



Dipartimento di Ingegneria Informatica Modellistica
Elettronica e Sistemistica

Corso di Laurea Magistrale in Ingegneria Informatica



Relazione per il Progetto di Sistemi Distribuiti

Studente:
Domenico Costantino
matr. 189168

Docente:
Prof. Domenico Talia
Ing. Loris Belcastro

Indice

1	Introduzione	2
2	Scenari di utilizzo	3
2.1	Funzionalità per gli utenti	5
2.2	Funzionalità per i negozianti	7
2.3	Alternative e possibili competitors	8
3	Telegram APIs	10
3.1	Telegram: storia, caratteristiche e competitors	10
3.2	Telegram Bot API	13
3.2.1	Come si crea un bot	16
3.3	La libreria TelegramBots	17
4	Scelte progettuali	18
4.1	Componenti del progetto	19
4.1.1	Il database Postgresql	22
4.2	Il PaaS usato: Heroku	22
5	Considerazioni finali e possibili scenari evolutivi	27

Capitolo 1

Introduzione

Supermarket Deal Bot, come si può intendere dal nome, è un **bot Telegram** pensato per assistere l’utente con la spesa quotidiana, velocizzando il confronto delle offerte presenti nei vari volantini e indirizzandolo verso quelle migliori.

I punti di forza del bot sono l’immediatezza e la semplicità di utilizzo, in un ambiente già conosciuto e in cui l’utente si sente a proprio agio. Tramite lo scambio di messaggi testuali e l’interazione tramite appositi button l’utente può usufruire delle seguenti *funzionalità*:

- Creazione di una lista della spesa interattiva.
- Confronto delle migliori offerte presenti nei volantini dei supermercati vicini, proponendo dei consigli basati sulla lista dell’utente.
- Indicazione della posizione del negozio da raggiungere.
- Comandi utili all’aggiornamento delle offerte e delle informazioni del punto vendita per i negozi che aderiscono all’iniziativa.

L’idea è quella di *supportare l’utente in tutte le fasi della spesa* (stilare la lista, decidere dove andare ad acquistare, recarsi sul posto e acquistare gli articoli) mantenendo una interazione semplice, intuitiva e veloce. Naturalmente cercando di far risparmiare tempo e contanti dato che verranno selezionate le offerte presenti nei volantini, in accordo alla lista stilata dall’utente.

Dopo questa breve introduzione, nel secondo capitolo, verranno presentati dei casi d’uso per spiegare nel dettaglio l’interazione con l’utente e approfondire le funzionalità. Verranno inoltre presentati delle soluzioni alternative (competitors) all’idea progettuale. Nel terzo capitolo si parlerà di Telegram, delle API che fornisce e delle librerie relative al sistema di messaggistica usate nel progetto. Il quarto capitolo è quello più tecnico in cui verranno esposte le scelte progettuali e presentate le principali tecnologie usate (Heroku, PostGreSQL e Google Maps). Infine il quinto capitolo contiene delle riflessioni finali sui punti di forza, debolezze e possibili scenari evolutivi.

Capitolo 2

Scenari di utilizzo

L’interfaccia utente che offre Telegram è simile a quella di molti altri sistemi di messaggistica istantanea, come WhatsApp e Messenger. È molto intuitiva e di semplice utilizzo. Presenta una schermata iniziale nella quale è possibile scorrere tutte le conversazioni, e cliccando su una di esse si apre la chat vera e propria. Naturalmente sono sempre richiamabili la funzione di ricerca di chat ed utenti e il menu delle impostazioni in cui si possono modificare tanto le proprie informazioni personali quanto tutte le impostazioni sia grafiche sia di funzionamento più avanzato dell’applicazione. Due schermate di esempio, per la versione mobile (a sinistra) e per la versione desktop sono presenti in Figura 2.1.

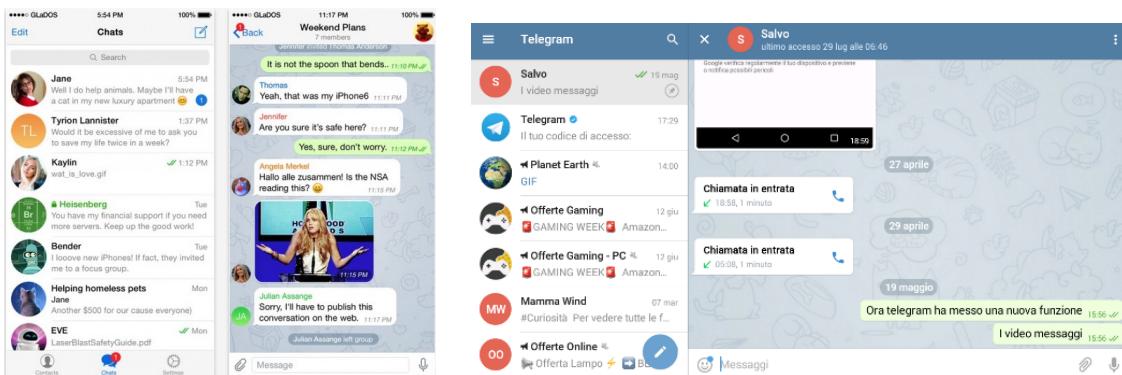


Figura 2.1: Schermate di esempio di Telegram

Naturalmente per poter utilizzare il bot è necessario innanzitutto ricercarlo, usando il nominativo **@Supermarket_deal_bot** e iniziare a interagire usando il comando `/start`, come mostrato nella seguente Figura 2.2.

Dopo il primo messaggio interattivo, in cui verrà presentato il bot, sarà possibile richiamare uno dei comandi dall’apposito menu a tendina richiamabile in qualsiasi momento premendo sul comando "/" (o digitando "/" da tastiera).

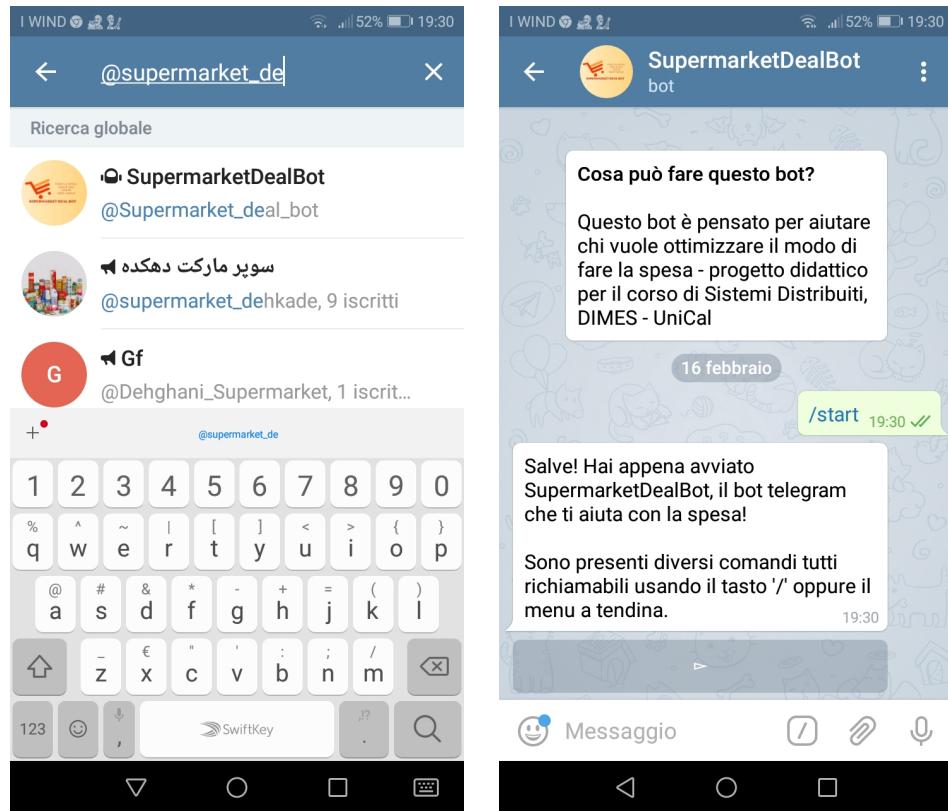


Figura 2.2: Ricerca del bot e avvio della conversazione

I comandi presenti sono sette, nel dettaglio:

- **start** avvia il bot e richiama la schermata iniziale.
- **lista** mostra l'attuale lista della spesa dell'utente.
- **aggiungi** permette di aggiungere un prodotto da acquistare alla lista.
- **svuota_lista** utile ad azzerare la lista dell'utente
- **spesa_ottimizzata** la principale funzionalità del bot, consiste in una procedura guidata che assiste l'utente durante tutta la spesa.
- **negozianti** richiama le funzionalità riservati ai negozi convenzionati.
- **info** informazioni e credits.

Andiamo ora a descrivere dettagliatamente l'interazione con l'utente, nel caso di creazione modifica della lista e della spesa guidata.

2.1 Funzionalità per gli utenti

Le funzionalità che vengono forniti agli utenti sono principalmente due: la possibilità di creare e usare una lista della spesa interattiva e il percorso guidato tra le offerte presenti nei volantini dei vari punti vendita. Lo use-case diagram in Figura 2.3 illustra a grandi linee il funzionamento del bot per quanto riguarda l’interazione con l’utente.

