dell'interfaccia con l'hardware; quindi assegna la memoria ad ogni processo e garantisce a più processi di essere in esecuzione contemporaneamente.

Passando al livello successivo, troviamo le librerie di basso livello, come SQLite, OpenGl et similia. Accanto ad esse ci sono il runtime Android (per un approfondimento di tale componente si rimanda il lettore alla Sottosezione 2.2.2).

Al penultimo livello troviamo il framework delle applicazioni. Fondamentalmente questo può essere inteso come l'ambiente in cui vivono le applicazioni lato developer; esso, quindi, deve necessariamente comprendere le librerie sia di basso livello

sia del runtime di Android, fondamentali per il corretto funzionamento di ogni applicazione. Questo è il livello in cui gli sviluppatori devono intervenire durante la fase di implementazione di un'applicazione.

Infine l'ultimo livello è il livello delle applicazioni, in cui le applicazioni risiedono lato cliente, pronte per l'esecuzione.

Architettura di un'applicazione Android

Un'applicazione Android è formata da diverse componenti (Figura 2.5), scritte come classi Java precodificate. Tali componenti sono:

- Activity: corrisponde a ciò che viene visualizzato sullo schermo, quindi rappresenta tutti gli elementi visuali su cui si ha il focus dell'utente. Si può considerare come la controparte Android delle finestre del PC Desktop. Inoltre risponde agli eventi innescati dall'utente.
- Service: ci serviamo di tale componente quando vogliamo che un'applicazione continui a funzionare anche quando ha perso il focus dell'utente (come, ad esempio, in seguito alla sua scomparsa dalla schermata principale). Esso rappresenta un'applicazione a cui non si associa necessariamente un'interfaccia utente, ma è relativa a qualcosa di eseguibile per un lungo periodo di tempo, come un music player o il GPS.
- Broadcast Receivers: è utilizzato per gestire quegli eventi generati dallo stato del telefono, come la batteria scarica o la perdita della connessione dati, oppure per inserire nella propria applicazione una qualunque logica, intesa come risposta ad eventi, quali il click o lo swipe.
- Content Providers: rappresentano componenti la cui funzione è quella di trasmettere dati da un'applicazione ad un'altra. Una componente di questo tipo è di fondamentale importanza poiché ogni applicazione viene eseguita su una macchina virtuale a se stante (meccanismo realizzato per proteggere le applicazioni da virus o malintenzionati) e che, quindi, risulta essere isolata da tutte le altre.
- Sync Adapters: da utilizzare per sincronizzare i dati con i servizi cloud.

Può essere d'aiuto nella comprensione di questo concetto immaginarsi di costruire un'applicazione Android come se si stesse costruendo una casa partendo da pezzi prefabbricati (che sarebbero le singole componenti dell'applicazione), anziché usando i mattoni: il risultato è lo stesso, ma viene raggiunto in modo sicuramente più rapido e agile. Tutte queste componenti vengono richiamate dall'Intent, un altro oggetto che chiede in input anche la tipologia di azione che si vuole eseguire.

2.2.2 Java e macchina virtuale

Il linguaggio di programmazione utilizzato per lo sviluppo di applicativi Android è Java, uno tra i linguaggi ad alto livello più conosciuti ed utilizzati. Orientato agli oggetti, tale linguaggio è stato pensato per essere il più possibile indipendente dall'hardware in cui esso viene fatto eseguire, con il conseguente difetto di non essere, dal punto di vista computazionale, prestante come alcuni dei suoi concorrenti.

Gli eseguibili Android, a differenza di come si potrebbe pensare, non sono file `.class`, ma `.dex`, un tipo di file che non può essere mandato in esecuzione su una

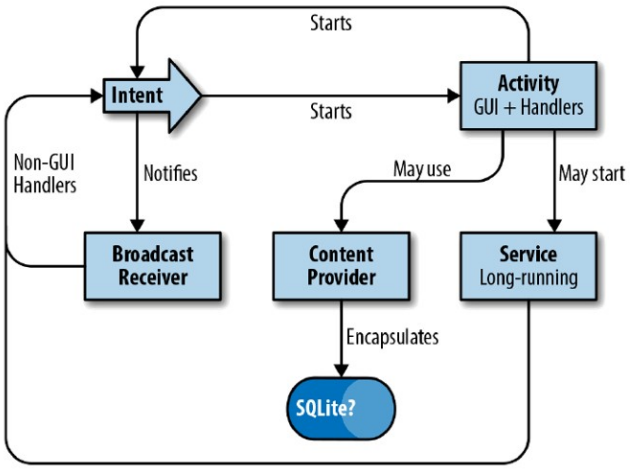

Figura 2.5. Struttura di un'applicazione Android

macchina virtuale come quella Java (JVM), cosa a cui siamo tipicamente abituati su un desktop, ma ha bisogno di una macchina virtuale personalizzata ed ottimizzata specificatamente per l'esecuzione su dispositivi a bassa potenza.

Un'applicazione Android, prima di poter essere mandata in esecuzione, subisce un processo di compilazione a tre passi (Figura 2.6): il codice sorgente è tipicamente un file Java, che poi viene compilato dal compilatore di Java, diventando un file `.class`. Successivamente il compilatore dex dà vita al file `.dex`, che subirà l'ultima trasformazione con il processo di "packaging" che genera il file `.apk`, che è tipicamente quello che viene installato nei dispositivi e mandato in esecuzione.

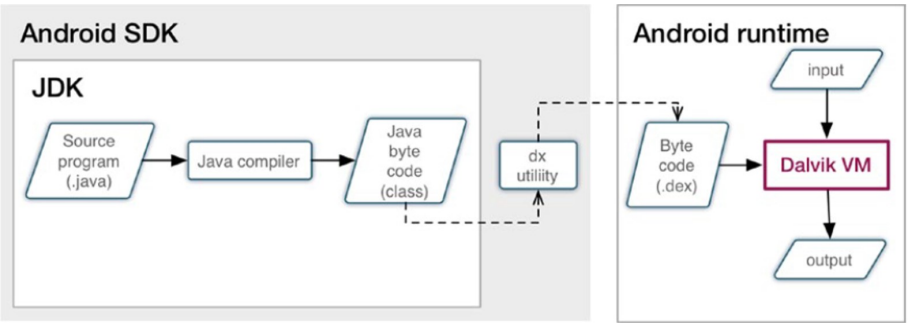

Figura 2.6. Compilazione di un'applicazione Android

Dalvik

Nella struttura dell'applicazione Android (Sezione 2.2.1), abbiamo evidenziato la presenza di un livello specifico per il runtime Android. Al suo interno troviamo le

librerie di classe Android e la macchina virtuale Dalvik. Tale macchina, sviluppata da Dan Bornstein, deve il nome ad un omonimo villaggio islandese. Il suo compito è quello di mandare in esecuzione gli applicativi Android, attività che inficia pesantemente sulle prestazioni di tutto il dispositivo. Il file eseguibile, ovvero un file `.dex`, permette di ottimizzare la gestione della memoria e del multithread, di fondamentale importanza in un dispositivo mobile.

Dalvik presenta, inoltre, una garbage collection migliorata ed ottimizzata per evitare spreco di risorse e, a partire dalla Versione 2.2, anche un compilatore JIT (Just In Time), che, come suggerisce il nome, effettua la fase di compilazione durante l'esecuzione e non prima, per avvicinare il più possibile le prestazioni a quelle di un linguaggio macchina.

2.2.3 Android Studio

Terminiamo il Capitolo 2 relativo ad Android presentando l'IDE (ambiente di sviluppo integrato) di riferimento, ovvero Android Studio (Figura 2.7).

Figura 2.7. Icona di Android Studio

Inizialmente, quando Android si presentava al pubblico ancora nella sua primitiva Versione 1.0, il kit di sviluppo messo a disposizione da Google consisteva in una serie di script predefiniti e in alcuni strumenti a riga di comando.

In una situazione di questo tipo, furono molti gli sviluppatori che si trovarono in difficoltà nella realizzazione delle proprie applicazioni, in quanto generalmente poco avvezzi all'utilizzo delle righe di comando, e ben più disposti a servirsi di un IDE che potesse supportarli in ogni fase, dall'evidenziazione degli errori nelle singole righe alle attività di debugging; quindi molti desistirono dallo sviluppo mobile.

Solo nel 2008 ad Android si associò il primo IDE, ovvero Eclipse, una scelta abbastanza scontata se si pensa che anche oggi è l'ambiente di sviluppo preferito per Java.

Nel 2013 Google rilascia Android Studio, basato su Intellij di Jetbrain. Esso ben presto divenne (a seguito di una fase di beta testing) il tool di sviluppo predefinito per chiunque volesse approcciarsi allo sviluppo per questo sistema operativo.

Tale ambiente di sviluppo risulta essere estremamente efficiente e stimolante, in quanto supporta lo sviluppatore indicando in tempo reale in modo chiaro ed

esplicito quali sono gli errori a livello di codice e, nel caso di errori fatali, genera un messaggio di errore dal quale è possibile il più delle volte risalire alla causa dell'interruzione dell'esecuzione dell'applicazione. La presenza di numerosissime librerie online, adeguatamente documentate, risulta essere un fattore determinante per l'implementazione di funzionalità di ogni tipo. Inoltre, essendo usato dalla stragrande maggioranza degli sviluppatori mobile, in rete si trova ogni informazione di cui si ha bisogno.

Infine, anche lo sviluppo delle varie interfacce utente è molto semplice, in quanto esso dà allo sviluppatore la possibilità di scrivere manualmente codice XML (Modalità "Text"), o di servirsi della componente grafica (Modalità "Design"), che permette, tramite il trascinamento di ciò che si vuole inserire, di costruire le interfacce in modo diretto, oppure manipolando il file di layout in entrambi i modi, per avere la massima libertà.

Nella Figura 2.8 riportiamo una schermata di questo IDE.

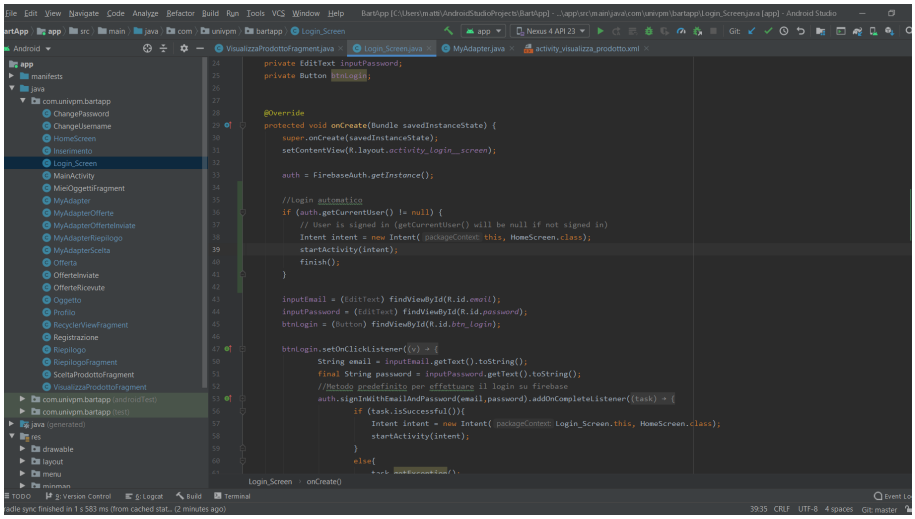

Figura 2.8. Una tipica schermata di Android Studio

3

Analisi dei requisiti e progettazione

In questo capitolo vengono descritti i requisiti funzionali e non, che dovranno caratterizzare l'applicazione. Successivamente verrà illustrata la sua progettazione.

3.1 Definizione dei requisiti

All'insieme dei requisiti funzionali si ascrivono tutte le caratteristiche di tipo implementativo della nostra applicazione. Invece, i requisiti non funzionali definiscono vincoli di tipo tecnico e realizzativo da far rispettare necessariamente.

La fase di analisi dei requisiti è fondamentale per la buona riuscita di un software, e ne precede il vero e proprio sviluppo. Di seguito verrà presentata l'analisi dei requisiti funzionali minimi di tale applicazione così com'è stata inizialmente pensata, e solo successivamente implementata, intesi come quelle funzionalità che il software dovrà fornire ai clienti. Ciò che deriva da un'analisi di questo tipo è la specifica dei requisiti, un documento che funge da guida per tutta la fase di progettazione ed implementazione del prodotto.

Ovviamente ci riserviamo la possibilità di apportare migliorie ed inserire nuove funzionalità nel caso in cui l'applicazione dovesse essere oggetto di un futuro rilascio in Google Play, o anche nel caso in cui la loro implementazione possa consentire una migliore rivendibilità del nostro prodotto.

Ogni requisito sarà descritto in una tabella che riporta le seguenti voci:

- *Nome*: il nome del requisito.
- *ID*: un codice identificativo univoco che permette una corretta distinzione da tutti gli altri.
- *Definizione*: una breve descrizione che esplicita quanto deve realizzare l'obiettivo.
- *Motivazione*: quali sono gli incentivi che spingono a realizzare il requisito.

Al fine di garantire ordine ed una maggiore leggibilità, si farà riferimento, per ciascun requisito, ad una tabella organizzata secondo il template mostrato nella Tabella 3.1.

Nome	
ID	
Definizione	
Motivazione	
Attore	

Tabella 3.1. Template della tabella dei requisiti

3.1.1 Descrizione del progetto

Come anticipato nel Capitolo 1, per baratto si intende formalmente uno "Scambio diretto di beni contro beni, senza uso della moneta". Sarà proprio questo l'obiettivo dell'applicazione oggetto della presente tesi, ovvero permettere uno scambio di oggetti tramite un accordo tra i rispettivi proprietari.

Pertanto l'applicazione Bartapp si rivolge a chiunque abbia intenzione di barattare un proprio oggetto, indipendentemente da quale esso sia. In particolar modo, si possono definire come clienti della nostra applicazione sia dei privati sia negozi che vogliano una sezione online dedicata allo scambio di prodotti.

Detto ciò, una applicazione di questo tipo deve permettere di realizzare i baratti nel modo più snello ed intuitivo possibile, ed è proprio questo l'obiettivo principale di Bartapp: essere veloce, leggero e "user-friendly".

3.1.2 Definizione dei requisiti funzionali

I principali requisiti funzionali e non, che vorremmo garantire nella nostra applicazione, si trovano in fondo al capitolo, riportati nelle tabelle 3.2-3.17.

Nome	Registrazione
ID	RF01
Definizione	L'applicazione permette di effettuare le operazioni standard di registrazione di un nuovo utente, ed il conseguente login o logout
Motivazione	Permette all'utente di accedere al catalogo degli oggetti o, nel caso di logout, di tornare alla schermata iniziale per poter accedervi con un altro account
Attore	Utente

Tabella 3.2. Descrizione del requisito funzionale RF01

Nome	Visualizzazione Catalogo
ID	RF02
Definizione	L'applicazione visualizza sullo schermo tutti gli oggetti correntemente in vendita, con aggiornamento in tempo reale
Motivazione	Permette all'utente di accedere ai dettagli di ogni singolo prodotto, nel caso in cui si fosse interessati allo scambio
Attore	Sistema

Tabella 3.3. Descrizione del requisito funzionale RF02

Nome	Filtro
ID	RF03
Definizione	Tramite la lente di ingrandimento o la lista delle categorie si permette il filtro del catalogo, per nome o per categorie
Motivazione	Funzionalità resa disponibile per quegli utenti che sono alla ricerca solo di un prodotto particolare, o appartenente ad una categoria predefinita
Attore	Sistema

Tabella 3.4. Descrizione del requisito funzionale RF03

Nome	Scambio
ID	RF04
Definizione	L'utente, una volta visualizzati i dettagli di un prodotto, può effettuare una richiesta di scambio
Motivazione	Dalla pagina di dettaglio del prodotto, si avvia la procedura di formulazione dell'offerta scegliendo dai propri oggetti e proponendo all'altro utente questa richiesta di scambio
Attore	Utente

Tabella 3.5. Descrizione del requisito funzionale RF04

Nome	Inserimento oggetto
ID	RF05
Definizione	L'utente, per effettuare gli scambi, deve prima inserire nel catalogo un oggetto in possesso, da usare come pedina di scambio
Motivazione	Permette di inoltrare offerte di baratto, offrendo i propri prodotti
Attore	Utente

Tabella 3.6. Descrizione del requisito funzionale RF05

Nome	Ritiro Prodotto
ID	RF06
Definizione	Nel caso in cui si volesse ritirare un prodotto dal catalogo, è disponibile tale funzionalità, tramite la quale verranno ritirate tutte le offerte inviate che prevedevano quel prodotto e verrà rifiutata qualunque proposta che lo vedeva coinvolto
Motivazione	Permette semplicemente di eliminare un proprio prodotto tra quelli disponibili
Attore	Utente

Tabella 3.7. Descrizione del requisito funzionale RF06

Nome	Pagina Profilo
ID	RF07
Definizione	Dal menù laterale è possibile accedere all'area riservata dell'utente, nella quale sono riepilogati i dettagli, come l'immagine del profilo, username e la mail
Motivazione	Entrando in questa pagina si possono modificare alcuni dati, come, ad esempio, la password, che dovrà ovviamente rispettare i requisiti
Attore	Sistema

Tabella 3.8. Descrizione del requisito funzionale RF07

Nome	Miei Oggetti
ID	RF08
Definizione	Dal menù laterale si può entrare nella pagina relativa alla lista dei propri oggetti, inseriti precedentemente in vendita tramite l'apposita funzionalità
Motivazione	Permette di visualizzare quali oggetti sono stati scelti per gli scambi, e visualizzare i dettagli di ciascuno di essi, con la possibilità di eliminarli
Attore	Sistema

Tabella 3.9. Descrizione del requisito funzionale RF08

Nome	Lista Offerte
ID	RF09
Definizione	L'applicazione permette la visualizzazione di tutte le offerte che sono state inviate precedentemente ad altri utenti, nonché quelle ricevute
Motivazione	Permette di avere un riepilogo delle offerte in corso; per ognuna di essa si può accettare/rifiutare (se è stata ricevuta) o ritirare (viceversa se inviata)
Attore	Sistema

Tabella 3.10. Descrizione del requisito funzionale RF09

Nome	Accettare o rifiutare offerta
ID	RF10
Definizione	Dalla lista delle offerte è possibile proseguire accettando o rifiutando un'offerta presente nella lista
Motivazione	Permette di giungere alla fase finale del baratto, consistente nel contatto via posta elettronica con l'utente
Attore	Utente

Tabella 3.11. Descrizione del requisito funzionale RF10

Nome	Eliminare offerte inviate
ID	RF11
Definizione	Dalla lista delle offerte, è possibile annullare una offerta che è stata precedentemente inviata
Motivazione	L'applicazione dà la possibilità di tornare indietro, ritirando una offerta, nel caso in cui fosse stata inviata per sbaglio, o nel caso in cui non si fosse più interessati a quel particolare baratto
Attore	Utente

Tabella 3.12. Descrizione del requisito funzionale RF11

Nome	Contatto
ID	RF12
Definizione	L'applicazione, una volta che un'offerta è stata accettata, dà la possibilità di entrare in contatto con il venditore, mostrando il suo indirizzo di posta elettronica, dalla pagina di "Riepilogo Scambi"
Motivazione	Questa è la funzionalità fulcro del software, in quanto permette, in modo informale, di far raggiungere un accordo tra i due utenti, e quindi di realizzare definitivamente il baratto
Attore	Utente

Tabella 3.13. Descrizione del requisito funzionale RF12

Nome	Cronologia
ID	RF13
Definizione	Dal menù laterale è presente anche una voce, "Riepilogo Scambi", che mostra una cronologia delle offerte accettate, sia che esse siano state inviate sia che queste siano state ricevute
Motivazione	Permette di avere una cronologia completa, e di cancellare dalla lista alcuni elementi nel caso in cui lo scambio fosse stato ultimato, o se non si volesse più mostrare un particolare elemento
Attore	Sistema

Tabella 3.14. Descrizione del requisito funzionale RF13

3.1.3 Definizione dei requisiti non funzionali

I requisiti non funzionali, che ripetiamo essere i vincoli cui l'applicazione deve conformarsi nell'eseguire le operazioni, sono riportati nelle tabelle 3.15-3.17.

