UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA TRIENNALE IN INFORMATICA

---

*Alessandro Maggio*

# SVILUPPO DI UN BOT TELEGRAM PER FACILITARE L'ORGANIZZAZIONE DI INCONTRI TRA AMICI E COLLEGHI

RELAZIONE PROGETTO FINALE

Relatore:
Chiar.mo Prof. Tramontana Emiliano Alessio

---

Anno Accademico 2017 – 2018

# INDICE

# Capitolo 1 - INTRODUZIONE

Viviamo in un mondo in cui vivere la vita vera è più difficile che viverla “virtualmente” su un social network. È vero, in qualche modo ci aiutano a sentirci meno soli e più vicini ad amici che non vediamo da parecchio tempo, amici che, per inseguire i propri sogni sono dovuti andare via dalla nostra città. Internet è pieno di servizi che aiutano a ridurre le distanze. Contattare le persone distanti a noi è un problema presente sin dall’antichità. Basti pensare che una volta per avere notizie di un nostro caro era necessario inviare un piccione viaggiatore ed aspettare per giorni una risposta. Grazie ad internet ed ai servizi di messaggistica instantea, oltre alla distanza sono stati ridotti anche i tempi di attesa.


*Figura 1.1 - Evoluzione della comunicazione*

Tra i tanti servizi vi è Telegram, un servizio di messaggistica istantanea basato sul cloud. Le sue caratteristiche principali 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. I client ufficiali di Telegram sono distribuiti come software libero per Android, GNU/Linux, iOS, MacOS, Windows NT e Windows Phone. 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.

Nonostante tutto, il miglior modo per vivere delle relazioni è quello di passare del tempo insieme, magari davanti ad una pizza o ad una birra. Questa tesi tramite l’utilizzo di un Bot Telegram si preoccupa di aiutare gruppi di amici ad organizzare incontri di vario tipo e decidere una destinazione finale nella quale vedersi. Analizzando i vari messaggi scambiati all’interno di un gruppo, il Bot capisce quando vi è l’intenzione di “fare qualcosa” tramite quindi una proposta iniziale e crea un sondaggio. Il sondaggio verrà

aggiornato dinamicamente ogni qualvolta un utente esprime la propria opinione oppure propone delle alternative a quella iniziale.

## 1.1 - CONTESTO

Il Bot opera all'interno di un gruppo Telegram e analizza tutti i messaggi scambiati all'interno dello stesso. Per ogni messaggio effettua un parsing delle parole presenti all'interno del testo e tramite la libreria TreeTagger classifica ogni parola all'interno della grammatica italiana. Giocano un ruolo molto importante i verbi e le preposizioni in quanto permettono di capire se la frase contiene l'intenzione di voler fare qualcosa, oppure vi è una nuova proposta ad un sondaggio già aperto.

Ad esempio, nella frase: "*Ciao ragazzi, come va? Volete fare qualcosa domani sera? Io pensavo di andare al cinema per passare un po' di tempo insieme."*, vi sono le parole chiave per creare un nuovo sondaggio e popolarlo già con una prima proposta. Innanzitutto, il testo viene suddiviso in tre frasi:

1. *Ciao ragazzi, come va?*
2. *Volete fare qualcosa domani sera?*
3. *Io pensavo di andare al cinema per passare un po' di tempo insieme.*

Nella prima frase non vi è alcuna parola di particolare importanza, dunque viene scartata. Nella seconda, invece, grazie al verbo *"fare"* coniugato all'infinito, seguito da *"domani sera"* è facile intuire che si sta proponendo al gruppo di voler intraprendere un'azione nella serata di domani, di conseguenza il Bot aprirà il sondaggio avente come titolo il messaggio originariamente inviato. Analizzando la terza frase inoltre, è possibile capire che l'utente sta persino proponendo un luogo in cui andare, in quanto, vi è presente il verbo *"andare"* coniugato all'infinito, seguito dalla preposizione articolata *"al"* seguita a sua volta da un nome, ovvero il cinema. Pertanto, il Bot aggiungerà automaticamente al sondaggio precedentemente aperto *"Andare al cinema"* come un'eventuale scelta per la serata aspettando che a sua volta gli altri componenti del gruppo esprimano le proprie opinioni.

## 1.2 - OBIETTIVO

Quando un gruppo è formato da tanti partecipanti non è sempre facile organizzarsi e trovare un punto d'incontro che vada bene a tutti. Inoltre, le varie proposte potrebbero essere sommerse da altri messaggi magari "off-topic". Per questo motivo il progetto si offre come un ottimo strumento di supporto raccogliendo sotto di esso tutte le proposte date prima che venga chiuso. Ogni proposta sarà accompagnata dal numero di voti ottenuti, i quali si aggiorneranno automaticamente ogni qualvolta un utente esprime la propria volontà. Il sondaggio potrà poi essere chiuso manualmente dall'utente utilizzando un comando del Bot oppure attendere che il Bot lo chiuda in automatico. Ogni indagine ha un "marcatore temporale" che indica giorno e orario per cui è stata aperta. Ad esempio, utilizzando la frase presente nel sotto capitolo 1.1, il Bot chiuderà automaticamente il sondaggio il giorno successivo alla sua apertura alle ore 20:00, in quanto, il testo conteneva le parole *"domani sera"*. Ogni qualvolta un sondaggio si conclude verrà inviato nel gruppo un messaggio contenente la proposta vincitrice.

# Capitolo 2 - REQUISITI

Come spiegato nei capitoli precedenti il Bot lavora analizzando stringhe di testo presenti nei vari messaggi inviati all'interno di un gruppo Telegram. Ogni stringa analizzata a sua volta viene categorizzata in base alla grammatica italiana.

## 2.1 - CASISTICHE

Tra le possibili casistiche si è prestata una maggiore attenzione a tutte le frasi di senso compiuto avente al suo interno almeno un verbo e/o un nome. Di particolare importanza anche gli avverbi, in quanto le parole come ad esempio *"domani"* sono classificati come tali. Preposizioni e articoli invece, vengono utilizzati per costruire un'eventuale nuova proposta da aggiungere al sondaggio.

Le seguenti frasi potrebbero essere tutte (seppur poco probabili) eventuali risposte alla domanda posta come d'esempio nel sotto capitolo 1.1.

| **Struttura frase** | **Esempio** |
|---|---|
| VERBO | Mangiare. |
| VERBO + NOME | Studiare programmazione. |
| VERBO + PREPOSIZIONE + NOME | Giocare a scacchi. |
| VERBO + ARTICOLO + NOME | Raccogliere i fiori. |

*Tabella 2.1 - Struttura delle frasi*

### 2.1.1 - Preposizioni

Ad ogni modo non tutte le preposizioni permettono alla frase di diventare possibile candidata per un eventuale proposta poiché, non tutte introducono un luogo dove è possibile recarsi o un'azione da poter intraprendere. Come ad esempio: *per, tra, fra, su, con*, ed altre. Inoltre, preposizioni come *in*, seppur sembri che introducono un logo potrebbero introdurre un mezzo di trasporto. Basti pensare alle due frasi seguenti:

- Io direi di andare *in* pizzeria.
- Oggi sono stanco, andiamoci *in* auto.

