PROIECT

-METODE DE DEZVOLTARE SOFTWARE -

MESSENGER BOT

Membrii:

- Buzoi Bianca-Nicoleta
- Necula Florin-Alexandru
- Oanea Ana Mălina
- Samoilescu Camelia

Cuprins

Alegerea proiectului şi împărţirea task-urilor

Primul pas a fost acela de a ne crea echipa, iar ulterior au urmat discuţii cu privire la tema pe care vrem să o dezvoltăm. Au fost nenumărate idei: implementare de jocuri, algoritmi matematici, algoritmi din geometria computaţională, etc. Însă, ne-am hotărât asupra unui Bot pentru o reţea de socializare, şi anume Facebook Messenger.

Cum ne-a venit această idee? Dată fiind situaţia de anul acesta în care cursurile face-to-face au fost oprite, am fost puşi în situaţia de a intra în contact unii cu alţii, prin intermediul aplicaţiilor online de socializare. Cea mai folosită a fost Discord. Astfel, ni s-a părut interesant să creăm un Bot, prin care să putem atât să comunicăm între noi, cât şi să creăm anumite feature-uri utile. Am ales să creăm un Messenger Bot, aceasta fiind una dintre cele mai utilizate reţele de socializare.

Un Bot este o aplicaţie software, implementată astfel încât să rezolve anumite task-uri. Acesta imită şi chiar înlocuieşte activitatea unui user.

Exista:

- ChatBots : Bot-uri care simulează o conversaţie între utilizatori prin faptul că răspunde la anumite întrebări/fraze bine definite, oferind un răspuns clar implementat
- GoogleBots : cei care extrag informaţii din paginile web
- SocialBots: cei realizaţi pentru platformele social media

Proiectul realizat implementează atât aspecte tipice unui ChatBot, cât şi al unui SocialBot, urmând a fi exemplificate caracteristicile fiecăruia.

Împărţirea task-urilor:

Crearea unui proiect trebuie să aibă la bază un plan bine-stabilit, în care fiecare membru are clar definite task-urile pe care le va implementa. Astfel, am definit câteva aspecte pe care să le avem în vedere în implementare:

- Să creăm structura efectivă a Botului: clase, metode şi să găsim limbaje de programare/module adecvate pentru a implementa această aplicaţie
- Să definim funcţionalităţile aferente acestui Bot
- Să creăm un server capabil să afişeze nişte interogări într-o pagina web
- Să testăm funcţionalităţile implementate
- Să asigurăm rularea continuă a serverului şi a Botului
- Să rezolvăm bug-urile care pot să apară pe parcurs

Am avut task-uri individuale, dar şi colective, pe care le-am rezolvat online, prin crearea de meeting-uri pe Discord în care fiecare dintre noi veneam cu idei, dezbăteam probleme apărute şi găseam soluţii.

Ca şi limbaj de programare am folosit Python deoarece acesta pune la dispoziţie librării ce ne-au fost de folos în implementare: Fbchat, Pyshorteners, Flask etc.

Pentru a implementa funcţia de vreme, am folosit API-ul de la OpenWeatherMap, o companie care oferă date despre vreme pentru milioane de utilizatori. Am utilizat varianta gratuită a acestui API, iar aceasta ne oferă 60 de request-uri pe minut şi 1.000.000 de request-uri pe lună. Aceste numere sunt suficiente pentru aplicaţia dezvoltată de noi. (https://openweathermap.org/)

Pentru partea de server, am apelat la Amazon AWS. Am reuşit să creăm o maşină virtuală, capabilă să primească request-uri de tip GET de la clientul nostru. Setarea nu a fost

foarte dificilă, procesul fiind unul foarte intuitiv. Pentru a rula maşina virtuală pe calculatorul personal, ne vom conecta la aceasta folosind protocolul SSH. Serverul nostru rulează pe portul 443, aferent protocolului HTTPS.

După configurarea iniţială, am început să instalăm dependinţele necesare rulării serverului nostru. Am folosit modulul Flask pentru a face serverul să ruleze.

Pentru a respecta standardele comunităţii Facebook(https://www.facebook.com/communitystandards/), am decis să adăugăm un url shortener la url-ul către serverul nostru. Am folosit tinyurl pentru aceasta.

Un aspect la care am avut foarte multă grijă a fost să creăm un cont de Facebook care să respecte standardele comunităţii, amintite mai sus. Pentru a evita situaţii de spam, din partea botului va fi trimis un singur răspuns per cerere. Totodată, conectarea la acest client se face cu consimţământul persoanelor dornice sa primească mesaje de la acesta. Aceste aspecte sunt vitale, deoarece Facebook are o politică foarte strictă cu privire la mesaje de tip Spam, care pot sau nu să conţină link-uri către site-uri necunoscute.

User stories

1. Necula Florin-Alexandru:

“Vreau să invit cât mai mulţi prieteni pe această aplicaţie, ca să ne bucurăm împreună de acest serviciu.”

2. Buzoi Bianca-Nicoleta:

“Vreau să-mi organizez programul mai bine, ca să am lucrurile sub control.”

3. Samoilescu Camelia:

“Vreau să comunic cu toţi colegii, ca să fiu la curent cu activităţile lor.”

4. Oanea Ana Mălina:

“Vreau să fiu mereu la curent cu datele meteo, ca să ştiu dacă am nevoie de umbrelă sau nu.”

5. Stoian Mihai:

“Vreau să schimb rapid culoarea la conversaţie, ca să nu fiu nevoit să o schimb din setări.”

6. Popescu Ion:

“Vreau să-mi gestionez echipa mai bine, ca să putem fii mai productivi.”

7. Ciobanu Ioan:

“Vreau o singură conversaţie în care să trimit mesaje, ca să nu trebuiască să dau mesaje separate la toţi prietenii mei.”

8. Roman Victoria:

“Vreau să mă integrez mai bine în lumea social media, ca să pot să fiu la curent cu tendinţele copiilor mei.”