Nome	Password
ID	RNF01
Definizione	La password che un utente può scegliere in fase di registrazione deve essere composta da almeno 6 caratteri, pena il reinserimento
Motivazione	Permette di avere una maggiore sicurezza rispetto ad un'applicazione che non presenta alcun sistema di prevenzione
Attore	Sistema

Tabella 3.15. Descrizione del requisito non funzionale RNF01

Nome	Integrità Prodotto
ID	RNF02
Definizione	Il sistema permette di inserire nel catalogo un proprio prodotto solo dopo che l'utente ha compilato tutti i campi richiesti, tra cui l'immagine
Motivazione	Permette di avere nel catalogo tutti oggetti caratterizzati da un nome, una descrizione, un prezzo ed una immagine descrittiva
Attore	Sistema

Tabella 3.16. Descrizione del requisito non funzionale RNF02

3.2 Mappa del sito

In questa sezione si illustra la mappa del sito (Figura 3.1), intesa come i percorsi che possono essere seguiti dall'utente, a partire da ogni schermata, attraverso il click in ciascuna di esse, per realizzare tutte le funzionalità offerte dal software, o per orientarsi.

Un esempio molto semplice per capire il funzionamento di una mappa di questo tipo può essere il seguente: si parte dalla schermata di "Login", dalla quale, una volta effettuato l'accesso, si può giungere alla "Home", con il catalogo dei prodotti. A questo punto, ad esempio, si può andare verso "Riepilogo Offerta", dove troviamo la cronologia delle offerte accettate, sia dall'account corrente, sia dagli altri utenti a cui sono state inviate delle offerte. Ora si può arrivare all'ultima schermata, che, come evidenziato dalla mappa del sito, è indicata con "Contatta", e consiste nel-

Nome	Mio Oggetto
ID	RNF03
Definizione	Nel caso in cui dal catalogo si selezionasse un proprio prodotto, il tasto relativo alla formulazione di una offerta viene sostituito da un tasto di eliminazione del prodotto, tipico della pagina relativa ai propri prodotti
Motivazione	Un requisito di questo tipo è fondamentale per evitare dei bug scomodi che possano permettere di effettuare degli scambi con se stessi
Attore	Sistema

Tabella 3.17. Descrizione del requisito non funzionale RNF03

l'applicazione di posta elettronica con cui inviare una mail di accordo con l'altro utente.

3.3 Diagramma dei casi d'uso

In questa sezione studieremo nel dettaglio i casi d'uso della nostra applicazione, intesi come l'insieme delle azioni che possono essere svolte dall'utilizzatore del software. Partendo da quanto si è detto nella precedente analisi dei requisiti, e tenendo in considerazione anche la mappa del sito, è nostra premura fornire un diagramma che delinei la struttura dal punto di vista logico del prodotto, tramite il flusso che l'utente dovrà seguire nell'espletamento delle sue funzioni e le informazioni necessarie all'esecuzione. Il nostro intento è definire il modello di comportamento del sistema quando un utente interagisce con esso. Dunque un diagramma, come quello in Figura 3.2, ci aiuta a descrivere in maniera chiara e coesa le funzionalità dell'utente, e mostra come i vari attori ed il sistema interagiscono tra di loro per ogni sequenza delle azioni.

A tale introduzione seguirà una lista dei casi d'uso, per ciascuno dei quali verrà definita una descrizione dettagliata, che ricalcherà le orme del template di Figura 3.3, e le interazioni tra gli attori coinvolti in un diagramma UML minimale.

- CU1: Registrazione di un nuovo utente;
- CU2: Login;
- CU3: Visualizzazione dei dettagli del proprio profilo;
- CU4: Visualizzazione dei dettagli di un prodotto;
- CU5: Formulazione di un'offerta;
- CU6: Visualizzazione della lista dei miei oggetti;
- CU7: Inserimento di un prodotto;
- CU8: Ritiro di un'offerta precedentemente inviata;

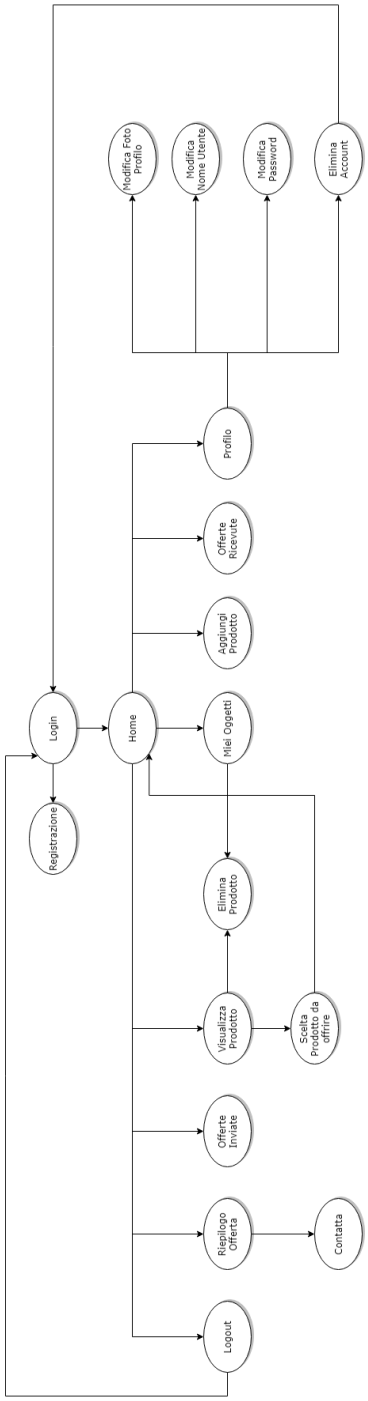

Figura 3.1. Mappa del sito di BartApp

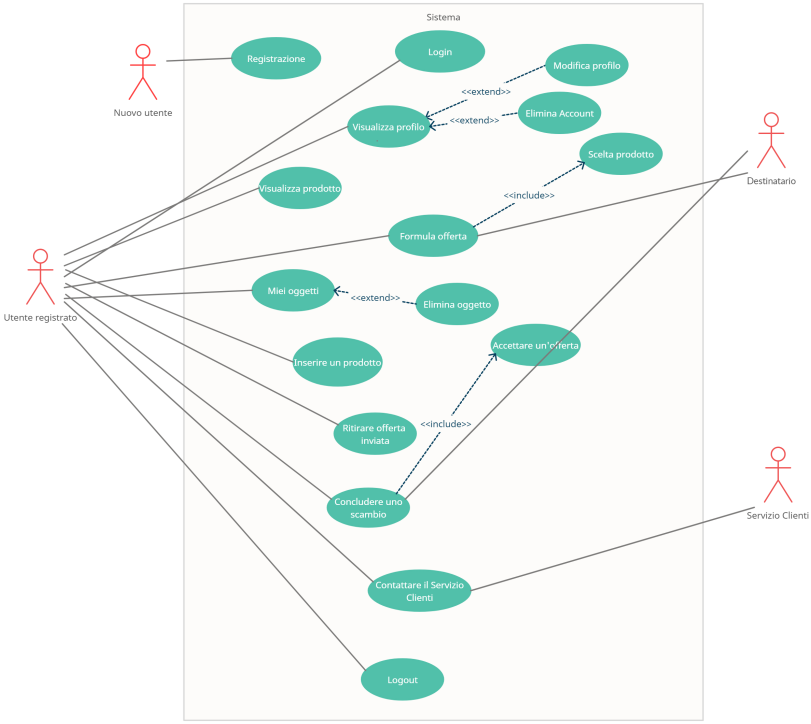

Figura 3.2. Diagramma dei casi d'uso

- CU9: Conclusione di uno scambio;
- CU10: Contatto del Servizio Clienti;
- CU11: Logout

3.3.1 Descrizione delle attività

A questo punto si passeranno in rassegna tutti i casi d'uso elencati nella precedente sezione, e si procederà nella descrizione di ciascun elemento.

CU1

Nel caso in cui l'utente non abbia un account associato all'applicazione, potrà cliccare sull'ancora della registrazione, che gli permetterà di registrarsi dopo aver inserito Nome, Cognome, Email e password, quest'ultima costituita da almeno 6 caratteri (Figura 3.4).

CU2

All'avvio, l'utente si troverà di fronte ad una schermata di login. Se ha già effettuato la registrazione, inserendo i campi email e password corretti, gli sarà permesso l'accesso al catalogo, altrimenti verrà avvisato dell'errore, ed invitato a ripopolare i campi fino a che non saranno corretti (Figura 3.5).

Figura 3.3. Template per la descrizione dei casi d'uso

CU3

L'utente, tramite il click in "Profilo", avrà accesso alle informazioni del proprio profilo con il quale si è collegato; a seguito di ciò saranno resi visibili l'immagine (modificabile cliccandoci sopra), il nome utente (modificabile cliccando sulla matita) ed, infine, l'email. Inoltre egli avrà la possibilità di modificare la password, anche qui con un click sull'immagine della matita. In fondo a questa lista si presenta un pulsante bianco con scritta rossa, che riporta "Elimina Account", e che, al click, eliminerà l'account (Figura 3.6).

CU4

Dopo il login, l'utente si troverà di fronte ad un catalogo di oggetti correntemente in vendita, dei quali si riportano il nome, il venditore, il prezzo virtuale e la posizione ad esso associati. Interagendo con una pressione sugli elementi della lista egli potrà visualizzare i dettagli del prodotto. La schermata che si troverà di fronte presenterà tutte le informazioni necessarie: oltre a quelle precedentemente riportate, si possono ritrovare la caratterizzazione del prodotto con una descrizione dettagliata, l'immagine del prodotto più grande e l'opzione di proposta di un'offerta, nel caso in cui si fosse interessati (Figura 3.7).

CU5

Una volta che l'utente si trova di fronte alla pagina dei dettagli del prodotto, e nel caso in cui fosse interessato, può formulare una offerta di scambio attingendo alla lista di oggetti che ha inserito precedentemente in vendita nella vetrina per il baratto. Una volta inviata l'offerta, che sarà possibile rivedere o annullare nella sezione "Offerte Inviate", quest'ultima sarà inviata all'altro utente, visibile nel suo menù "Offerte Ricevute", con le opzioni per rifiutare o accettare l'offerta proposta (Figura 3.8).

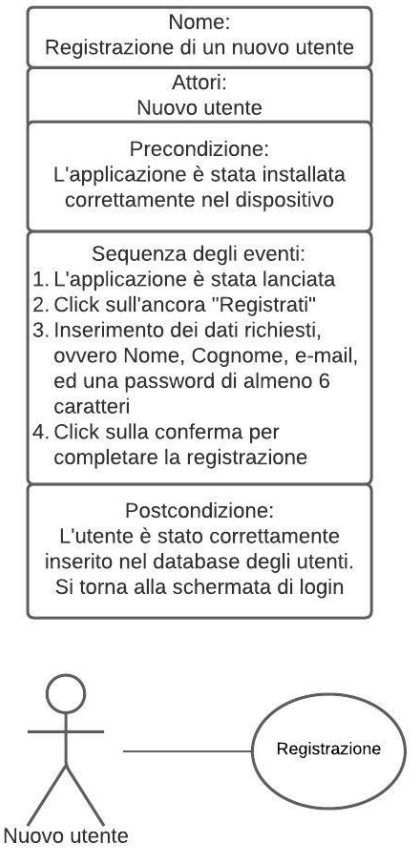

Figura 3.4. Descrizione del caso d'uso CU1 (Registrazione) e diagramma UML associato

CU6

Come anticipato, l'utente potrà inserire prodotti nel catalogo, con una descrizione che deve essere quanto più esauriente possibile, una foto (che sarà presa dalla galleria solo dopo che l'utente ha dato il permesso all'applicazione di accedere alle foto) ed un prezzo di vendita (Figura 3.9). Inoltre è bene riportare che l'applicazione è stata implementata in modo tale che all'oggetto appena inserito si associ una posizione, presa dalla posizione corrente del dispositivo (acquisibile solo dopo che l'utente ha dato il permesso), così che nel catalogo ci sia anche questo dato aggiuntivo che può essere sicuramente determinante, sia in positivo che in negativo, per l'acquisto del prodotto da parte di alcuni utenti.

L'utilizzatore dell'applicazione viene avvisato sullo schermo anche del successo dell'operazione di inserimento.

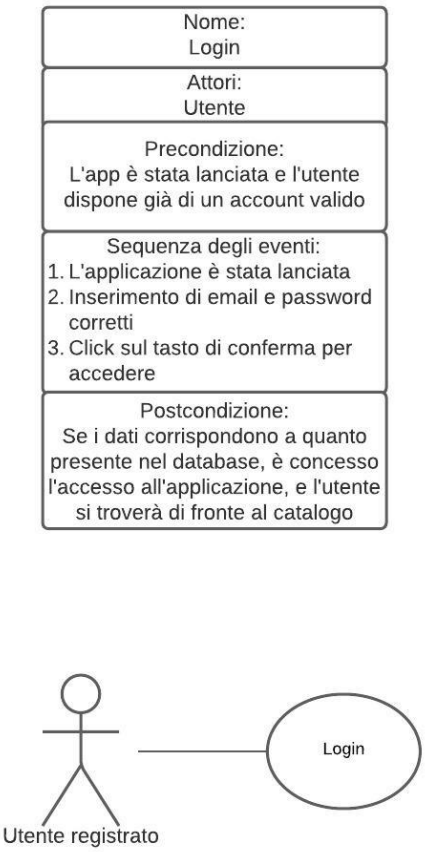

Figura 3.5. Descrizione del caso d'uso CU2 (Login) e diagramma UML associato

CU7

L'utente può visualizzare tutti i suoi oggetti attualmente in vendita in "Miei Oggetti", accessibile tramite menù laterale. Nella pagina che si apre sarà riportato un ulteriore catalogo, nel quale è stato applicato un filtro in modo tale che l'utente possa vedere solo gli oggetti in proprio possesso. Come nel catalogo principale, premendo sulla riga associata ad un elemento, si aprirà la pagina dei dettagli del prodotto, con la possibilità, tramite la pressione sul tasto relativo, di eliminare il prodotto dal mercato (Figura 3.10).

CU8

Dal menù laterale si può aprire la pagina delle offerte inviate. Per ciascuna di esse si offrono i dettagli, sia del prodotto richiesto che di quello offerto in scambio. È , inoltre, presente un tasto relativo all'eliminazione dell'offerta che, se premuto,

Nome: Visualizzare i dettagli del proprio profilo
Attori: Utente
Precondizione: È stata completata la procedura di Login
Sequenza degli eventi: 1. Tramite uno swipe verso destra o premendo sulle 3 linee orizzontali in angolo in altro a sinistra dello schermo, si accede al menù laterale 2. Click su "Profilo" per accedere all'area riservata 3. Click sulla matita per modificare il nome utente o la password 4. Click sull'immagine del profilo per selezionarne una nuova 5. Pressione del tasto di conferma per rendere effettivi i cambiamenti apportati 6. Se si volesse procedere con l'eliminazione del proprio account, si preme sul tasto rosso che riporta la scritta "Elimina Account"
Postcondizione: Se la modifica è effettuata con successo, l'utente verrà avvisato a schermo. Se l'utente ha eliminato il proprio account, verrà reindirizzato alla schermata iniziale del login.

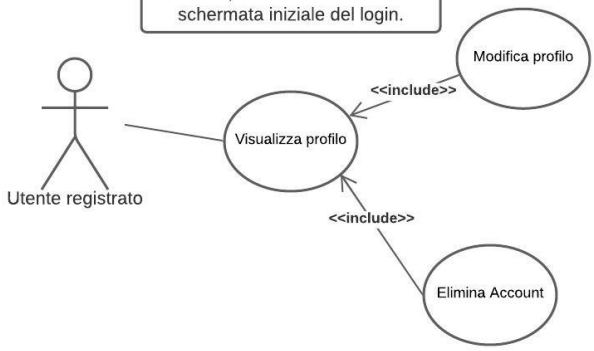

Figura 3.6. Descrizione del caso d'uso CU3 (Visualizzare i dettagli del proprio profilo) e diagramma UML associato

permette il ritiro ufficiale della proposta, con conseguente cancellazione del suo elemento dal database e scomparsa dalla lista[1].

[1] L'operazione elimina l'offerta anche per chi la stava ricevendo, non permettendogli di accettarla o rifiutarla

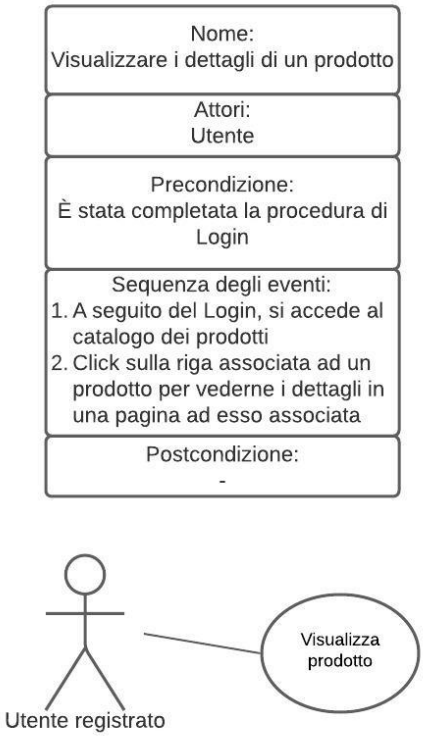

Figura 3.7. Descrizione del caso d'uso CU4 (Visualizzare i dettagli di un prodotto) e diagramma UML associato

CU9

Quando un utente riceve una offerta di scambio, la si può trovare nella sezione "Offerte ricevute". Qui si troveranno tutti i dettagli, e ci sarà un pulsante per decidere se accettare o rifiutare l'offerta.

Nel caso d'uso in questione si suppone che l'utente abbia giudicato come soddisfacente l'offerta propostagli; pertanto procederà con l'accettazione (Figura 3.12).

Ogni volta che viene accettata una offerta, un suo riepilogo verrà inserito nella sezione "Riepilogo Offerte"; in essa l'utente vedrà la cronologia delle offerte che ha inviato e che sono state accettate. A questo punto potrà procedere contattando il venditore, di cui si allega l'indirizzo di posta elettronica e, successivamente, eliminando il riepilogo, nel caso in cui voglia fare una pulizia della cronologia.