Come soluzione a questi problemi sono state create delle variabili di supporto utili a scartare le preposizioni che escludono una nuova proposta. Riguardo alla preposizione

*"in"*, è stato adottato un file dizionario contente una buona raccolta di mezzi di trasporto. Se il nome successivo ad *"in"* è un mezzo allora la frase viene scartata altrimenti, vista la tipologia del problema, verrà considerato come luogo in cui recarsi.

### 2.1.2 - Avverbi

Come descritto nel capitolo 2.1 anche gli avverbi vengono analizzati in quanto ci permettono di capite se il sondaggio si riferisce a ciò che è chiamato all'interno del progetto *"trasformatore temporale"*. Gli unici avverbi ad essere presi in considerazione sono: *domani* e *dopodomani*, in quanto, indicano che il sondaggio avrà fine uno o due giorni dopo la sua chiusura.

### 2.1.3 - Nomi

Tra i nomi possiamo sicuramente riconoscere la classe più ampia di parole presenti nel dizionario italiano, basti pensare che parole come: *casa, pizza, sera, lunedì, cena, pranzo* vengono tutte classificate come nomi. Tra gli esempi riportati però è bene notare che la parola *"sera"* non è un oggetto sulla quale si vuole eseguire un'azione o un luogo dove si ha intenzione di andare ma bensì quando la si vuole svolgere. Pertanto, alcune parole vengono classificate nel progetto come *"puntatori temporali"*. Tutte queste parole sono contenute all'interno di una variabile di supporto, dove, ad ognuna di esse è stato associato un orario indicativo utilizzabile dal Bot per chiudere automaticamente il sondaggio.

| **Parola** | **Orario indicativo** |
|---|---|
| colazione | 08:00 |
| pranzo | 13:00 |
| pomeriggio | 16:00 |
| sera | 20:00 |
| cena | 21:00 |
| in giornata | 00:00 |

*Tabella 2.2 - Puntatori temporali*

Ulteriori nomi che potrebbero confondere il Bot sono i giorni della settimana. Anch'essi come gli avverbi indicano il giorno di chiusura del sondaggio senza andare però ad agire sul giorno corrente.

## 2.2 - FUNZIONALITÀ

Tutto il progetto è basato sul concetto di sondaggio, ovvero, una ricerca ed elaborazione di dati statistica con lo scopo di conoscere l'opinione di un gruppo di persone relative ad un dato argomento.

### 2.2.1 - Apertura di un sondaggio

Innanzitutto, al centro di un sondaggio c'è un determinato argomento che deve essere proposto da un utente del gruppo. Tutti i nuovi sondaggi vengono aperti se nel testo c'è il verbo *andare* o *fare*. Nella frase d'esempio del sotto capitolo 1.1 era presente il verbo *fare*, banalmente un altro esempio potrebbe essere: *Dove andiamo martedì?* Ogni sondaggio è caratterizzato in particolare dal titolo, orario e giorno di chiusura. Se nel messaggio non viene specificato un giorno o un orario, di default il sondaggio verrà chiuso alla mezzanotte del giorno in cui è stato aperto.

### 2.2.2 - Chiusura di un sondaggio

L'intento è quello di sapere e capire cosa ne pensano gli altri utenti, pertanto è importante avere dei risultati finali. Per far ciò è stato inserito il comando */close_pool* all'interno del Bot il quale, se utilizzato, avrà il compito di chiudere il sondaggio e determinare il vincitore tra le tante proposte. Il sondaggio può essere chiuso solo da chi lo ha aperto. Nel caso in cui un utente abbia aperto più indagini, il Bot chiederà quale si vuole chiudere. Inoltre, dato che ogni sondaggio ha come caratteristica giorno e orario di chiusura, nel caso in cui non venga chiuso manualmente il Bot in automatico provvederà a farlo informando il gruppo sui risultati finali.

### 2.2.3 - Votazione

Le votazioni appaiono nel modo più semplice e intuitivo. Ogni sondaggio avrà sotto di esso una lista di proposte. L'utente interessato potrà cliccare sulla proposta esprimendo così la propria opinione.

### 2.2.4 - Identificazione nuove proposte

Le idee spesso sono molteplici, viene proposto un determinato luogo nel quale passare la serata ma non sempre va bene tutti. Proprio per questo chiunque può aggiungere nuove proposte sotto un sondaggio aperto semplicemente scrivendolo nel gruppo. Se vi è una sola indagine aperta il Bot inserirà automaticamente questa proposta all'interno di esso, nel caso in cui i sondaggi sono più di uno cercherà di capire tramite le parole chiave relative al giorno e/o al tempo di chiusura del messaggio a quale sondaggio l'utente si sta riferendo. Se invece il messaggio di proposta è un po' troppo vago tramite dei pulsanti verrà chiesto all'utente stesso di selezionare il sondaggio di riferimento.

# Capitolo 3 - PROGETTAZIONE

Il progetto è stato sviluppato in Python, un linguaggio di programmazione ad alto livello, orientato agli oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing.

Il file principale è denominato *main.py* ed ha il compito di istanziare tutte le classi e librerie utili al fine ultimo del progetto. In particolare, le librerie che giocano un ruolo importante sono:

- python-telegram-bot
- pymongo
- treetaggerwrapper

Tutti in sondaggi invece sono gestiti tramite il database MongoDB.

## 3.1 - TELEGRAM BOT API

Le API di Telegram offrono un'interfaccia HTTP creata per gli sviluppatori che vogliono realizzare il proprio Bot Telegram. Ad ogni Bot viene assegnato un token univoco necessario per autenticarsi con le API di Telegram. Tutte le chiamate devono essere in HTTPS e si presentano nella forma: https://api.telegram.org/bot<token>/METHOD_NAME. Le richieste supportano le chiamate HTTP di tipo GET e POST, inoltre, i parametri possono essere passati come:

- URL query string
- application/x-www-form-urlencoded
- application/json (except for uploading files)
- multipart/form-data (use to upload files)

Le risposte invece, contengo un oggetto JSON, avente all'interno il campo "*ok"* di tipo booleano e altri campi opzionali come ad esempio "*description*". Tutti gli endpoint disponibili sono ben documentati all'interno del sito ufficiale.

