Bot Telegram per la ricerca di esami dei corsi di laurea appartenenti all'Università degli Studi di Perugia

Ingegneria del Software 2020-2021
Prof. Alfredo Milani

Bruno Lazo La Torre Montalvo
Matricola 300667

Sommario:

1. Obiettivo del progetto

Sviluppo di un bot Telegram che, dato l'input del nome del professore e/o nome della materia, appartenenti all'Università degli Studi di Perugia,da parte di un utente, restituisca le informazioni relative ai prossimi esami che si svolgeranno, compreso di collegamento al link della corrispondente aula virtuale

2. Glossario

-Telegram: applicazione di messaggistica istantanea gratuita, open source e multipiattaforma che permette anche di effettuare chiamate, videochiamate ed invio di contenuto multimediale
-Bot: programmi sviluppati dai programmatori che permettono l'esecuzione di determinate istruzioni in automatico oppure in risposta all'utente
-Web scraping: operazione che permette di collezionare dati ed informazioni da pagine web, tramite l'utilizzo di un programma software
-API: acronimo di Application programming interface e si tratta di applicazioni che, mediante modalità standard espongono le funzionalità di altre applicazioni

3. Requisiti

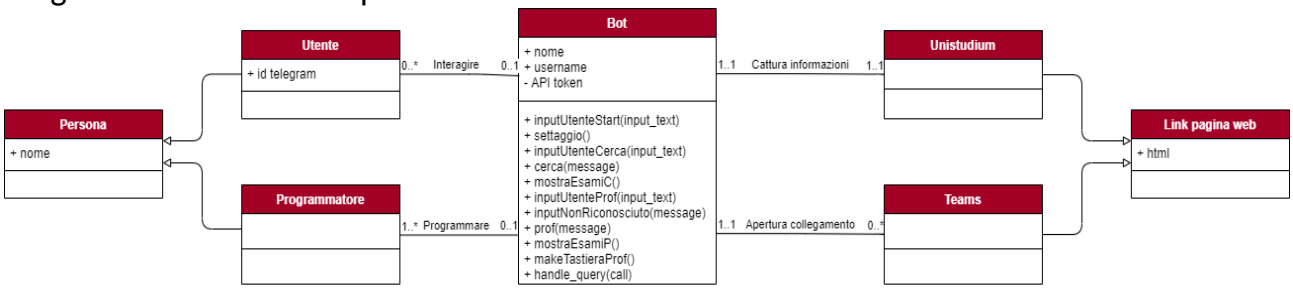
Definizione dei requisiti dell'utente:
-Il bot "RICERCA ESAMI UNIPG" invierà messaggi in chat riportando il nome del corso, docente, corso di laurea, data, ora dell'appello e link all'aula virtuale

Definizione dei requisiti del sistema:
-il sistema deve avere una breve guida tramite la quale poter facilmente utilizzare il bot
-sarà data la possibilità all'utente, tramite dei comandi, di filtrare le informazioni relative alle ricerche effettuate
-prima di mostra i risultati all'utente verrà avvertito del numero di risultati trovati e verrà richiesta una sua conferma, in modo tale da evitare un invio di messaggi non desiderati
-se si cerca il nome di un professore sarà data la possibilità di visualizzare gli esami relativi ad un determinato corso di laurea in cui insegna
-l'accesso al bot sarà disponibile 24 ore su 24