CU10

Specialmente considerando il fatto che l'applicazione oggetto della presente tesi non è stata pubblicata in alcuno store online, è stato giudicato come doveroso dare la

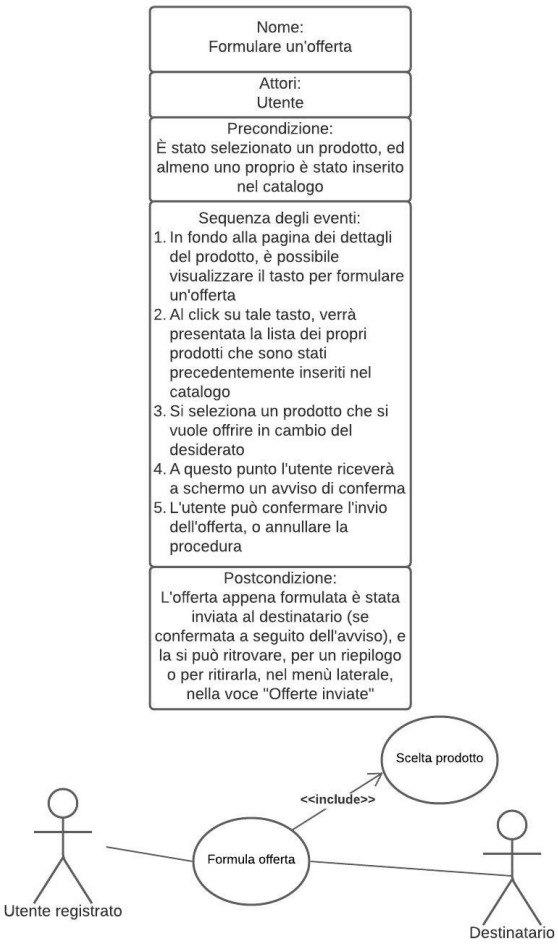

Figura 3.8. Descrizione del caso d'uso CU5 (Formulare un'offerta) e diagramma UML associato

possibilità agli utilizzatori di questo software di contattare gli sviluppatori, nel caso in cui volessero inviare un messaggio al team di sviluppo (Figura 3.13).

Per coerenza con quanto fatto in precedenza, si riporta un caso d'uso adatto nel momento in cui l'applicazione sarà distribuita, vale a dire un contesto dove, al posto degli sviluppatori, ci sarà un Servizio Clienti, immaginato come uno o più addetti con il compito di ricevere e gestire email inviate dai clienti, che ragionevolmente potranno essere lamentele, critiche (costruttive e non) o, semplicemente, particolari domande.

CU11

Inoltre riportiamo la presenza dell'ancora "Logout", che permette all'utente di scollegarsi ed essere riportato nella schermata di login, nel caso in cui avesse bisogno di

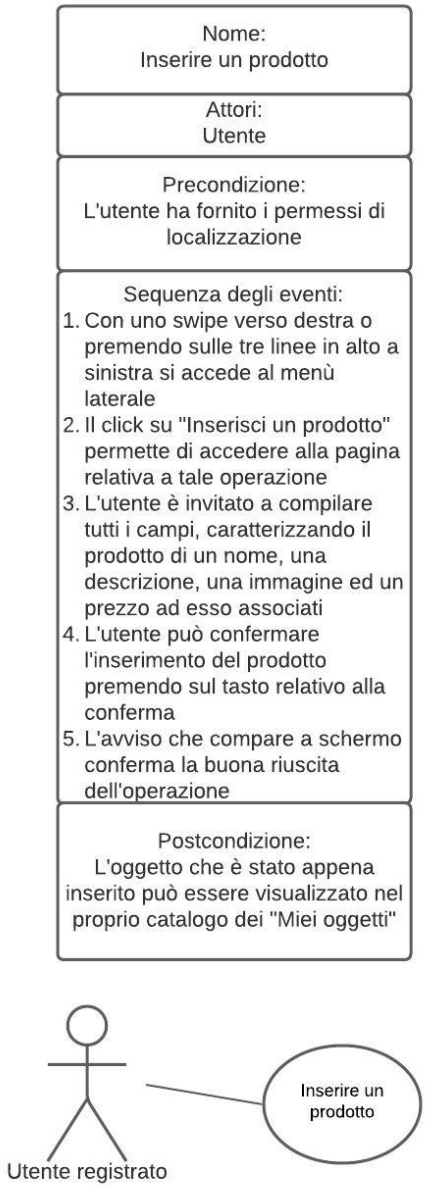

Figura 3.9. Descrizione del caso d'uso CU6 (Inserire un prodotto) e diagramma UML associato

effettuare l'accesso con un account differente (Figura 3.14).

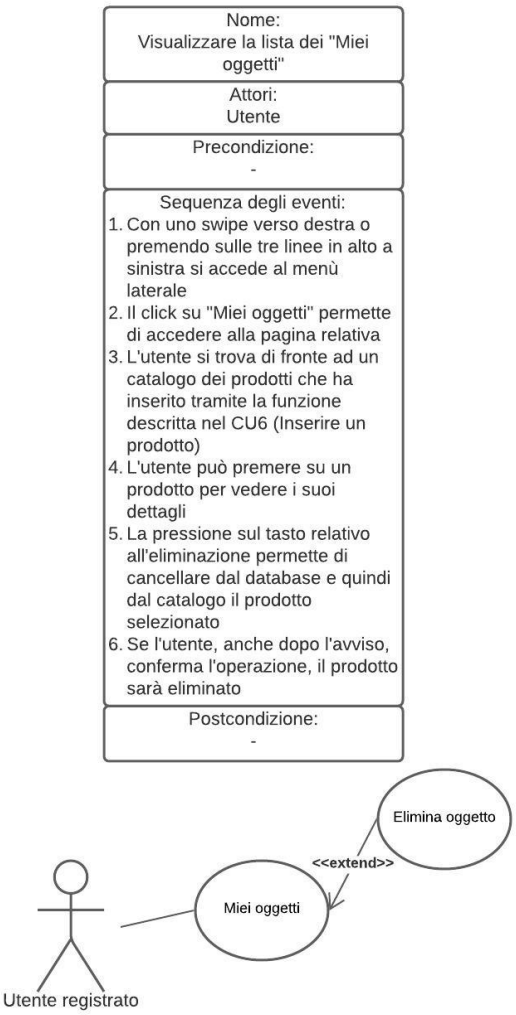

Figura 3.10. Descrizione del caso d'uso CU7 (Visualizzare la lista dei Miei oggetti) e diagramma UML associato

3.4 Struttura del database

La nostra applicazione, per garantire il corretto funzionamento, si serve di più strutture dati, ciascuna delle quali contenente al proprio interno dati di tipo diverso, organizzati in modo opportuno.

Nello specifico facciamo riferimento a quattro strutture dati: "oggetti", "utenti", "scambi", e "riepilogoscambi", realizzate e strutturate attraverso la piattaforma "Firebase", che ci ha dato la possibilità di dar vita ad un database con aggiornamento in tempo reale e altri due, invece, di tipo asincrono, che cioè richiedono la ricarica della pagina per visualizzare eventuali aggiornamenti dei dati in esse contenuti.

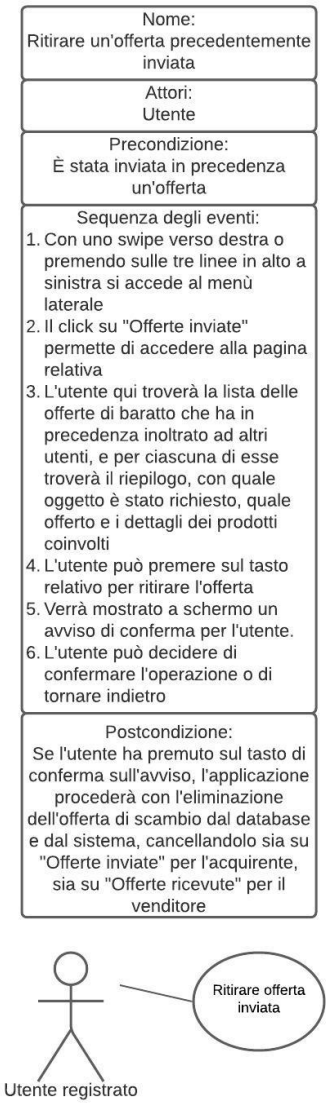

Figura 3.11. Descrizione del caso d'uso CU8 (Ritirare un'offerta precedentemente inviata) e diagramma UML associato

3.4.1 Oggetti

Il primo database che analizziamo è "oggetti". La scelta del privilegio dell'aggiornamento in tempo reale è ricaduta su di esso, in quanto traeva un vantaggio maggiore, soprattutto perché, in questo modo, è possibile vedere i nuovi prodotti e la rimozione dei presenti nel momento stesso in cui essi non sono più disponibili, a differenza di quanto avviene con le altre due strutture.

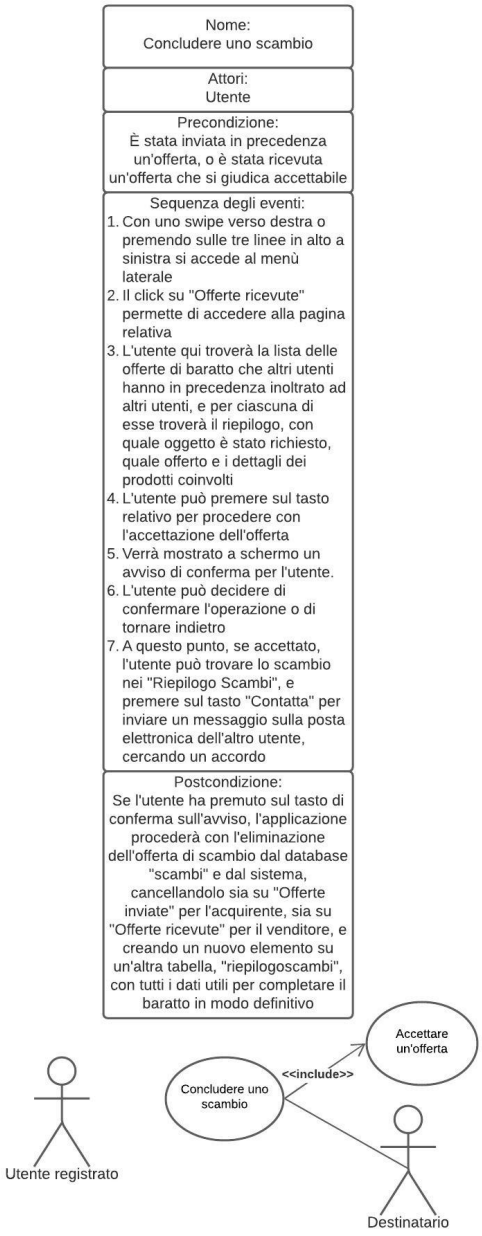

Figura 3.12. Descrizione del caso d'uso CU9 (Concludere uno scambio) e diagramma UML associato

Come è ben visibile nella Figura 3.15, tale struttura si presenta come una lista di codici identificativi, ciascuno dei quali è associato in modo univoco ad un oggetto del catalogo, e, accedendo ai singoli elementi, si può visualizzare la lista dei campi

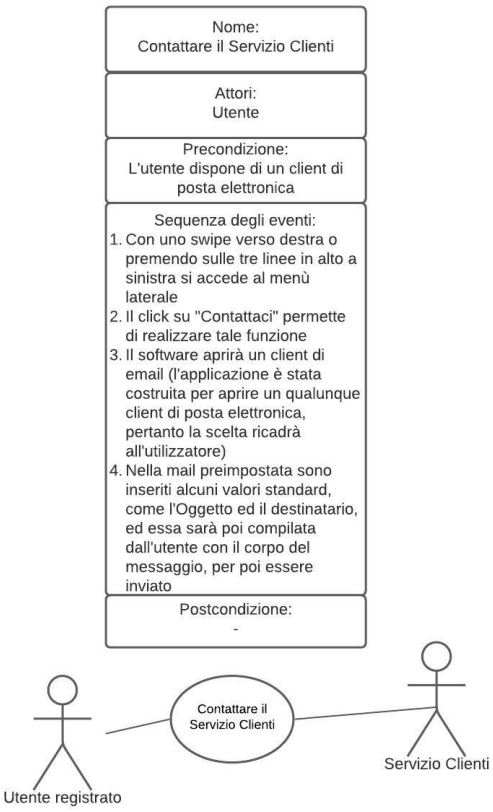

Figura 3.13. Descrizione del caso d'uso CU10 (Contattare il Servizio Clienti) e diagramma UML associato

popolati.

Ogni elemento della struttura "oggetti" è descritto con una serie di voci, che permettono di rappresentarlo al meglio, senza appesantire la struttura dati. Tali voci sono:

- *categoria*: rappresenta la categoria che è stata scelta dall'utente in fase di inserimento del prodotto, e serve principalmente per la funzione di filtro.
- *descrizione*: permette una descrizione più accurata dell'oggetto proposto.
- *idUser*: semplicemente è l'ID del proprietario del prodotto.
- *nome*: il nome dell'oggetto, che è stato scelto in fase di inserimento; rappresenta la voce che compare nella visualizzazione del catalogo.
- *nomeVenditore*: è il nome del venditore.
- *posizione*: nella fase di inserimento del prodotto, automaticamente viene associata all'oggetto la posizione corrente dell'utente, basandosi sulla posizione del dispositivo.
- *prezzo*: il prezzo simbolico che è stato deciso dal proprietario.

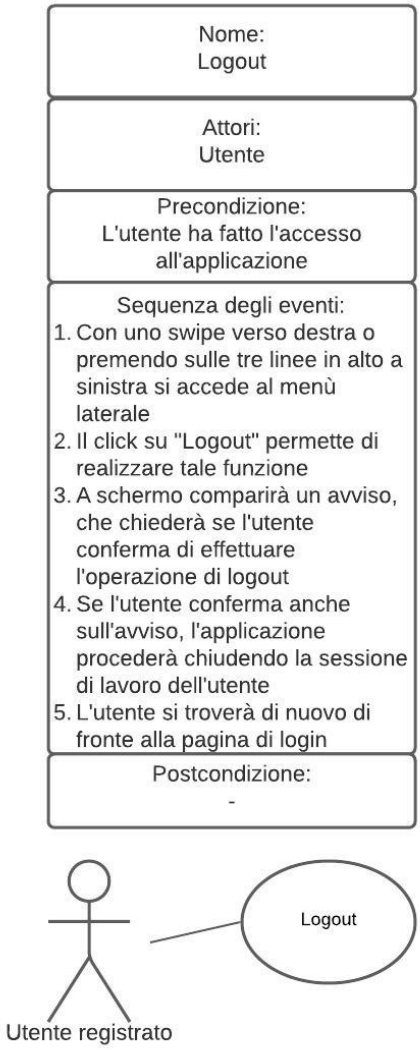

Figura 3.14. Descrizione del caso d'uso CU11 (Logout) e diagramma UML associato

Come già puntualizzato nella definizione dei requisiti, i campi saranno necessariamente tutti popolati, in quanto c'è il controllo, nella fase di inserimento, della presenza di informazioni nei campi nei quali l'utente deve inserire qualcosa.

3.4.2 Utenti

La successiva struttura dati analizzata è "utenti". Essa è relativamente semplice: come la precedente, si presenta come una lista di ID, che saranno, poi, quelli che

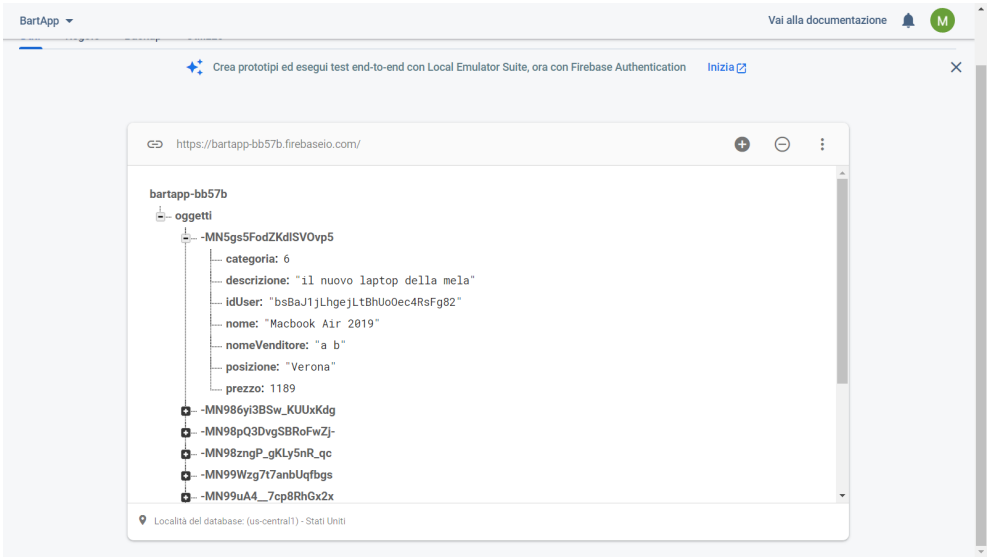

Figura 3.15. La schermata di Firebase relativa alla struttura dati "oggetti"

popoleranno la voce "idUser" di "oggetti"; per ciascun elemento ci troveremo di fronte ai seguenti campi:

- *nome*: il nome scelto in fase di registrazione.
- *cognome*: il cognome scelto in fase di registrazione.
- *email*: l'indirizzo di posta elettronica. Deve rispettare un vincolo di integrità per cui deve essere un indirizzo del tipo "aaaaa@bbbbb.it", pena il reinserimento del dato.

I dati provenienti da questa tabella saranno poi utilizzati dall'applicazione per visualizzare nel catalogo le generalità dei possessori dei prodotti visualizzati. Inoltre l'email comparirà nel "Riepilogo Scambi", in quanto informazione necessaria per far raggiungere un accordo tra gli utenti.

La Figura 3.16 permetterà, sicuramente, di capire al meglio come è organizzato il database.

3.4.3 Scambi

Una volta che un utente ha formalizzato uno scambio, selezionando un prodotto e offrendone in cambio uno in proprio possesso, viene aggiunto un elemento nella tabella "scambi", una struttura dati che raccoglie tutte le offerte. Sarà proprio su questa tabella che l'applicazione si baserà per riempire la lista delle offerte inviate e ricevute (con i filtri gestiti in modo opportuno), visibili nelle pagine apposite presenti nel menù laterale.

Ogni elemento di questa struttura si compone di diverse voci che lo caratterizzano; in particolare, saranno presenti le seguenti voci:

- `emailVend`: l'indirizzo di posta elettronica del venditore.

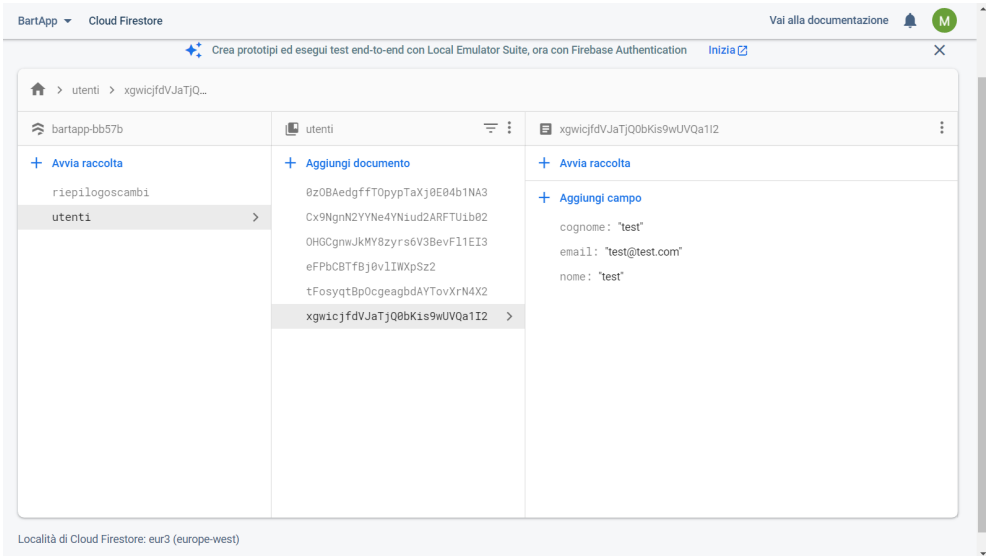

Figura 3.16. La schermata di Firebase relativa alla struttura dati "utenti"

- `idAcq`: l'id dell'utente acquirente, ottenuto dalla tabella "utenti".
- `idProdAcq`: il codice del prodotto che sta per essere acquistato da colui che ha inoltrato l'offerta.
- `idProdVend`: codice relativo al prodotto offerto in cambio.
- `idVend`: codice dell'utente venditore.
- `nomeAcq`: nome dell'utente acquirente.
- `nomeOggettoAcq`: nome dell'oggetto che sta per essere acquistato.
- `nomeOggettoVend`: nome dell'oggetto offerto in cambio.
- `nomeVend`: nome di chi ha inviato la proposta di baratto.
- `posizioneAcq`: posizione dell'acquirente associata al prodotto.
- `posizioneVend`: posizione del venditore associata al prodotto.
- `prezzoAcq`: prezzo associato al prodotto che si vuole ricevere.
- `prezzoOggettoVend`: prezzo associato al prodotto che si offre in cambio.