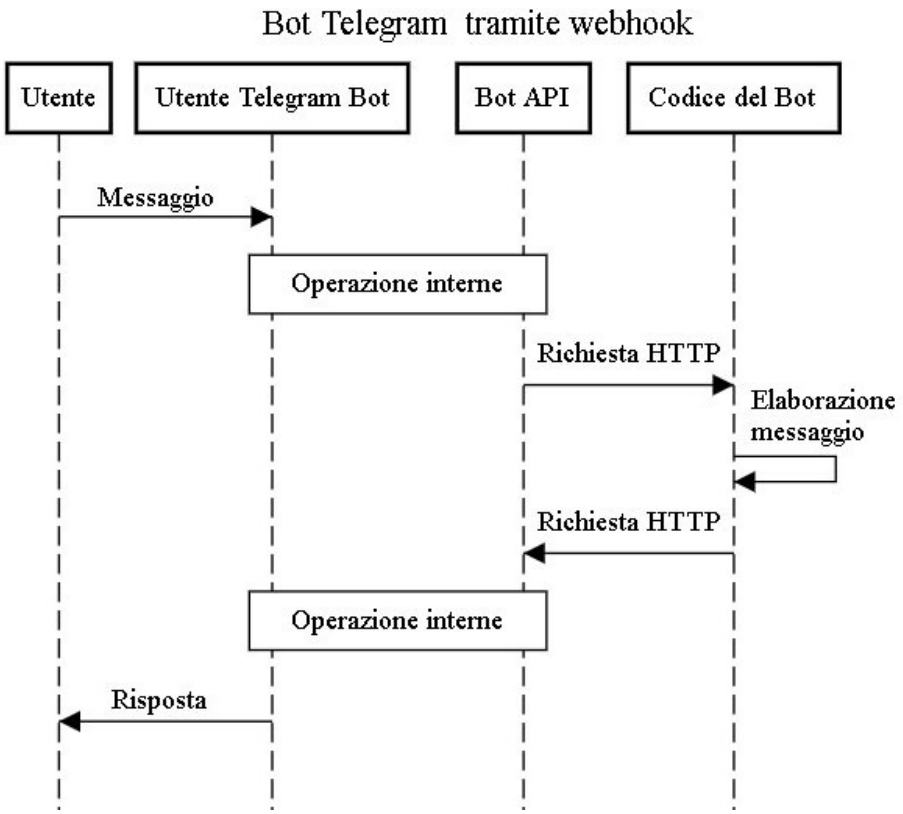


*Figura 3.1 - Funzionamento API Bot Telegram*

### 3.1.1 - python-telegram-bot

La libreria python-telegram-bot fornisce un'interfaccia per le Bot API di Telegram. Tra le varie librerie presenti sul web questa è tra le migliori in quanto offre molteplici classi di alto livello per rendere lo sviluppo di un Bot semplice ed immediato. Le classi principalmente utilizzate sono:

| Campo | Descrizione |
|---|---|
| Updater | Questa classe si occupa di inviare tutti gli aggiornamenti agli handlers registrati. |
| CommandHandler | Questa classe gestisce tutti i comandi inviati al Bot Telegram. Solitamente sono messaggi che iniziano con /, opzionalmente seguite da @ e il nome del bot e/o del testo aggiuntivo. |
| CallbackQueryHandler | Questa classe gestire le richieste di Callback associate ad ogni Handler. |
| InlineKeyboardButton | Questo oggetto rappresenta il singolo pulsante di una tastiera. |
| InlineKeyboardMarkup | Questo oggetto rappresenta la tastiera associata la messaggio. |

*Tabella 3.1 – Classi di python-telegram-bot utilizzate*

## 3.2 - TREETAGGER

TreeTagger è uno strumento che permette di classificare le parole in base al contesto. Su ogni parola viene annotato il lemma di esso e il tipo di tag assegnatogli. Il progetto è stato sviluppato da Helmut Schmid nell'istituto per la computazione linguistica dell'università di Stuttgart. TreeTagger è stato utilizzato con successo per etichettare molteplici lingue tra cui anche l'italiano. Il tagset italiano utilizzato anche all'interno di questo progetto è il seguente:

| | |
|---|---|
| AB | abbreviation |
| ADJ | adjective |
| ADV | adverb |
| CON | conjunction |
| DET:def | definite article |
| DET:indef | indefinite article |
| FW | foreign word |
| INT | interjection |
| LS | list symbol |
| NOM | noun |
| NPR | name |
| NUM | numeral |
| PON | punctuation |
| PRE | preposition |
| PRE:det | preposition+article |
| PRO | pronoun |
| PRO:demo | demonstrative pronoun |
| PRO:indef | indefinite pronoun |
| PRO:inter | interrogative pronoun |
| PRO:pers | personal pronoun |
| PRO:poss | possessive pronoun |
| PRO:refl | reflexive pronoun |
| PRO:rela | relative pronoun |
| SENT | sentence marker |
| SYM | symbol |
| VER:cimp | verb conjunctive imperfect |
| VER:cond | verb conditional |
| VER:cpre | verb conjunctive present |
| VER:futu | verb future tense |
| VER:geru | verb gerund |
| VER:impe | verb imperative |
| VER:impf | verb imperfect |
| VER:infi | verb infinitive |
| VER:pper | verb participle perfect |
| VER:ppre | verb participle present |
| VER:pres | verb present |
| VER:refl:infi | verb reflexive infinitive |
| VER:remo | verb simple past |

*Tabella 3.2 - Italian tagset used in the TreeTagger parameter file. (Copyright Prof. Achim Stein, University of Stuttgart)*

L'utilizzo di TreeTagger all'interno del progetto è stato possibile grazie al wrapper *treetaggerwrapper*.

```
import treetaggerwrapper

tagger = treetaggerwrapper.TreeTagger(TAGLANG='it', TAGDIR='./TreeTagger',
TAGPARFILE='./TreeTagger/lib/italian.par')

tags_encoded = tagger.tag_text( "Ciao ragazzi. Volete uscire questa sera?" )
tags = treetaggerwrapper.make_tags( tags_encoded )
```

| **word** | **pos** | **lemma** |
|---|---|---|
| Ciao | INT | ciao |
| ragazzi | NOM | ragazzo |
| . | SENT | . |
| Volete | VER:pres | volere |
| uscire | VER:infi | uscire |
| questa | PRO:demo | questo |
| sera | NOM | sera |
| ? | PON | ? |

*Tabella 3.3 - Output TreeTagger*

## 3.3 - MONGODB

MongoDB (da "humongous", enorme) è un DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL, MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico (MongoDB chiama il formato BSON), rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce. Il database è stato denominato come *"pool_group"* e contiene al suo interno le collezioni *pool* e *pending_propose*.