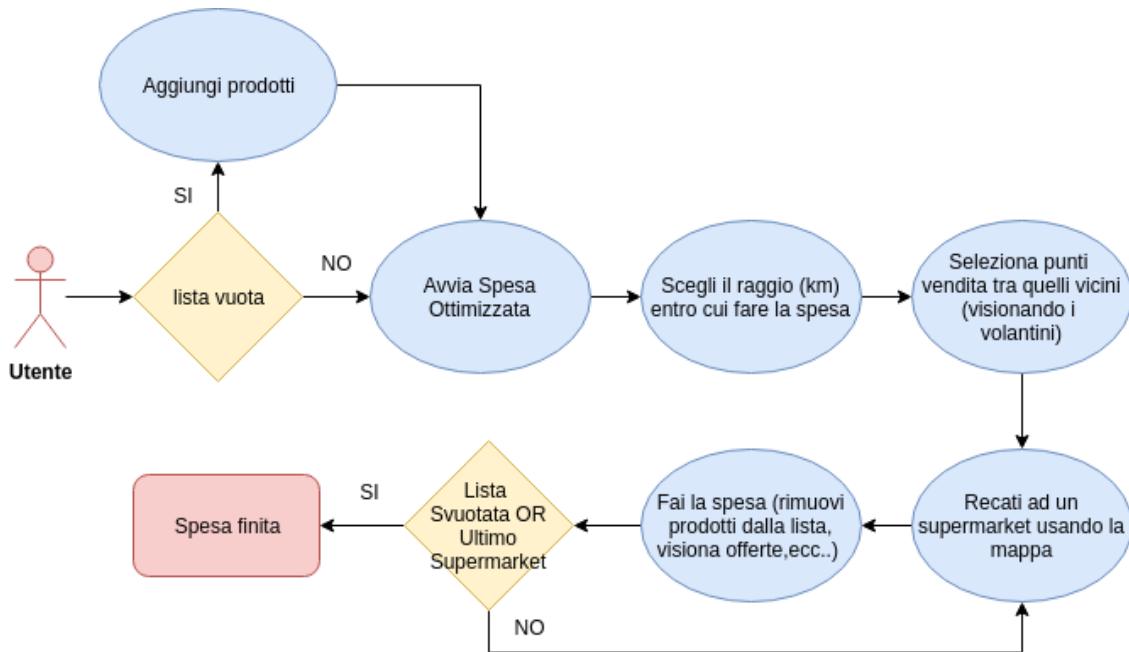


Figura 2.3: Use-case Diagram Utente

La gestione della lista interattiva è molto semplice. Al richiamo del comando **/aggiungi** si aprirà una nuova tastiera (*CustomKeyboard*) dotata di bottoni per selezionare le categorie di profotti che sarà possibile memorizzare nella lista. Attraverso il comando **/lista** si potrà visionare la lista interattiva e premendo su uno qualsiasi dei prodotti presenti a schermo (*InlineKeyboard*) sarà possibile eliminarlo. Nella Figura 2.4 sono mostrate le schermate con la lista e l’aggiunta di un prodotto.

Il comando **/svuota_list** come già detto, permette di azzerare la lista dell’utente.

La **gestione guidata della spesa** invece è più articolata. Innanzitutto richiede che sia presente una lista della spesa (non vuota) e poi, per evitare di interrompere l’interazione necessaria al bot, viene richiesto all’utente di *non richiamare nessun altro comando* mentre si usa questa funzione.

L’interazione è arricchita da diverse Custom ed Inline Keyboard (bottoni a schermo (sul messaggio come per la lista) o su tastiera virtuale) e inizia richiedendo la distanza massima (raggio, in linear retta) dalla posizione dell’utente entro la quale

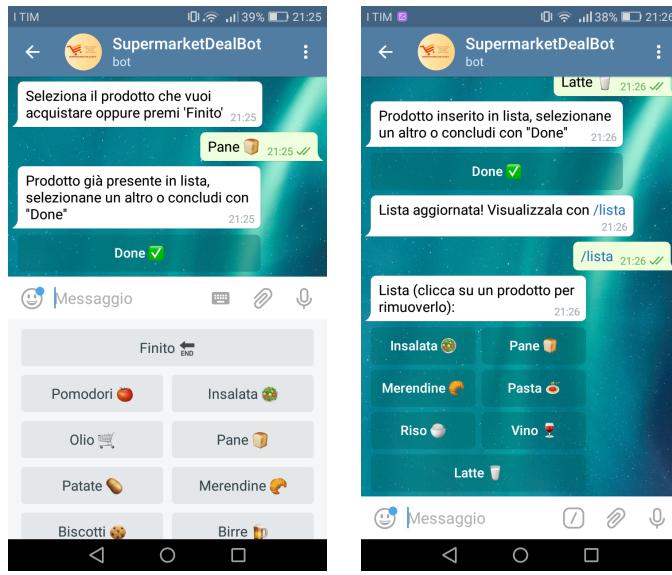


Figura 2.4: Comandi "aggiungi" e "lista"

ricercare eventuali punti vendita. Vengono fornite tre possibilità: 5, 10 oppure 20 Km. Si richiede poi all’utente di inviare la propria posizione mediante un apposito pulsante e di scegliere (potendo visionare le offerte di ogni punto vendita relative ai prodotti in lista) una lista di supermarket tra quelli presenti nel raggio scelto. L’ordine di scelta è importante in quanto il bot guiderà l’utente ai vari supermercati in base alla sequenza indicata.

Una volta finita questa parte preliminare si può passare alla fase attiva, quindi il bot invierà la posizione del primo supermarket (Figura 2.5), che permetterà, cliccandoci sopra di aprire l’apposito itinerario dentro l’applicazione di Google Maps.

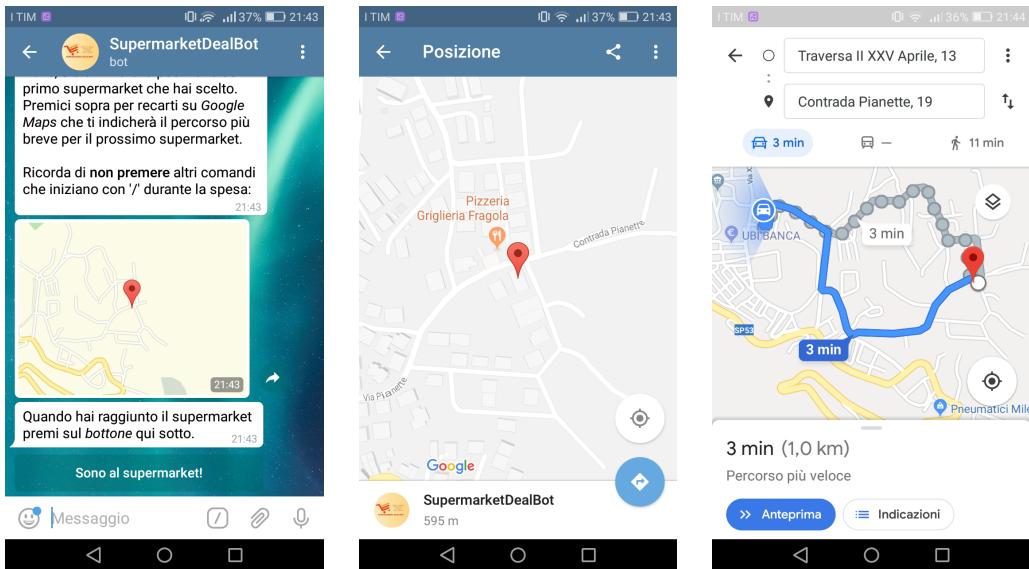


Figura 2.5: Integrazione con Google Maps

Una volta raggiunto il punto vendita sarà possibile visionare le offerte (come in precedenza, anche quelle degli altri punti vendita vicini), eliminare prodotti dalla lista oppure ripristinare la lista iniziale in caso di errore nella rimozione di un prodotto. Si può quindi passare al prossimo supermarket tra quelli elencati.

Nel caso nel prossimo supermarket non siano presenti offerte per i prodotti selezionati, e ci siano ancora supermarket tra quelli elencati, si può decidere di non visitare il supermarket e andare direttamente al successivo.

Quando la lista sarà svuotata oppure si è già acquistato nell'ultimo supermarket l'interazione finisce premendo su un apposito bottone.

2.2 Funzionalità per i negozianti

Per quanto riguarda i responsabili dei punti vendita sono state pensate delle funzioni appropriate che permettano di:

- **Creare** un nuovo punto vendita tra quelli presenti.
- **Modificare** i dati di un punto vendita.
- **Aggiornare le offerte** relative ad un supermarket.