La Figura 3.17 permette una rappresentazione grafica di quanto detto.

3.4.4 Riepilogo scambi

Il quarto ed ultimo database che esploreremo in questa trattazione è quello relativo agli scambi effettuati, che si presenta sotto il nome di "riepilogoscambi".

Al suo interno, come anticipato, troveremo tutti quegli scambi che sono stati proposti e che sono stati accettati dall'utente che ha ricevuto l'offerta.

Su una struttura dati di questo tipo è stato deciso di implementare un filtro che per selezionare tutte le offerte che coinvolgevano l'utente correntemente collegato all'applicazione, e riempire con i dati corretti la pagina relativa, vale a dire "Riepilogo scambi", accessibile tramite menù laterale.

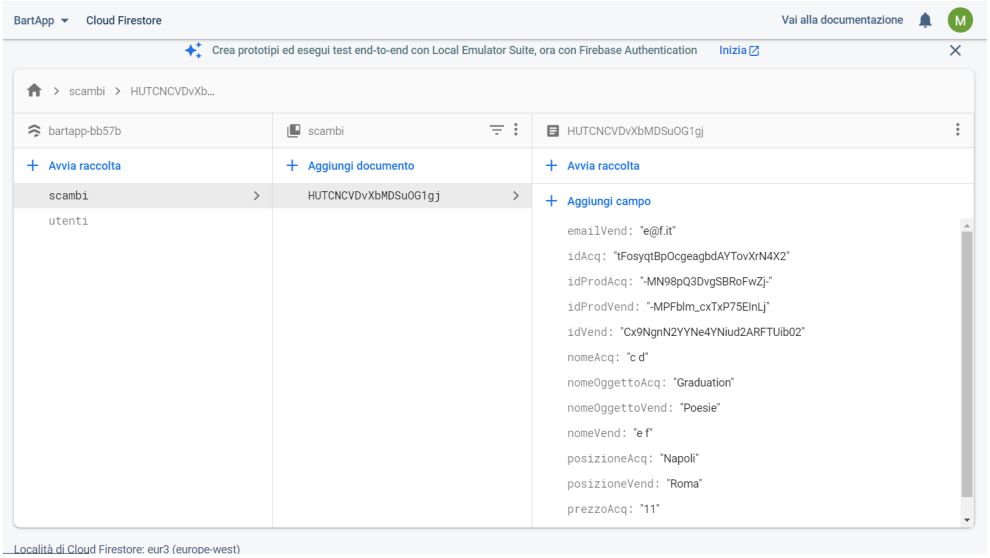

Figura 3.17. La schermata di Firebase relativa alla struttura dati "scambi"

Ogni singolo elemento della tabella, contraddistinto da un codice identificativo, presenta diversi campi che, per quanto icastici, verranno dettagliati ed organizzati come di seguito specificato:

- `emailAcq`: rappresenta l'email dell'acquirente, che verrà utilizzata per popolare la riga relativa al riepilogo.
- `emailVend`: rappresenta l'email del venditore, che l'acquirente utilizzerà per raggiungere degli accordi.
- `idAcq`: questa voce contiene l'id dell'utente acquirente.
- `idVend`: questa voce contiene l'id dell'utente venditore.
- `nomeOggettoAcq`: come suggerisce la voce, rappresenta il nome dell'oggetto proposto dall'acquirente.
- `nomeOggettoVend`: definisce il nome dell'oggetto richiesto, di possesso del venditore.
- `nomeUtenteVend`: mostra il nome utente del venditore [2].
- `posizioneAcq`: mostra la posizione dell'acquirente.
- `posizioneVend`: mostra la posizione del venditore.

La Figura 3.18 esplica la strutturazione dei dati appena analizzati.

[2] Come si può notare, diversamente da come ci si potesse aspettare, è assente la voce `"nomeUtenteAcq"`, in quanto tale dato può essere estrapolato attraverso funzioni predefinite di Firebase.

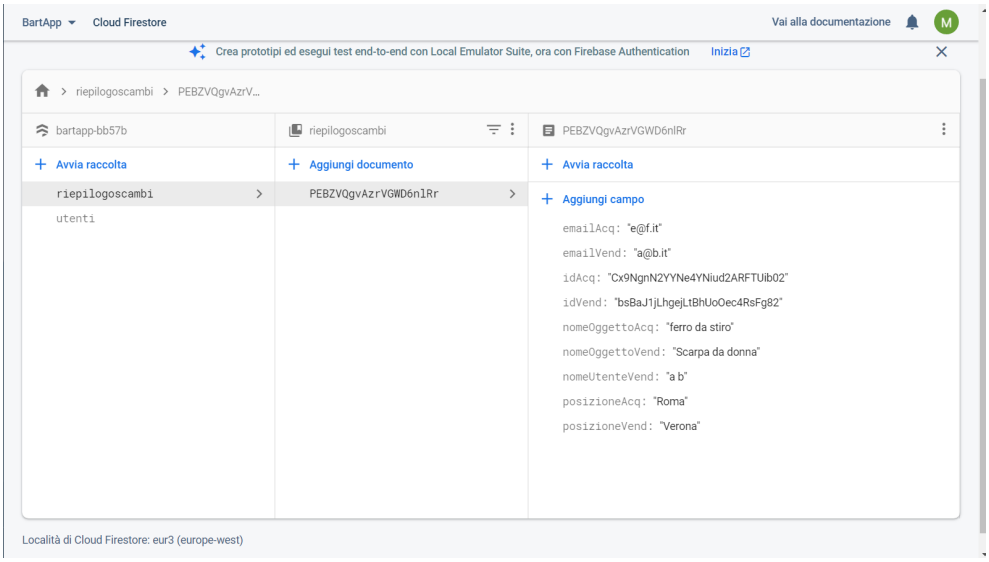

Figura 3.18. La schermata di Firebase relativa alla struttura dati “riepilogoscambi”

4

Implementazione dell'app e manuale utente

Questo capitolo si può dividere in due macrosezioni: implementazione e manuale utente. Nella prima parte ci soffermeremo sul lato tecnico della nostra applicazione, descrivendo le principali funzionalità e come esse siano state implementate nel codice.

La seconda, invece, consiste in un vero e proprio manuale, quindi una guida all'uso per gli utenti anche meno esperti, in modo tale da orientarli nel compiere ogni operazione che la nostra applicazione permette.

4.1 Implementazione dell'app

All'interno di questa sezione procederemo con una spiegazione tecnica dell'applicazione oggetto della presente tesi, soffermandoci sul codice e sul suo funzionamento.

4.1.1 Registrazione e Login

La prima funzione, con cui avremo a che fare con Bartapp, è quella relativa alla registrazione: un'operazione resa molto semplice dalle tre librerie `FirebaseAuth`, `FirebaseUser` e `UserProfileChangeRequest`, offerte dalla piattaforma Firebase. La prima di questa applicazione consiste di richiamare il metodo chiamato `createUserWithEmailAndPassword(email, password)` attraverso cui creare un nuovo utente sulla piattaforma associandogli l'email e la password da lui inserite in fase di registrazione (Listato 4.1).

```
mAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if(task.isSuccessful()){
                            final FirebaseUser user = mAuth.getCurrentUser();
                            UserProfileChangeRequest profileChangeRequest= new UserProfileChangeRequest.Builder()
                                    .setDisplayName(nome+ " "+ cognome)
                                    .build();
                            user.updateProfile(profileChangeRequest)
                            .addOnCompleteListener(new OnCompleteListener<Void>() {
                                @Override
                                public void onComplete(@NonNull Task<Void> task) {
                                    writeuserToDb(nome, cognome, user.getUid(), email);
                                    Intent intent= new Intent();
                                    intent.putExtra("nome", textNome.getText().toString());
```

```
                                intent.putExtra("cognome", textCognome.getText().toString());
                                intent.putExtra("email", textEmail.getText().toString());
                                intent.putExtra("password",textPassword.getText().toString());
                                Intent intent1 = new Intent(Registrazione.this, Login_Screen.class);
                                startActivity(intent1);
                            }
                        });

                    }
                    else{
                        task.getException().printStackTrace();
                        Toast.makeText(Registrazione.this, getString(R.string.errorsignup),Toast.LENGTH_SHORT)
                        .show();
                    }
                }
            });
        }
        catch (NullPointerException e) {
            Toast.makeText(Registrazione.this, getString(R.string.inforequired), Toast.LENGTH_SHORT).show();
        } catch (IllegalArgumentException e){
            Toast.makeText(Registrazione.this, getString(R.string.inforequired), Toast.LENGTH_SHORT).show();
        } /* oppure semplicemente un unico catch con Exception e*/

    }
});
```

Listato 4.1. Codice per la registrazione

Degna di menzione è la funzione `writeuserToDb`, con cui si inserisce un nuovo elemento nella tabella "utenti" di Firebase (Listato 4.2), caratterizzato da un nome, un cognome ed un'email. Questi dati saranno poi utilizzati nel catalogo quando in ogni riga bisognerà inserire anche il nome del proprietario, e nel riepilogo degli scambi, quando sarà necessario anche l'indirizzo di posta elettronica.

```
private void writeuserToDb(String nome, String cognome, String uid, String email){
       Map<String, Object> user= new HashMap<>();
       user.put("nome", nome);
       user.put("cognome", cognome);
       user.put("email", email);
       FirebaseFirestore db= FirebaseFirestore.getInstance();
       db.collection("utenti"). document(uid).set(user);
   }
```

Listato 4.2. Codice per inserire un nuovo utente nel database

Per quanto riguarda le operazioni di login, ci siamo serviti del metodo predefinito `signInWithEmailAndPassword(email,password)` (Listato 4.3), che viene richiamato quando l'utente fa click sul tasto associato al login (`btnLogin`). Quest'ultimo, nel caso di successo nell'autenticazione, permetterà l'accesso alla `HomeScreen`, vista associata al catalogo dei prodotti.

```
btnLogin.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            String email = inputEmail.getText().toString();
            final String password = inputPassword.getText().toString();


            //Metodo predefinito per effettuare il login su firebase

            auth.signInWithEmailAndPassword(email,password)
            .addOnCompleteListener(new OnCompleteListener<AuthResult>() {

                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {

                    if (task.isSuccessful()) {

                        Intent intent = new Intent(Login_Screen.this, HomeScreen.class);
                        startActivity(intent);
                    }
                    else{

                        task.getException();
                        Toast.makeText(Login_Screen.this, "Email/Password errati!",Toast.LENGTH_SHORT).show();
```

```
                }
            }
        });

    }
});
```

Listato 4.3. Codice per il login

4.1.2 Catalogo

Come abbiamo detto, subito dopo l'operazione di login, l'utente viene reindirizzato nella schermata principale, che mostra il catalogo dei prodotti ed un menù laterale (visibile previo swipe verso destra), che elencherà tutte le possibili funzioni della nostra applicazione.

Il codice si profila relativamente corposo (Listato 4.4); al suo interno sono presenti metodi per la gestione del menù laterale (`ActionBarDrawerToggle`), la gestione delle varie schermate che verranno visualizzate in seguito ad azioni dell'utente (`setNavigationItemSelectedListener`) ed il nome dell'utente mostrato nell'header del menù (`nomeHeader.setText(firebaseUser.getDisplayName())`).

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_home_screen);

        menuItem = (MenuItem) findViewById(R.id.Contattaci);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        drawerLayout = findViewById(R.id.drawer_layout);
        actionBarDrawerToggle = new ActionBarDrawerToggle(this,drawerLayout,toolbar,R.string.open,R.string.close);
        drawerLayout.addDrawerListener(actionBarDrawerToggle);
        actionBarDrawerToggle.syncState();
        NavigationView navView = (NavigationView) findViewById(R.id.navigation);
        navView.setNavigationItemSelectedListener(this);
        firebaseAuth = FirebaseAuth.getInstance();
        firebaseUser = firebaseAuth.getCurrentUser();
        View headerView = navView.getHeaderView(0);
        nomeHeader = (TextView) headerView.findViewById(R.id.nomeHeader);
        nomeHeader.setText(firebaseUser.getDisplayName());
        imageHeader = (ImageView) headerView.findViewById(R.id.button_add);
        firebaseStorage = FirebaseStorage.getInstance();
        storageReference = firebaseStorage.getReference();

        //inserisco l'immagine del profilo nell'header della navigation view
        storageReference.child("Image").child("Profile␣Pic").child(firebaseAuth.getUid())
        .getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
            @Override
            public void onSuccess(Uri uri) {
                Picasso.get().load(uri).fit().centerCrop().into(imageHeader);
            }
        });
        //inserimento fragment della recycler per la lista dei prodotti che sono sul mercato
        RecyclerViewFragment recyclerViewFragment = new RecyclerViewFragment();
        FragmentManager fm = getSupportFragmentManager();
        FragmentTransaction ft = fm.beginTransaction();
        ft.add(R.id.fragment_container_visualizza, recyclerViewFragment, "");
        ft.commit();
    }
```

Listato 4.4. Codice per la schermata principale dell'applicazione

Inoltre riportiamo anche le operazioni per inserire l'immagine del profilo dell'utente nell'header del menù, presente direttamente sopra il nome utente ed i metodi associati alla visualizzazione del catalogo, inseriti nella classe chiamata `RecyclerViewFragment`, che costituisce il vero e proprio catalogo.

All'interno di questa classe è di fondamentale importanza il metodo che si chiama `OnCreateView`, che permette la corretta visualizzazione dei prodotti, opportu-

namente inseriti in un array tramite l'oggetto `options`, della classe che prende il nome di `FirebaseRecyclerOptions`, che costituirà l'insieme dei prodotti visibili nel catalogo (Listato 4.5).

La classe vanta anche i metodi per i filtri, per quanto riguarda sia una ricerca per nome che per categorie. Come mostrato, per il nome nel Listato 4.6, per la categoria nel Listato 4.7, sono dunque due filtri che operano semplicemente una query che richiede che i prodotti mostrati sullo schermo abbiano delle caratteristiche particolari, compatibili con il filtro apportato.

```
databaseReference = FirebaseDatabase.getInstance().getReference().child("oggetti");
        databaseReference.keepSynced(true);

        //Estraggo tutti gli oggetti che verranno inseriti nella Recycler
        FirebaseRecyclerOptions<Oggetto> options = new FirebaseRecyclerOptions.Builder<Oggetto>()
                .setQuery(databaseReference, Oggetto.class)
                .build();


        adapter = new MyAdapter(options);
        recyclerView.setAdapter(adapter);
        return view;
    }
```

Listato 4.5. Codice per la vista dei prodotti

```
protected void firebaseSearch(String searchText) {
        final Query query = databaseReference.orderByChild("nome").equalTo(searchText);


        options = new FirebaseRecyclerOptions.Builder<Oggetto>().setQuery(query, Oggetto.class).build();


        firebaseRecyclerAdapter = new FirebaseRecyclerAdapter<Oggetto, MyAdapter.FirebaseViewHolder>(options) {
            @Override
            protected void onBindViewHolder(@NonNull final MyAdapter.FirebaseViewHolder firebaseViewHolder,
            final int i, @NonNull final Oggetto oggetto) {
                firebaseViewHolder.nome.setText(oggetto.getNome());
                firebaseViewHolder.nomeVenditore.setText(oggetto.getNomeVenditore());
                firebaseViewHolder.prezzo.setText(String.valueOf(oggetto.getPrezzo()));
                firebaseViewHolder.idUser.setText(oggetto.getIdUser());
                firebaseViewHolder.posizione.setText(oggetto.getPosizione());
                final FirebaseStorage firebaseStorage = FirebaseStorage.getInstance();
                StorageReference storageReference = firebaseStorage.getReference();
                final String keyId = this.getRef(i).getKey();
                final String descrizione = oggetto.getDescrizione();

                String idUser = oggetto.getIdUser();
                String nome = oggetto.getNome();
                storageReference.child("Image").child("ImmaginiOggetti").child(idUser).child(nome).getDownloadUrl()
                .addOnSuccessListener(new OnSuccessListener<Uri>() {
                    @Override
                    public void onSuccess(Uri uri) {
                        Picasso.get().load(uri).fit().centerCrop().into(firebaseViewHolder.immagineOggetto);
                    }
                });
                firebaseViewHolder.itemView.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {

                        TextView nome1 = v.findViewById(R.id.nome_oggetto);
                        String Nome1 = nome1.getText().toString();
                        TextView nomeVend = v.findViewById(R.id.nome_venditore);
                        String NomeVend = nomeVend.getText().toString();
                        TextView prezzo1 = v.findViewById(R.id.prezzo);
                        String Prezzo1 = prezzo1.getText().toString();
                        TextView idUser1 = v.findViewById(R.id.id_venditore);
                        String IdUser1 = idUser1.getText().toString();
                        TextView posizione1= v.findViewById(R.id.posizione);
                        String Posizione1= posizione1.getText().toString();

                        Bundle bundle= new Bundle();
                        bundle.putString("IdOggetto", keyId);
                        bundle.putString("Nome1", Nome1);
                        bundle.putString("NomeVend", NomeVend);
                        bundle.putString("Prezzo1", Prezzo1);
                        bundle.putString("idUser", IdUser1);
                        bundle.putString("descrizione", descrizione);
                        bundle.putString("posizione", Posizione1);
                        bundle.putInt("categoria", oggetto.getCategoria());
                        AppCompatActivity abc= (AppCompatActivity) v.getContext();
                        VisualizzaProdottoFragment visualizzaProdottoFragment= new VisualizzaProdottoFragment();
```

```
                    visualizzaProdottoFragment.setArguments(bundle);
                    firebaseRecyclerAdapter.stopListening();
                    abc.getSupportFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container_visualizza, visualizzaProdottoFragment, "")
                    .addToBackStack(null).commit();

                }
            });
        }

        @NonNull
        @Override
        public MyAdapter.FirebaseViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
            return new MyAdapter.FirebaseViewHolder(LayoutInflater.from(parent.getContext())
            .inflate(R.layout.rv_row, parent, false));
        }
    };
  }
```

Listato 4.6. Codice per la ricerca per nome

```
protected void firebaseSearchCategory(int position) {
      final Query query;
      if(position != 0) {
          query = databaseReference.orderByChild("categoria").equalTo(position - 1);
      }
      else query= databaseReference;


      options = new FirebaseRecyclerOptions.Builder<Oggetto>().setQuery(query, Oggetto.class).build();


      firebaseRecyclerAdapter = new FirebaseRecyclerAdapter<Oggetto, MyAdapter.FirebaseViewHolder>(options) {
          @Override
          protected void onBindViewHolder(@NonNull final MyAdapter.FirebaseViewHolder firebaseViewHolder,
          final int i, @NonNull final Oggetto oggetto) {
              firebaseViewHolder.nome.setText(oggetto.getNome());
              firebaseViewHolder.nomeVenditore.setText(oggetto.getNomeVenditore());
              firebaseViewHolder.prezzo.setText(String.valueOf(oggetto.getPrezzo()));
              firebaseViewHolder.idUser.setText(oggetto.getIdUser());
              firebaseViewHolder.posizione.setText(oggetto.getPosizione());
              final FirebaseStorage firebaseStorage = FirebaseStorage.getInstance();
              StorageReference storageReference = firebaseStorage.getReference();
              final String keyId = this.getRef(i).getKey();
              final String descrizione = oggetto.getDescrizione();