4. Architettura del sistema

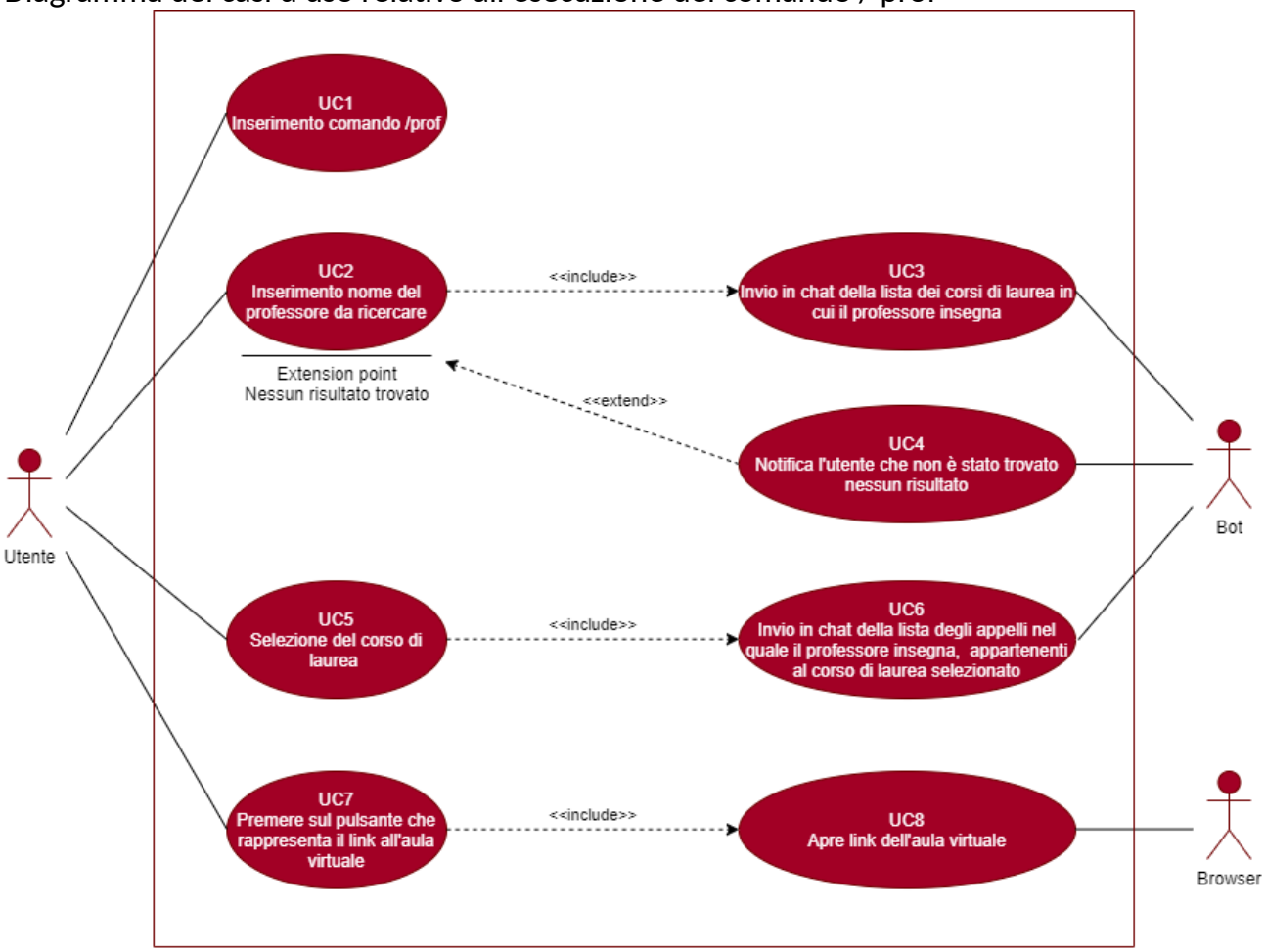
4.1. Diagramma di classe

Diagramma di classe completo



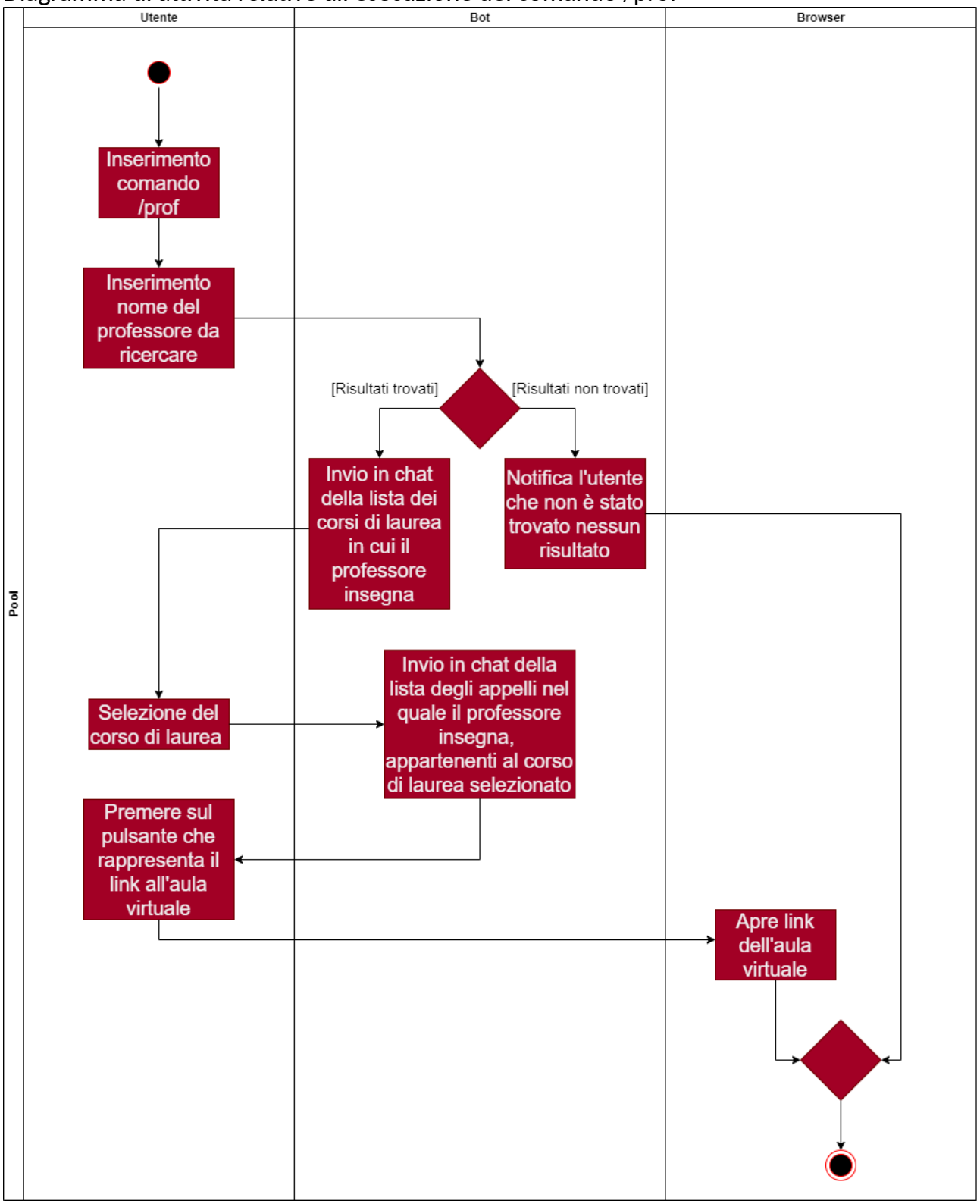
4.2. Diagramma dei casi d'uso

Diagramma dei casi d'uso relativo all'esecuzione del comando / prof



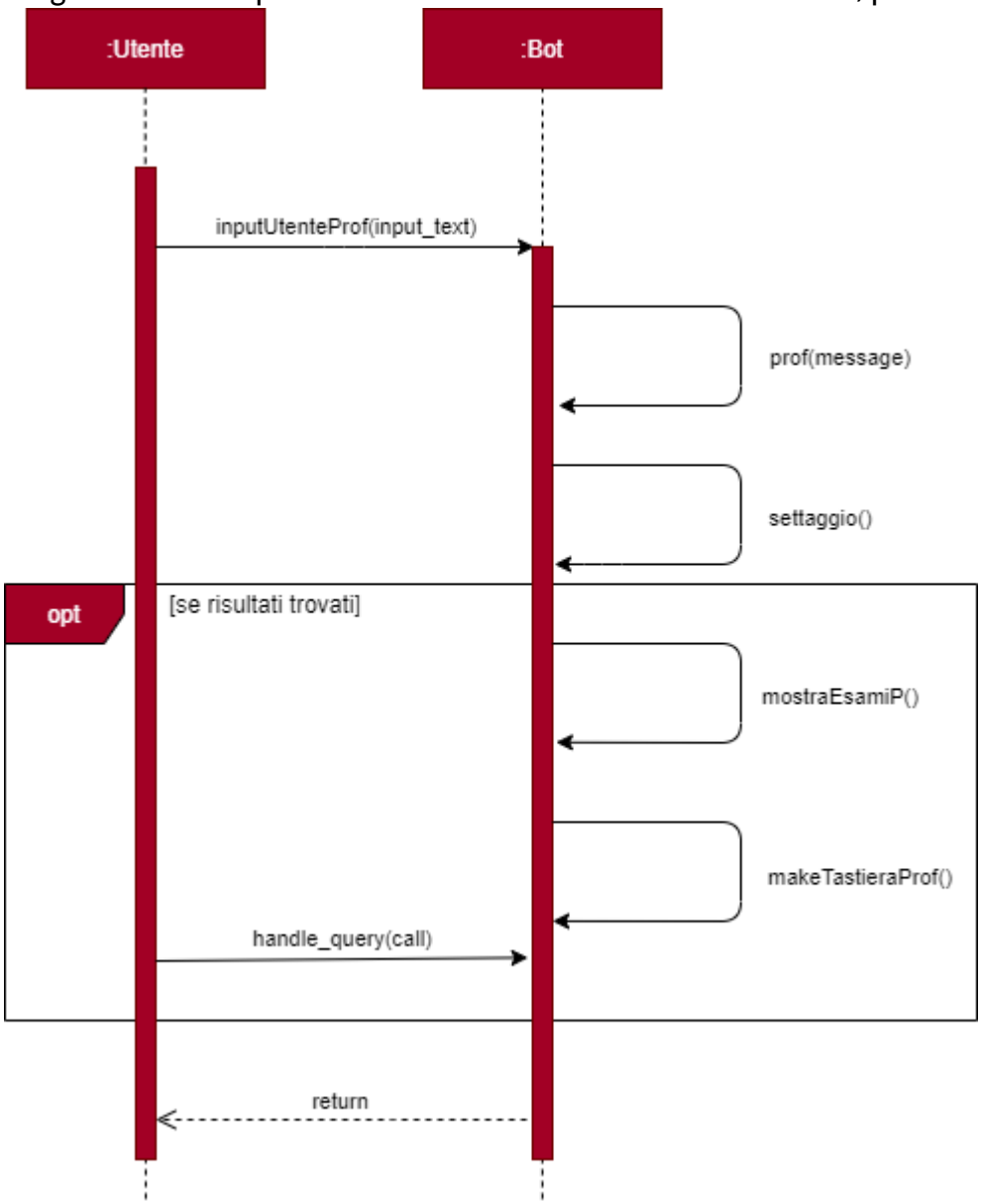
4.3. Diagramma di attività

Diagramma di attività relativo all'esecuzione del comando /prof



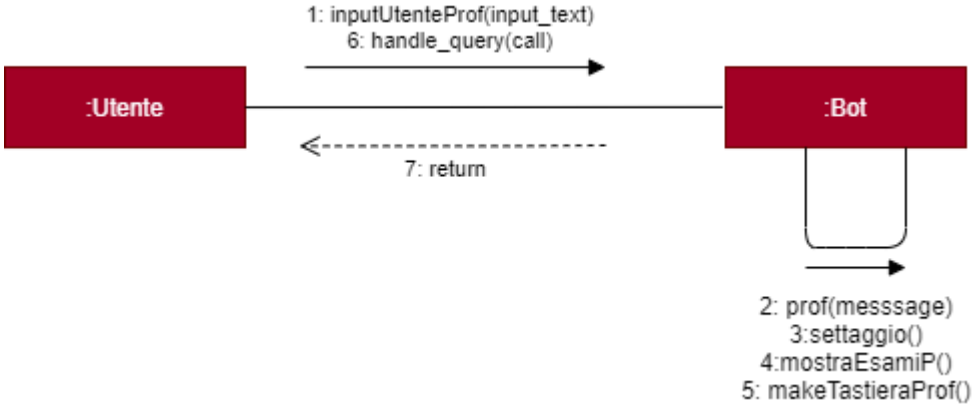
4.4. Diagramma di sequenza

Diagramma di sequenza relativo all'esecuzione del comando /prof



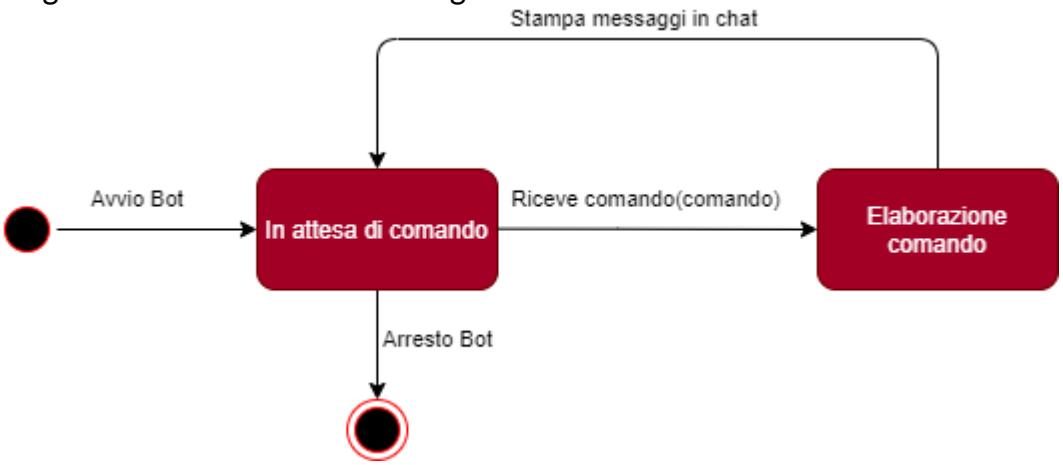
4.5. Diagramma di collaborazione

Diagramma di collaborazione relativo all'esecuzione del comando /prof



4.6. Diagramma di stato

Diagramma di stato del bot Telegram



5. Realizzazione e implementazione

Possibili modalità d'implementazione:
-implementare il bot telegram in modo che effettui delle operazioni di web scraping su www.unistudium.unipg.it e restituire direttamente all'utente i dati da lui richiesti.
-creare un database locale nel quale memorizzare il contenuto estratto da unistudium, per poi consentire al bot di accedervi e restituire all'utente i dati richiesti. Questo passaggio in più però è ridondante e si può bypassare tramite la prima implementazione proposta

La scelta del linguaggio che si occcuperà della programmazione del bot telegram è ricaduta su Python, il quale dispone di librerie con ampia documentazione che permettono l'implementazione di web scraping e di lavorare con l'API del bot Telegram.

Nella programmazione in Python è stato fatto uso dei package “pyTelegramBotAPI” per l'implementazione dei comandi, invio di messaggi e pulsanti all'interno del bot telegram, “beautifulsoup4” per web scraping quindi estrazione dati da pagine html, “requests” per accedere a siti web tramite richieste http

Il programma è in esecuzione all'interno di Heroku, una piattaforma che fornisce un servizio di cloud diretto verso la programmazione, operativa 24 ore su 24, 7 giorni su 7.