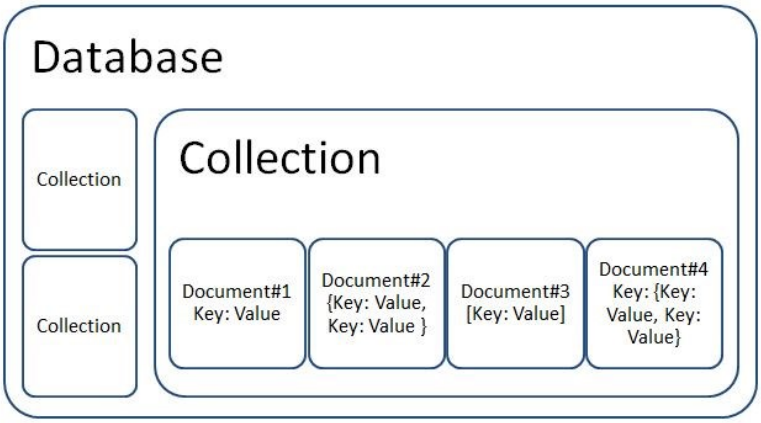


*Figura 3.2 - Struttura dei database NoSql*

### 3.3.1 - pool

La collezione *pool* contiene al suo interno documenti i quali identificano ogni sondaggio che è stato aperto all'interno dei vari gruppi ove il Bot è presente. Un esempio di documento presente all'interno della collezione è il seguente:

```
{
  "_id" : ObjectId("5be2b1e620d8e705b5504863"),
  "title" : "Ciao ragazzi, come va? Volete fare qualcosa questa sera….",
  "closed" : false,
  "time_value" : {
    "close_datetime" : ISODate("2018-11-14T20:00:00.000Z"),
    "pool_day" : "mercoledì",
    "time_pointers" : "sera"
  },
  "owner" : 58012332,
  "chat_id" : -297619357,
  "proposals" : [
    {
      "propose" : "andare al cinema",
      "voted_by" : [ 45345, 345345 ]
    }
    {
      "propose" : "andare in pizziera",
      "voted_by" : [ 45123 ]
    }
  ],
  "message_id" : 2606
}
```

*Code snippet 3.1 - Struttura pool collection*

| Campo | Tipo | Descrizione |
|---|---|---|
| _id | ObjectId | Identificativo del sondaggio. |
| title | string | Titolo del sondaggio. |
| closed | boolean | Flag che indica se il sondaggio è stato chiuso o meno. |
| time_value.close_datetime | ISODate | Scadenza del sondaggio. |
| time_value.pool_day | string | Giorno in cui il sondaggio avrà termine. |
| time_value.time_pointers | string | Periodo temporale in cui il sondaggio avrà termine. |
| owner | number | Identificavo (telegram) dell'utente che ha aperto il sondaggio. |
| chat_id | number | Identificavo (telegram) della chat in cui è stato aperto il sondaggio. |
| proposals | array | Array contente tutte le proposte relative al sondaggio. |
| proposals.$.propose | string | Nome della proposta. |
| proposals.$.voted_by | array | Array contente tutti gli identificativi (telegram) dei votanti. |
| message_id | number | Identificavo (telegram) del messaggio a cui è stato associato il sondaggio. |

*Tabella 3.4 - Struttura pool collection*

### 3.3.2 - pending_propose

La collezione *pending_propose* contiene al suo interno documenti i quali identificano tutte le proposte identificate come nuove, in attesa di essere associate a un sondaggio non automaticamente riconosciuto. Un esempio di documento presente all'interno della collezione è il seguente:

```
{
  "_id" : ObjectId("5bdefe2120d8e70bd7c37fef"),
  "pools" : [
    {
      "title" : "Ciao ragazzi volete fare qualcosa? Io volevo ….",
      "_id" : ObjectId("5bdefdf620d8e70bd7c37fed")
    },
    {
      "title" : "Si ragazzi ma martedì sera cosa facciamo?",
      "_id" : ObjectId("5bdefe0720d8e70bd7c37fee")
    }
  ],
  "proposals" : [
    "giochiamo a scacchi"
  ],
  "from_user" : 58012332
}
```

*Code snippet 3.2 - Struttura pending_propose collection*

| Campo | Tipo | Descrizione |
|---|---|---|
| _id | ObjectId | Identificativo della nuova proposta. |
| pools | array | Array contente tutti i sondaggi alla quale la proposta potrebbe essere assegnata. |
| pools.$.title | string | Titolo del possibile sondaggio |
| pools.$._id | ObjectId | Identificativo del possibile sondaggio |
| proposals | array<string> | Array contente le nuova proposte identificate. |
| from_user | number | Identificavo (telegram) dell'utente che ha effettuato una nuova proposta. |

*Tabella 3.5 - Struttura pending_propose collection*

## 3.4 - POOLMANAGER

La classe *PoolManager* si occupa di gestire tutti i sondaggi e interfacciarsi con la collection *pool* di MongoDB.

| Nome | Parametri | Descrizione |
|---|---|---|
| init_pool | | Restituisce un dict con la struttura per un nuovo sondaggio con i valori di default. |
| get_pool | _id | Ritorna un sondaggio in base al suo ObjectID |
| close_pool | _id | Chiude un sondaggio in base al suo ObjectID |
| pools_same_day | chat_id, time_value | Ritorna True nel caso in cui vi è già un sondaggio aperto nello stesso gruppo e con gli stessi valori temporali. |
| get_opened_pools | chat_id, owner | Ritorna tutti i sondaggi aperti nel gruppo. Se passato anche il parametro owner ritorna quelli aperti da un singolo utente. |
| get_pools_toclose | time_pointer | Ritorna tutti i sondaggi da chiudere aventi valori temporali corrispondenti al parametro. |
| new_pool | pooldict | Inserisce un nuovo sondaggio all'interno della collection. |
| update_pool | _id, pooldict | Aggiorna un sondaggio. |

*Tabella 3.6 - Struttura classe PoolManager*

## 3.5 - PROPOSEMANAGER

La classe *ProposeManager* si occupa di gestire eventuali nuove proposte e interfacciarsi con la collection *pending_propose* di MongoDB.

| Nome | Parametri | Descrizione |
|---|---|---|
| get_propose | _id | Ritorna un'eventuale proposta pendente tramite il suo ObjectId. |
| new_propose | proposedict | Salva una nuova proposta pendente all'interno del database. E ne ritorna il suo ObjectId. |
| update_propose | _id, proposedict | Aggiorna una proposta pendente all'interno della collection. Solitamente questo aggiornamento viene fatto per assegnargli il message_id ove è presente il dubbio di assegnazione. |
| add_proposal | pool_propose, new_propose | Dato un array di proposte prese da un sondaggio ne aggiunge di nuove escludendo eventuali duplicati. |

*Tabella 3.7 - Struttura classe ProposeManager*

## 3.6 - DAYMANAGER

La classe DayManager tramite l'utilizzo delle librerie *locale, calendar* e *datetime* gestisce tutti i parametri temporali presenti all'interno del progetto.

| Nome | Parametri | Descrizione |
|---|---|---|
| get_day | day, index | Ogni giorno della settimana è ordinato all'interno di un array. Se viene passato il parametro day viene ritornato il giorno corrispondente ad oggi più il parametro. Se invece viene passato l'indice viene ritornato direttamente il giorno corrispondente. |
| get_today | | Ritorna il giorno della settimana corrispondente al giorno corrente. |
| day_to_add | _datetime, day | Dato in input un oggetto di tipo datetime e l'indice corrispondente a un giorno della settimana, questa funzione ritorna la differenza di giorni. |
| add_day | day | Aggiunge il giorno passato come parametro al datetime corrente. |
| get_close_pool | _datetime, hour | Ritorna il datetime passato come parametro avente però come orario il parametro hour. |
| is_day | word | Data una parola ritorna l'indice corrispondete al giorno della settimana. |

*Tabella 3.8 - Struttura classe DayManager*

## 3.7 - MAIN.PY

Come detto all'inizio del capitolo il file *main.py* si occupa di gestire tutte le classi e librerie descritte finora. Quando uno script python viene lanciato da riga di comando il codice ad essere eseguito è quello sottostante al *"__main__"*. Innanzitutto, viene inizializzato l'Updater importato dalla libreria *telegram.ext* e ad esso vengono registrati tutti gli handler necessari. Più eventuali Job.