              String idUser = oggetto.getIdUser();
              String nome = oggetto.getNome();
              storageReference.child("Image").child("ImmaginiOggetti").child(idUser).child(nome).getDownloadUrl()
              .addOnSuccessListener(new OnSuccessListener<Uri>() {
                  @Override
                  public void onSuccess(Uri uri) {
                      Picasso.get().load(uri).fit().centerCrop().into(firebaseViewHolder.immagineOggetto);
                  }
              });
              firebaseViewHolder.itemView.setOnClickListener(new View.OnClickListener() {
                  @Override
                  public void onClick(View v) {

                      TextView nome1 = v.findViewById(R.id.nome_oggetto);
                      String Nome1 = nome1.getText().toString();
                      TextView nomeVend = v.findViewById(R.id.nome_venditore);
                      String NomeVend = nomeVend.getText().toString();
                      TextView prezzo1 = v.findViewById(R.id.prezzo);
                      String Prezzo1 = prezzo1.getText().toString();
                      TextView idUser1 = v.findViewById(R.id.id_venditore);
                      String IdUser1 = idUser1.getText().toString();
                      TextView posizione1= v.findViewById(R.id.posizione);
                      String Posizione1= posizione1.getText().toString();

                      Bundle bundle= new Bundle();
                      bundle.putString("IdOggetto", keyId);
                      bundle.putString("Nome1", Nome1);
                      bundle.putString("NomeVend", NomeVend);
                      bundle.putString("Prezzo1", Prezzo1);
                      bundle.putString("idUser", IdUser1);
                      bundle.putString("descrizione", descrizione);
                      bundle.putString("posizione", Posizione1);
                      bundle.putInt("categoria", oggetto.getCategoria());
                      AppCompatActivity abc= (AppCompatActivity) v.getContext();
                      VisualizzaProdottoFragment visualizzaProdottoFragment= new VisualizzaProdottoFragment();
                      visualizzaProdottoFragment.setArguments(bundle);
                      firebaseRecyclerAdapter.stopListening();
                      abc.getSupportFragmentManager().beginTransaction()
                      .replace(R.id.fragment_container_visualizza, visualizzaProdottoFragment, "")
                      .addToBackStack(null).commit();

                  }
              });
```

```
            }

            @NonNull
            @Override
            public MyAdapter.FirebaseViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
                return new MyAdapter.FirebaseViewHolder(LayoutInflater
                .from(parent.getContext()).inflate(R.layout.rv_row, parent, false));
            }
        };
    }
```

Listato 4.7. Codice per la ricerca per categoria

Infine una struttura di questo tipo ha necessità anche di un `Adapter`, presentato nel Listato 4.8. Come previsto dalla teoria, questo compone di 2 metodi, cioè `onBindViewHolder` e `onCreateViewHolder` e della classe `FirebaseViewHolder`.

Il primo metodo si occupa di popolare la singola riga dell'elemento presente nel catalogo, inserendo il nome, il nome del venditore, il prezzo e la posizione.

Per completare la riga, è necessario aggiungere anche l'immagine del prodotto, operazione che abbiamo deciso di realizzare servendoci della libreria "Picasso", alla quale forniamo l'immagine ottenuta mediante una query sulla tabella "ImmaginiOggetti", e, tramite i metodi predefiniti in questa libreria, riusciremo ad inserire l'immagine. Infine abbiamo associato una serie di istruzioni che si verificano al click sulla singola riga; infatti si crea e si popola un bundle, inserendo le informazioni che serviranno nella schermata successiva, relativa ai dettagli del prodotto.

Il secondo metodo semplicemente specifica quale modello della riga verrà utilizzato nella lista.

Infine, come anticipato, c'è bisogno di una nuova classe, `FirebaseViewHolder`, che, per comodità, è stata definita internamente e che ha il compito di recuperare il riferimento ad un oggetto collocato nell'interfaccia utente.

```
public MyAdapter(@NonNull FirebaseRecyclerOptions<Oggetto> option) {
        super(option);
    }

    @Override
    protected void onBindViewHolder(@NonNull final FirebaseViewHolder firebaseViewHolder,
     final int i, @NonNull final Oggetto oggetto) {
        //Effettuo il binding per ogni singolo elemento della recycler
        firebaseViewHolder.nome.setText(oggetto.getNome());
        firebaseViewHolder.nomeVenditore.setText(oggetto.getNomeVenditore());
        firebaseViewHolder.prezzo.setText(String.valueOf(oggetto.getPrezzo()));
        firebaseViewHolder.idUser.setText(oggetto.getIdUser());
        firebaseViewHolder.posizione.setText(oggetto.getPosizione());
        final String descrizione = oggetto.getDescrizione();
        final String keyId = this.getRef(i).getKey(); //prendo la chiave nel database dell'elemento
        final FirebaseStorage firebaseStorage = FirebaseStorage.getInstance();
        StorageReference storageReference = firebaseStorage.getReference();

        databaseReference = FirebaseDatabase.getInstance().getReference().child("oggetti");
        databaseReference.keepSynced(true);

        final String idUser = oggetto.getIdUser();
        String nome = oggetto.getNome();
        //Inserisco l'immagine dell'oggetto nella Recycler a seconda dell'oggetto nella lista
        storageReference.child("Image").child("ImmaginiOggetti").child(idUser).child(nome).getDownloadUrl()
        .addOnSuccessListener(new OnSuccessListener<Uri>() {
            @Override
            public void onSuccess(Uri uri) {
                Picasso.get().load(uri).fit().centerCrop().into(firebaseViewHolder.immagineOggetto);
            }
        });

        firebaseViewHolder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                TextView nome1 = v.findViewById(R.id.nome_oggetto);
                String Nome1 = nome1.getText().toString();
                TextView nomeVend = v.findViewById(R.id.nome_venditore);
                String NomeVend = nomeVend.getText().toString();
                TextView prezzo1 = v.findViewById(R.id.prezzo);
                String Prezzo1 = prezzo1.getText().toString();
                TextView idUser1 = v.findViewById(R.id.id_venditore);
```

```
                String IdUser1 = idUser1.getText().toString();
                TextView posizione1= v.findViewById(R.id.posizione);
                String Posizione1= posizione1.getText().toString();
                StorageReference storageReference = firebaseStorage.getReference();
                //Creo un bundle per portare i dati dalla recycler alla vista in dettaglio del Prodotto
                // con tutti i dati relativi ad esso
                Bundle bundle= new Bundle();
                bundle.putString("IdOggetto", keyId);
                bundle.putString("Nome1", Nome1);
                bundle.putString("NomeVend", NomeVend);
                bundle.putString("Prezzo1", Prezzo1);
                bundle.putString("idUser", IdUser1);
                bundle.putString("descrizione", descrizione);
                bundle.putString("posizione", Posizione1);
                bundle.putInt("categoria", oggetto.getCategoria());
                AppCompatActivity abc= (AppCompatActivity) v.getContext();
                VisualizzaProdottoFragment visualizzaProdottoFragment= new VisualizzaProdottoFragment();
                visualizzaProdottoFragment.setArguments(bundle);
                abc.getSupportFragmentManager().beginTransaction()
                .replace(R.id.fragment_container_visualizza, visualizzaProdottoFragment, "")
                        .addToBackStack(null).commit();
            }
        });

    }

    @NonNull
    @Override
    public FirebaseViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        return new FirebaseViewHolder(LayoutInflater.from(parent.getContext())
        .inflate(R.layout.rv_row, parent, false));

    }


    static class FirebaseViewHolder extends RecyclerView.ViewHolder {
        public ImageView immagineOggetto;
        public TextView nome;
        TextView nomeVenditore, prezzo, idUser, posizione;

        public FirebaseViewHolder(@NonNull View itemView) {
            super(itemView);
            immagineOggetto = itemView.findViewById(R.id.immagine_oggetto);
            nome = itemView.findViewById(R.id.nome_oggetto);
            nomeVenditore = itemView.findViewById(R.id.nome_venditore);
            prezzo = itemView.findViewById(R.id.prezzo);
            idUser = itemView.findViewById(R.id.id_venditore);
            posizione= itemView.findViewById(R.id.posizione);

        }
    }
```

Listato 4.8. Codice per l'Adapter

4.1.3 Profilo

Dal menù laterale possiamo entrare nell'area riservata dell'utente, all'interno della quale saranno riassunti tutti i dati del profilo.

La schermata che ci troviamo di fronte è relativamente semplice; in essa abbiamo l'immagine del profilo ingrandita e sensibile al click (per la sua gestione abbiamo utilizzato la libreria "Glide", come mostrato nel Listato 4.9), l'email, il nome utente e la password. In fondo alla schermata abbiamo, anche, la possibilità di eliminare l'account, operazione implementata tramite il metodo `delete()` della libreria `FirebaseAuth`.

```
    @Override
      public void onActivityResult(int requestCode, int resultCode, Intent intent) {
          super.onActivityResult(requestCode, resultCode, intent);
          if (requestCode == PICK_IMAGE) {
              Bitmap bitmap = null;
              if (resultCode == RESULT_OK) {
                  if (getPickImageResultUri(intent) != null) {
                      imagePath = getPickImageResultUri(intent);
                      try {
                          bitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), imagePath);
                      } catch (IOException e) {
                          e.printStackTrace();
                      }
```

```
            } else {
                bitmap = (Bitmap) intent.getExtras().get("data");
            }
        }

        Glide.with(this)
                .load(bitmap)
                .centerCrop()
                .into(proPic);
    }
}
```

Listato 4.9. Codice per l'immagine del profilo

Per quanto riguarda l'implementazione, è di un certo rilievo l'operazione associata alla modifica dell'immagine del profilo. Come previsto dalla teoria, abbiamo fatto in modo che l'applicazione, prima di poter accedere alle immagini del dispositivo ed alla fotocamera, richieda i permessi necessari, e nel caso in cui non siano dati, non sarà permesso ad essa di modificare l'immagine (Listato 4.10).

```
proPic.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            permissions.add(Manifest.permission.CAMERA);
            permissions.add(Manifest.permission.WRITE_EXTERNAL_STORAGE);
            permissions.add(Manifest.permission.READ_EXTERNAL_STORAGE);

            permissionsToRequest = findUnaskedPermissions(permissions);
            if(permissionsToRequest.size() > 0) {
                requestPermissions(permissionsToRequest
                .toArray(new String[permissionsToRequest.size()]), ALL_PERMISSIONS_RESULT);
            } else {
                startActivityForResult(getPickImageChooserIntent(), PICK_IMAGE);
            }
            btnSalvaModifica.setVisibility(View.VISIBLE);
            btnSalvaModifica.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    StorageReference imageReference = storageReference.child("Image").child("Profile Pic")
                    .child(mAuth.getUid());
                    if (imagePath != null) {
                        sendUserData(); // metodo per caricare l'immagine nello storage di Firebase
                    }
                    else{
                        Toast.makeText(Profilo.this, "Non hai inserito alcuna immagine!", Toast.LENGTH_SHORT)
                        .show();
                    }
                }
            });
        }
    });

}

private void getCurrentLocation(){
    LocationManager locationManager= (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    if (locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER) || locationManager
    .isProviderEnabled(LocationManager.NETWORK_PROVIDER)){
        fusedLocationProviderClient.getLastLocation().addOnCompleteListener(new OnCompleteListener<Location>() {
            @Override
            public void onComplete(@NonNull Task<Location> task) {
                final Location location= task.getResult();
                if (location != null){
                    //posizione.setText(location.toString());
                    Geocoder geocoder= new Geocoder(Profilo.this, Locale.getDefault());
                    try {
                        List<Address> addresses= geocoder
                        .getFromLocation(location.getLatitude(), location.getLongitude(), 1);
                        posizione.setText(addresses.get(0).getLocality());
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                } else{
                    LocationRequest locationRequest= new LocationRequest()
                    .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY).setInterval(10000)
                    .setFastestInterval(1000).setNumUpdates(1);
                    LocationCallback locationCallback = new LocationCallback(){
                        public void onLocationResult(LocationResult locationResult){
                            Location location1= locationResult.getLastLocation();
                            posizione.setText(String.valueOf(location1.getLatitude()));
                        }
                    };

                    fusedLocationProviderClient.requestLocationUpdates(locationRequest,
                    locationCallback, Looper.myLooper());
```

```
                }

            }
        });
    } else{
        startActivity(new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS)
        .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
    }

}

private Uri getPickImageResultUri(Intent data) {
    boolean isCamera = true;
    if (data != null) {
        String action = data.getAction();
        isCamera = action != null && action.equals(MediaStore.ACTION_IMAGE_CAPTURE);
    }

    return isCamera ? getCaptureImageOutputUri() : data.getData();
}

private Intent getPickImageChooserIntent() {

    Uri outputFileUri = getCaptureImageOutputUri();

    List<Intent> allIntents = new ArrayList<>();
    PackageManager packageManager = this.getPackageManager();

    Intent captureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    List<ResolveInfo> listCam = packageManager.queryIntentActivities(captureIntent, 0);
    for(ResolveInfo res : listCam) {
        Intent intent = new Intent(captureIntent);
        intent.setComponent(new ComponentName(res.activityInfo.packageName, res.activityInfo.name));
        intent.setPackage(res.activityInfo.packageName);
        if(outputFileUri != null) {
            intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
        }
        allIntents.add(intent);
    }

    Intent galleryIntent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    List<ResolveInfo> listGallery = packageManager.queryIntentActivities(galleryIntent, 0);
    for(ResolveInfo res : listGallery) {
        Intent intent = new Intent(galleryIntent);
        intent.setComponent(new ComponentName(res.activityInfo.packageName, res.activityInfo.name));
        intent.setPackage(res.activityInfo.packageName);
        allIntents.add(intent);
    }

    Intent mainIntent = allIntents.get(allIntents.size()-1);
    for(Intent intent : allIntents) {
        if(intent.getComponent().getClassName().equals("com.android.documentsui.DocumentsActivity")) {
            mainIntent = intent;
            break;
        }
    }
    allIntents.remove(mainIntent);

    Intent chooserIntent = Intent.createChooser(mainIntent, getString(R.string.selsorgente));

    chooserIntent.putExtra(Intent.EXTRA_INITIAL_INTENTS, allIntents.toArray(new Parcelable[allIntents.size()]));

    return chooserIntent;
}

public Uri getCaptureImageOutputUri() {
    Uri outputFileUri = null;
    File getImage = this.getExternalCacheDir();
    if(getImage != null) {
        outputFileUri = Uri.fromFile(new File(getImage.getPath(), "propic.png"));
    }
    return outputFileUri;
}

private ArrayList findUnaskedPermissions(ArrayList<String> wanted) {
    ArrayList<String> result = new ArrayList<>();

    for(String perm : wanted) {
        if(!(this.checkSelfPermission(perm) == PackageManager.PERMISSION_GRANTED)) {
            result.add(perm);
        }
    }

    return result;
}

public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    if(requestCode == ALL_PERMISSIONS_RESULT) {
        for(String perm: permissionsToRequest) {
            if(!(this.checkSelfPermission(perm)==PackageManager.PERMISSION_GRANTED)) {
                permissionsRejected.add(perm);
            }
        }
        if(permissionsRejected.size() > 0) {
```

```
                if(shouldShowRequestPermissionRationale(permissionsRejected.get(0))) {
                    Toast.makeText(this,"Approva tutto", Toast.LENGTH_SHORT).show();
                }
            }
            else {
                startActivityForResult(getPickImageChooserIntent(), PICK_IMAGE);
            }
        }
    }
```

Listato 4.10. Codice per la richiesta di permessi e per la modifica dell'immagine del profilo

In pratica, il codice mostrato nel Listato 4.10 popola inizialmente un array, `permissionsToRequest` tramite il metodo `findUnaskedPermissions`. Nel caso in cui l'array venisse effettivamente popolato significherà che ci sono dei permessi che devono essere richiesti, sia a seguito di un rifiuto o, semplicemente, perché non sono ancora mai stati richiesti.

Il metodo `findUnaskedPermissions` chiama iterativamente il metodo `checkSelfPermission` su ogni elemento dell'array che riceve in input, e verifica lo stato del permesso; nel caso in cui da questa operazione risultasse che il permesso correntemente valutato non è concesso, allora lo aggiunge all'array di output, che sarà, quindi, costituito da tutti quei permessi che hanno dato esito positivo nell'operazione contenuta nel ciclo.

I metodi `getPickImageResultUri`, `getPickImageChooserIntent`, ed anche `getCaptureImageOutputUri` sono relativi, invece, alla gestione della fotocamera e della galleria delle foto.

Infine il metodo `sendUserData` si occupa semplicemente di caricare nello storage "Profile Pic" della tabella "Image" l'immagine del profilo selezionata; esso, inoltre, mostra sullo schermo un avviso che segnala l'esito positivo o negativo del caricamento.

4.1.4 Aggiunta di un prodotto

Anche la funzione di inserimento di un nuovo prodotto nel catalogo si serve della libreria `Glide` precedentemente citata. Come mostrato nel Listato 4.11, questa libreria ci permette di visualizzare sullo schermo l'immagine che abbiamo selezionato per il prodotto.

```
    @Override
        public void onActivityResult(int requestCode, int resultCode, Intent intent) {

            super.onActivityResult(requestCode, resultCode, intent);
            if (requestCode == PICK_IMAGE) {

                Bitmap bitmap = null;
                if (resultCode == RESULT_OK) {

                    if (getPickImageResultUri(intent) != null) {

                        imagePath = getPickImageResultUri(intent);

                        try {
                            bitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), imagePath);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    } else {
                        bitmap = (Bitmap) intent.getExtras().get("data");
                    }
                }

                Glide.with(this)
                        .load(bitmap)
```

```
                .centerCrop()
                .into(immagineOggetto);
        }
    }
```

Listato 4.11. Codice per la visualizzazione dell'immagine sullo schermo

Menzioniamo, anche, l'utilizzo dello `Spinner`, cioè la lista delle categorie tra cui scegliere per associarne una al prodotto che stiamo per inserire. Lo `Spinner` necessita di un adapter generico; ad esso diamo la lista di categorie ed il layout dei singoli item della lista, per garantirne la visualizzazione nel modo corretto.

Inoltre, è bene anche riportare la funzione relativa all'ottenimento della posizione corrente del dispositivo, funzione fondamentale in quanto abbiamo deciso, in fase progettuale, di assegnare ad ogni prodotto una posizione, così che gli utenti possano servirsene per valutare l'acquisto di un oggetto anche sulla base della distanza (Listato 4.12).