9. Tănase Sorin:

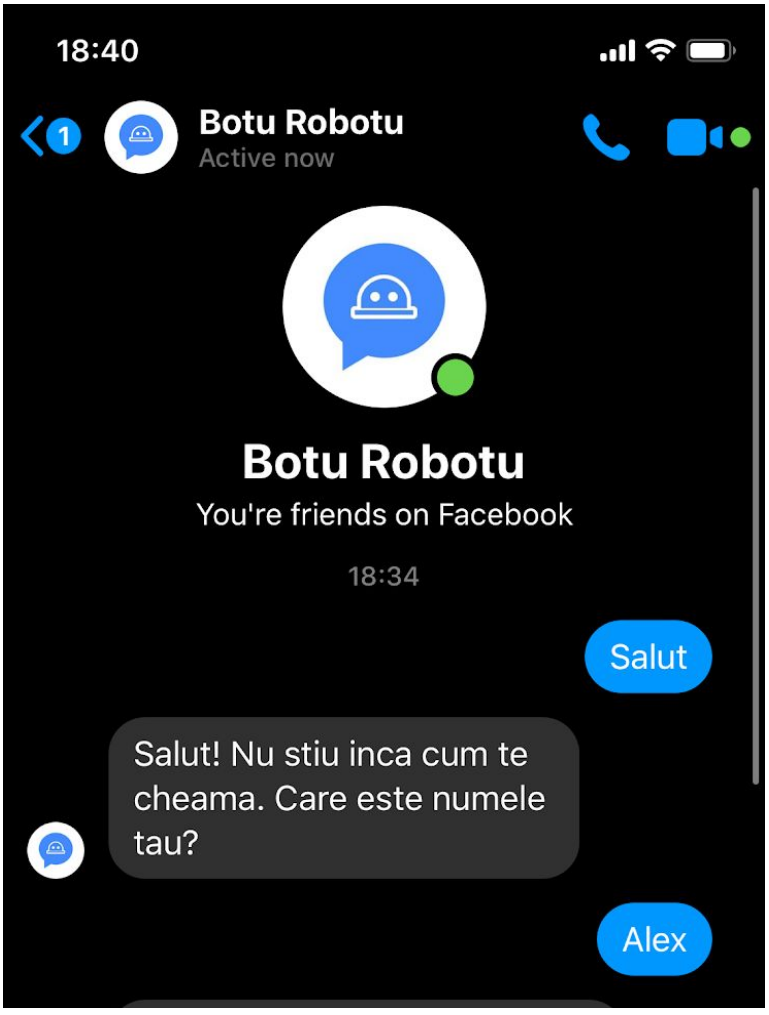
“Vreau să vorbesc cu toţi colegii mei în acelaşi timp, ca să putem învăţa împreună pentru examene.”

10. Tudor Anghel:

“Vreau să le trimit la toţi prietenii mei postări de pe Facebook, ca să vadă şi ei noutăţile din lumea jocurilor video.”

Functionalităţi:

- La primirea unui mesaj de la un autor necunoscut, Botul cere informaţii despre acesta: nume şi îl adaugă la lista sa de utilizatori.



Se creează o listă de User unici, în care sunt reţinuţi cei care se loghează

- Dupa ce s-a conectat, i se trimite un mesaj de
”Bun venit”

- La conectarea mai multor utilizatori, aplicația funcționează ca un ChatGroup: fiecare mesaj trimis de un utilizator conectat este văzut de toți ceilalți membrii ai conversației.

Pentru a se cunoaşte autorul mesajului, acesta va fi însoţit şi de numele celui care l-a expediat.

- De asemenea, Botul poate răspunde la dorinţa utilizatorului de a cere vremea într-o anumită locaţie.
  Pentru aceasta, este necesar să se trimită un mesaj cu textul:

“botule cum e vremea la X”
, unde X este oraşul pentru care se doreşte a se afla vremea

Botul se foloseşte de API-ul: OpenWeather API https://openweathermap.org/api, prin care găseşte temperatura unei locaţii, pe care o trimite ulterior în conversaţie.

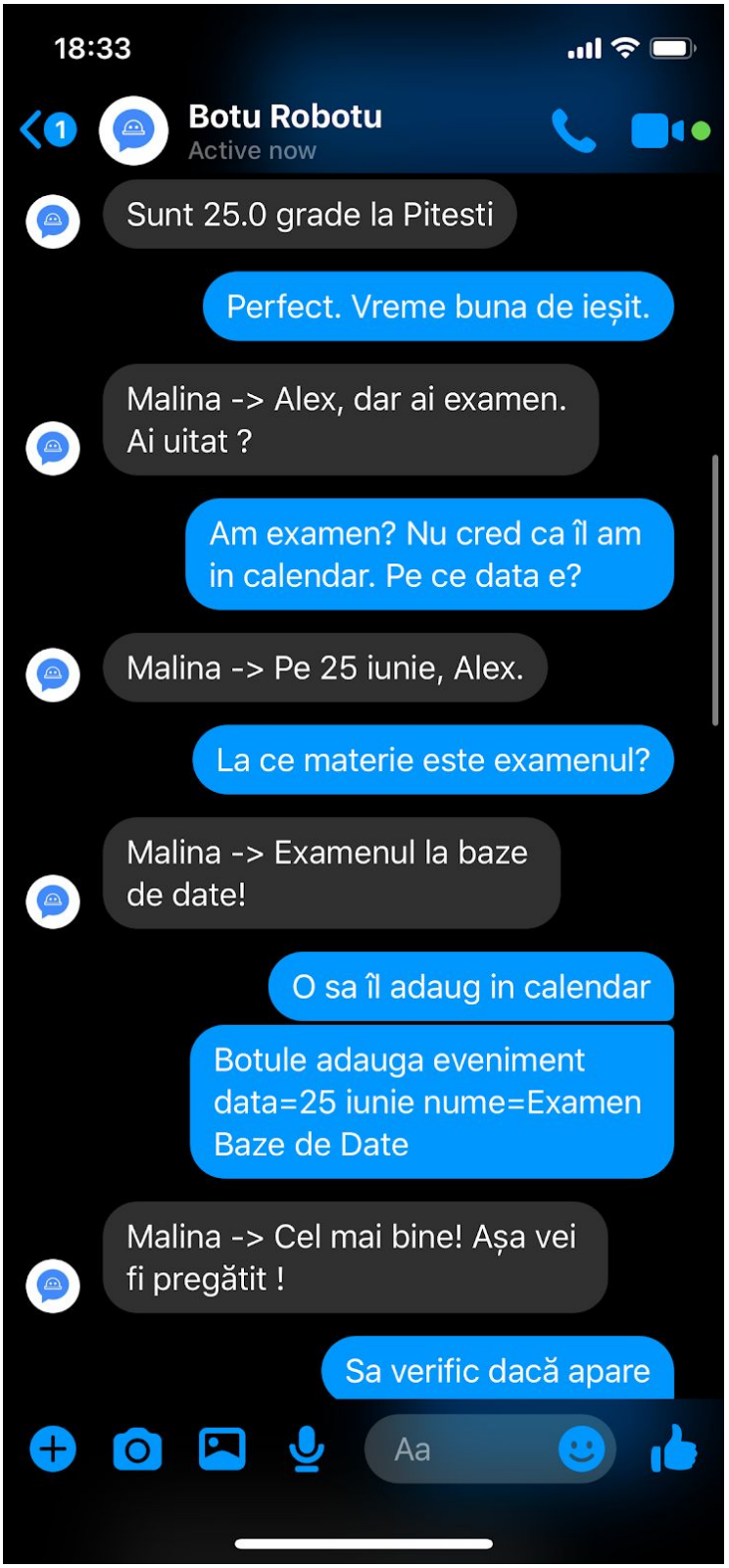
- Pentru a imita cât mai mult un comportament al unei aplicaţii de socializare des utilizată, am implementat comanda de a schimba nickname-ul unui utilizator:

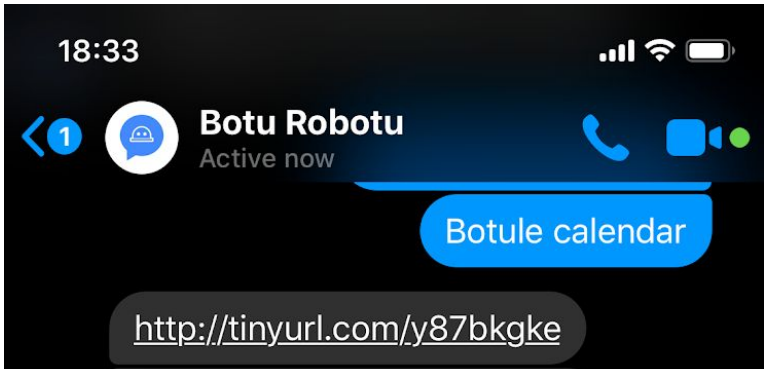
Prin mesajul:
“botule nickname X Y”
, unde X este numele actual al user-ului, iar Y reprezintă porecla

Dacă se dorește să se schimbe numele unui user inexistent, se va afișa un mesaj corespunzător.

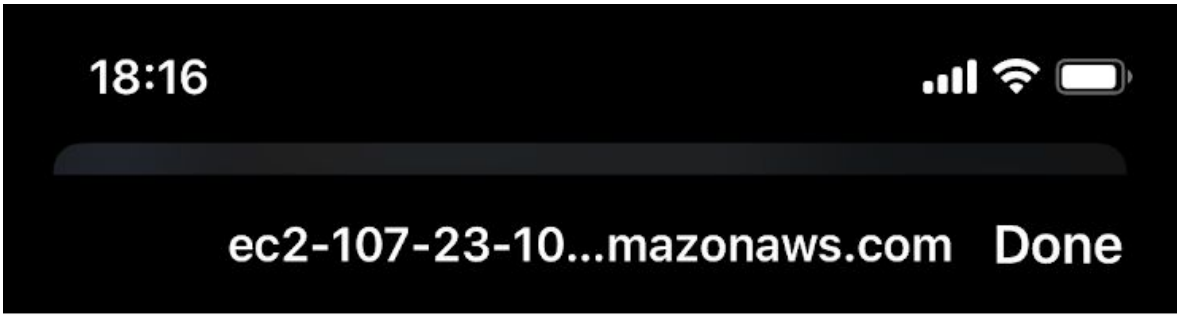
- Totodată, am dorit să ţinem evidenţa unui calendar al evenimentelor fiecărui utilizator:
  Clasa User are un câmp calendar de tip dicţionar, în care se reţine o listă de evenimente de tipul:
  Data : [Eveniment1, Eveniment2, ..]
  Există comanda:
  “botule adauga eveniment data=X nume=Y”
  , unde X este data evenimentului, iar Y numele acestuia

User-ul îşi poate vedea calendarul prin comanda:
,,botule calendar”.


18:33
Botu Robotu
Active now
Sunt 25.0 grade la Pitesti
Perfect. Vreme buna de ieșit.
Malina -> Alex, dar ai examen. Ai uitat ?
Am examen? Nu cred ca îl am in calendar. Pe ce data e?
Malina -> Pe 25 iunie, Alex.
La ce materie este examenul?
Malina -> Examenul la baze de date!
O sa îl adaug in calendar
Botule adauga eveniment data=25 iunie nume=Examen Baze de Date
Malina -> Cel mai bine! Așa vei fi pregătit !
Sa verific dacă apare
Aa



Odată cu trimiterea acestui mesaj, este generat un link propriu . La accesarea acestuia, User este redirecţionat către o pagina web în care apare calendarul lunii curente şi evenimentele sale.



June 2020

Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Evenimente:
La data de 25iunie aveti urmatoarele evenimente: examenbazededate,

- O altă funcţionalitate a aplicaţiei este capacitatea unui user de a schimba culoarea conversaţiei.
  Există comanda:
  ,,Botule new color {red, blue, orange, coral, rose}


16:54
Botu Robotu
Active now
Botu Robotu
You're friends on Facebook
16:54
Salut!
Salut! Nu stiu inca cum te cheama. Care este numele tau?
Alex
Alex a intrat pe server! Hai sa-l salutam
Nu prea îmi place culoarea asta. Vreau sa o schimb!
Aa


16:54
Botu Robotu
Active now
Botu Robotu
You're friends on Facebook
16:54
Salut!
Salut! Nu stiu inca cum te cheama. Care este numele tau?
Alex
Alex a intrat pe server! Hai sa-l salutam
Nu prea îmi place culoarea asta. Vreau sa o schimb!
Botule new color coral
Botu changed the chat theme to Coral Pink.
E mult mai bine așa!
Aa

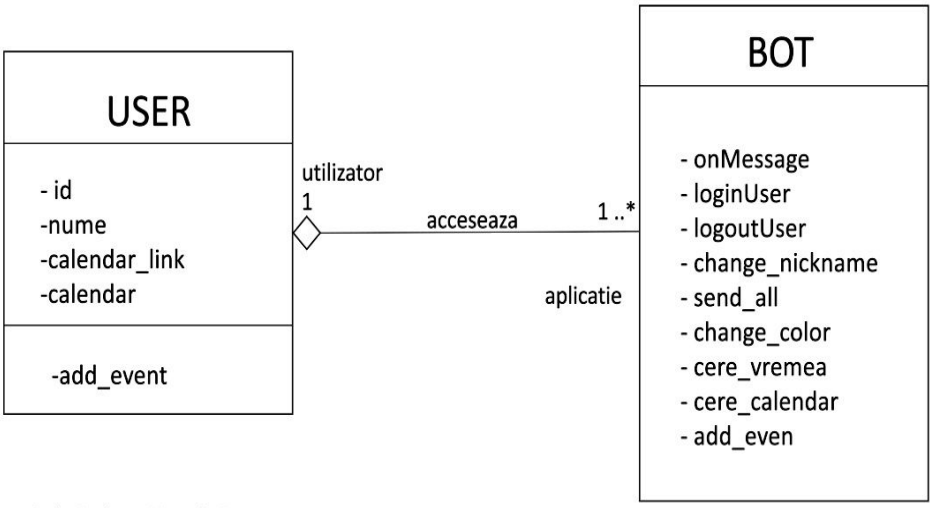
- Dacă se dorește deconectarea de la chat, User trimite comanda:

,,Botule logout”

prin care este șters din lista de Useri. La trimiterea unui nou mesaj, User-ului i se cer din nou date de conectare.