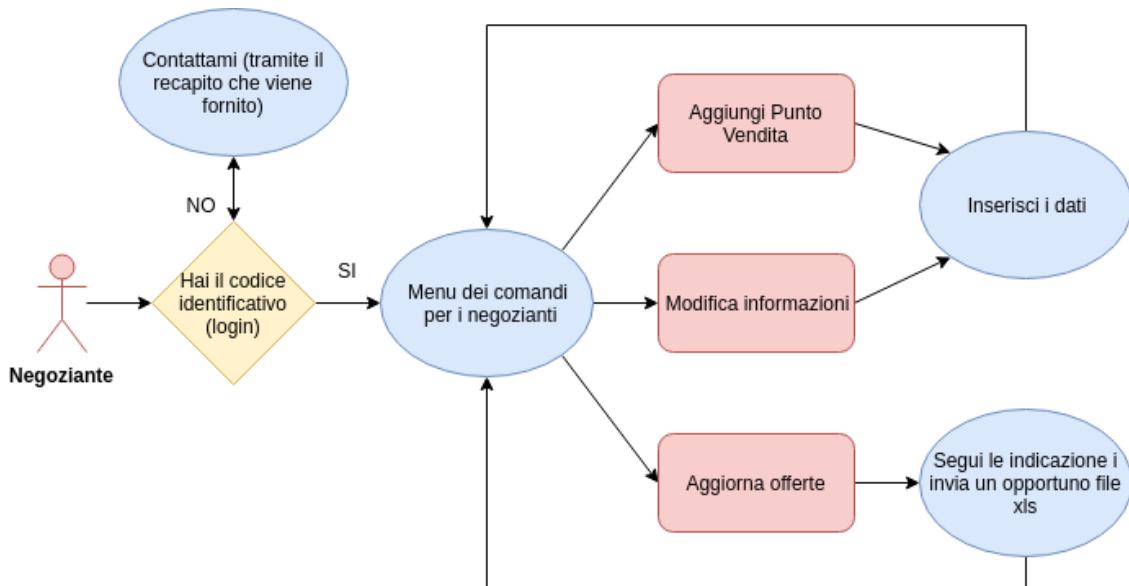


Figura 2.6: Use-case Diagram Negozianti

È importante sottolineare che ogni utente (negoziante) può avere associato un solo punto vendita.

Per l'autenticazione si usa il **chatId** (un intero di 9 cifre) visibile al bot e relativo alla conversazione con quell'utente. Quindi per ottenere l'accesso a questi comandi

un utente che voglia diventare "negoziante" deve contattarmi affinchè io possa fornirgli il codice identificativo. Questo meccanismo basilare è stato scelto in quanto Telegram non fornisce meccanismi di autenticazione come fanno invece applicazioni e siti web.

La parte relativa all'aggiornamento delle offerte consiste in una breve interazione in cui viene inviato dal bot un documento xls come *template* da seguire per creare un opportuno file xls contenente le nuove offerte. Questo nuovo file, una volta inviato al bot permetterà a quest'ultimo di aggiornare il database delle offerte, eliminando quelle già presenti per il supermarket associato all'identificativo. Possibili scenari di interazione con i negozianti sono mostrati nella Figura 2.7.

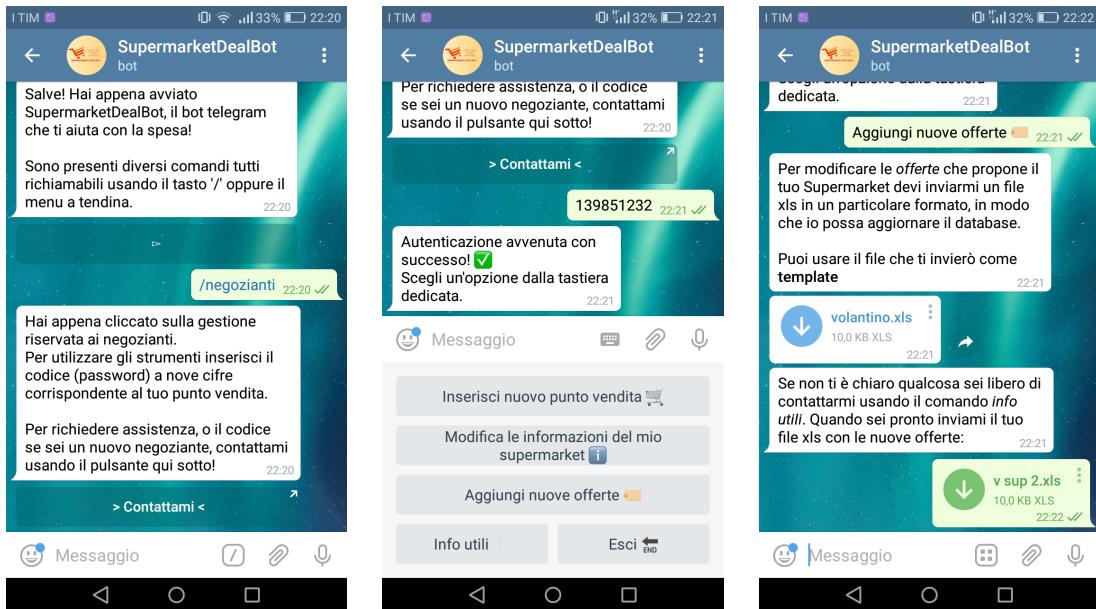


Figura 2.7: Interazione con i negozianti

2.3 Alternative e possibili competitors

Da una veloce ricerca non sono risultati presenti altri bot Telegram con funzionalità di aggregatori di offerte per i supermercati fisici. Invece sono presenti diversi bot che effettuano servizi di price tracking e bot per la creazione di liste della spesa.

Da questo punto di vista i servizi web e le applicazioni iOS e Android che permettono di ricercare offerte o creare liste condivise o personalizzate sono molteplici. I più noti sono:

- **PromoQui:** è un'applicazione con motore di ricerca integrato, per trovare in pochi istanti le offerte più convenienti nei negozi vicino casa; non solo supermercati ma anche outlet di abbigliamento, store di bricolage e giardinaggio, negozi al dettaglio e tutto quello che ti può servire. Puoi gestire una lista della

spesa personalizzata, salvare i dettagli dei volantini, cercare un punto vendita specifico, ecc.

- **DoveConviene:** è una delle app più scaricate del momento, apprezzata dai consumatori soprattutto per il pratico shopping alert, che tiene aggiornati costantemente sul lancio di nuovi prodotti, delle promozioni flash e sulle date di scadenza delle offerte salvate tra i preferiti, per ricordarti quanto tempo hai ancora a disposizione. [7]
- **Out Of Milk Shop List:** che permette di creare liste della spesa generiche e liste multiple suddivisibili anche per categorie. È possibile anche sincronizzare le liste con più dispositivi, in modo da condividerle ad esempio con un familiare che, e in caso si dimentichi qualcosa si potrà effettuare una modifica aggiornando le varie liste in tempo reale. [8]

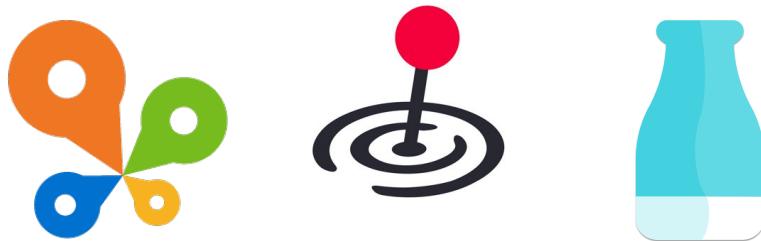


Figura 2.8: Possibili competitori: PromoQui, DoveConviene, OutOfMilk

La peculiarità di un bot rispetto ad un'applicazione, oltre all'immediatezza, è la totale assenza di programmazione del client, in quanto non è necessario scaricare nessun eseguibile o installare alcunchè. La computazione è tutta ad appannaggio del server.

Capitolo 3

Telegram APIs

Per lo sviluppo del bot ci si è appoggiati ad una libreria che ha permesso di programmare la logica di business in Java, in maniera semplice e trasparente. Per introdurla occorre, però, prima parlare in generale di *Telegram*, delle API che fornisce per lo sviluppo e la creazione di bot (e applicazioni), e infine della particolare libreria usata.

3.1 Telegram: storia, caratteristiche e competitors

Telegram è un **servizio di messaggistica istantanea** basato su cloud ed erogato senza fini di lucro dalla società *Telegram LLC*, una società a responsabilità limitata con sede nel Regno Unito, fondata dall'imprenditore Russo Pavel Durov. I client ufficiali di Telegram sono distribuiti come software libero per *Android*, *GNU/Linux*, *iOS*, *MacOS*, *Windows NT* e *Windows Phone*. Caratteristiche di Telegram sono la possibilità di stabilire conversazioni tra due o più utenti, effettuare chiamate vocali cifrate "punto-punto", scambiare messaggi vocali, videomessaggi, fotografie, video, stickers e file di qualsiasi tipo grandi fino a 1,5 GB. È possibile usare telegram anche direttamente in versione Web da browser, senza scaricare ed installare nessun client, rinunciando, però, a diverse funzionalità.

Telegram è stato fondato nel 2013 dai fratelli Nikolai e Pavel Durov, i fondatori del social network russo VK. Nikolai ha creato il nuovo protocollo MTProto sul quale Telegram è basato mentre Pavel ha fornito sostegno finanziario e infrastrutture attraverso il suo fondo denominato Digital Fortress.