I comandi implementati a disposizione dell'utente sono:
- “/start” è un comando che permette di visualizzare l'intera lista di comandi e una breve descrizione di essi
- “/cerca” è un comando che permette di ricercare un insegnamento e/o nome del docente e di ricevere un messaggio in risposta contenente il numero di risultati trovati e se ricevuta conferma da parte dell'utente, gli verranno inviati dei messaggi in risposta contenti le informazioni relative all'esame con annesso collegamento ad aula virtuale
- “/prof” è un comando che permette di ricercare il nome del docente e darà la possibilità di visualizzare solo i risultati di un determinato corso di laurea nel quale il professore insegna

```python
import Costanti
import telebot
import bs4, requests

API_KEY=Costanti.API_KEY
bot=telebot.TeleBot(API_KEY)
LINK = "https://www.unistudium.unipg.it/cercacorso.php?p=1"

@bot.message_handler(commands=['start'])
def inputUtenteStart(input_text):
  bot.send_message(input_text.chat.id, "Lista di comandi:\n/start ti permetterà di visulizzare questa lista di comandi\n/ce

def settaggio():
  if(len(messaggioInput.text)<=200):
    response = requests.post(LINK, data={'query': messaggioInput.text})
    response.raise_for_status()
    soup = bs4.BeautifulSoup(response.text,'html.parser')
    return soup
  else:
    bot.send_message(messaggioInput.chat.id, "Il testo inserito non deve superare i 200 caratteri")
    return 0

@bot.message_handler(commands=['cerca'])
def inputUtenteCerca(input_text):
  sent = bot.send_message(input_text.chat.id, "Digita il nome dell'insegnamento e/o nome del docente")
  bot.register_next_step_handler(sent, cerca)

def cerca(message):
  global messaggioInput
  global htmlRighe
  messaggioInput = message
  soup=settaggio()
  if soup!=0:
    if soup.find('table'):
      htmlOrario = soup.find('table')
      htmlRighe = htmlOrario.find_all('tr')
      del htmlRighe[0]
      tastieraCerca = telebot.types.InlineKeyboardMarkup()
      tastieraCerca.add(telebot.types.InlineKeyboardButton('mostrare', callback_data='mostrare'))
      tastieraCerca.add(telebot.types.InlineKeyboardButton('non mostrare', callback_data='non mostrare'))
      bot.send_message(messaggioInput.chat.id,"Sono state trovate " + str(len(htmlRighe)) + " corrispondenze, mostrarle o

    else:
      bot.send_message(messaggioInput.chat.id,'Non è stata trovata nessuna corrispondenza.\nInserisci nuovamente il comando


def mostraEsamiC():
  for htmlRiga in htmlRighe:
    htmlCaselle = htmlRiga.find_all('td')
    testoRiga = ''
    linkEsame = ''
    for htmlCasella in htmlCaselle:
      if htmlCasella.find('a'):
        htmlEsame = htmlCasella.find('a')
        linkEsame = str(htmlEsame.get('href'))
      else:
        testoCasella = str(htmlCasella.text)
        testoRiga += (testoCasella + '\n')
    tastieraCercaEsame = telebot.types.InlineKeyboardMarkup()
    tastieraCercaEsame.add(telebot.types.InlineKeyboardButton(testoRiga, url=linkEsame))