```
private void getCurrentLocation(){

       LocationManager locationManager= (LocationManager) getSystemService(Context.LOCATION_SERVICE);
       if (locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER) ||
        locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER)){
           fusedLocationProviderClient.getLastLocation().addOnCompleteListener(new OnCompleteListener<Location>() {
               @Override
               public void onComplete(@NonNull Task<Location> task) {
                   final Location location= task.getResult();
                   if (location != null){

                       Geocoder geocoder= new Geocoder(Inserimento.this, Locale.getDefault());
                       try {
                           List<Address> addresses= geocoder
                           .getFromLocation(location.getLatitude(), location.getLongitude(), 1);
                           oggetto.setPosizione(addresses.get(0).getLocality());
                           databaseReference.push().setValue(oggetto);


                       } catch (IOException e) {
                           e.printStackTrace();
                       }
                   } else{
                       LocationRequest locationRequest= new LocationRequest()
                       .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY).setInterval(10000)
                               .setFastestInterval(1000).setNumUpdates(1);
                       LocationCallback locationCallback = new LocationCallback(){
                           public void onLocationResult(LocationResult locationResult){
                               Location location1= locationResult.getLastLocation();

                           }
                       };

                       fusedLocationProviderClient.requestLocationUpdates(locationRequest,
                        locationCallback, Looper.myLooper());
                   }

               }

           });
       } else{
           startActivity(new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS)
           .setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
       }
   }
```

Listato 4.12. Codice per la cattura della posizione del dispositivo

4.1.5 Offerte inviate e ricevute

Per quanto riguarda le offerte inviate, la query che è stata applicata sul database è relativamente semplice, come mostrato nel Listato 4.13. Infatti è bastato accedere alla tabella `"scambi"` che contiene tutte le offerte che sono state inviate tra tutti

gli utenti, e poi filtrare su di esse solo quelle che hanno, come ID del venditore (cioè chi ha inoltrato l'offerta), quelle dell'utente correntemente autenticato, così che in modo ragionevole si mostrano solo le offerte inviate.

Successivamente, per garantire all'utente una esperienza quanto più possibile comprensibile all'utente è stato inserito anche un controllo che verifica se il risultato di quella query fornisce una quantità nulla. Infatti, in questo caso, la schermata sarebbe altresì nulla, e quindi un utente potrebbe ritrovarsi confuso. Pertanto è stato deciso di inserire una riga di testo che compare solo se dal controllo risulta che non sono state inviate offerte. In questo caso quindi sullo schermo verrà mostrata una riga di testo che comunica all'utente che al momento non ci sono dati da mostrare.

```
@Override
    public View onCreateView(@NonNull LayoutInflater inflater,
     @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.recycler_nosearch, container, false);
        RecyclerView recyclerView= (RecyclerView) view.findViewById(R.id.recycler_view);
        layoutManager = new LinearLayoutManager(this.getContext());
        recyclerView.setLayoutManager(layoutManager);

        String mAuth= FirebaseAuth.getInstance().getCurrentUser().getUid();
        FirebaseFirestore db=FirebaseFirestore.getInstance();
        Query query = db.collection("scambi").whereEqualTo("idVend",mAuth);

        FirestoreRecyclerOptions<Offerta> options = new FirestoreRecyclerOptions.Builder<Offerta>()
                .setQuery(query , Offerta.class)
                .build();
            final TextView textView= view.findViewById(R.id.textView8);

            query.addSnapshotListener(new EventListener<QuerySnapshot>() {
                @Override
                public void onEvent(@Nullable QuerySnapshot queryDocumentSnapshots,
                 @Nullable FirebaseFirestoreException e) {

                    if (queryDocumentSnapshots.size() != 0) {
                        textView.setVisibility(View.INVISIBLE);
                    }
                }
            });

        adapter = new MyAdapterOfferteInviate(options, this.getContext());
        recyclerView.setAdapter(adapter);
        return view;
    }
```

Listato 4.13. Codice relativo al metodo `onCreateView`

Per quanto riguarda, invece, le offerte ricevute, la situazione è speculare alla precedente, e la query adottata è simile, con la sola differenza che l'ID, che deve coincidere con l'utente correntemente autenticato, non è quello del venditore, bensì quello dell'acquirente, mostrando, così tutte le offerte ricevute dall'utilizzatore dell'applicazione (nel Listato 4.14 mostriamo la funzione di accettazione e di rifiuto delle offerte ricevute).

```
public void accetta(final String keyId, final String idProdAcq, final String idProdVend, final String idAcq, final
                    String idVend, final String nomeOggettoAcq, final String nomeOggettoVend,
                     final String nomeVend, final String emailVend,
                    final String posizioneAcq, final String posizioneVend) {

        AlertDialog.Builder dialog = new AlertDialog.Builder(context);
        dialog.setTitle("Attenzione!");
        dialog.setCancelable(false);
        dialog.setMessage("Sei␣sicuro␣di␣voler␣accettare␣l'offerta?");
        dialog.setPositiveButton("Accetta", new DialogInterface.OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {

                    FirebaseFirestore firebaseFirestore= FirebaseFirestore.getInstance();
                    final Map<String, Object> map= new HashMap<>();

                    map.put("emailAcq", FirebaseAuth.getInstance().getCurrentUser().getEmail());
                    map.put("emailVend", emailVend);
                    map.put("nomeOggettoAcq", nomeOggettoAcq);
                    map.put("nomeOggettoVend", nomeOggettoVend);
```

```
                    map.put("nomeUtenteVend", nomeVend);
                    map.put("idAcq", idAcq);
                    map.put("idVend", idVend);
                    map.put("posizioneAcq", posizioneAcq);
                    map.put("posizioneVend", posizioneVend);

                    firebaseFirestore.collection("riepilogoscambi").document().set(map);

                    operazione1(idProdAcq);
                    operazione2(idProdAcq);
                    operazione3(idProdVend);
                    operazione4(idProdVend, idProdAcq, idAcq, idVend, nomeOggettoAcq, nomeOggettoVend);

        }
    });

    dialog.setNegativeButton("Annulla", new DialogInterface.OnClickListener() {

        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
        }
    });

    AlertDialog alertDialog = dialog.create();
    alertDialog.show();

}


//Elimino tutte le offerte che mi sono state fatte sullo stesso oggetto

public void operazione1 (String idProdAcq) {

    FirebaseFirestore db = FirebaseFirestore.getInstance();
    db.collection("scambi").whereEqualTo("idProdAcq", idProdAcq)
            .get().addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {

        @Override
        public void onComplete(@NonNull Task<QuerySnapshot> task) {

            if (task.isSuccessful()) {
                for (DocumentSnapshot document : task.getResult()) {

                    document.getReference().delete();
                    Toast.makeText(context, "Scambio effettuato", Toast.LENGTH_SHORT).show();
                }

            } else {
            }
        }
    });
}


//Elimino tutte le offerte che IO ho inviato ad altri dell'oggetto che scambio con l'utente.

public void operazione2 (String idProdAcq) {

    FirebaseFirestore db0 = FirebaseFirestore.getInstance();
    db0.collection("scambi").whereEqualTo("idProdVend", idProdAcq).get()
    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {

        @Override
        public void onComplete(@NonNull Task<QuerySnapshot> task) {
            if (task.isSuccessful()) {

                for (DocumentSnapshot document : task.getResult()) {
                    document.getReference().delete();
                    Toast.makeText(context, "Scambio effettuato", Toast.LENGTH_SHORT).show();
                }
            } else {
            }
        }
    });

}


//Elimino tutte le offerte in cui l'oggetto che mi offre l'altro utente Ã¨ stato richiesto da altri utenti.

public void operazione3 (String idProdVend) {

    FirebaseFirestore db1 = FirebaseFirestore.getInstance();
    db1.collection("scambi").whereEqualTo("idProdAcq", idProdVend).get()
    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
        @Override
        public void onComplete(@NonNull Task<QuerySnapshot> task) {
            if (task.isSuccessful()) {

                for (DocumentSnapshot document : task.getResult()) {
                    document.getReference().delete();
                }
            } else {
```

```
            }
        }
    });
}


//Elimino tutte le offerte in cui l'oggetto che mi offre l'altro utente Ã¨ stato proposto ad altri utenti.

public void operazione4 (final String idProdVend, final String idProdAcq, final String idAcq,
final String idVend, final String nomeOggettoAcq, final String nomeOggettoVend) {

    FirebaseFirestore db2 = FirebaseFirestore.getInstance();
    db2.collection("scambi").whereEqualTo("idProdVend", idProdVend).get()
    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {

        @Override
        public void onComplete(@NonNull Task<QuerySnapshot> task) {
            if (task.isSuccessful()) {

                for (DocumentSnapshot document : task.getResult()) {
                    document.getReference().delete();
                    Toast.makeText(context, "Scambio effettuato", Toast.LENGTH_SHORT).show();
                    eliminaProdotti(idProdVend, idProdAcq, idAcq,
                    idVend, nomeOggettoAcq, nomeOggettoVend);
                }
            } else {
                }
            }
    });
}
```

Listato 4.14. Codice per la gestione delle offerte ricevute

Fondamentalmente la funzione di accettazione mostra all'utente un avviso sullo schermo, che permette di annullare l'accettazione dell'offerta o di confermare la propria decisione.

Nel caso in cui l'utente accetti, verrà creata una `Map<String, Object>` che sarÃ poi popolata con i campi necessari per il riepilogo degli scambi, quindi:

- `emailAcq`
- `emailVend`
- `nomeOggettoAcq`
- `nomeOggettoVend`
- `nomeUtenteVend`
- `idAcq`
- `idVend`
- `posizioneAcq`
- `posizioneVend`

Successivamente questa mappa verrà inserita proprio nella tabella con il nome di `"riepilogoscambi"`. A questo punto verranno mandate in esecuzione quattro funzioni in sequenza, ovvero `operazione1`, `operazione2`, `operazione3`, `operazione4`, che permettono di eliminare tutte le altre offerte che coinvolgono gli oggetti impegnati nel baratto appena accettato, per evitare qualche truffa che preveda lo stesso oggetto interessato in più scambi contemporaneamente.

Infine, l'ultima operazione ha a sé associata anche la funzione `eliminaProdotti` (sempre visualizzabile nel Listato 4.14), che, come suggerisce il nome, elimina i due prodotti barattati e le loro immagini dallo storage.

4.1.6 Riepilogo Scambi

Per quanto riguarda la pagina del riepilogo degli scambi, abbiamo adottato una implementazione piuttosto particolare; abbiamo infatti impostato due diversi layout

sovrapposti, con la selezione di uno piuttosto che un altro a seconda se a visualizzarlo è il venditore o l'acquirente. Del resto abbiamo optato per una scelta simile nel momento in cui avevamo imposto all'oggetto "acquisito" il caratteristico colore verde ed il rosso per quello che, poi, verrà ceduto all'altro utente interessato nello scambio.

Dunque un controllo iniziale, come mostrato nel Listato 4.15, fa capire se l'utente correntemente autenticato è il venditore o l'acquirente nel riepilogo che sta per essere visualizzato; verranno, quindi, impostate delle combinazioni di visibilità particolari degli elementi sullo schermo, per permettere la definizione del layout adatto.

```
@Override
  protected void onBindViewHolder(@NonNull MyAdapterRiepilogo.FirestoreViewHolder holder,
  int position, @NonNull final Riepilogo model) {
      if (model.getIdAcq().equals(mAuth.getUid())) {
          holder.relativeLayout2.setVisibility(View.INVISIBLE);
          holder.nomeOggettoAcq.setText(model.getNomeOggettoAcq());
          holder.nomeOggettoVend.setText(model.getNomeOggettoVend());
          holder.nomeVend.setText(model.getNomeUtenteVend());
          holder.emailVend.setText(model.getEmailVend());
           holder.posizioneVend.setText(model.getPosizioneVend());


          DocumentSnapshot documentSnapshot = getSnapshots().getSnapshot(position);
          final String id = documentSnapshot.getId();
          holder.btnCancella.setOnClickListener(new View.OnClickListener() {
              @Override
              public void onClick(View v) {
                  eliminaRiepilogo(id);
              }
          });

          holder.btnContatta.setOnClickListener(new View.OnClickListener() {
              final String email = model.getEmailVend();
              final String nomeOggetto = model.getNomeOggettoVend();
              final String nomeOggetto2 = model.getNomeOggettoAcq();

              @Override
              public void onClick(View v) {
                  contatta(email, nomeOggetto, nomeOggetto2, context);
              }
          });
      } else if (model.getIdVend().equals(mAuth.getUid())) {
          holder.relativeLayout1.setVisibility(View.INVISIBLE);
          holder.email2.setText("Email:␣" + model.getEmailAcq());
          holder.nomeOggettoProposto.setText("Oggetto␣proposto:␣" + model.getNomeOggettoVend());
          holder.nomeOggettoRichiesto.setText("Oggetto␣richiesto:␣" + model.getNomeOggettoAcq());
          holder.posizioneRichiesto.setText(model.getPosizioneAcq());
          holder.btnContatta2.setOnClickListener(new View.OnClickListener() {
              final String email = model.getEmailAcq();
              final String nomeOggetto = model.getNomeOggettoVend();
              final String nomeOggetto2 = model.getNomeOggettoAcq();

              @Override
              public void onClick(View v) {
                  contatta2(email, nomeOggetto, nomeOggetto2, context);
              }
          });
      } else {
          holder.itemView.setVisibility(View.INVISIBLE);
      }
  }
```

Listato 4.15. Codice per la gestione dei layout del riepilogo delle offerte

Una volta verificata la visualizzazione corretta del riepilogo delle offerte, valutata riga per riga, l'utente ha il compito di portare a termine lo scambio, operazione che noi abbiamo deciso di rendere possibile mediante un accordo per via telematica, mettendo a disposizione per entrambi gli utenti interessati l'indirizzo di posta elettronica dell'altro utente, ed un tasto che riporta la scritta "Contatta", che aprirà il client di posta elettronica selezionato dall'utente, ed un messaggio predefinito, che invita a completare in via definitiva lo scambio, ed avente come destinatario l'altro utente, mediante i metodi `contatta` e `contatta2` del Listato 4.16.

A questo punto l'utente potrà servirsi di questa schermata per ricontrollare

tutti gli scambi che sono stati precedentemente portati a termine, da cui il nome "Riepilogo Scambi" che, appunto, suggerisce quale sia il suo reale obbiettivo.

Ciononostante, è probabile che un utente più accorto voglia avere sempre una cronologia pulita, o semplicemente cancellare alcuni scambi per una questione di ordine. A tal proposito abbiamo inserito una immagine di un cestino stilizzato, che, al click, eliminerà quel particolare riepilogo dalla lista, tramite il metodo `eliminaRiepilogo` del Listato 4.16.

```
public void eliminaRiepilogo(final String id) {
       AlertDialog.Builder dialog = new AlertDialog.Builder(context);
       dialog.setTitle("Attenzione!");
       dialog.setCancelable(false);
       dialog.setMessage("Sei␣sicuro␣di␣voler␣il␣riepilogo␣dell'offerta?
␣␣␣␣␣␣␣␣Perderai␣tutti␣i␣dettagli␣relativi␣ad␣essa!");
       dialog.setPositiveButton("Elimina", new DialogInterface.OnClickListener() {
           @Override
           public void onClick(DialogInterface dialog, int which) {
               FirebaseFirestore db = FirebaseFirestore.getInstance();
               db.collection("riepilogoscambi").document(id).delete();
           }
       });
       dialog.setNegativeButton("Annulla", new DialogInterface.OnClickListener() {
           @Override
           public void onClick(DialogInterface dialog, int which) {
               dialog.dismiss();
           }
       });

       AlertDialog alertDialog = dialog.create();
       alertDialog.show();
   }

   public void contatta(String email, String nomeOggetto, String nomeOggetto2, Context context) {
       Intent intent = new Intent(Intent.ACTION_SENDTO, Uri.fromParts("mailto", email, null));
       intent.putExtra(Intent.EXTRA_SUBJECT, "Scambio␣Oggetto");
       intent.putExtra(Intent.EXTRA_TEXT, "Sono␣interessato␣all'oggetto␣" +
               nomeOggetto + "␣che␣ha␣inserito,␣in␣cambio␣del␣mio␣" + nomeOggetto2
                + "␣possiamo␣effettuare␣lo␣scambio!");
       intent.putExtra(Intent.EXTRA_EMAIL, email);
       context.startActivity(Intent.createChooser(intent, "Invia␣email.."));
   }
   public void contatta2(String email, String nomeOggetto, String nomeOggetto2, Context context) {
       Intent intent = new Intent(Intent.ACTION_SENDTO, Uri.fromParts("mailto", email, null));
       intent.putExtra(Intent.EXTRA_SUBJECT, "Scambio␣Oggetto");
       intent.putExtra(Intent.EXTRA_TEXT, "Sono␣interessato␣all'oggetto␣" +
               nomeOggetto + "␣che␣ha␣inserito,␣in␣cambio␣del␣mio␣" + nomeOggetto
               + "␣possiamo␣effettuare␣lo␣scambio!");
       intent.putExtra(Intent.EXTRA_EMAIL, email);
       context.startActivity(Intent.createChooser(intent, "Invia␣email.."));
   }
```

Listato 4.16. Codice per la gestione della cronologia delle offerte

4.1.7 Contattaci e Logout

Terminiamo questa sezione riportando, per completezza, i codici relativi alle ultime due opzioni disponibili nel menu laterale che non abbiamo ancora approfondito, vale a dire "Contattaci" e "Logout", mostrate, rispettivamente, nei listati 4.17 e 4.18.

```
 public void invioMail() {
       Intent intent = new Intent(Intent.ACTION_SENDTO);
       intent.putExtra(Intent.EXTRA_SUBJECT, "Help␣Request");

       intent.setData(Uri.parse("mailto:BartApp@developers.com"));
       startActivity(intent);
   }
```

Listato 4.17. Codice per l'invio della mail agli sviluppatori

```
protected void logout() {
```

```
        AlertDialog.Builder dialog = new AlertDialog.Builder(HomeScreen.this);
        dialog.setTitle("Attenzione");
        dialog.setCancelable(false);
        dialog.setMessage("Sei sicuro di voler effettuare il Logout?");
        dialog.setPositiveButton("Conferma", new DialogInterface.OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                FirebaseAuth mAuth = FirebaseAuth.getInstance();
                mAuth.signOut();
                Intent intent = new Intent(HomeScreen.this, Login_Screen.class);
                startActivity(intent);
                finish();
            }
        });
        dialog.setNegativeButton("Annulla", new DialogInterface.OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
            }
        });