Il 24 marzo 2014 Telegram ha annunciato di aver raggiunto i 35 milioni di utenti attivi mensilmente e 15 milioni di utenti giornalieri, il 15 settembre 2015 c'erano 60 milioni di utenti attivi al mese. Nel mese di febbraio 2016 si sono raggiunti i



Figura 3.1: Logo di Telegram

100 milioni di utenti attivi mensili, 350 000 nuovi utenti al giorno e 15 miliardi di messaggi scambiati giornalmente. A febbraio 2018 i download giornalieri dell'app su Android e iOS si assestavano rispettivamente a oltre 500 000 e circa 100 000.[9]

Questi numeri da capogiro sono destinati ad aumentare in quanto a fine marzo 2018, sul blog ufficiale, è stato annunciato di aver superato il traguardo di *200 milioni di utenti attivi mensilmente*. [6]

I client ufficiali di Telegram sono **software libero**. Il codice sorgente del lato server invece non è stato rilasciato. Ciò significa che non è possibile verificare la segretezza delle conversazioni effettuate sul cloud e non è possibile fornire in proprio questo servizio di messaggistica (ad esempio da un proprio server). I messaggi inviati sono salvati sul cloud di Telegram, così da garantire la sincronizzazione istantanea. Il risultato consente all'utente di poter accedere ai messaggi da diversi dispositivi contemporaneamente, inclusi tablet e computer.

Quando si registra, l'utente può scegliere se permettere ad altri di cercarlo inserendo il nickname scelto durante la registrazione preceduto da @ nella barra di ricerca. Questa funzione facoltativa consente l'identificazione di un utente senza conoscere necessariamente il suo numero di telefono. Dalla versione 3.9, i messaggi inviati si possono modificare fino a due giorni dall'invio.

Le chat sono di due tipi:

- **Chat cloud** (classiche): la chat utilizza *una cifratura client-server*, ovvero è cifrata dal dispositivo fino ai server di Telegram e viceversa. Quindi la conversazione dovrebbe restare salvata in maniera cifrata sui server di Telegram per poter essere sincronizzata fra più dispositivi (es. smartphone, tablet, PC). La chat classica permette l'invio di messaggi di testo, messaggi vocali, videomesaggi, posizione GPS attuale, coordinate GPS sulla mappa, contatti e qualsiasi tipo di file di dimensione massima di 1,5 GB.
- **Chat segrete**: la chat utilizza una *cifratura end-to-end*, ossia è cifrata fra i due dispositivi coinvolti nella conversazione. Di conseguenza la conversazione non rimane salvata sui server di Telegram. Se da un lato queste conversazioni sono notevolmente più sicure dal punto di vista della privacy, da un lato occorre sapere che di conseguenza la chat non può essere sincronizzata fra più dispositivi ma si può visualizzare solo dal dispositivo dal quale è stata avviata. Le chat segrete attualmente non funzionano sulla versione Web e su Telegram Desktop, mentre sono fruibili sui client mobile, vari client alternativi e su Telegram per Mac.

Oltre agli utenti su telegram sono presenti diverse "entità" che permettono di perseguire diversi scopi comunicativi. Nel dettaglio esistono:

- **I Gruppi** che possono contenere fino a 200 000 membri, in cui è possibile impostare amministratori con permessi personalizzati. Sono naturalmente presenti molte impostazioni dedicate appositamente ai gruppi. Questi possono essere resi pubblici impostando un username, in tal modo sarà possibile trovarli dalla ricerca e leggere i messaggi senza unirsi.

- I **canali**, nati a settembre 2015, sono chat in cui chiunque sia amministratore può inviare messaggi ai membri del canale, anche se questi ultimi non possono rispondere né commentare. Un canale può contenere un numero illimitato di iscritti e può essere pubblico o privato; al canale pubblico si può associare un indirizzo link e un username, ricercabile dalla funzione di ricerca dell'app.
- Da giugno 2015 Telegram ha introdotto una piattaforma per permettere, a sviluppatori terzi, di creare i **Bot**. I Bot sono degli *account* Telegram, gestiti da un programma, che offrono molteplici funzionalità con risposte immediate e completamente automatizzate. Dal 4 gennaio 2016 è disponibile una nuova modalità *inline* per i bot, che permette di utilizzare un bot semplicemente citandolo con il proprio username in qualsiasi chat (chat cloud, canali e gruppi). I bot inline ufficiali sono i seguenti:
 - @gif – Per cercare e mandare GIF;
 - @vid – Per condividere video da YouTube;
 - @bing – Per condividere immagini con Bing;
 - @pic – Per condividere immagini con Yandex;
 - @wiki – Per condividere voci di Wikipedia in tutte le lingue;
 - @Imdb – Per condividere informazioni sui film;
 - @bold – Per formattare del testo.

In seguito sono stati introdotti altri inline bot, @youtube, @music, @foursquare, @sticker, @gamee e @gamebot.

Per creare e gestire i propri bot è stato creato **Botfather**.

Sono inoltre presenti gli **Sticker** (o adesivi), ovvero immagini ad alta definizione salvate sul Cloud con lo scopo di rendere più espressive e convincenti le emoji e i messaggi. Gli Sticker sono raggruppati nei set, chiamati "Sticker Pack" che si possono aggiungere alla propria libreria tramite un link oppure cliccando sopra uno sticker mandato da un altro utente. Telegram offre un pacchetto di Sticker preinstallato, con la possibilità per gli utenti di crearne altri tramite il bot ufficiale @Stickers. [9]

Il confronto naturale che viene da fare quando si parla di applicazioni per la messaggistica istantanea vede contrapporre Telegram e **WhatsApp**. Quest'ultimo, forte di una base di utenti attivi mensile enorme, e superiore al rivale, sembra prevalere nei numeri. Per quanto riguarda invece le funzionalità e i servizi offerti, questi fanno pendere l'ago della bilancia a favore di Telegram, che è riuscito negli anni ad anticipare su quasi tutte le funzioni il rivale (bots, crittografia, canali, gifs, stickers, pagamenti, ecc.), trainato dalla forza di essere un progetto (almeno per la parte client) di vocazione open source. Nella Figura 3.2 un confronto (ironico) tra i due servizi di messaggistica (relativo a giugno 2018).



Figura 3.2: Confronto Telegram-WhatsApp [5]

3.2 Telegram Bot API

Telegram mette a disposizione vari tipi di API open source e un protocollo di comunicazione (MTProto) free.

In particolare fornisce le *Bot API* che permettono di creare programmi che usano i messaggi telegram come interfacce e le **Telegram API** che permettono di creare un client basato su quello di Telegram ma personalizzabile in tutti gli aspetti (comunicazione, sicurezza, storage, ecc.). Per facilitare l'uso di questo tipo di interfaccia sono presenti le *TDLib* (Telegram Database Library) che fornisce molte librerie di terze parti riconosciute, che permettono uno sviluppo più semplice e supportano diversi linguaggi di programmazione. [2]

Per quanto riguarda il presente progetto sono di rilevanza le **Telegram Bot API**, un'interfaccia HTTP-based che permette di collegare un bot al sistema Telegram.

I *bots* sono applicazioni sviluppate da terze parti che "vivono" all'interno di Telegram. Gli utenti possono interagirci inviando messaggi di vario tipo, tramite i *comandi* o le *richieste inline*. Il funzionamento del bot si basa principalmente su richieste HTTP che vengono inviate alle API Telegram.

Un bot può essere programmato per:

- Notificare notizie e aggiornamenti personalizzati.
- Integrarsi con altri servizi arricchendo le chat Telegram di contenuti esterni (ad esempio da Wikipedia, Youtube, GitHub, ecc.).
- Accettare pagamenti e operare come un virtual frontend per un negozio.
- Creare servizi ad hoc come allert per il meteo, servizi basati sulla geolocalizzazione (come SupermarketDealBot), eseguire traduzioni, connettere utenti in base a particolari criteri, o qualsiasi altra cosa che potrebbe operare un web service.
- Fornire giochi single e multiplayer, sviluppati in HTML5, all'interno di Telegram. [1]

Concettualmente il funzionamento di un bot Telegram è abbastanza semplice: quando un utente interagisce con il bot, le API inviano dettagli sull'interazione, tramite una richiesta HTTP, al codice sorgente del bot (che gira su un qualche server), che produrrà una risposta adeguata e la invierà come risposta HTTP.

Esistono due modi, mutuamente esclusivi, di ricevere aggiornamenti dalle API: il metodo *getUpdates* da un lato e il meccanismo dei *Webhooks* dall'altro. Indipendentemente da quale si scelga si riceverà un oggetto **Update** serializzato in formato JSON. L'Update contiene qualsiasi informazione utile riguardante la chat, il mittente e il messaggio ricevuto (con eventuali allegati). In Figura 3.3 è presente un esempio di messaggio Telegram.

```
{
  "update_id":646911460,
  "message":{
    "message_id":93,
    "from":{
      "id":10000xxxx,
      "is_bot":false,
      "first_name":"Jiayu",
      "username":"jiayu",
      "language_code":"en-US"
    },
    "chat":{
      "id":10000xxxx,
      "first_name":"Jiayu",
      "username":"jiayu",
      "type":"private"
    },
    "date":1509641174,
    "text":"eevee"
  }
}
```

Figura 3.3: Esempio di messaggio Telegram

Approfondendo i due modi di ricevere messaggi, si avrà:

- **getUpdates** si basa sul *LongPolling*, un meccanismo di *push notification*, evoluzione del classico polling. A differenza del tradizionale polling, in cui se si fa richiesta e non sono presenti nuovi aggiornamenti si riceve un messaggio vuoto come risposta, con il Long Polling, il server mantiene la richiesta attiva finché non si avranno nuovi aggiornamenti. Quindi appena si avrà un aggiornamento disponibile invierà una risposta consistente, diminuendo anche la latenza per il client, a costo di una gestione lato server più onerosa e complessa. Per motivi di consistenza è possibile avere un solo ascoltatore per volta, quindi il metodo getUpdates eventualmente richiamato dal bot deve essere unico. In altre parole, può girare solo un'istanza di un determinato bot. Un esempio di comunicazione tramite getUpdates è fornito in Figura 3.4.