    bot.send_message(messaggioInput.chat.id, testoRiga, reply_markup=tastieraCercaEsame)

@bot.message_handler(commands=['prof'])
def inputUtenteProf(input_text):
  sent = bot.send_message(input_text.chat.id, "Digita il nome del docente")
  bot.register_next_step_handler(sent, prof)

@bot.message_handler(func=lambda message: True)
def inputComandoNonRiconosciuto(message):
 bot.send_message(message.chat.id, "Inserire un comando valido, per maggiori informazioni riguardo ai comandi disponibili s

def prof(message):
  global messaggioInput
  global htmlRighe
  global listaCorsi
  messaggioInput = message
  soup = settaggio()
  if soup != 0:
    if soup.find('table'):
      htmlOrario = soup.find('table')
      htmlRighe = htmlOrario.find_all('tr')
      del htmlRighe[0]
      listaCorsi = []
      mostraEsamiP()
      bot.send_message(message.chat.id, "I corsi relativi al docente ricercato sono:", reply_markup=makeTastieraProf())
    else:
      bot.send_message(message.chat.id,'Non è stata trovata nessuna corrispondenza.\nInserisci nuovamente il comando /prof


def mostraEsamiP():
  for htmlRiga in htmlRighe:
    htmlCaselle = htmlRiga.find_all('td')
    numeroColonna = 0
    for htmlCasella in htmlCaselle:
      numeroColonna += 1
      testoCasella = str(htmlCasella.text)
      if numeroColonna == 3:
        presente = 0
        for corsoPresenteLista in listaCorsi:
          if corsoPresenteLista == testoCasella:
            presente += 1
        if presente == 0:
          listaCorsi.append(testoCasella)

def makeTastieraProf():
  tastieraProf = telebot.types.InlineKeyboardMarkup()
  for corsoPresenteLista in listaCorsi:
    corsoPresenteLista = corsoPresenteLista[:64]
    tastieraProf.add(telebot.types.InlineKeyboardButton(text=str(corsoPresenteLista), callback_data=str(corsoPresenteLista)
  return tastieraProf
```

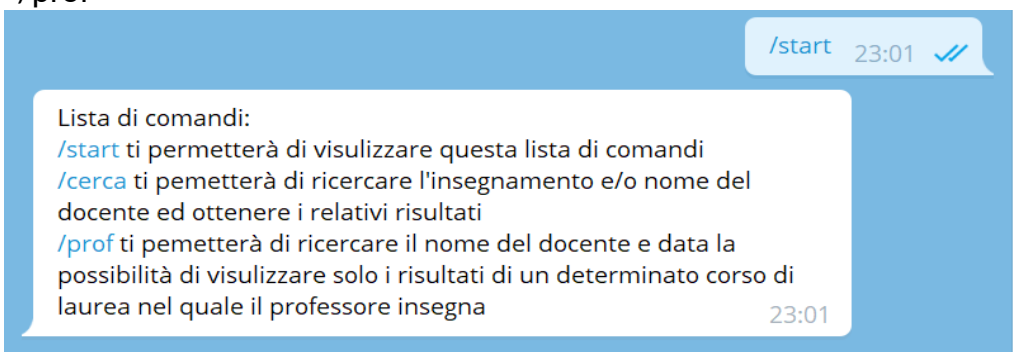
```python
@bot.callback_query_handler(func=lambda call: True)
def handle_query(call):
  if call.data == "mostrare":
    mostraEsamiC()
  elif call.data == "non mostrare":
    bot.send_message(messaggioInput.chat.id,"Inserisci nuovamente il comando /cerca e prova ad essere più specifico nella ri
  else:
    for htmlRiga in htmlRighe:
      htmlCaselle = htmlRiga.find_all('td')
      testoRiga = ''
      linkEsame = ''
      numeroColonna = 0
      corso = ''
      for htmlCasella in htmlCaselle:
        numeroColonna += 1
        if htmlCasella.find('a'):
          htmlEsame = htmlCasella.find('a')
          linkEsame = str(htmlEsame.get('href'))
        else:
          testoCasella = str(htmlCasella.text)
          testoRiga += (testoCasella + '\n')
          if numeroColonna == 3:
            corso = testoCasella
            corso = corso[:64]
      if call.data == corso:
        tastieraProfEsame = telebot.types.InlineKeyboardMarkup()
        tastieraProfEsame.add(telebot.types.InlineKeyboardButton(testoRiga, url=linkEsame))
        bot.send_message(messaggioInput.chat.id, testoRiga, reply_markup=tastieraProfEsame)

bot.polling()
```