Gli handler registrati in questo caso sono solamente tre: *analyzes_message, close_pool e button.*

```
if __name__ == '__main__':
    updater = Updater(token, request_kwargs={'read_timeout': 20, 'connect_timeout': 20})
    dp = updater.dispatcher
    job = updater.job_queue
    for pointer in time_pointers:
        job.run_daily(
            tick_pool,
            datetime.time(time_pointers[pointer], 00, 00),
            context=pointer,
            name='POOL TICK {}:00 - {}'.format(time_pointers[pointer], pointer.capitalize())
        )
    dp.add_handler(CommandHandler("close_pool", close_pool))
    dp.add_handler(MessageHandler(Filters.text, analyzes_message))
    dp.add_handler(CallbackQueryHandler(button))
    dp.add_error_handler(error)
    updater.start_polling(timeout=25)
    updater.idle()
```

*Code snippet 3.3 - __main__*

### 3.7.1 - analyzes_message

La funzione *analyzes_message* viene richiamata ogni qualvolta un utente invia un messaggio all'interno del gruppo che sia solamente di tipo testo. Il messaggio viene suddiviso in più sotto frasi analizzando la punteggiatura qualora fosse presente. Per ogni sotto frase tramite TreeTagger vengono classificate tutte le parole presenti all'interno di essa. Vengono contante le negazioni per ogni sotto frase in quanto, un numero dispari di negazioni identifica la volontà di *non* voler compiere una determinata azione. Nel caso in cui le negazioni fossero pari l'esecuzione del codice continua iterando ogni parola, denominata *tag* all'interno del codice. Tutti i controlli vengo eseguiti in cascata.

| Condizione | Conseguenza |
|---|---|
| La parola è un ADV ed è identificata come *day_transformers.* | All'interno del dict inizializzato tramite poolmanager.*init_pool()* poco prima dell'iterazione delle frasi viene aggiornato all'interno di *time_value* il valore *pool_day* corrispondente al giorno identificato dalla parola. |
| La parola è un NOM. | Se la parola è un giorno allora viene aggiornato *pool_day* all'interno di *time_value* per il sondaggio che si sta cercando di aprire.<br>Se la parola viene identificata come *time_pointer* viene aggiornata la variabile *time_pointers* all'interno di *time_value*.<br>Se la parola è un mezzo di trasporto allora viene ignorata. |
| La parola è un VER o un NOM. | Vengono eseguiti i controlli spiegati nel capitolo 2.1. per determinare se vi è un eventuale proposta. Analizzando la parola successiva e precedente a quella iterante. In caso affermativo se la proposta non è già presente viene inserita in un array. |
| La parola è un VER. | Come spiegato nel sotto capitolo 2.1.1 se il verbo è *fare* o *andare* e nella frase è presente il punto interrogativo allora la frase indica la volontà di aprire un nuovo sondaggio. |
| La parola è un DET o una PRE | Ne viene salvato l'indice corrispondente e si aggiorna un flag che verrà controllato alla prossima iterazione. Più precisamente nella condizione in cui la parola è sia un verbo che un nome. |

*Tabella 3.9 - Condizioni per la ricerca di una nuova proposta e/o sondaggio*

Completate tutte le iterazioni vengono analizzati i dati raccolti. Se è stata trovata l'intenzione di voler aprire un nuovo sondaggio si chiede alla classe PoolManager se vi è la possibilità di aprire un nuovo sondaggio che non vada in contrasto con quelli già aperti. In caso affermativo viene inserito il sondaggio all'interno del database e inviato come messaggio di risposta nel gruppo. In caso negativo viene controllato la lista *proposal*

popolata con tutte le nuove proposte eventualmente presenti nel messaggio. Se il sondaggio aperto nel gruppo è uno solo la proposta viene semplicemente aggiunta ad esso. Se i sondaggi sono più di uno vengono controllati i valori temporali presenti in *time_value*. Se viene identificato un sondaggio che soddisfa tutte le condizioni allora la proposta viene automaticamente inserita nel sondaggio, altrimenti viene chiesto all'utente di selezionare un'indagine di riferimento.

### 3.7.2 - button

Ogni qualvolta viene premuto un pulsante all'interno della chat viene richiamata la funzione *button*. I pulsati possono corrispondere a *proposte, sondaggi da chiudere* e *sondaggi a cui assegnare la proposta.* Ogni pulsante contiene al suo interno una stringa da 1 a 64 bytes identificata come *callback_data.* All'interno di ogni *callback_data* è presente un JSON che aiuta la funzione button a capire il motivo per il quale è stato premuto il bottone. Distinguiamo appunto tre casi:

| **Tipo** | **Operazione** |
|---|---|
| VOTE | Si sta cercando di aggiungere un voto ad una determinata proposta. Si ricerca all'interno del JSON l'ObjectId corrispondete al sondaggio e l'indice corrispondente alla proposta. Successivamente viene inserito il voto nel database. |
| CLOSE | Si sta cercando di chiudere uno tra i tanti sondaggi aperti. Anche qui all'interno del JSON è presente l'ObjectId del sondaggio da chiudere. |
| CHOICE | Vi era un dubbio sull'assegnamento di una proposta ad uno dei tanti sondaggi aperti. All'interno del JSON troviamo l'ObjectId della *pending_propose* in modo tale da poter completare l'assegnamento. |