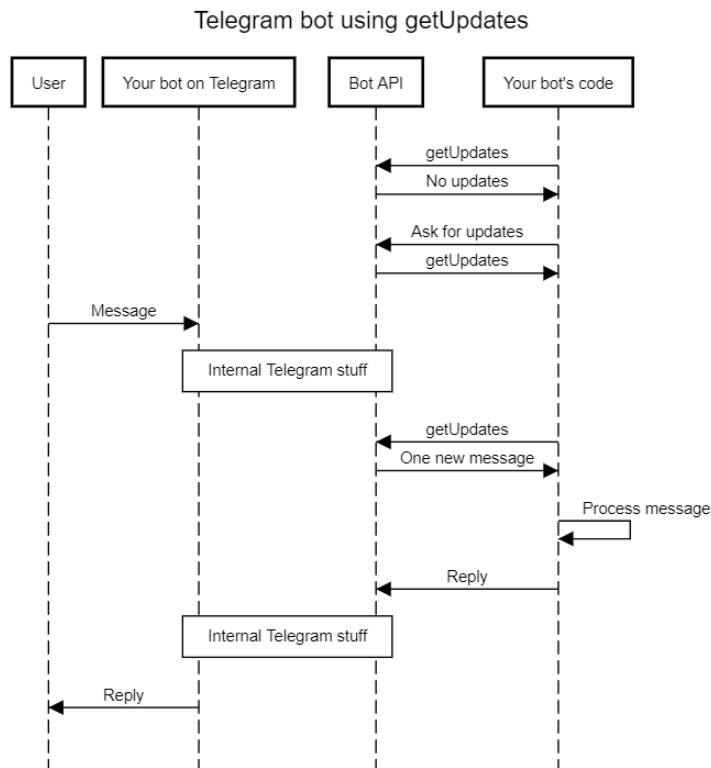


Figura 3.4: Sequence diagram metodo getUpdates

- **Webhooks** sono delle callback HTTP, che, scatenate da un evento (in questo caso l'arrivo di un messaggio/comando), indirizzano richieste HTTP a una particolare destinazione. Concettualmente è come inoltrare la richiesta che è arrivata al server Telegram anche al codice sorgente del bot. Per settare un Webhook le API telegram forniscono delle apposite chiamate. Anche in questo caso per ogni bot telegram si può assegnare un solo Webhook (che punti al codice del bot). In Figura 3.5 è presente un sequence diagram per la comunicazione tramite Webhook.

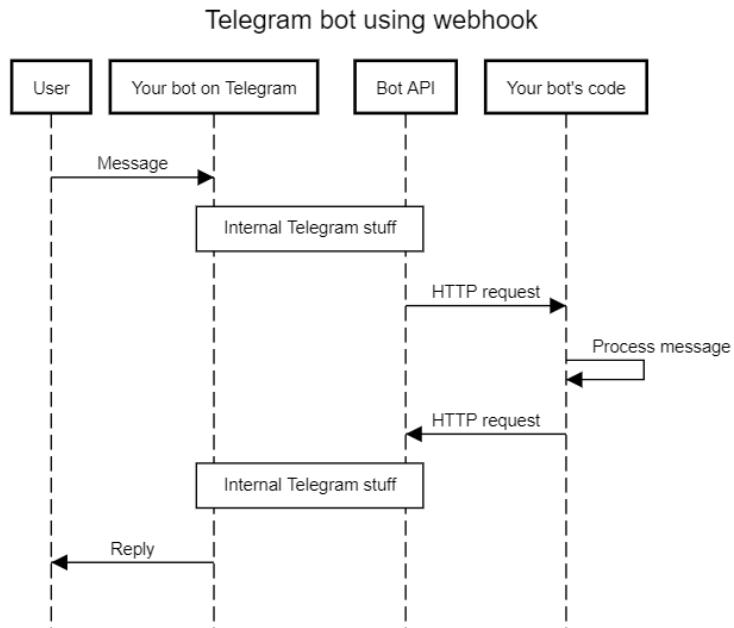


Figura 3.5: Sequence diagram Webhook

Anche in assenza di ricezione da parte del codice del bot, tutti i messaggi sono salvati sui server Telegram per almeno 24 ore. In quel lasso di tempo possono ancora essere consegnati ad una successiva riattivazione del bot. [3] [10]

Per una descrizione dettagliata delle API dei bot Telegram, e dei vari metodi forniti, riferirsi alla pagina <https://core.telegram.org/bots/api>.

3.2.1 Come si crea un bot



Per quanto riguarda la **creazione** di un nuovo bot, ci si avvale di un altro bot "speciale" **BotFather**.

Una volta avviato questo bot sarà possibile interagire tramite il comando `/newbot` che farà partire una sequenza guidata alla fine della quale si otterrà un **Token**, ovvero l'identificativo univoco del nuovo bot all'interno del sistema Telegram. Questo processo è illustrato

nella Figura 3.6.

BotFather esporta anche altre funzionalità come ad esempio quello di poter customizzare un bot (immagine, descrizione), aggiungere dei comandi ad un bot, richiedere un nuovo Token per un bot, ecc.

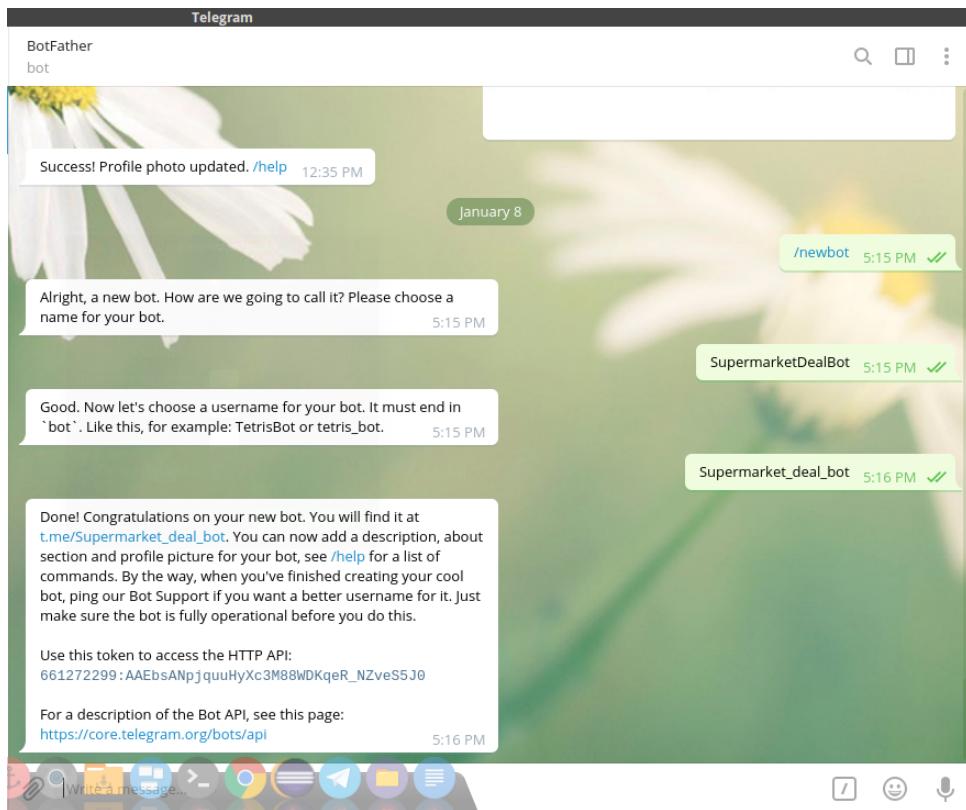


Figura 3.6: Creazione di SupermarketDealBot tramite BotFather

3.3 La libreria TelegramBots

La libreria TelegramBots, visionabile all’indirizzo <https://github.com/rubenlagus/TelegramBots>, è una libreria Java pensata per lo sviluppo di bot Telegram che si interfacciano con le Telegram Bots API.

Il codice sorgente del repository GitHub può essere importato in un progetto tramite Maven o come semplice libreria esterna jar (includendo tutte le dipendenze). Viene fornita sotto MIT License e semplifica di molto la creazione di un bot. Onestamente ho trovato il progetto su GitHub di facile interpretazione e corredata da esempi adeguati.

È possibile implementare sia l’ascolto di aggiornamenti tramite Webhook che tramite polling. L’ultimo metodo viene consigliato nella libreria quindi il bot sviluppato usa quel meccanismo.

Capitolo 4

Scelte progettuali

Il bot è stato sviluppato come progetto **Maven** utilizzando l'IDE *Eclipse* e la versione 8 di *Java*. Nella sua interezza il progetto presenta gli elementi mostrati nella Figura 4.1.