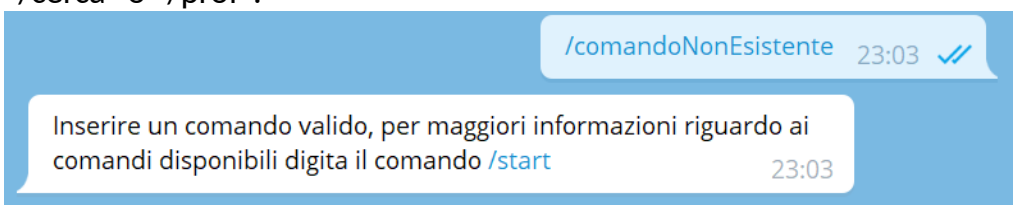
6. Casi di test

Il bot mette a disposizione dell'utente tre comandi: "/start", "/cerca", "/prof". Sono quindi presenti:

- una classe di test validi nel caso la stringa inserita dall'utente sia "/start" o "/cerca" o "/prof"

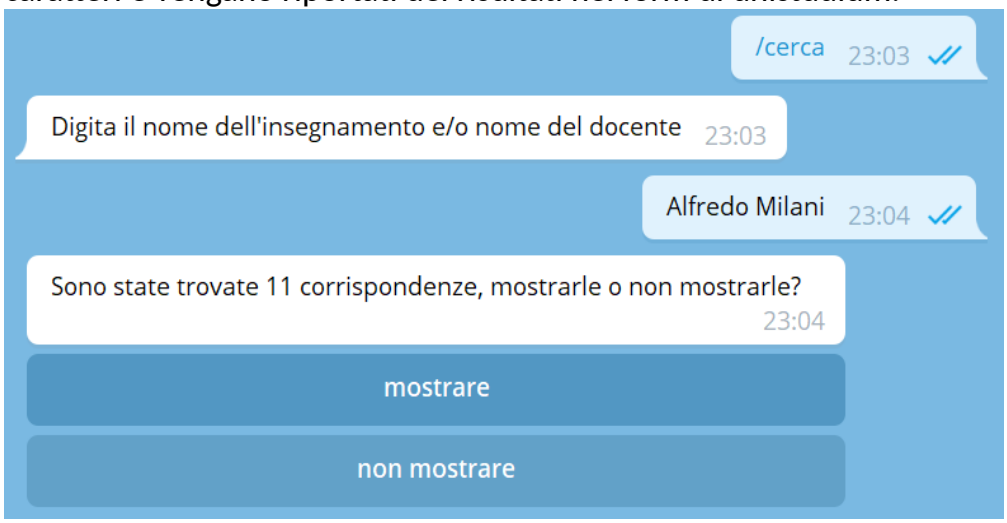


- una classe di test non validi nel caso la stringa inserita dall'utente sia diversa da "/start" o "/cerca" o "/prof".