        AlertDialog alertDialog = dialog.create();
        alertDialog.show();
    }
```

Listato 4.18. Codice per il logout

4.2 Manuale utente

Dopo aver illustrato l'implementazione a livello di codice della nostra applicazione, si è ritenuta necessaria la stesura di un manuale utente, vale a dire una spiegazione dettagliata di come la nostra applicazione deve funzionare, in base alle intenzioni dell'utente. Guideremo passo passo il lettore, indicando i passaggi da eseguire e corredando le parole con degli screenshot presi direttamente dall'applicazione, per far sì che l'utente non si senta spaesato, anzi orientato in modo mirato nella navigazione tra le varie schermate che gli si pongono davanti.

Figura 4.1. Icona di Bartapp

4.2.1 Primo avvio

La prima cosa da fare è avviare la nostra applicazione, tramite un click sull'icona corrispondente, mostrata in Figura 4.1. A questo punto, nel caso di primo avvio,

l'utente vedrà una schermata di questo tipo (Figura 4.2), nella quale è invitato ad inserire l'email ed una password relative ad un account preesistente.

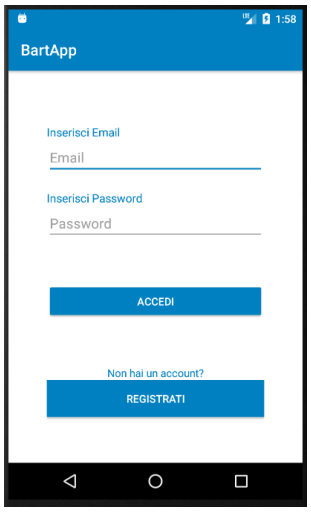

Figura 4.2. Schermata di Login

Nel nostro caso l'utente ha appena avviato l'applicazione per la prima volta, e quindi non dispone di un account, pertanto si recherà nella Sezione "Registrazione", che si mostrerà come in Figura 4.3. A questo punto i placeholder presenti in ogni riga suggeriscono all'utente quali informazioni devono essere inserite per compilare il proprio account e completare la fase di registrazione: verrà richiesto l'inserimento delle proprie generalità, quali nome e cognome, inoltre verrà inserita una mail valida ed, infine, una password, che dovrà contenere almeno 6 caratteri.

Una volta compilati tutti i campi, l'utente premerà sul tasto "Registrati", presente in fondo allo schermo, per andare avanti.

Nel caso in cui l'operazione non sia andata a buon fine (ad esempio la password contiene meno di 6 caratteri), l'utente sarà avvisato, e conseguentemente invitato ad inserire di nuovo i dati. Questa operazione si ripeterà fino a che l'utente non inserirà tutti i dati compatibili con un account utente valido.

4.2.2 Login e Catalogo

A questo punto l'utente avrà di fronte la schermata già mostrata in Figura 4.2, con la sola differenza che questa volta si dispone di un account. Inserendo nei campi "email" e "password" i dati precedentemente specificati, sarà verificata l'autenticazione e sarà permesso l'accesso all'applicazione vera e propria, che si presenta come in Figura 4.4, cioè una lista di prodotti caratterizzati da un'immagine, un nome, il nome del proprietario, il prezzo e la posizione.

Nel caso in cui l'utente cercasse un prodotto appartenente ad una specifica categoria, egli può filtrare i prodotti del catalogo tramite il menù a tendina, come in

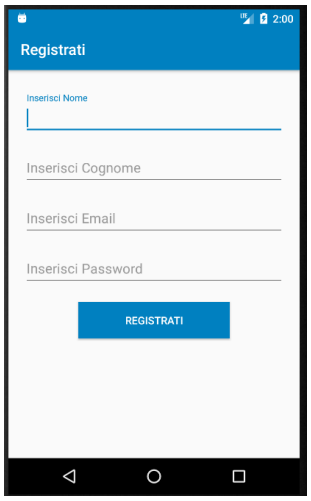

Figura 4.3. Schermata di Registrazione

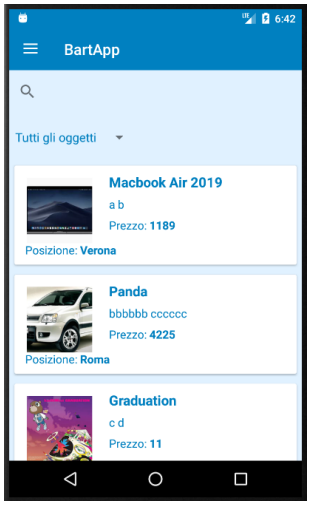

Figura 4.4. Catalogo dei prodotti

Figura 4.5, nel quale è evidenziato in rosso il menù dal quale selezionare la categoria del prodotto che si vuole cercare. Ad oggi si contano ben 10 categorie:

- Abbigliamento;
- Veicoli;
- Film e Musica;
- Elettrodomestici;
- Fotografia;
- Libri;
- Informatica;
- Sport;

- Alimentari;
- Altro.

In caso contrario, la lente d'ingrandimento, posta immediatamente sopra il menù delle categorie, permette all'utente l'inserimento di un nome di un prodotto, nel caso in cui si conosca quello cercato.

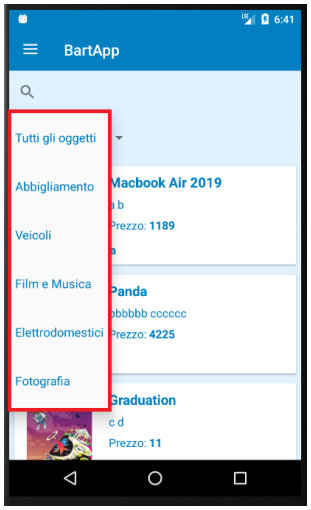

Figura 4.5. Filtro delle categorie

4.2.3 Visualizzazione prodotto

Nel caso in cui un particolare oggetto destasse la curiosità dell'utilizzatore del software, potrà premere sulla riga corrispondente all'oggetto desiderato per accedere alla pagina dei dettagli del singolo prodotto, come mostrato in Figura 4.6. Oltre alle informazioni reperibili già dal catalogo, ora si caratterizza l'oggetto con una descrizione più esauriente.

Come vedremo in seguito, l'applicazione offre la possibilità di inserire propri oggetti; nel caso in cui fosse stata premuta la riga di un proprio oggetto, il tasto dell'offerta è sostituito da quello per l'eliminazione.

4.2.4 Formulare un'offerta

In fondo alla pagina dei dettagli è presente un tasto (visibile in Figura 4.6) che riporta la voce "Formula offerta". Come suggerisce il nome, questo tasto ci porta alla schermata da cui partirà la proposta di scambio: infatti, al click sul tasto, vedremo il catalogo dei prodotti inseriti dall'utente (Figura 4.7), con la possibilità di sceglierne uno da offrire in cambio di quello desiderato.

Seguirà un avviso sullo schermo che chiederà conferma all'utente di formulare l'offerta; se l'utente dà conferma, l'applicazione procederà ad inoltrare l'offerta al ricevente.

Figura 4.6. Schermata dei dettagli del prodotto

Figura 4.7. Schermata di scelta del prodotto da scambiare

4.2.5 Profilo

Mediante uno swipe verso destra, o tramite il click sulle tre linee poste in alto a sinistra dello schermo, evidenziate in Figura 4.8, si mostrerà sullo schermo il menu laterale (Figura 4.9), che presenta l'immagine del profilo, il nome utente e tutte le funzioni che la nostra applicazione offre.

Partendo con ordine, la prima dell'elenco è "Profilo". Premiamo sull'opzione corrispondente ed entriamo nell'area riservata all'utente (Figura 4.10). Al suo interno troviamo l'immagine del profilo (ingrandita rispetto a quella che vediamo nel menù laterale), il nome utente, la e-mail e la password.

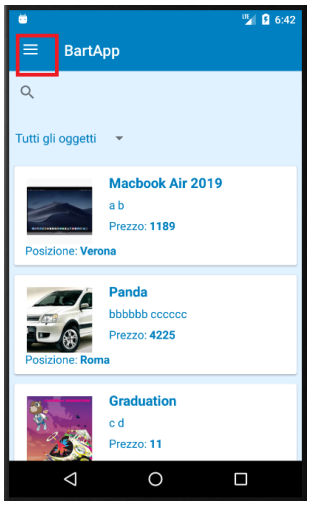

Figura 4.8. Accesso al menù laterale

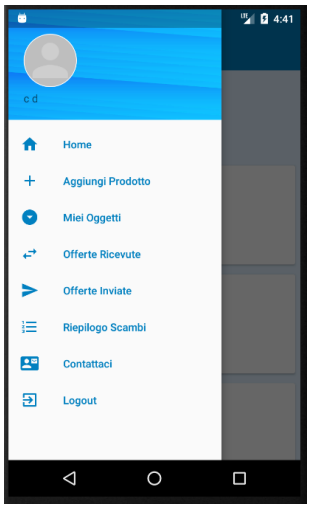

Figura 4.9. Menù laterale

Premendo sull'immagine del profilo l'applicazione ci permetterà di cambiare immagine: una volta forniti i permessi di accesso al rullino ed alla fotocamera (operazioni richieste una sola volta), potremo inserire una foto dal rullino o scattarne una all'istante. Confermando sul tasto corrispondente avremo modificato l'immagine del profilo.

Successivamente, le matite stilizzate presenti accanto al nome utente e password aprono alla possibilità di modifica di quei campi. Per quanto riguarda la modifica del nome utente, l'operazione è semplicissima: inseriamo il nuovo nome utente e confermiamo con il tasto sottostante (Figura 4.11).

Per quanto riguarda la modifica della password, l'utente può notare due livelli di

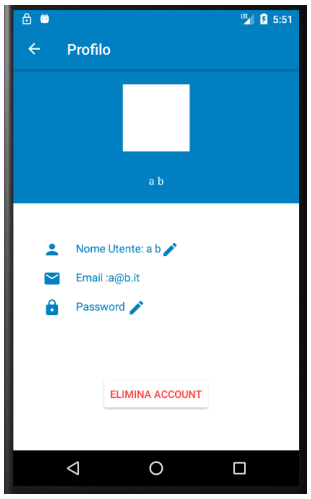

Figura 4.10. Area riservata dell'utente

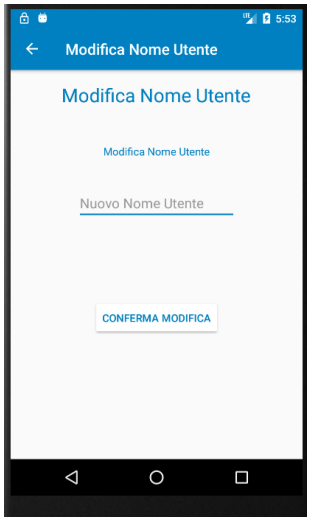

Figura 4.11. Schermata per la modifica del nome utente

sicurezza: la conferma della password inserita, che verrà realizzata tramite l'input contrassegnato con "Conferma password", ed il minimo di 6 caratteri come lunghezza della password (Figura 4.12). Il tasto in fondo allo schermo permette la conferma dell'operazione, una volta cliccato.

4.2.6 Inserisci un prodotto

L'operazione imprescindibile per la formulazione di un'offerta di baratto consiste nell'inserimento di un proprio prodotto da utilizzare come pedina di scambio. Del resto, il menu laterale riporta anche la voce "aggiungi prodotto". Il click su di essa

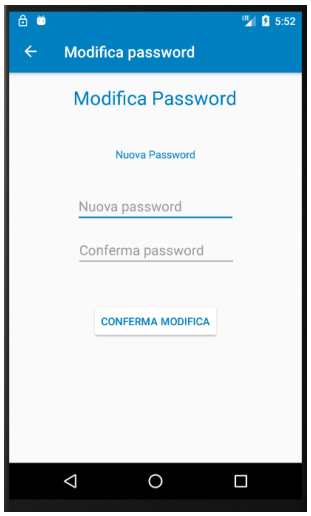

Figura 4.12. Schermata per la modifica della password

apre la schermata di Figura 4.13, nella quale viene richiesto di inserire un'immagine del prodotto che si sta per inserire, presa dal rullino o scattata sul momento, un nome, una descrizione che garantisca un grado di dettaglio sufficiente per il lettore, la categoria di appartenenza ed il prezzo virtuale ad esso associato.

Cliccando sul tasto in fondo allo schermo, verrà completata l'operazione di inserimento; a questo punto verrà visualizzato un avviso che comparirà sullo schermo per testimoniare il successo dell'operazione.

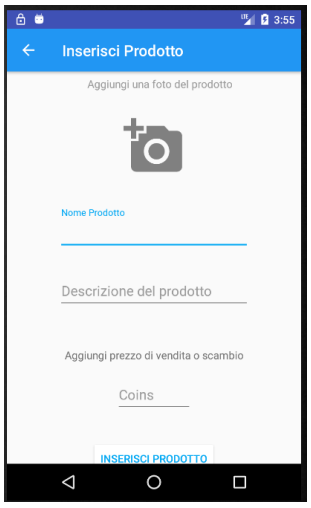

Figura 4.13. Schermata per l'inserimento di un oggetto nel catalogo

4.2.7 Miei oggetti

Una volta che sono stati inseriti uno o più oggetti, può essere importante per un utente vedere la lista di tutti gli oggetti inseriti (Figura 4.14). La lista è presente in "Miei oggetti", accessibile premendo sulla voce corrispondente presente nel menù laterale.

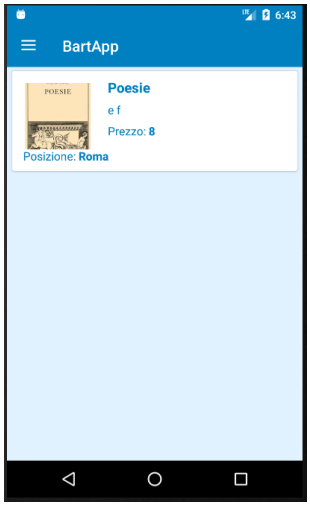

Figura 4.14. Lista degli oggetti dell'utente

La lista presentata in tale schermata è molto simile a quella presente nel catalogo; quindi ogni oggetto sarà descritto dalle stesse voci presentate in precedenza (nome, posizione etc.); anche qui ciascuna riga permetterà, al click su di essa, di entrare nella schermata dei dettagli del prodotto.

Questa volta, invece del tasto per formulare un'offerta, ragionevolmente ci sarà il tasto associato all'operazione di eliminazione di quest'oggetto dalla propria lista.

4.2.8 Offerte inviate e ricevute

Entrando nella sezione "Offerte inviate", è possibile visualizzare la lista delle offerte che l'utente ha formulato in precedenza. Chiaramente, questa volta la riga che descrive i singoli elementi è ben diversa da quella del catalogo; del resto, come descritto in Figura 4.15, ogni elemento della lista ha il compito di descrivere in modo dettagliato il baratto che è stato proposto. A caratterizzarlo, dunque, ci saranno, ovviamente, i due prodotti coinvolti, distinguibili dai colori rosso e verde, associati, rispettivamente, all'oggetto proposto e a quello richiesto. Anche qui, come nel catalogo, ogni oggetto sarà descritto da un'immagine, un nome, un prezzo ed una posizione.

L'unico tasto presente è quello di "Annulla offerta", che, come suggerisce il testo presente al suo interno, sarà preposto all'annullamento della proposta, con

Figura 4.15. Schermata delle offerte inviate dall'utente

conseguente eliminazione di questo elemento dalla lista, nonchè dalla lista del ricevente.

Ovviamente, oltre alla voce relativa alle offerte inviate, è presente anche la voce per le offerte ricevute (Figura 4.16). Ogniqualvolta un altro utente si interessa ad un prodotto dell'utente e formula un'offerta. Questa comparirà nella lista; quest'ultima sarà molto simile alla lista precedente, con due sostanziali differenze: la prima è che i due colori (rosso e verde) sono invertiti, e la seconda è che i tasti presenti questa volta sono due, uno per accettare ed uno per rifiutare l'offerta. Anche qui, al click, comparirà un avviso sullo schermo che chiederà conferma dell'utente, e quando l'utente fornirà tale conferma, procederà con l'eliminazione di questo scambio da questa lista e da quella delle offerte inviate da parte del mittente.

4.2.9 Riepilogo scambi

A questo punto, nel caso in cui l'utente abbia accettato l'offerta, comparirà una voce in questa sezione, "Riepilogo Scambi" (Figura 4.18). Tale voce riepilogherà tutte le offerte che l'utente ha accettato, e tutte quelle da lui inviate che sono state accettate dal destinatario, come fosse una sorta di "cronologia" degli scambi.

Caratterizzati dalla stessa coppia di colori delle offerte ricevute ed inviate, i due oggetti saranno le due informazioni fondamentali presenti in ogni elemento di questa lista.

A corredo del riepilogo dell'offerta, ci sarà anche un tasto per contattare l'altro utente, per raggiungere un accordo finale sullo scambio, ed un cestino nero stilizzato (Figura 4.17), che, una volta cliccato, cancellerà l'elemento ad esso associato alla lista, per permettere così una pulizia della storia degli scambi dell'utente.

Figura 4.16. Schermata delle offerte che sono state ricevute

Figura 4.17. Focus sul cestino per eliminare il riepilogo

4.2.10 Contattaci e Logout

Le ultime due voci presenti nel menù laterale sono “Contattaci” e “Logout” (Figura 4.19).

La prima permette di contattare via posta elettronica il team di sviluppo (o in futuro il Servizio Clienti, se necessario), nel caso in cui ci fossero delle domande da sottoporre a chi di dovere, o se si avessero perplessità, dubbi, suggerimenti, ed, infine, per riportare eventuali problemi che saranno presi in carico dal team di manutenzione del software.

La seconda, invece, realizza l’operazione di logout dell’account correntemente autenticato, riportando l’utilizzatore alla schermata di Login per permettergli l’accesso

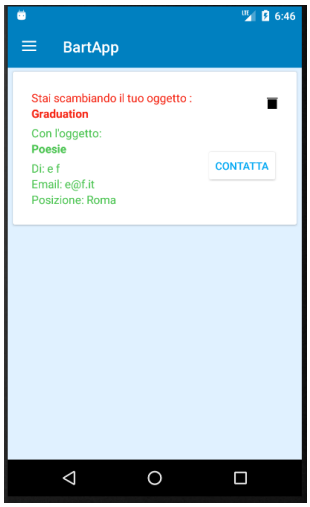

Figura 4.18. Schermata per il riepilogo degli scambi effettuati in precedenza

con un altro account.

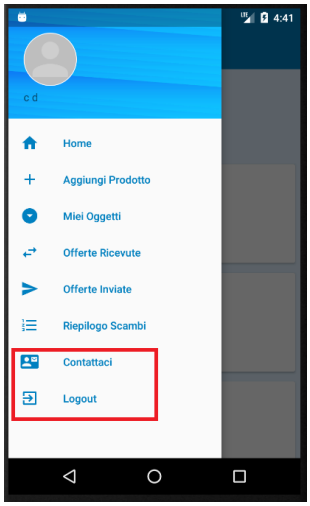

Figura 4.19. Focus sulle ultime due funzionalità offerte dalla nostra applicazione

5

Discussione in merito al lavoro svolto

In questo capitolo faremo il punto della situazione, con una discussione di larga veduta su tutto il lavoro che concerne lo sviluppo dell'applicazione.

Nello specifico, la prima sezione esplorerà tutte le conoscenze che sono state acquisite nella realizzazione della presente tesi. Nella seconda sezione faremo un'analisi del risultato ottenuto, presentando tutte le possibili criticità mostrate dall'applicazione, i suoi limiti ed eventuali idee di miglioramento.

5.1 Lezioni apprese

La seguente sezione si pone l'obiettivo di descrivere le principali conoscenze che sono state acquisite durante tutto il lavoro effettuato nella realizzazione dell'applicazione oggetto della presente tesi.

5.1.1 Android e programmazione mobile

Bartapp è un'applicazione nativa per il sistema operativo mobile Android. Come abbiamo già discusso nel Capitolo 1, Android (Figura 5.1) è il sistema operativo per dispositivi mobili più diffuso in tutto il mondo, ed è stata questa la principale caratteristica che ci ha spinti verso l'adozione di questo software. Un altro punto a favore è dato, sicuramente, dalla facilità ed agilità di sviluppo, garantita grazie a strumenti software come Android Studio e Firebase, di cui parleremo nei paragrafi successivi.

Sviluppare per Android, in realtà, ha richiesto che ci ponessimo di fronte a diversi bivi, come, ad esempio, la scelta se impostare la nostra applicazione puntando principalmente sulle funzionalità offerte oppure su una facilità d'utilizzo.

Una scelta di questo tipo ha richiesto delle considerazioni che abbracciavano diversi aspetti, e possiamo affermare che è stata preferita un'esperienza utente agevole soprattutto perché diverse ricerche hanno mostrato come l'utente medio (quindi il target a cui questa applicazione è rivolto) spenda davvero un tempo minimo di fronte allo schermo; quindi è ragionevole pensare che il nostro software debba necessariamente permettere anche all'utente più affrettato di accede con successo ad una qualunque delle possibili funzionalità.

Figura 5.1. Icona di Android

Quindi il lato progettuale ha evidenziato come la mappa del sito debba essere relativamente semplice, garantendo così un utilizzo rapido ed efficace, mostrando come l'applicazione riesca a passare da una schermata all'altra facilmente, evitando schermate annidate e permettendo all'utente di tornare alla schermata iniziale in modo quanto più rapido possibile, così da districarsi efficacemente tra le varie funzioni.

È stata oggetto di discussione anche la grafica offerta dalla nostra applicazione, ed anche in questo caso ci siamo soffermati sulla concorrenza, osservando come le classifiche delle applicazioni più vendute (consultabili nel Google Play Store) siano dominate da applicazioni con una veste grafica minimale, semplice e davvero pulita, in quanto, appunto, va ad avvicinarsi al concetto di applicazione mobile, cioè un qualcosa di semplice.

Dunque abbiamo evitato che la nostra applicazione potesse avere una grafica pesante o animazioni relativamente complesse, preferendo dei colori moderni e l'adozione del "Material Design" (Figura 5.2), che, come riporta il sito stesso, "è un sistema di progettazione creato da Google per aiutare i team a creare esperienze digitali di alta qualità per Android". L'adozione del Material Design è stata fondamentale per avere un'applicazione snella ma che fosse, comunque, quanto più d'impatto possibile per un utente interessato all'utilizzo del software.

Figura 5.2. Icona di Material Design

5.1.2 Android Studio

Lo strumento con cui abbiamo dato vita al software è Android Studio. Già illustrato ampiamente nel Capitolo 2, Android Studio (Figura 5.3) è l'ambiente di sviluppo

preferito dagli sviluppatori di applicazioni rivolte a dispositivi con sistema operativo Android.

Figura 5.3. Icona di Android Studio

Lo sviluppo con Android Studio è stato estremamente formativo; la sua estrema produttività e il suo essere adatto anche a sviluppatori meno esperti ci ha permesso di capire fin da subito quali fossero le sue peculiarità, così come quelle di Android stesso, e soprattutto le sue potenzialità, essendo un ambiente talmente flessibile da essere consigliabile sia per progetti semplici sia per progetti estremamente complessi (sempre con le dovute accortezze).

Come esempio della sua semplicità possiamo prendere lo sviluppo della schermata utente. In questo caso basta trascinare da una palette gli elementi grafici con cui si vuole popolare la schermata e poi modificare i parametri in modo estremamente intuitivo, ed eventualmente solo successivamente implementare le interazioni che possano intervenire tra gli elementi e l'utilizzatore, nonché le funzionalità associate.

Inoltre si ritiene di fondamentale importanza, almeno per i prossimi anni, la conoscenza di questo ambiente di sviluppo, dato che il mondo mobile e tutto ciò che ne concerne sta avendo un ruolo sempre più di rilievo nella tecnologia moderna, e la sua crescita esponenziale suggerisce come aver deciso di incentrare la presente tesi su un argomento di questo tipo sia una decisione che guarda al futuro, avendo ora dimostrato, dunque, la capacità di implementazioni software mobili; questa conoscenza è, inoltre, facilmente estensibile ai più svariati ambiti applicativi.