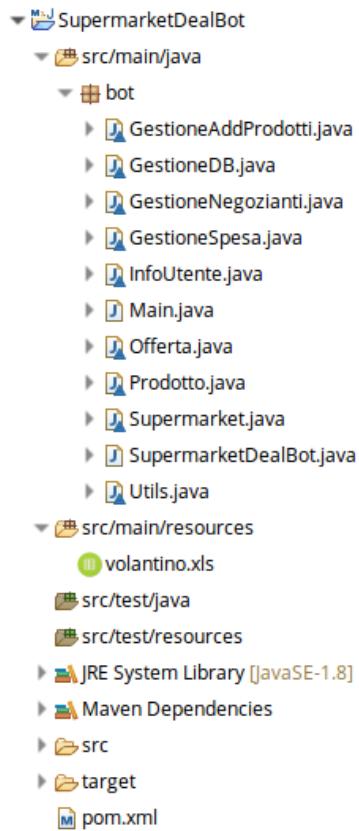


Figura 4.1: Progetto su Eclipse

È composto da 11 classi Java con una lunghezza totale di circa 2600 righe di codice.

4.1 Componenti del progetto

Tutto gira intorno alla classe *SupermarketDealBot* che ha il compito di gestire l'interazione con l'utente, mantenere le strutture dati necessarie e richiamare i metodi presenti in tutti le altre classi. Infatti le classi nominate *GestioneXXX* e la classe *Utils* sono classi di utilità statiche, che esportano metodi per alleggerire SupermarketDealBot. Tutte le restanti classi tranne il Main rappresentano oggetti utili ai fini del servizio offerto. Il diagramma UML del progetto è raffigurato in Figura 4.2.

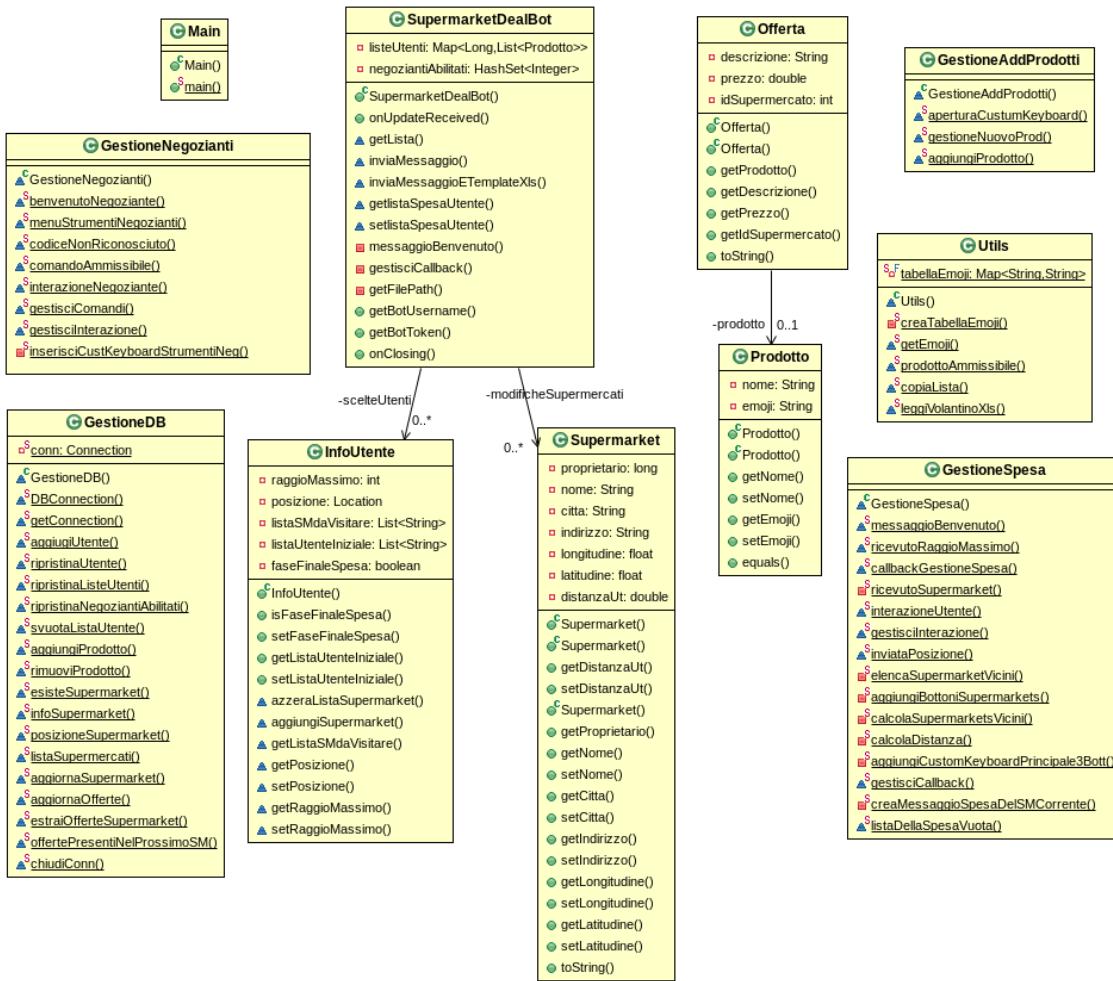


Figura 4.2: Class Diagram dell'intero progetto

Prima di descrivere nel dettaglio quello che fanno le classi è bene dare un'occhiata alle dipendenze presenti nel file pom.xml di Maven (Figura 4.3).

Nel dettaglio: *telegrambots* è la libreria di alto livello usata per lo sviluppo del bot; *emoji-java* permette una più semplice ingrazione delle emoji nei messaggi; *postgresql* sono i driver JDBC fondamentali per interfacciarsi con il database; *poi* e

```

<dependencies>
    <dependency>
        <groupId>org.telegram</groupId>
        <artifactId>telegrambots</artifactId>
        <version>4.1</version>
    </dependency>
    <dependency>
        <groupId>com.vdurmont</groupId>
        <artifactId>emoji-java</artifactId>
        <version>3.1.3</version>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.2.5</version>
    </dependency>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>3.17</version>
    </dependency>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>3.17</version>
    </dependency>
</dependencies>

```

Figura 4.3: Dettaglio delle dipendenze del progetto

poi-ooxml sono librerie, fornite da *Apache*, che permettono di accedere ai file xls ed estrarre il contenuto (vengono usate quando si aggiornano le offerte).

Concentrandoci invece sulle classi, andiamo a descriverle nel dettaglio:

- **Prodotto:** definisce una categoria di prodotti che può essere aggiunta alla lista.
- **Offerta:** definisce un'offerta resa disponibile da un supermercato. Contiene un prodotto specifico (descrizione), l'id del supermercato al quale è associato, la categoria (oggetto di classe Prodotto) in cui ricade l'offerta e il prezzo. Sono presenti gli opportuni metodi accessori.
- **Supermarket:** definisce un punto vendita con tutte le informazioni utili come la posizione (latitudine e longitudine) il nome, l'indirizzo e il responsabile associato.
- **InfoUtente:** è un oggetto usato solo quando viene chiamato il comando *spesa_ottimizzata* e permette di memorizzare tutte le scelte effettuate dall'utente.
- **SupermarketDealBot:** è il cuore del progetto. Questa classe eredita da *TelegramLongPollingBot* e implementa i metodi:
 - *onUpdateReceived(Update)* che ha il compito di intercettare la notifica che viene inviata dalle API di Telegram all'arrivo di un nuovo messaggio

e svolgere l'opportuna computazione. È il metodo più importante della classe, da qui vengono chiamate tutti gli altri metodi delle classi di utilità.

- *getBotUsername*: autoesplicativo.
- *getBotToken*: autoesplicativo.
- *onClosing*: chiude la connessione con il database.

Sono inoltre presenti molti metodi privati e con visibilità package utili alla computazione.

Quando si è finito di calcolare la risposta si crea un oggetto di classe *SendMessage* con all'interno tutto l'occorrente per la risposta e si invia all'utente desiderato richiamando il metodo *execute* della classe da cui eredita il bot.

Le strutture dati presenti in questa classe sono 4 e aiutano a tenere conto delle liste degli utenti (comunque salvate sul database), delle modifiche apportate ai supermercati, dei negozianti abilitati e delle scelte degli utenti nella spesa ottimizzata. Sono principalmente strutture di tipo *HashMap*.

- **Main**: ha lo scopo di creare il bot e avviare il servizio. È mostrata in Figura 4.4.

```

1 package bot;
2
3 import org.telegram.telegrambots.ApiContextInitializer;
4 import org.telegram.telegrambots.meta.TelegramBotsApi;
5 import org.telegram.telegrambots.meta.exceptions.TelogramApiException;
6
7 public class Main {
8     public static void main(String[] args) {
9         ApiContextInitializer.init();
10        TelegramBotsApi botsApi = new TelegramBotsApi();
11
12        try {
13            botsApi.registerBot(new SupermarketDealBot());
14        } catch (TelegramApiException e) {
15            e.printStackTrace();
16        }
17        System.out.println("SupermarketDealBot successfully started!");
18    }
19
20
21 }
22
23

```

Figura 4.4: La classe Main

Considerando ora le classi di utilità, si avranno:

- **Utils**: classe di utilità generica che aiuta a gestire le emoji e implementa la funzione di lettura di un file xls formattato opportunamente (Figura 4.5).
- **GestioneAddProdotti**: crea la (lunga) tastiera personalizzata per l'inserimento di prodotti e gestisce opportunamente le strutture, in questo caso.