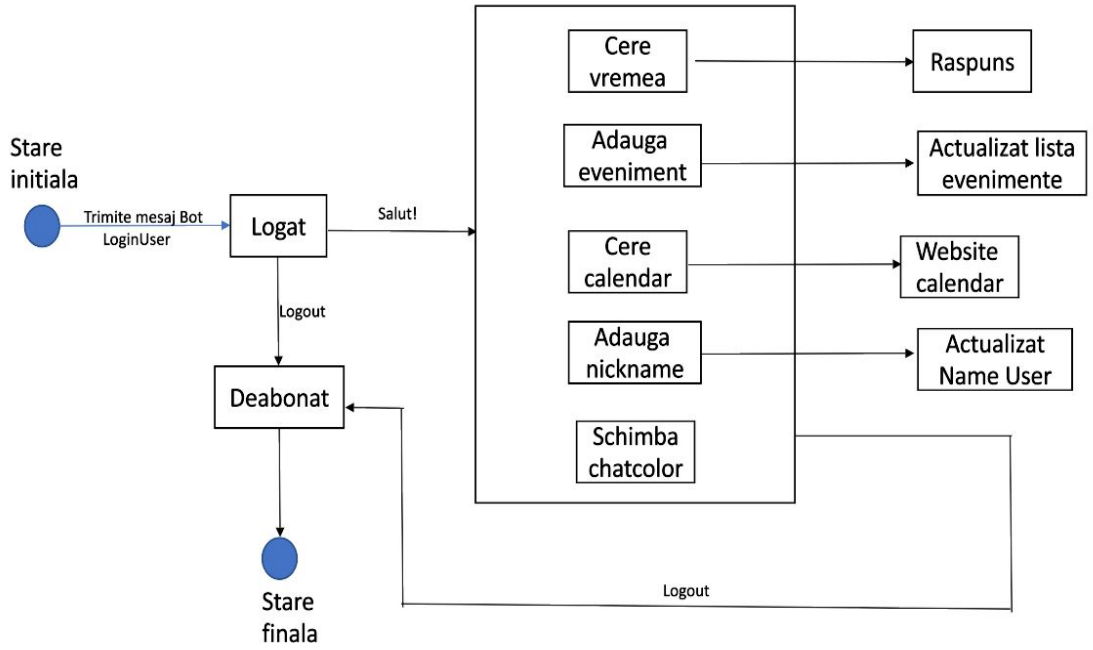
Diagrame

Diagrama clase



Cele 2 clase: User & Bot.
Intre ele exista o relatie de agregare (relatie parte-intreg).
Bot-ul reprezinta partea "mai mare", adica intregul, fiind format din mai multe parti mai mici, reprezentate de obiecte de tip User.

Diagrama stari



Clasa User

Clasa User defineşte utilizatorul. Această clasă conţine următoarele metode:

1. **Constructorul** clasei.

```
class User():


    def __init__(self, id="", name="", calendar_link="", calendar={}):
        '''
        initializam un obiect de tip user,
        folosind argumentele date
        Argumente:
                    id -> id-ul user-ului
                    nume -> numele user-ului
                    calendar_link -> link-ul catre pagina calendarului
                    calendar -> calendarul care initial nu contine niciun eveniment
        '''
        self.id = id
        self.name = name
        self.calendar_link = calendar_link
        self.calendar = calendar

```

- Este utilizat pentru a iniţializa un nou obiect de tip “User”.
- Constructorul foloseşte următorii parametrii:
    - id -> id-ul de pe Facebook Messenger (fiecărui user de pe Facebook i se asociază un id unic).
    - nume -> numele cu care s-a logat utilizatorul.
    - calendar_link -> link-ul către calendarul acestui user.
    - calendar -> dicţionarul care va conţine evenimentele userului (acest dicţionar este iniţial vid).

2. **Adăugare eveniment (metoda add_event).**

```
32      def add_event(self, event_name, date):
33          '''
34          Argumente: event_name -> numele evenimentului
35                     date -> data evenimentului
36
37          '''
38
39
40          if date in self.calendar:
41              # daca data aleasa exista, adaugam evenimentul
42              self.calendar[date].append(event_name)
43          else:
44              # daca data aleasa nu exista o adaugam impreuna cu evenimentul
45              self.calendar[date] = [event_name]
46
```

- Funcţia este apelată în momentul în care dorim să se adauge un eveniment nou la un anumit user.
- Parametrii sunt:
  - event_name -> numele evenimentului.
  - date -> data la care va avea loc acest eveniment.
- Funcţia adaugă evenimentul precizat în dicţionarul calendar definit anterior.

Clasa Bot

Clasa Bot se ocupă de acţiunile bot-ului. Acestuia i se adresează toate comenzile:

1. Setează nickname

```
71      def change_nickname(self, author_id, target_user, nickname):
72          target_user.set_name = nickname
73          print("Merge")
74          if author_id == target_user.id:
75              message = text = f"Ti-ai schimbat numele in {nickname}"
76          else:
77              message = text = f"{get_user(author_id).name} ti-a schimbat numele in {nickname}"
78
```

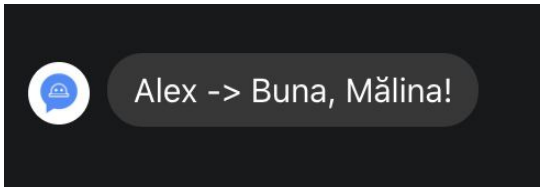
- Aceasta funcţie se ocupă de nickname-ul persoanelor din chat, mai precis când un utilizator trimite un mesaj se va schimba numele care apare înainte de ‘->’.

Parametrii:

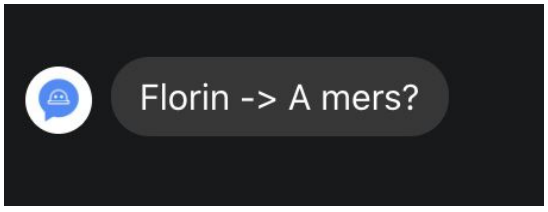
- author_id = numele utilizatorului care apelează funcţia
- target_user = numele utilizatorului pentru care se doreşte un nickname
- nickname = porecla dorită

Se setează numele user-ului cu noul user.
Se setează în funcţie de utilizator mesajul care trebuie să apară în conversaţie.