*Tabella 3.10 - Tipologie callback_data*

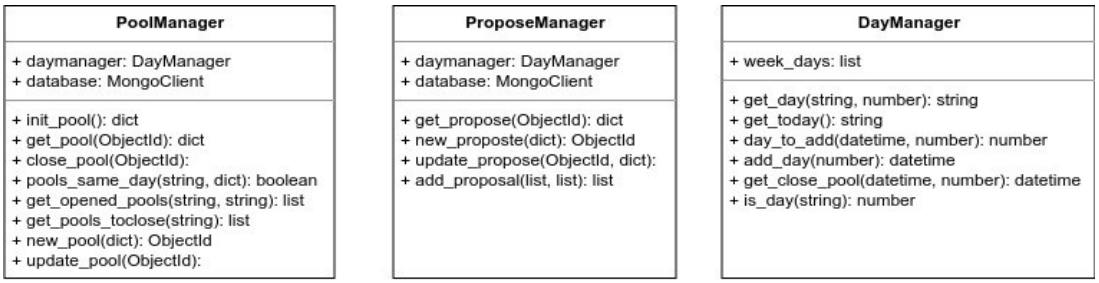


*Figura 3.4 - UML Classi PoolManager, ProposeManager e DayManager*

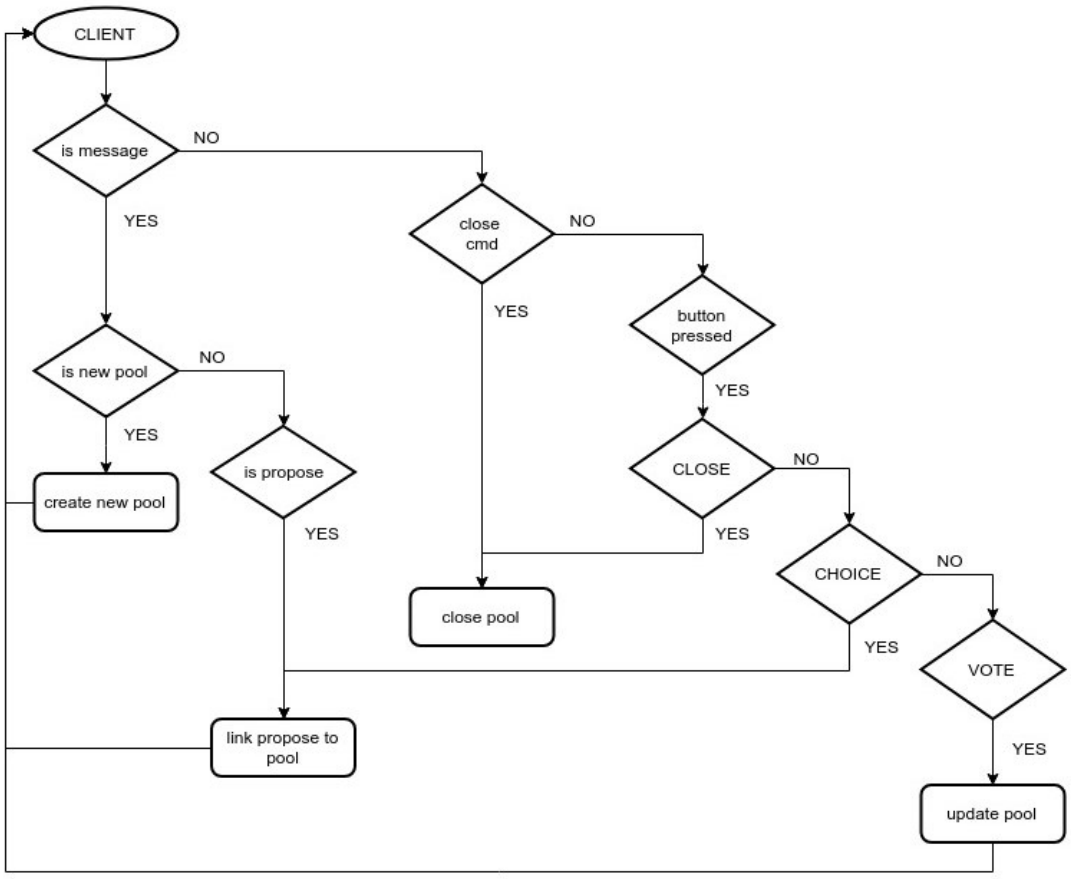


*Figura 3.3 - Diagramma di flusso del progetto*

# Capitolo 4 - RISULTATI E CONCLUSIONI

Il Bot è stato sviluppato sullo storico di alcune chat già in mio possesso, ed è stato successivamente testato su alcuni gruppi di colleghi, amici e coinquilini. I risultati ottenuti non sono stati del tutto deludenti e il Bot non è risultato nemmeno troppo invasivo all'interno del gruppo. Purtroppo, sono state trovate delle imprecisioni sulle risposte dirette dove, la nuova proposta era composta solamente da un nome, come ad esempio:

- D: Ciao ragazzi, cosa facciamo questa sera?
- R: Cinema!

Un eventuale sviluppo futuro potrebbe essere quello di storicizzare tutte le proposte avute precedentemente in modo tale che il Bot possa automaticamente apprendere e identificare le parole come un determinato luogo.

Un altro problema riscontrato durante l'utilizzo del Bot è l'incapacità di riconoscere i nomi dei locali. Un'eventuale soluzione potrebbe essere quella di ricercare il locale online e capire se si tratta di una discoteca, pizzeria e/o altro.

Di seguito un esempio di conversazione realmente avvenuta la quale ha portato alla creazione di più sondaggi e multiple proposte.

1. *"Weilà ragazzi come state?"*

| **Sotto frasi generate** | **Azione** | **Corretto** |
|---|---|---|
| *Weilà ragazzi come state?* | Ignorata | SI |

*Tabella 4.1 - Analisi del messaggio 1*

| **word** | **pos** | **lemma** |
|---|---|---|
| weilà | NOM | weilà |
| ragazzi | NOM | ragazzo |
| come | CON | come |
| state | VER:pper | stare |
| ? | PON | ? |

*Tabella 4.2 - Risultato del tagger sul messaggio 1*

2. *"Tutto bene! Avete impegni per questa sera? Volete andare da qualche parte?"*

| **Sotto frasi generate** | **Azione** | **Corretto** |
|---|---|---|
| *Tutto bene!* | Ignorata | SI |
| *Avete impegni per questa sera?* | Ignorata | SI |
| *Volete andare da qualche parte?* | Apertura di un nuovo sondaggio | SI |

*Tabella 4.3 - Analisi del messaggio 2*

| **word** | **pos** | **lemma** | **word** | **pos** | **lemma** |
|---|---|---|---|---|---|
| tutto | PRO:indef | tutto | avete | VER:pres | avere |
| bene | NOM | bene | impegni | NOM | impegno |
| ! | PON | ! | per | PRE | per |
| | | | questa | PRO:demo | questo |
| | | | sera | NOM | sera |
| | | | ? | PON | ? |

| **word** | **pos** | **lemma** |
|---|---|---|
| volete | VER:pres | volere |
| andare | VER:infi | andare |
| da | PRE | da |
| qualche | ADJ | qualche |
| parte | NOM | parte |
| ? | PON | ? |

*Tabella 4.4 - Risultato del tagger sul messaggio 2*

3. *"No, sono libero. Potremmo andare in discoteca"*

| **Sotto frasi generate** | **Azione** | **Corretto** |
|---|---|---|
| *No, sono libero.* | Ignorata | SI |
| *Potremmo andare in discoteca* | Aggiunta di una nuova proposta | SI |

*Tabella 4.5 - Analisi del messaggio 3*

| word | pos | lemma | word | pos | lemma |
|---|---|---|---|---|---|
| no | ADV | no | potremmo | VER:cond | potere |
| , | PON | , | andare | VER:infi | andare |
| sono | VER:pres | essere | in | PRE | in |
| libero | ADJ | libero | discoteca | NOM | discoteca |
| . | SENT | . | | | |

*Tabella 4.6 - Risultato del tagger sul messaggio 3*

4. *"uhmm io pensavo più ad una serata relax. Magari andiamo a prendere una birra"*

| Sotto frasi generate | Azione | Corretto |
|---|---|---|
| *uhmm io pensavo più ad una serata relax* | Ignorata | SI |
| *Magari andiamo a prendere una birra* | Aggiunta di una nuova proposta | SI |