	A	B	C	D
1				
2	ID Supermercato	categoria	prezzo	descrizione
3		10 Pomodori		0.99 Pomodori Casale Prov. Italia
4		10 Insalata		0.89 Lattuga Romana
5		10 Birre		1.00 Tuborg cl. 66
6		10 Birre		1.29 Carlsberg cl. 66
7		10 Birre		2.50 Leffe cl. 33
8		10 Birre		1.99 Peroni X3 xl. 33
9		10 Vino		3.89 Vino Spadafora rosso cl. 70
10		10 Vino		3.49 Vino Ciro rosato cl.70
11		10 Vino		6.49 Vino Amarone cl.70
12		10 Vino		5.99 Vino Salento cl.70
13		10 Patate		0.99 Patate Francia kg 1.5
14		10 Patate		1.39 Patate della sila kg 1.5
15				
16				
17				
18	Il file da inviare deve seguire questo template per essere caricato correttamente, in cui:			
19	ID Supermercato è il codice di autenticazione fornito al negoziante			
20	prezzo e descrizione dei prodotti sono autoesplicativi			
21	le categorie ammissibili sono:			
22	Pomodori			
23	Insalata			

Figura 4.5: Esempio di file xls per aggiornamento delle offerte

- **GestioneNegozianti:** gestisce la computazione relativa ai messaggi ricevuti in seguito al comando /negozianti.
- **GestioneSpesa:** è la classe più corposa del progetto e implementa la funzionalità relativa alla spesa guidata. Tra le altre cose, viene anche calcolata la distanza, in linea d'aria, tra l'attuale posizione dell'utente e quella dei punti vendita; usando la *Haversine Formula*, per il calcolo di distanze tra punti su superficie sferica.
- **GestioneDB:** è una classe cruciale in quanto usa gli strumenti forniti dalla libreria JDBC per connettersi con il database e garantire la persistenza e la correttezza di liste, supermarket, offerte e negozianti abilitati.

4.1.1 Il database Postgresql

Per garantire la consistenza e non perdere i dati quando si riavvia il bot (persistenza) si è fatto ricorso ad un *database relazionale* gestito dal DBMS Postgresql.

Durante lo sviluppo e il test del progetto è stata usata sia l'interfaccia a riga di comando che grafica (*pgAdmin*). Ciò sia in locale in fase di sviluppo che sul PaaS nella fase finale.

Il database in uso è formato da 4 tabelle relative alle offerte, supermarketi, negozianti e offerte.

4.2 Il PaaS usato: Heroku

Per quanto riguarda il funzionamento del bot, dato il meccanismo del polling sarebbe bastato anche usare il notebook usato per lo sviluppo, collagato ad Internet, come server su cui hostare il bot.

Dato che però, ho ritenuto interessante approfondire il funzionamento e lo sviluppo di applicazioni pensate per girare su Cloud, la mia scelta è ricaduta su questo servizio online, che sta riscuotendo discreto successo.

Heroku è un platform as a service (PaaS) su cloud che supporta diversi linguaggi di programmazione. Incominciò ad essere sviluppata nel giugno 2007, quando supportava solamente Ruby, in seguito è stato aggiunto supporto per Java, Node.js, Scala, Clojure, Python, PHP, e Go. Nel 2010 è stata acquisita da Salesforce.com.

Il fatto che sia un PaaS vuol dire che Heroku mette a disposizione di chi sviluppa una piattaforma dove poter rilasciare rapidamente una applicazione senza dover conoscere come configurare l'infrastruttura che c'è dietro. In Figura 4.7 sono presenti dei dati di utilizzo relativi alla piattaforma. [4]



Figura 4.6: Logo di Heroku



Figura 4.7: Informazioni sull'utilizzo di Heroku

Heroku permette di far girare applicazioni nei cosiddetti **dynos** ovvero istanze di macchine virtuali che possono essere facilmente scalate sia orizzontalmente (aumentandone il numero) che verticalmente (aumentandone la taglia). Nella versione free Heroku fornisce un singolo dyno al giorno (provvisto di 512 MB di RAM), per un totale di 550 ore mensili.

C'è da precisare che sebbene Heroku fornisca i servizi, l'intera piattaforma PaaS, come ogni applicazione deployata è caricata su *Amazon Web Services* (AWS), in quanto Heroku non possiede l'infrastruttura fisica.

Verrebbe quindi da chiedersi perché non usare direttamente AWS. Oltre alle differenti politiche di pricing, Heroku fornisce un'interfaccia a riga di comando (**Heroku CLI**) e anche un'interfaccia di controllo grafica, di immediato utilizzo per gli sviluppatori. Inoltre è un progetto open ed estendibile che comprende supporto per Nodejs, Ruby, PHP, Python, Java, ecc.

Infine sono presenti circa 150 add-ons built-in, diversi dei quali gratuiti (in versione limitata), facilmente integrabili. Sono presenti Add-ons per: Data Stores, Caching, Errors and Exceptions, Monitoring, Network Services, User Management,

Data Analysis e molto altro. Per ogni Add-on aggiunto vengono create delle opportune variabili d'ambiente richiamabili nelle applicazioni. In Figura 4.8 è presente uno schema di funzionamento degli Add-ons.

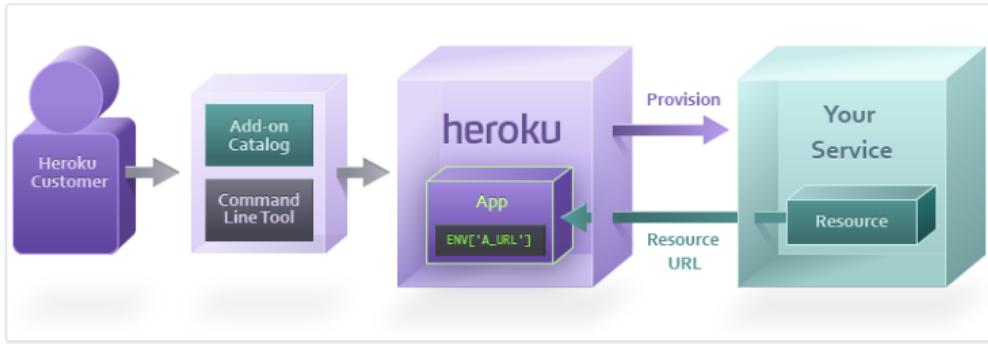


Figura 4.8: Heroku Add-ons

L'unico Add-on usato nel progetto è stato il database PostGresql (*Heroku PostGres Hobby Dev*), che fornisce un limite di 10000 tuple totali e 20 connessioni simultanee. Per quanto riguarda la documentazione l'ho trovata sufficientemente chiara e molto corposa.

Lavorare con Heroku non è stato particolarmente complicato, una volta comprese le convenzioni. L'interfaccia grafica web (*Heroku DashBoard*) si presenta nella seguente forma (Figura 4.9).

Heroku Dashboard Screenshot:

- Installed add-ons:** \$0.00/month (Heroku Postgres Hobby Dev)
- Dyno formation:** \$0.00/month (This app is using free dynos)
- worker**: java -jar target/SupermarketDealBot-1.0.jar
- Collaborator activity:** domcost93@gmail.com (19 deploys)
- Latest activity:**
 - Feb 14 at 9:06 PM - v27 (Deployed)
 - Feb 14 at 9:05 PM - View build log (Build succeeded)
 - Feb 14 at 8:59 PM - v26 (Deployed)
 - Feb 14 at 8:59 PM - View build log (Build succeeded)
 - Feb 12 at 3:49 PM - v25 (Deployed)
 - Feb 12 at 3:48 PM - View build log (Build succeeded)

Figura 4.9: Heroku Dashboard

Per quanto riguarda lo sviluppo si può benissimo operare in locale e poi deployare in maniera corretta sulla piattaforma. Tutti i progetti Heroku sono dotati di un

ProcFile che definisce il tipo (o i tipi) di processo che girerà nei dynos. Per un progetto Java sono presenti due tipi: *web* e *worker*. SupermarketDealBot è una app di tipo **worker** in quanto il meccanismo del polling non richiede che sia disponibile un’interfaccia web da contattare esplicitamente (si fa uso di callbacks).

Il deploy di un progetto Java deve avvenire mediante Maven e il PaaS si occupa della build finale. Per inviare i dati sulla piattaforma si utilizza *Git* (o anche collegando un repository GitHub) utilizzando la Heroku CLI, come mostrato nella Figura 4.10.

```

Thu, Feb 14 21:00
Applications
+ provaDB pgadmin4 SDB:git Home
(5 rows)

supermarket-deal-bot::DATABASE> Select * from offerte;
supermarket-deal-bot::DATABASE> \q
domenico@domenico-RC530-RC730:~/Desktop/SDB$ heroku scale worker=0
Scaling dynos... done, now running worker at 0:Free
domenico@domenico-RC530-RC730:~/Desktop/SDB$ git add .
domenico@domenico-RC530-RC730:~/Desktop/SDB$ git commit -am "si spera commit finale"
[master 95bach5] si spera commit finale
 18 files changed, 47 insertions(+), 33 deletions(-)
 rewrite target/SupermarketDealBot-1.0.jar (95%)
 rewrite target/classes/bot/GestioneAddProdotti.class (94%)
 rewrite target/classes/bot/GestioneDB.class (99%)
 rewrite target/classes/bot/GestioneNegozianti.class (88%)
 rewrite target/classes/bot/InfoOfferente.class (100%)
 rewrite target/classes/bot/Offerta.class (100%)
 rewrite target/classes/bot/Prodotto.class (100%)
 rewrite target/classes/bot/Supermarket.class (99%)
 rewrite target/classes/bot/SupermarketDealBot.class (63%)
 rewrite target/classes/bot/Utils.class (99%)
 domenico@domenico-RC530-RC730:~/Desktop/SDB$ git push heroku master
Counting objects: 27, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (27/27), 79.64 KiB | 0 bytes/s, done.
Total 27 (delta 12), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: ----> Java app detected
remote:      ----> Installing JDK 1.8... done
remote:      ----> Installing Maven 3.3.9... done
remote:      ----> Executing: mvn -DskipTests clean dependency:list install
remote: [INFO] Scanning for projects...
remote: [WARNING]
remote: [WARNING] Some problems were encountered while building the effective model for telegrambot:SupermarketDealBot:jar:1.0
remote: [WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-jar-plugin is missing. @ line 68, column 12
remote: [WARNING]
remote: [WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
remote: [WARNING]
remote: [WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
remote: [INFO]

```

Figura 4.10: Deploy del progetto su Heroku

Le altre istruzioni maggiormente usate a riga di comando sono: *heroku pg:psql* per controllare il database remoto, *heroku logs -tail* per stampare il log (la console) del worker e *heroku scale worker=NUM_DYNOS* per attivare o disattivare i dynos sul PaaS. In Figura 4.11 degli esempi d’uso.