Înainte de apelarea funcţiei:



După apelarea funcţiei (Alex s-a schimbat în Florin):



2. Cere vremea

- Implementează o utilitate a botului
- Primeşte ca parametru oraşul pentru care utilizatorul se interesează de vreme
- Foloseşte api-ul descris mai sus pentru a afla temperatura, după care trimite mesajul pe chat

Parametrii:

- Message_object : obiect din modulul Fbchat, reprezentând mesajul unui user
- Message_object.text : text-ul propriu zis

Variabila “oras” va fi inițializată cu orașul pentru care se dorește a se afla vremea, fiind ‘cules’ din mesajul trimis.
Se realizează un request la API-ul oferit de “http://api.openweathermap.org”, având ca și parametru “oras”.
Răspunsul oferit este un JSON, iar prin intermediul funcției json.loads(response.text), se convertește la un dicționar.
Temp va fi data[‘main’][‘temp’].
Temp este temperatura în grade Kelvin, pe care o convertim la grade Celsius.
Mesajul corespunzător “Sunt x grade la Oraș” este trimis ulterior fiecărui user, prin apelul:

```
self.send(Message(text=f"Sunt {str(temp)} grade la {oras}"), thread_id=address, thread_type=ThreadType.USER)
```

```
115     def cere_vremea(self, message_object):
116         oras = message_object.text.strip('?').split('la ')[1]
117         response = requests.get(
118             "http://api.openweathermap.org/data/2.5/weather?q=" + oras + "&appid=d00e3b645e7b14cceb37659193a27d72"
119         data = json.loads(response.text)
120         temp = data['main']['temp']
121 
122         temp = round(temp - 273.15, 2)
123         for address in get_addresses():
124             self.send(Message(text=f"Sunt {str(temp)} grade la {oras}"), thread_id=address, thread_type=ThreadType.
125 
```

3. Send all

```
82      def send_all(self, message, author_id=""):
83          for user in users:
84              if user.id != author_id:
85                  self.send(Message(text=message), thread_id=user.id, thread_type=ThreadType.USER)
```

- Implementează trimiterea mesajelor de către bot
- Mesajul ce trebuie trimis este dat ca parametru, iar acesta este trimis tuturor, mai puțin emițătorului

4. Add event

```
134     def add_even(event_name, date, user):
135         user.add_event(event_name, date)
```

- Implementează adăugarea unui nou eveniment pentru un user
- Primește ca parametru numele eventului, data și user-ul
  Parametrii:
- event_name : numele evenimentului dat de utilizator
- date: string, reprezentând data
- user: un obiect din clasa User, reprezentând utilizatorul care apelează funcția

5. Autentificare user

```
137     def loginUser(self, author_id, message_object):
138         global users
139         global got_login
140         if not got_login:
141             message = "Salut! Nu stiu inca cum te cheama. Care este numele tau?"
142             self.send(Message(text=message), thread_id=author_id, thread_type=ThreadType.USER)
143             got_login = True
144 
145         elif got_login:
146 
147             got_login = False
148             name = message_object.text
149             user = User(id=author_id, name=name)
150             users.append(user)
151             message = name + ' a intrat pe server! Hai sa-l salutam'
152             self.send_all(message=message)
153 
154         return message
```

Parametrii:

- author_id : id-ul autorului care apelează metoda
- message_object: mesajul primit

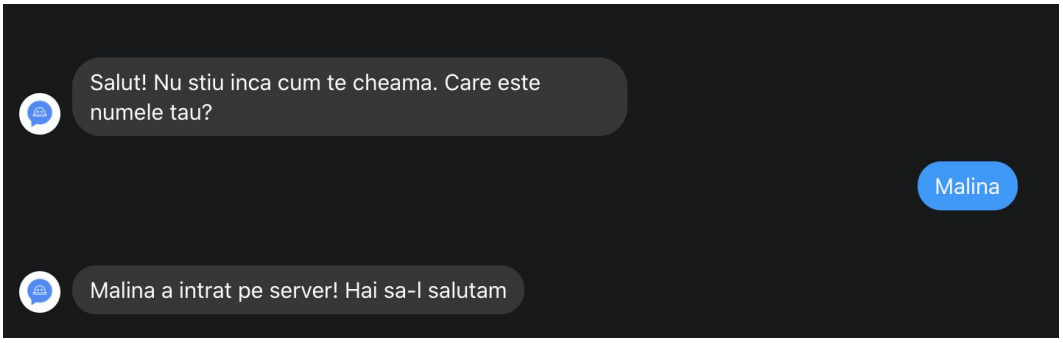
Funcția:

- global users: reține lista de utilizatori

- global got_login: variabila folosită pentru a reţine dacă un utilizator există sau nu în listă.
  La primul mesaj trimis, got_login=False şi se trimite un mesaj prin care se cere numele. Se apelează funcţia self.send(), care trimite doar user-ului care a trimis mesajul.
  Următorul mesaj al utilizatorului este cel în care îşi spune numele.
  Astfel, message_object.text reprezintă numele user-ului.
  Se creează astfel un nou user, care se adaugă în lista de users.
  Se trimite un mesaj tuturor utilizatorilor, apelându-se funcţia send_all(message).

```
self.send_all(message=message)
```

- Aceasta este perspectiva unui user:



6. Dezautentificare user

```
156     def logoutUser(self, author_id):
157         global users
158         user = get_user(author_id)
159         name = user.name
160         users.remove(user)
161         message = name + ' a parasit conversatia'
162         self.send_all(message=message, author_id=author_id)
163         return message
```

- Aşa se poate autentifica câte un user nou, fiecare se poate şi dezautentifica.