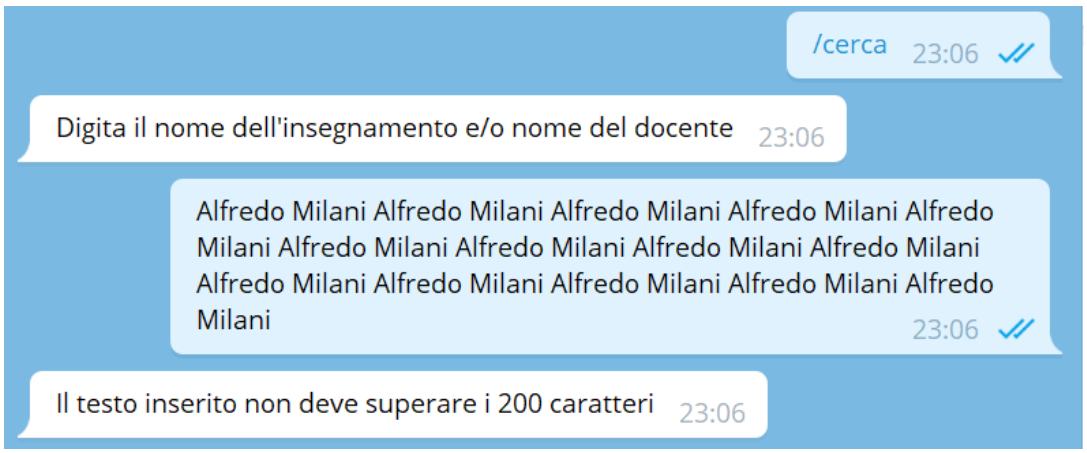


Nel caso venga inserito un comando tra "/cerca" o "/prof" e venga richiesto all'utente di inserire il nome del docente o dell'insegnamento, tale stringa potrà essere composta al massimo da 200 caratteri. Sono quindi presenti:

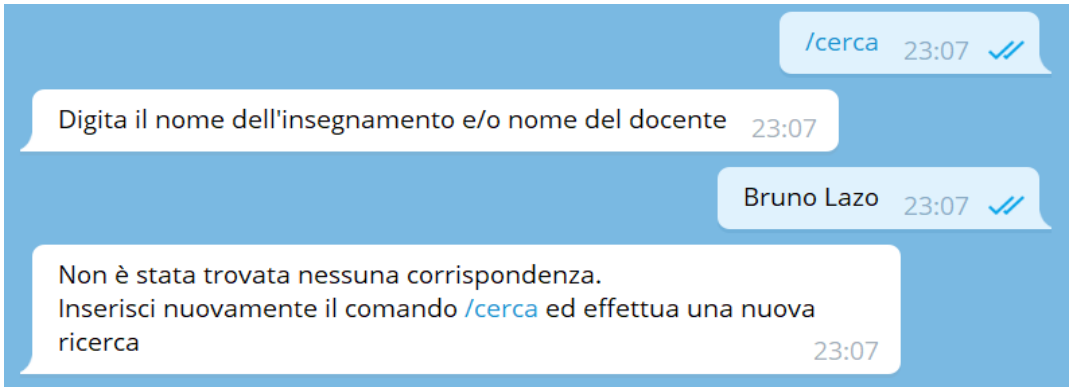
-una classe di test validi nel caso la stringa inserita dall'utente sia compresa tra 1 e 200 caratteri e vengano riportati dei risultati nel form di unistudium.



- una classe di test non validi nel caso la stringa inserita dall'utente sia superiore ai 200 caratteri.



- una classe di test non validi nel caso in cui stringa inserita dall'utente non riporti nessun risultato nel form di unistudium



7. Design Pattern

Nella programmazione in python del progetto è stato implementato indirettamente il Design pattern Chain of Responsibility, all'inserimento in chat di una qualsiasi stringa, il programma controllerà, nell'ordine in cui il codice è stato strutturato, quale tra i presenti gestori di messaggio è in grado di gestire l'input inserito; quindi la richiesta verrà trasmessa da un gestore ad un altro fino a quando non troverà un gestore che la gestisca.

```
@bot.message_handler(commands=['start'])
def inputUtenteStart(input_text):
  bot.send_message(input_text.chat.id, "Lista di comandi:\n/start ti permetterà di visulizzare questa lista di comandi\n
```

```
@bot.message_handler(commands=['cerca'])
def inputUtenteCerca(input_text):
  sent = bot.send_message(input_text.chat.id, "Digita il nome dell'insegnamento e/o nome del docente")
  bot.register_next_step_handler(sent, cerca)
```

```
@bot.message_handler(commands=['prof'])
def inputUtenteProf(input_text):
  sent = bot.send_message(input_text.chat.id, "Digita il nome del docente")
  bot.register_next_step_handler(sent, prof)
```

```
@bot.message_handler(func=lambda message: True)
def inputComandoNonRiconosciuto(message):
 bot.send_message(message.chat.id, "Inserire un comando valido, per maggiori informazioni riguardo ai comandi disponibili
```