Per qualsiasi approfondimento relativo all’implementazione si rimanda al codice allegato.

The screenshot shows two windows side-by-side. The left window is a terminal session titled 'domenico@domenico-RC530-RC730:~/Desktop/SDB\$' running the command 'heroku pg:psql'. It connects to a PostgreSQL database on Heroku, showing the schema and relations for the 'supermarket-deal-bot' application. The right window is a 'heroku logs --tail' session, also titled 'domenico@domenico-RC530-RC730:~/Desktop/SDB\$'. It displays the log output for the application's worker process, showing the startup of the Java application and its connection to the database.

```

domenico@domenico-RC530-RC730:~/Desktop/SDB$ heroku pg:psql
--> Connecting to postgresql-closed-52639
pgsql (9.5.14, server 10.6 (Ubuntu 10.6-1.pgdg16.04+1))
WARNING: pgsql major version 9.5, server major version 10.
        Some pgsql features might not work.
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

supermarket-deal-bot::DATABASE=> \d
              List of relations
 Schema |      Name      | Type  |  Owner
-----+---------------+-----+-
 public | listaprodotti | table | ufqlbefsodobyf
 public | negozianti   | table | ufqlbefsodobyf
 public | offerte      | table | ufqlbefsodobyf
 public | offerte_IDProdotto_seq | sequence | ufqlbefsodobyf
 public | supermarkets  | table | ufqlbefsodobyf
(5 rows)

supermarket-deal-bot::DATABASE=> select * from supermarkets;
   id   |      nome      |      citta      |      indirizzo      | latitudine | longitudine
-----+-----+-----+-----+-----+
    4 | Carrefour Market | Crotone | via Passovecchio | 39.115646 | 17.092609
    2 | Pianette Discount | Strongoli | via pianette | 39.265549 | 17.059707
    3 | Mare             | Strongoli Marina | prima fila | 39.250649 | 17.106441
    1 | Gerardo          | Strongoli | via Lala Farm | 39.26557 | 17.050963
139851232 | Nuovo nome       | Questa   | La               | 12.3      | 45.6
(5 rows)

supermarket-deal-bot::DATABASE=>

heroku logs --tail
[2]+ Stopped                  heroku logs --tail
domenico@domenico-RC530-RC730:~/Desktop/SDB$ heroku scale worker=0
Scaling dynos... done, now running worker at 0:Free
domenico@domenico-RC530-RC730:~/Desktop/SDB$ heroku scale workers=1
Scaling dynos... done, now running worker at 1:Free
domenico@domenico-RC530-RC730:~/Desktop/SDB$ heroku logs --tail
2019-02-14T20:17:57.796881+00:00 heroku[worker.1]: Stopping all processes with SIGTERM
2019-02-14T20:17:58.186393+00:00 heroku[worker.1]: Process exited with status 143
2019-02-16T18:28:20.269563+00:00 app[api]: Scaled to worker@1:Free by user domcost93@gmail.com
2019-02-16T18:28:23.987548+00:00 heroku[worker.1]: Starting process with command `java -jar target/SupermarketDealBot-1.0.jar`
2019-02-16T18:28:24.850685+00:00 heroku[worker.1]: State changed from starting to up
2019-02-16T18:28:25.845982+00:00 app[worker.1]: Picked up JAVA_TOOL_OPTIONS: -Xmx300m -Xss512k -XX:CICompilerCount=2 -Dfile.encoding=UTF-8
2019-02-16T18:28:28.407005+00:00 app[worker.1]: Opened database successfully
2019-02-16T18:28:28.419444+00:00 app[worker.1]: Table created successfully
2019-02-16T18:28:28.960582+00:00 app[worker.1]: SupermarketDealBot successfully started!
2019-02-16T18:28:29.255298+00:00 app[worker.1]: 139851232
2019-02-16T18:30:27.962845+00:00 app[worker.1]: 761514537
2019-02-16T20:25:26.300761+00:00 app[worker.1]: 139851232
2019-02-16T20:25:34.153274+00:00 app[worker.1]: 139851232
2019-02-16T20:25:34.164966+00:00 app[worker.1]: Pane

```

Figura 4.11: Esempi di utilizzo della Heroku CLI

Capitolo 5

Considerazioni finali e possibili scenari evolutivi

Come già accennato in precedenza, ho ritenuto molto interessante sfruttare lo sviluppo di questo progetto andandomi a confrontare con una tecnologia (quella del Cloud e dei PaaS) che è ormai dominante sul mercato, oltre ad essere parte integrante del programma di questo corso.

Per quanto riguarda il bot si può sicuramente dire che i punti di forza sono la totale assenza di codice (e quindi applicativo da installare) client, come anche l'immediatezza di interazione in un ambiente familiare che si conosce già.

Di contro, le Telegram Bot API non mettono a disposizione funzioni per una più approfondita autenticazione degli utenti, che non sia il codice relativo alla chat aperta. Inoltre il fatto di essere un bot preclude la presenza di un'interfaccia grafica più accattivante e maggiormente intuitiva, che potrebbe offrire un'applicazione mobile.

Ragionando su possibili miglioramenti futuri si può pensare di integrare un database di immagini o di file contenenti i volanti stampati (oltre alla lista stampata a schermo), aggiungere delle funzionalità per rendere una lista collaborativa e condivisibile, inserire più informazioni e riferimenti ai supermarket e pensare ad una qualche forma di gamification che possa incentivare l'utilizzo e l'acquisto nei punti vendita selezionati (tramite punti fedelta e premi).

Elenco delle figure

2.1	Schermate di esempio di Telegram	3
2.2	Ricerca del bot e avvio della conversazione	4
2.3	Use-case Diagram Utente	5
2.4	Comandi "aggiungi" e "lista"	6
2.5	Integrazione con Google Maps	6
2.6	Use-case Diagram Negoziante	7
2.7	Interazione con i negozianti	8
2.8	Possibili competitors: PromoQui, DoveConviene, OutOfMilk	9
3.1	Logo di Telegram	10
3.2	Confronto Telegram-WhatsApp [5]	13
3.3	Esempio di messaggio Telegram	14
3.4	Sequence diagram metodo getUpdates	15
3.5	Sequence diagram Webhook	16
3.6	Creazione di SupermarketDealBot tramite BotFather	17
4.1	Progetto su Eclipse	18
4.2	Class Diagram dell'intero progetto	19
4.3	Dettaglio delle dipendenze del progetto	20
4.4	La classe Main	21
4.5	Esempio di file xls per aggiornamento delle offerte	22
4.6	Logo di Heroku	23
4.7	Informazioni sull'utilizzo di Heroku	23
4.8	Heroku Add-ons	24
4.9	Heroku Dashboard	24
4.10	Deploy del progetto su Heroku	25
4.11	Esempi di utilizzo della Heroku CLI	26

Bibliografia

- [1] Bots: An introduction for developers. <https://core.telegram.org/bots>.
- [2] Telegram apis. <https://core.telegram.org/api>.
- [3] Telegram bot api. <https://core.telegram.org/bots/api>.
- [4] What is heroku? <https://www.heroku.com/what>, 2019.
- [5] DashMagazine. Infographic: Telegram vs whatsapp. <https://hackernoon.com/infographic-telegram-vs-whatsapp-e2655d4b0550>, 2018.
- [6] Pavel Durov. 200,000,000 monthly active users. <https://telegram.org/blog/200-million>, 2018.
- [7] Maurizio Fiorino. Le 5 migliori app volantini e offerte. <https://www.telefoninostop.com/le-migliori-app-volantini-e-offerte/>, 2018.
- [8] Enrico Sanzo. Lista della spesa su smartphone: ecco le migliori applicazioni. <http://android.caotic.it/2014/11/android-applicazioni-lista-spesa/>, 2018.
- [9] Wikipedia. Telegram. <https://it.wikipedia.org/wiki/Telegram>, 2019.
- [10] Jiayu Yi. Introduction to the telegram bot api, part 1. <https://chatbotslife.com/introduction-to-the-telegram-bot-api-part-1-2ae36f7b30a4>, 2017.