- Funcţia primeşte id-ul celui care doreşte să se dezautentifice
- Ca o confirmare, se trimite mesajul Nume + “a părăsit conversaţia.” tuturor participanţilor.
  Parametrii:
- author_id= id-ul celui care apelează funcţia

- user= get_user(author_id) -> returnează user-ul corespunzător id-ul respectiv
- users.remove(user) -> şterge din lista de users, user-ul dat ca parametru
- self.send_all(message, author_id) -> trimite mesajul tuturor utilizatorilor, cu excepţia celui care s-a dezabonat(author_id)

7. OnMessage

- Poate fi văzută ca o funcţie principala în cadrul clasei Bot
- Aceasta apelează efectiv toate utilităţile în funcţie de comanda data de user

```
281     def onMessage(self, author_id, message_object, thread_id, thread_type, **kwargs):
282
283         '''
284         Aceasta functie este apelata la trimiterea unui mesaj din partea unui utilizator
285
286         Fiecarui utilizator ii este asociat un id (author_id)
287         '''
288
289         global addresses
290         global users
291         self.markAsDelivered(thread_id, message_object.uid)
292         self.markAsRead(thread_id)
293
294         if author_id != self.uid:
295
296             # verificam daca user-ul se afla deja printre cei conectati
297
298             if author_id not in get_addresses():
299                 # nu se afla deci trebuie sa-si faca login
300
301                 self.loginUser(author_id, message_object)
302
303             else:
304
305                 # daca user-ul se afla deja in conversatie, mesajul lui va fi trimis catre toti ceilalti utilizatori
306                 self.send_all(message=get_user(author_id).name + " -> " + message_object.text, author_id=author_id)
307
308                 cere_vremea = False
309                 cere_calendar = False
310                 cere_logout = False
311                 change_nickname = False
312                 change_color = False
313
314
```

```
315                 # in functie de ce mesaj a dat user-ul, stabilim ce trebuie sa-i raspunda bot-ul
316                 if 'botule' in message_object.text.lower() and 'vremea' in message_object.text.lower():
317                     ''' botule ..vremea .. la ORAS'''
318                     # user-ul cere informatii despre vreme
319                     cere_vremea = True
320
321                 if 'botule' in message_object.text.lower() and 'calendar' in message_object.text.lower():
322                     '''botule ... calendar '''
323                     # user-ul cere informatii despre calendar
324                     cere_calendar = True
325
326                 if 'botule' in message_object.text.lower() and 'logout' in message_object.text.lower():
327                     '''botule .... logout'''
328                     # user-ul vrea sa iasa din conversatie
329                     cere_logout = True
330
331                 if 'botule' in message_object.text.lower() and 'eveniment' in message_object.text.lower():
332                     '''botule ... eveniment .. data=DATA_EVENIMENT ... nume=NUME_EVENIMENT'''
333                     # user-ul vrea sa adauge evenimente in calendar
334                     cere_adauga_eveniment = True
335
336                     # extragem data si evenimentul
337                     sir = message_object.text.lower()
338                     sir2 = message_object.text.lower()
339
340                     data = sir.split('data=')
341                     nume = sir.split('nume=')
342
343                     data = data[1].split(" ")
344                     data2 = data[0] + data[1]
345                     nume = nume[1]
346
347                     # adaugam evenimentul in calendar
348                     self.add_event(nume, data2, get_user(author_id))
349
```

```
            # botule schimba nickname bianca new_nick
            if 'botule' in message_object.text.lower() and 'nickname' in message_object.text.lower():
                '''botule .... nickname target_user NEW_NI'''
                change_nickname = True

                substring = message_object.text.split("nickname ")

                # extragem numele user-ului caruia vrem sa-i schimbam numele
                name = substring[1].split(" ")[0]

                # extragem porecla pe care vrem sa i-o dam
                nickname = substring[1].split(" ")[1]

                # gasim user-ul avand numele cautat
                target = get_user_by_name(name.lower())
                if target != -1:
                    self.change_nickname(author_id, target, nickname)
                else:
                    # daca nu exista, trimitem celui care a initiat aceasta actiune, ca nu exista user-ul
                    self.send(Message(text="Userul ales nu exista"), thread_id=author_id,
                              thread_type=ThreadType.USER)

            if 'botule' in message_object.text.lower() and 'new color' in message_object.text.lower():
                ''' botule ... new color rose
                                        blue
                                        orange
                                        red
                                        coral
                '''

                change_color = True

                # extragem textul corespunsator
                color = message_object.text.split("new color ")[1]
                self.change_color(color, author_id)


            if cere_vremea:
                self.cere_vremea(message_object)
            elif cere_calendar:
                self.cere_calendar(author_id)
            elif cere_logout:
                self.logoutUser(author_id)
```

8. Cere calendar

```
    def cere_calendar(self, author_id, thread_type):
        url = 'http://ec2-107-23-107-21.compute-1.amazonaws.com:443/Calendar?name=' + get_user(
            author_id).name + '&dict=' + str(get_user(author_id).calendar).replace('\'', "\"").replace('
        s = pyshorteners.Shortener()
        generated_link = s.tinyurl.short(url)
        self.send(Message(text=generated_link), thread_id=author_id, thread_type=thread_type)
        return generated_link
```

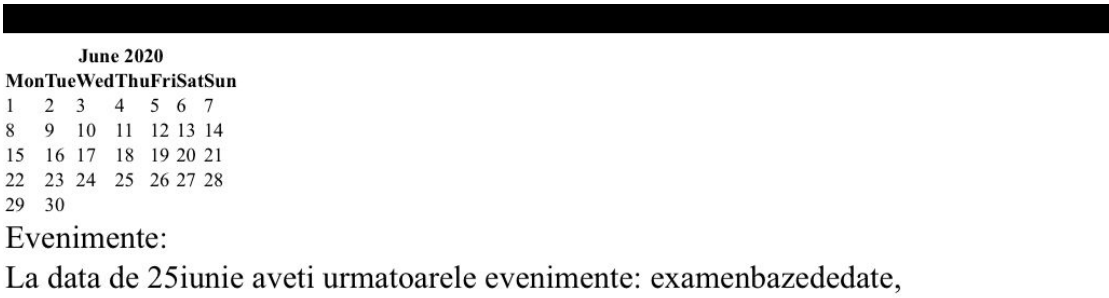
- Userii aplicaţiei pot cere un link pentru calendar în care îşi salvează evenimentele
  Parametrii:

- author_id ->id-ul celui care a apelat metoda
- Thread_type -> tipul thread-ului, corespunzător unui obiect FbChat. Aceasta poate fi: `ThreadType.USER sau ThreadType.GROUP`

- Se creează un url-ul corespunzător(formarea sa este explicată ulterior la partea de server)
- Se trimite un mesaj utilizatorului cu link-ul creat.

```
self.send(Message(text=generated_link),
thread_id=author_id,
thread_type=ThreadType.USER)
```

- Se returnează un link catre calendar



June 2020

Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Evenimente:

La data de 25iunie aveti urmatoarele evenimente: examenbazededate,

9. Schimba culoarea

```python
    def change_color(self, color, author_id):

        '''
        In cadrul acestei functii este schimbata culoarea conversatiei
        a userului cu id-ul author_id, astfel fiecare user isi poate alege culoarea dorita

        Argumente: author_id -> id-ul user-ului care doreste schimbarea
                             -> textul introdus de user care contine culoarea dorita

        '''

        # verificam culoarea pe care acesta o doreste
        if 'red' in color.lower():
            self.changeThreadColor(ThreadColor.RADICAL_RED, thread_id=author_id)
        elif 'blue' in color.lower():
            print("yessss")
            self.changeThreadColor(ThreadColor.MESSENGER_BLUE, thread_id=author_id)
        elif 'orange' in color.lower():
            self.changeThreadColor(ThreadColor.PUMPKIN, thread_id=author_id)
        elif 'coral' in color.lower():
            self.changeThreadColor(ThreadColor.LIGHT_CORAL, thread_id=author_id)
        elif 'rose' in color.lower():
            self.changeThreadColor(ThreadColor.BRILLIANT_ROSE, thread_id=author_id)
```

- User-ul poate opta pentru a cere schimba culoarea conversaţiei
  Parametrii:
- color: culoarea care se doreşte
- author_id : id-ul celui care a apelat metoda

Fbchat pune la dispoziţie metoda “changeThreadColor” care primeşte ca parametru culoarea şi id-ul utilizatorului care doreşte sa i se schimbe culoarea.