*Tabella 4.7 - Analisi del messaggio 4*

| word | pos | lemma | word | pos | lemma |
|---|---|---|---|---|---|
| uhmm | NOM | uhmm | potremmo | VER:cond | potere |
| io | PRO:pers | io | andare | VER:infi | andare |
| pensavo | VER:impf | pensare | in | PRE | in |
| più | ADV | più | discoteca | NOM | discoteca |
| ad | PRE | ad | | | |
| una | DET:indef | una | | | |
| serata | NOM | serata | | | |
| relax | NOM | relax | | | |

*Tabella 4.8 - Risultato del tagger sul messaggio 4*

5. *"uhmm io passo, ho da fare"*

| Sotto frasi generate | Azione | Corretto |
|---|---|---|
| *io passo, ho da fare* | Aggiunta di una nuova proposta | NO |

*Tabella 4.9 - Analisi del messaggio 5*

| word | pos | lemma |
|---|---|---|
| io | PRO:pers | io |
| passo | ADJ | passo |
| , | PON | , |
| ho | VER:pres | avere |
| da | PRE | da |
| fare | VER:infi | fare |

*Tabella 4.10 - Risultato del tagger sul messaggio 5*

6. *"Io non ci sono. Non posso uscire con voi"*

| Sotto frasi generate | Azione | Corretto |
|---|---|---|
| *Io non ci sono* | Ignorata | SI |
| *Non posso uscire con voi* | Ignorata | SI |

*Tabella 4.11 - Analisi del messaggio 6*

| word | pos | lemma | word | pos | lemma |
|---|---|---|---|---|---|
| no | ADV | no | non | ADV | non |
| , | PON | , | posso | VER:pres | potere |
| sono | VER:pres | essere | uscire | VER:infi | uscire |
| libero | ADJ | libero | con | PRE | con |
| . | SENT | . | voi | NOM | voi |

*Tabella 4.12 - Risultato del tagger sul messaggio 6*

7. *"Sabato a pranzo invece cosa facciamo?"*

| Sotto frasi generate | Azione | Corretto |
|---|---|---|
| *Sabato a pranzo invece cosa facciamo?* | Apertura di un nuovo sondaggio | SI |

*Tabella 4.13 - Analisi del messaggio 7*

| word | pos | lemma |
|---|---|---|
| sabato | NOM | sabato |
| a | PRE | a |
| pranzo | NOM | pranzo |
| invece | ADV | invece |
| cosa | NOM | cosa |
| facciamo | VER:pres | fare |
| ? | PON | ? |

*Tabella 4.14 - Risultato del tagger sul messaggio 7*

8. *"Io direi di mangiare una bella carbonara da Sapori di Pasta"*

| Sotto frasi generate | Azione | Corretto |
|---|---|---|
| *Io direi di mangiare una bella carbonara da Sapori di Pasta* | Identificata una nuova proposta con dubbio sul sondaggio | NO.<br>La proposta identificata è stata errata |

*Tabella 4.15 - Analisi del messaggio 8*

| word | pos | lemma |
|---|---|---|
| io | PRO:pers | io |
| direi | VER:cond | dire |
| di | PRE | di |
| mangiare | VER:infi | mangiare |

| | | |
|---|---|---|
| una | DET:indef | una |
| bella | NOM | bella |
| carbonara | ADJ | carbonaro |
| da | PRE | da |
| sapori | NOM | sapore |
| di | PRE | di |
| pasta | NOM | pasta |

*Tabella 4.16 - Risultato del tagger sul messaggio 8*

9. *"Io invece voglio andare a mangiare un panino"*

| **Sotto frasi generate** | **Azione** | **Corretto** |
|---|---|---|
| *Io invece voglio andare a mangiare un panino* | Identificata una nuova proposta con dubbio sul sondaggio | SI |

*Tabella 4.17 - Analisi del messaggio 9*

| **word** | **pos** | **lemma** |
|---|---|---|
| io | PRO:pers | io |
| invece | ADV | invece |
| voglio | VER:pres | volere |
| andare | VER:infi | andare |
| a | PRE | a |
| mangiare | VER:infi | mangiare |
| un | DET:indef | un |
| panino | NOM | panino |

*Tabella 4.18 - Risultato del tagger sul messaggio 9*

10. *"naah, sabato a pranzo mangiamo una piadina"*

| Sotto frasi generate | Azione | Corretto |
|---|---|---|
| *naah, sabato a pranzo mangiamo una piadina* | Identificata una nuova proposta e assegnata in automatico al sondaggio | SI |

*Tabella 4.19 - Analisi del messaggio 10*

| word | pos | lemma |
|---|---|---|
| naah | NOM | naah |
| , | PON | , |
| sabato | NOM | sabato |
| a | PRE | a |
| pranzo | NOM | pranzo |
| mangiamo | VER:pres | mangiare |
| una | DET:indef | una |
| piadina | NOM | piadina |

*Tabella 4.20 - Risultato del tagger sul messaggio 10*

11. *"Anche io voglio mangiare la pasta"*

| Sotto frasi generate | Azione | Corretto |
|---|---|---|
| *Anche io voglio mangiare la pasta* | Identificata una nuova proposta con dubbio sul sondaggio. Ignorata poi in fase di assegnazione. | SI |

*Tabella 4.21 - Analisi del messaggio 11*

| word | pos | lemma |
|---|---|---|
| anche | ADV | anche |
| io | PRO:pers | io |
| voglio | VER:pres | volere |
| mangiare | VER:infi | mangiare |
| la | DET:def | il |
| pasta | NOM | pasta |

*Tabella 4.22 - Risultato del tagger sul messaggio 11*

12. *"Cosa volete fare venerdì pomeriggio?"*

| Sotto frasi generate | Azione | Corretto |
|---|---|---|
| *Cosa volete fare venerdì pomeriggio?"* | Apertura di un nuovo sondaggio | SI |

*Tabella 4.23 - Analisi del messaggio 12*

| word | pos | lemma |
|---|---|---|
| cosa | NOM | cosa |
| volete | VER:pres | volere |
| fare | VER:infi | fare |
| venerdì | NOM | venerdì |
| pomeriggio | NOM | pomeriggio |
| ? | PON | ? |

*Tabella 4.24 - Risultato del tagger sul messaggio 12*

Come dimostrano gli esempi sopra riportati i risultati si sono rilevati piuttosto soddisfacenti. Su dodici messaggi inviati sono state create ben diciassette sotto frasi generando a sua volta tre sondaggi e un totale di sette proposte. Purtroppo, due delle proposte sono state categorizzate in maniera errata. La frase numero 5 ha portato alla creazione della proposta *“fare”* assegnata al sondaggio *"Tutto bene! Avete impegni per*

*questa sera? ... "*, la quale però non ha alcun senso nel contesto. Nella frase numero 8 invece la nuova proposta identificata è stata solamente *"mangiare una bella"* rendendo vaga così la proposta fatta dall'utente.