5.1.3 Firebase

La piattaforma a cui ci siamo appoggiati per alcune operazioni, anche essenziali, è Firebase (Figura 5.4).

Firebase è una piattaforma Mobile backend che consente di interfacciare applicazioni mobili e web app ad un cloud backend, fornendo allo sviluppatore servizi utili per la gestione degli utenti, storage, notifiche push ed altri strumenti di analisi e sviluppo. Il modello su cui si basa la piattaforma è relativamente recente poiché, appoggiandosi al cloud computing, fornisce uno servizio globale e uniforme per connettere client differenti offrendo una sincronizzazione dei dati in tempo reale.

Pertanto ci siamo serviti di Firebase per dar vita a tutto il servizio di autenticazione (login) presente nella prima schermata dell'applicazione. È proprio su tale servizio che sono riepilogati tutti i profili degli utenti che si sono registrati nel tempo. Per ciascuno di tali profili possiamo visualizzare (accedendo alla pagina principale di Firebase) il codice identificativo, la mail ed alcune opzioni tra cui la cancellazione, nel caso eventuale di comportamenti illeciti che meritano tale attività.

Figura 5.4. Icona di Firebase

Oltre al servizio di autenticazione, ci viene fornito un database con aggiornamento in tempo reale che permette all'applicazione di utilizzare una lista di oggetti che saranno, poi, quelli nel catalogo.

Ma non solo, in quanto abbiamo la possibilità di utilizzare diverse tabelle, questa volta asincrone, adottate per altre tipologie di dati (utenti, offerte, riepiloghi degli scambi).

Infine abbiamo usato Firebase anche come servizio di storage, inserendo al suo interno, servendoci del suo cloud, le immagini dei profili degli utenti, ed anche le immagini dei singoli prodotti.

Anche qui, seguendo un discorso analogo a quanto fatto a proposito di Android, la conoscenza di Firebase è sicuramente una caratteristica da non sottovalutare, data la sempre maggiore importanza che stanno avendo i servizi in cloud, cioè quell'insieme dei servizi che consentono di memorizzare i contenuti multimediali in uno spazio virtuale, rendendoli accessibili su ogni dispositivo e ovunque ci si trova, riuscendo, così, a limare l'annoso problema dello spazio di archiviazione limitato del dispositivo.

5.2 Revisione critica

Ciò che seguirà sarà un'analisi approfondita dell'applicazione sviluppata. Si ritiene necessario un lavoro di questo tipo in quanto evidenziare le principali criticità ed eventuali limiti può mostrare un forte spirito autocritico, oltre ad un senso di consapevolezza del lavoro che si è svolto, in totale trasparenza, senza sottovalutare anche l'apertura ad ogni forma di miglioramento che può aiutare l'inserimento dell'app nel mercato mobile.

5.2.1 Criticità, limiti e miglioramenti apportabili

Innanzitutto possiamo illustrare una prima forma di limite dal punto di vista concettuale. Un'app di questo tipo si predispone molto bene ad una trasformazione verso un'app di e-commerce, realizzabile semplicemente introducendo delle forme di pagamento garantite, associando, quindi, delle modalità per completare delle transazioni di tipo monetario.

Quanto detto non va a stravolgere il lavoro svolto, in quanto il baratto, e il suo utilizzo come forma di acquisizione di prodotti rimarrano, e potranno essere preferiti da alcuni utenti che si vogliono disfare di oggetti inutilizzati. Ma una forma di pagamento di certo allargherebbe il bacino d'utenza dell'app, rendendola un prodotto più fruibile, e quindi più appetibile.

Nel capitolo precedente, inoltre, è stato fatto un accenno alla possibilità che associare ad ogni prodotto la sua posizione possa renderlo più attraente per alcuni utenti piuttosto che per altri, in funzione della distanza, appunto. Quindi potremmo sicuramente potenziare questa caratteristica, introducendo, nella schermata principale dell'app, un filtro più complesso, che, oltre al nome e alla categoria, permetterebbe di filtrare prodotti in base anche alla distanza, selezionabile ovviamente dall'utilizzatore. In questo modo si permetterebbe all'utente di vedere sullo schermo solo prodotti che si trovano entro un determinato range di distanza da sé.

Un altro importante limite che caratterizza la nostra app è il vincolo ad offire un solo prodotto in cambio di un altro. Sicuramente una proposta che vede in cambio di un solo oggetto un insieme di più oggetti può rendere più interessante una offerta, ed aprire a nuove possibilità di scambi, rendendo sempre più elastica e versatile la nostra app.

Oltre a ciò, non possiamo non menzionare la sezione del profilo ed il fatto che essa consti di funzionalità pressocché elementari, ma di certo essenziali e funzionali.

Ma questo non significa che una revisione approfondita di quella sezione non possa porre le basi per un miglioramento decisamente corposo. Le idee partono già dalla fase di registrazione, durante la quale possiamo inserire più criteri di sicurezza nella scelta della password, se non anche realizzare una registrazione servendoci di social network, permettendo ad un utente che ha già un account in un social network di usarlo anche per Bartapp, evitando le sempre più tediose ed evitabili procedure di registrazione.

Inoltre, a seguito di un'analisi delle sezioni dei profili delle principali app concorrenti, si può notare l'assenza di una qualunque procedura di recupero di una eventuale password dimenticata. Attraverso tale procedura, servendosi della e-mail inserita in fase di registrazione, si può ottenere una nuova password, evitando così la perdita dell'account e di tutti gli scambi ad esso associati.

Un'ultima implementazione migliorabile potrebbe, sicuramente, essere tutta la fase conclusiva di uno scambio; la nostra applicazione è stata impostata per delegare in modo totalmente libero la buona riuscita di uno scambio, mediante il tasto che permette l'invio delle mail all'altro utente interessato al baratto.

Ma la comunicazione tramite e-mail, non è sempre la via prediletta dagli utenti, che, eventualmente, vogliono evitare di intasare la casella di posta elettronica con messaggi di scambi, nel caso di utenti particolarmente insistenti. Quindi sarebbe sicuramente una idea eccellente l'introduzione di una chat, accessibile una volta raggiunto un accordo, per permettere agli utenti di concludere scambi in modo decisamente più moderno, rapido e funzionale, e per confinare le interazioni con l'app all'interno della stessa, senza e-mail che richiedono altri client di cui il dispositivo deve necessariamente essere fornito.

6

Conclusioni

In questa tesi è stata presentata un'applicazione per dispositivi Android che permette di scambiare oggetti tra utenti. Nella tesi sono state trattate sia la sua progettazione che la sua implementazione.

Per garantire la massima contestualizzazione al lettore, è stata fatta inizialmente un'introduzione al concetto di baratto, soffermandoci sulla sua definizione e sulla pratica in sé, ponendo, in seguito, particolare attenzione alla sua storia, mostrando come una pratica così antica sia ancora oggi sicuramente valida ed adottabile, in quanto evolutasi in varie forme decisamente più moderne. Pertanto da qui nasce spontaneamente la divisione della storia del baratto in due macrosezioni:

- "Tempi antichi"
- "Tempi moderni"

Una suddivisione di questo genere è stata realizzata proprio per mostrare la sua evoluzione ed il modo in cui sia rimasta nel corso degli anni, sotto diverse spoglie.

Successivamente è stato presentato un quadro complessivo di cosa sia Android, la sua storia e la sua struttura interna, ponendoci dunque dal punto di vista prima di un semplice consumatore e poi di uno sviluppatore.

Fondamentalmente un capitolo del genere ha lo scopo di illustrare al lettore il sistema operativo in sé ed i suoi vantaggi, ed in particolar modo quali possano essere le motivazioni che ci hanno spinti a sviluppare in Android piuttosto che facendo riferimento alla concorrenza, sia vestendo i panni di un cliente, adducendo principalmente la diffusione, sia di uno sviluppatore di software, menzionando gli strumenti utilizzabili, come Firebase ed Android Studio.

Il capitolo successivo ci ha permesso di mostrare in modo quanto più dettagliato possibile tutta la fase di progettazione e il lavoro che c'è stato dietro a determinate decisioni, per far capire al lettore il motivo per cui l'applicazione si mostra in un certo modo ed è caratterizzata da una determinata struttura ed impostazione.

Dunque ad una analisi dei requisiti funzionali e non, segue la mappa del sito, l'elenco dei casi d'uso ed, infine, tutta la struttura del database di cui ci siamo serviti, ponendo l'accento sui dati in esso contenuti, giustificando le scelte progettuali compiute, sia per quanto riguarda la struttura dati sia per ciò che concerne il codice dell'applicazione stessa.

Una volta terminata la spiegazione della fase progettuale, ci si sofferma sull'implementazione di Bartapp, inizialmente mostrando funzione per funzione quale sia il codice che realizza quanto offerto dall'app, spiegando in modo esauriente il funzionamento delle singole istruzioni, riga per riga, per garantire la massima comprensione al lettore.

Di seguito ha una particolare rilevanza la presenza di un manuale utente, in cui le funzioni non sono mostrate attraverso il codice, ma spiegate punto per punto, indicando al lettore tutte le istruzioni da eseguire per portare a termine determinate azioni, corredando il testo di una serie di immagini che rappresentano l'applicazione stessa per ottenere la massima comprensione.

Infine, la tesi si conclude con una discussione in merito al lavoro svolto, in cui inizialmente si presentano tutte le lezioni che sono state apprese nel corso della lunga fase di sviluppo dell'app e di tutto ciò che è stato acquisito. A tal proposito abbiamo citato tutte quelle caratteristiche di cui deve disporre una qualunque applicazione mobile. Abbiamo illustrato il concetto stesso di "Mobile App Development". Infine, abbiamo introdotto una sezione per descrivere quanto appreso riguardo alla piattaforma di "Firebase", per la strutturazione delle tabelle dei dati, spiegando in quali occasioni ci siamo serviti di tale strumento. Solo in un secondo momento, con un forte spirito critico, è stata realizzata un'analisi del lavoro svolto, portando alla luce le criticità che la nostra applicazione mostra, oltre ai suoi limiti, e dunque quali possano essere tutti quei miglioramenti per renderla più appetibile, e per ottenere un qualche riscontro positivo nello store delle applicazioni mobili.

Ovviamente la presenza di un paragrafo così critico deve fungere da spia al lettore per la forte versatilità del lavoro svolto, mettendo alla luce la possibilità, se non il sincero interesse, di continuare a lavorare su questo progetto, non considerandolo come un lavoro completato, ma, viceversa, come solo una base che è stata fondata per permettere all'app di rivaleggiare con i suoi principali concorrenti. Del resto abbiamo citato una lunga serie di correzioni ed aggiunte che possono essere apportate all'applicazione, e grazie all'ambiente di sviluppo utilizzato, e a tutti gli strumenti di cui ci siamo serviti, sarà sicuramente possibile in modo agevole dar vita a tutti questi miglioramenti.

Inoltre, un lavoro di questo tipo può avere anche una visione di più ampio respiro. Esso mostra, infatti, come sia stata possibile l'implementazione di un'applicazione mobile, e quindi ci estende anche alla possibilità di realizzarne di nuove, a seconda delle innumerevoli necessità di chi le richiede, avendo mostrato la capacità di lavorare in team, lavorare con un IDE, maneggiare codice Java e realizzare tutta una fase progettuale che anticipa quella implementativa.

Riferimenti bibliografici

1. G. Aranguena and D. Jegerson. *I pagamenti elettronici. Dal baratto ai portafogli digitali*. GoWare, 2016.
2. R. Ballard. *Relational Databases for Agile Developers*. CreateSpace Independent Publishing Platform, 2017.
3. R. Bellotti. *Il cloud computing nelle imprese e nella pubblica amministrazione. Profili giuridico-economici e applicazioni nel campo sanitario*. Giuffre, 2019.
4. E. Ercolani Cocchi. *Dal baratto all'euro. Storia della moneta dalle origini ai giorni nostri*. Editoriale Olimpia, 2003.
5. T. Hayden, T. Webster, and B. Sansone. *E-commerce per dispositivi mobili. La rivoluzione degli acquisti con tablet e smartphone*. Tecniche Nuove, 2015.
6. K. Hoffman and S. Dalin. *The Art of Barter: How to Trade for Almost Anything*. Skyhorse Publishing, 2010.
7. C. Humphrey. *Barter, Exchange and Value: An Anthropological Approach*. Cambridge University Press, 1992.
8. M. Jenkins and S. Kitamura. *The History of Money: From Bartering to Banking*. Candlewick, 2015.
9. P. Kotler and G. Stigliano. *Retail 4.0. 10 regole per l'era digitale*. Mondadori Electa, 2018.
10. S. Kumar. *Mastering Firebase for Android Development*. Packt Publishing, 2018.
11. F. Marinuzzi and M. Liciani. *Basi di dati e Big Data: come estrarre valore dai propri dati*. Youcanprint, 2016.
12. S. Mishra. *Android Practical Books, Android Phones And Tablets Development: Android,SQL Lite, Firebase And Unity*. Independently published, 2019.
13. M. Passalacqua. *Swap! Guida all'Arte del Baratto*. Ultra, 2013.
14. A. Petrov. *Database Internals: A Deep Dive into How Distributed Data Systems Work*. O'Reilly and Associates Inc, 2019.
15. T. Raffles. *The History of Java*. University of Michigan Library, 2009.
16. G. Reese. *Cloud computing. Architettura, infrastrutture, applicazioni*. Tecniche Nuove, 2010.
17. N. Ruparelia. *Cloud Computing (The MIT Press Essential Knowledge Series)*. The MIT Press, 2016.
18. N. Smyth. *Firebase Essentials: Android Edition, Second Edition*. Createspace Independent Pub, 2017.
19. G. Tillman. *Usage-Driven Database Design: From Logical Data Modeling through Physical Schema Definition*. Apress, 2017.
20. J. Work. *The Barter (The Barter And Reckoning Series Book 2)*. CreateSpace, 2013.

Ringraziamenti

Mi sembra davvero incredibile essere arrivato fin qui, a scrivere questa parte conclusiva della mia tesi, in cui ringraziare chiunque sia stato al mio fianco durante questo percorso universitario.

Di persone da ringraziare ce ne sarebbero a decine, ma prima fatemi dire quanto io sia in primis fiero di me stesso: per una volta nella mia vita ho preso una decisione più di cuore che di testa, quella di seguire la mia passione nel mondo dell'informatica, e frequentare una Facoltà notoriamente impegnativa come quella di Ingegneria.

Di difficoltà ne ho avute davvero tante, quasi innumerevoli, e chiunque mi è stato vicino in questi tre anni lo può testimoniare; ma, nonostante questo, non mi sono mai arreso, ho sempre lavorato a testa bassa, con tanti sacrifici e un dispendio di energie che non sapevo neppure di avere.

In questi tre anni ho assunto una consapevolezza di me stesso davvero ottima ed ho visto una crescita in me stesso al punto che non vedo l'ora di mettermi in gioco nel mondo del lavoro, una volta terminato il percorso magistrale. Per questo devo necessariamente ringraziare ogni professore che mi ha accompagnato in questo corso di studi, dandomi un bagaglio di capacità e conoscenze che voglio assolutamente sfruttare nel futuro.

Vorrei spendere alcune parole in particolare per il Professore Domenico Ursino, che ringrazierò infinitamente per la sua disponibilità, pazienza e chiarezza sia a lezione sia nelle direttive da seguire durante la stesura della presente tesi, e per il rapporto sempre positivo che sono riuscito ad avere con lui. Ovviamente i ringraziamenti vanno anche al suo collaboratore Enrico Corradini, tempestivo nel rispondere a qualunque domanda o dubbio che io sottoponessi, ed anche ai miei compagni di corso Nicola Mori e Matteo Vitellozzi, che mi hanno aiutato nello sviluppo della presente applicazione mobile.

Lasciatemi ringraziare anche tutti i miei amici compagni di corso, che mi hanno davvero aiutato, dandomi quel divertimento e quel quotidiano ambiente positivo ed amichevole che è stato di fondamentale importanza per non crollare nei momenti di difficoltà, strappandomi un sorriso ogni giorno. Vi devo davvero tanti caffè, vi aspetto al bar dell'università!

Ovviamente ringrazierò tutti i miei più cari amici, che sono stati come una spalla per me, standomi sempre accanto e regalandomi momenti di gioia e divertimento, senza i quali, forse, non sarei arrivato fin qui. Vi voglio bene, a tutti quanti.

Ringrazio anche tutta la mia famiglia, i miei zii ed i nonni, che sono stati per me un supporto costante in tutto questo tempo; so che ponevate fiducia in me e spero di avervi resi tutti fieri.

Ed, infine, un ringraziamento speciale va ai miei genitori, che erano, sono, e saranno sempre la mia forza, che hanno fatto un numero enorme di sacrifici per permettermi di studiare, e sono stati il mio principale stimolo a non arrendermi mai. Mi hanno visto in ogni condizione, da quella più triste, dopo una bocciatura, a quella ansiosa e a tratti insopportabile prima di un esame, e quella euforica, dopo aver superato brillantemente un esame che sembrava impossibile. A voi devo davvero tutto, siete due genitori fantastici; non mi avete mai fatto mancare nulla, e so di avervi reso immensamente fieri.

Una menzione speciale va a mio Nonno Marcello, che è stato capace di accendere una scintilla in un me insicuro, in una giornata d'estate di ormai 4 anni fa, quando mi strinse delicatamente un braccio e mi guardò negli occhi, dicendomi che sarei stato capace di ottenere questa laurea. Una frase semplice, ma che per me fu di una importanza smisurata, e che è stata ciò che mi ha spinto ad iscrivermi a questa Facoltà. Nonno, ce l'ho fatta!