Alte funcţii ajutătoare

1. *get_addresses()* -> funcţie care returnează o listă cu toate id-urile utilizatorilor.

```python
def get_addresses():
    '''
        returneaza un array care contine
        toate id-urile utilizatorilor
    '''
    addresses = []
    for user in users:
        addresses.append(user.id)

    return addresses
```

2. *get_user(id)* -> funcţie care returnează obiectul de tip User care are id-ul specificat.

```python
def get_user(id):
    '''
        returneaza user-ul avand id-ul id, dat ca parametru

    '''
    for user in users:
        if user.id == id:
            return user
```

3. *get_user_by_name(name)* -> funcţie care returnează obiectul de tip User care are numele dat ca parametru.

```python
def get_user_by_name(name):
    '''
        returneaza user-ul avand numele name, dat ca parametru

    '''
    for user in users:
        if user.name.lower() == name:
            return user
    return -1
```

Server Calendar

```python
import calendar
from flask import Flask
from flask import request
from datetime import datetime
import json

app = Flask(__name__)

@app.route('/Calendar')
def calendar_user():
    name = request.args.get('name', None)
    dictionary = request.args.get('dict', None)
    dicti = json.loads(dictionary)
    currentMonth = datetime.now().month
    currentYear = datetime.now().year
    text_cal = calendar.HTMLCalendar(firstweekday = 0)
    year = currentYear
    month = currentMonth

    items = dicti.items()

    if len(items) > 0:
        param = "Evenimente: <br/>"
        for item in items:
            param += f" La data de {item[0]} aveti urmatoarele evenimente: "

            for even in item[1]:
                param += f"{even}, "
            param += "<br/>"
    else:
        param = "Nu aveti evenimente in aceasta luna<br/>"


    return text_cal.formatmonth(year, month) + param

if __name__ == '__main__':
    app.run(host = '0.0.0.0', port = 443)
```

Pentru realizarea serverului Calendar, am utilizat modulul Flask din Python.

Am reuşit să creăm un server care rulează pe maşina noastră virtuala de la Amazon AWS, pe portul 443. Url-ul acestui serviciu este de forma "http://ec2-107-23-107-21.compute-1.amazonaws.com:443/Calendar?name=nume&dict={}". Acest url primeşte doi parametrii:

- name = Numele persoanei care a cerut calendarul.
- dict = Un dicţionar din Python, în care sunt reţinute evenimentele din calendarul persoanei definită mai sus.

Acest serviciu funcţionează ca o interfaţă, oferindu-i utilizatorului o reprezentare mai intuitivă a calendarului acestuia.

Serviciul afişează calendarul lunii curente, urmat de evenimentele utilizatorului. Pentru a determina luna şi anul curent, se foloseşte modulul datetime. Folosim modulul json pentru a

putea interpreta dicţionarul transmis ca parametru, iar calendarul este generat de către modulul calendar, care genereaza o pagină html care conţine calendarul specificat.

Pentru a asigura rularea continuă a acestui server, am apelat la tmux, care permite rularea în paralel a mai multor terminale.

Testare

Testarea produsului software are o importanţă majoră în procesul de dezvoltare al acestuia. În viaţa de zi cu zi, testarea manuală ar întârzia lansarea pe piaţă a produsului, de aceea o componentă importantă a procesului de testare este testarea automată.

Testarea automată presupune scrierea de cod care să testeze un anumit produs software înaintea lansării acestuia, sau predarea acestuia clienţilor.

Acest tip de testare ne oferă posibilitatea de a descoperi erori fizice şi umane, pe care prin testearea manuală nu le-am putea găsi. De asemenea, testele automate pot fi rulate simultan pe mai multe maşini, timpul de testare fiind unul semnificativ mai mic.

Un dezavantaj inevitabil este că automatizarea nu poate fi făcută intru totul, unele teste necesitând testele manuale.

Scopul testarii este acela de a identifica erorile software şi de a le rezolva. Principiile generale ale testarii sunt testele, cazurile de test şi testarea efectivă.

Un **test** reprezintă alegerea unui set de date convenabil pentru a verifica dacă rezultatul obţinut în urmă executării programului este corect sau nu.

Seturile de date de intrare convenabil alese împreună cu seturile de date de ieşire la care ne aşteptăm să rezulte la execuţia programului formează **cazurile de test.**

Testarea înglobează aceste principii, astfel ca această reprezintă atât formarea cazurilor de test, executarea testelor cât şi compararea rezultatelor.

În cadrul proiectului am ales să folositm testarea unitară(Unit testing). Aceasta presupune existenţa unor teste specifice pentru o unitsate dintr-un program, validându-l sau nu. Când spunem unitate, ne referim la o funcţie individuală, o metoda dintr-o clasă sau la o clasă întreagă. Testele sunt uşor de citit, scurte şi rulează foarte repede.

Pentru a face posibilă testarea, am folosit framework-ul de testare unitară din python **unittest** şi am creat un fişier nou, **test_helper.py** în cadrul căruia am extras funcţiile care implementează logica aplicaţiei în afara funcţiilor care răspund la cereri pe server, pentru a le putea testa independent.

Într-un fişier separat **test.py**, în care exista codul de testare, am definit o clasă **Test** care moşteneşte modul unittest(unittest.TestCase). În această clasă au fost definite metodele pe care le-aş vrea ca testele unitare să le verifice în conformitate cu comportamentele definite în fişierul test_helper.py.

Clasa TestCase oferă mai multe metode în ideea de a verifica, raporta şi corecta eventualele erori. Noi ne-am oprit asupra metodelor **assertEqual(actual, expected)** şi **assertTrue(fact).**

Metoda **assertEqual(actual, expected)** verifică dacă actual este egal cu expected, în mod contrar testul va eşua.