La struttura del database finale sulla base della conversazione d'esempio è la seguente:

| **Titolo** | **Proposte** | **Giorno** | **Orario** |
|---|---|---|---|
| *Tutto bene! Avete impegni per questa sera? Volete andare da qualche parte?* | Andare in discoteca<br>Prendere una birra<br>Fare | Lunedì | Sera |
| *Sabato a pranzo invece cosa facciamo?* | Mangiare una bella<br>Mangiare un panino<br>Mangiamo una piadina<br>Mangiare la pasta | Sabato | Pranzo |
| *Cosa volete fare venerdì pomeriggio?* | | Venerdì | Pomeriggio |

*Tabella 4.25 - Struttura finale del database*

Per concludere, tre screen che mostrano come il bot si interfaccia con l'utente.


*Figura 4.1 - Sondaggio classico con 1 vincitore*

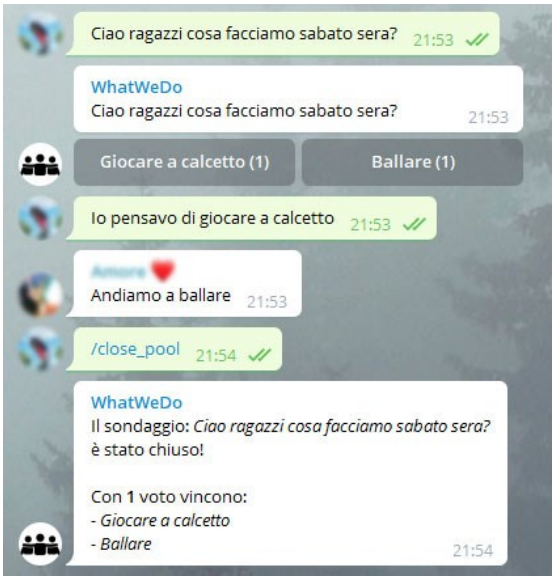


*Figura 4.2 - Sondaggio concluso con un pareggio*

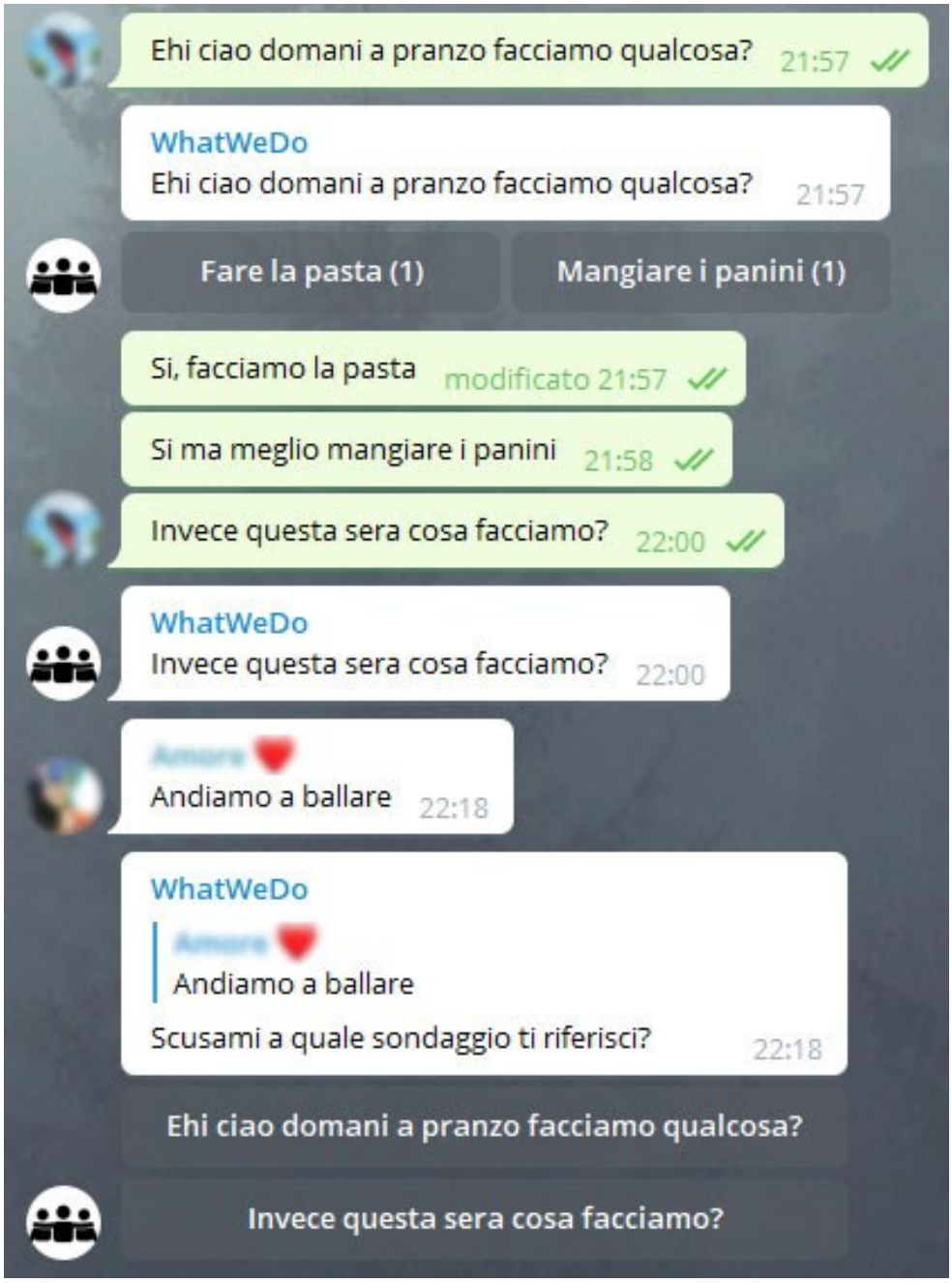


*Figura 4.3 - Inserimento di una nuova proposta senza parole chiavi*

# RIFERIMENTI


- http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
- http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger2.pdf
- http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger1.pdf
- http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/italian-tagset.txt
- https://treetaggerwrapper.readthedocs.io/en/latest/
- https://www.json.org/
- https://www.mongodb.com/it
- https://api.mongodb.com/python/current/
- https://core.telegram.org/bots/api
- https://www.python.org/
- https://docs.python.org/3/library/datetime.html
- https://python-telegram-bot.org/
- https://it.wikipedia.org/wiki/Elaborazione_del_linguaggio_naturale
- https://github.com/napolux/paroleitaliane

# RINGRAZIAMENTI

Ci tenevo a ringraziare innanzitutto il Prof. Tramontana per il tempo dedicatomi durante lo sviluppo di questa tesi e per la disponibilità avuta nei miei confronti durante il mio percorso accademico.

Un ringraziamento particolare va alla mia fidanzata la quale, ha sempre creduto in me e mi è stata vicina nei momenti in cui credevo di non farcela.

Un grazie immenso a tutti i miei amici e colleghi divenuti ormai compagni di avventure. Amici che, nonostante le mie ideologie e il mio carattere molto particolare mi sono sempre stati accanto.

Infine, la mia famiglia. Per tutti i sacrifici fatti fino ad ora e che stanno continuando a fare. Se sono qui è soprattutto grazie a loro che mi sostengono e credono nei miei sogni.