Metoda **assertTrue(fact)** testează dacă fact este adevărăt. Dacă acesta este adevărat, testul se va încheia cu succes, altfel va eşua.

Pentru testare am ales câteva funcţii din clasa Bot, cum ar fi: add_event, change_nickname, cere_calendar, logout, login.

Iniţial am introdus în fişierul test_helper.py o listă de utilizatori formată dintr-un singur utilizator, pe baza căreia am testat functionalităţile

```
users = [User(123, 'cami')]
```

În fişierul test.py am importat fişierul test_helper.py pentru a avea acces la aceste funcţii extrase din clasa Bot şi libraria unittest.

```
import unittest
from test_helper import *
```

Apoi am luat fiecare funcţie în parte şi le-am testat.

Pentru **functia add_event**, corespondenţa acesteia din test_helper.py este:

```
def add_event(event_name, date, user):
    user.add_event(event_name, date)
```

Şi a fost testată în următorul fel:

La adăugarea evenimentului, utilizatorul ar trebui să aibă în calendar, la data de ‘26 mai’, examenul la ‘engleză.

```
def test_add_event(self):
    add_event('engleza', '26 mai', users[0])
    self.assertTrue(users[0].calendar['26 mai'][0] == 'engleza')
```

Apoi a urmat şi testarea următoarelor funcţii.

Functia de logout:

```
def logoutUser(author_id):
    global users
    user = get_user(author_id)
    name = user.name
    users.remove(user)
    message = name + ' a parasit conversatia'
    return message
```

Testarea acesteia:

Funcţia logout trebuie să returneze ‘cami a părăsit conversaţia’, mesaj ce este trimis tuturor utilizatorilor.

```
def test_logout(self):
    actual = logoutUser(users[0].id)
    expected = 'cami a parasit conversatia'
    self.assertEqual(actual, expected)
```

Functia de login:

```
def loginUser( author_id, message_object):
    global users
    global got_login

    got_login = True
    if not got_login:
        message = "Salut! Nu stiu inca cum te cheama. Care este numele tau?"
        got_login = True

    elif got_login:
        got_login = False
        name = message_object
        user = User(id=author_id, name=name)
        users.append(user)
        message = name + ' a intrat pe server! Hai sa-l salutam'

    return message
```

Testarea acesteia:

Am presupus că user-ul a dat deja mesajul cu numele lui, şi am testat această parte a funcţiei, adăugând un nou user cu numele ‘Andrei’.

```
def test_login(self):


    actual = loginUser(id, 'Andrei')
    expected = 'Andrei a intrat pe server! Hai sa-l salutam'
    self.assertEqual(actual, expected)
```

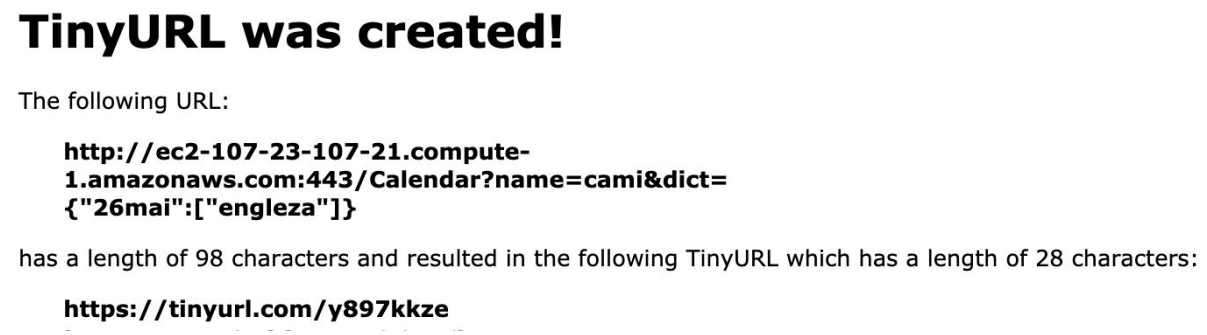
Functia cere_calendar

```
def cere_calendar(author_id):
    url = 'http://ec2-107-23-107-21.compute-1.amazonaws.com:443/Calendar?name=' + get_user(
        author_id).name + '&dict=' + str(get_user(author_id).calendar).replace('\'', "\"").replace(' ', '')
    s = pyshorteners.Shortener()
    generated_link = s.tinyurl.short(url)
    return generated_link
```

Testarea acesteia:

Verificăm că link-ul către calendar să fie cel corespunzător utilizatorului cami, având un singur eveniment, engleza, la data de ‘26 mai’.

Am găsit site-ul corespunzator folosind site-ul tinyurl, iar rezultatul așteptat este acesta:



TinyURL was created!

The following URL:

http://ec2-107-23-107-21.compute-1.amazonaws.com:443/Calendar?name=cami&dict={"26mai":["engleza"]}

has a length of 98 characters and resulted in the following TinyURL which has a length of 28 characters:

https://tinyurl.com/y897kkze

```
def test_cere_calendar(self):
    actual = cere_calendar(123)
    expected = 'http://tinyurl.com/y897kkze'
    self.assertEqual(actual, expected)
```

Functia change_nickname

```
def change_nickname(author_id, target_user, nickname):
    target_user.set_name = nickname

    if author_id == target_user.id:
        message = text = f"Ti-ai schimbat numele in {nickname}"
    else:
        message = text = f"{get_user(author_id).name} ti-a schimbat numele in {nickname}"

    return message
```

Testrea acesteia:

În cadrul acestei funcţii, testăm faptul că un user işi schimbă singur numele în ‘Ana’, iar acest lucru duce la trimiterea mesajului ‘Ţi-ai schimbat numele în Ana’

```
def test_change_nickname(self):
    actual = change_nickname(123, users[0], 'Ana')
    expected = 'Ti-ai schimbat numele in Ana'
    self.assertEqual(actual, expected)
```

Teste au avut succes la sfârsitul execuţiei programului, afişându-se aceasta:

```
.....
----------------------------------------------------------------------
Ran 5 tests in 0.500s

OK

Process finished with exit code 0
```

Pe parcurs am întâmpinat şi unele probleme, cum ar fi generarea incorectă a link-ului calendarului:

```
.F...
======================================================================
FAIL: test_cere_calendar (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/camelia/PycharmProjects/bot.py/venv/test.py", line 23, in test_cere_calendar
    self.assertEqual(actual, expected)
AssertionError: 'http://tinyurl.com/ybywsenj' != 'http://tinyurl.com/y897kkze'
- http://tinyurl.com/ybywsenj
?                     ^^^^ --
+ http://tinyurl.com/y897kkze
?                     ^^^^^^


----------------------------------------------------------------------
Ran 5 tests in 0.505s

FAILED (failures=1)
```

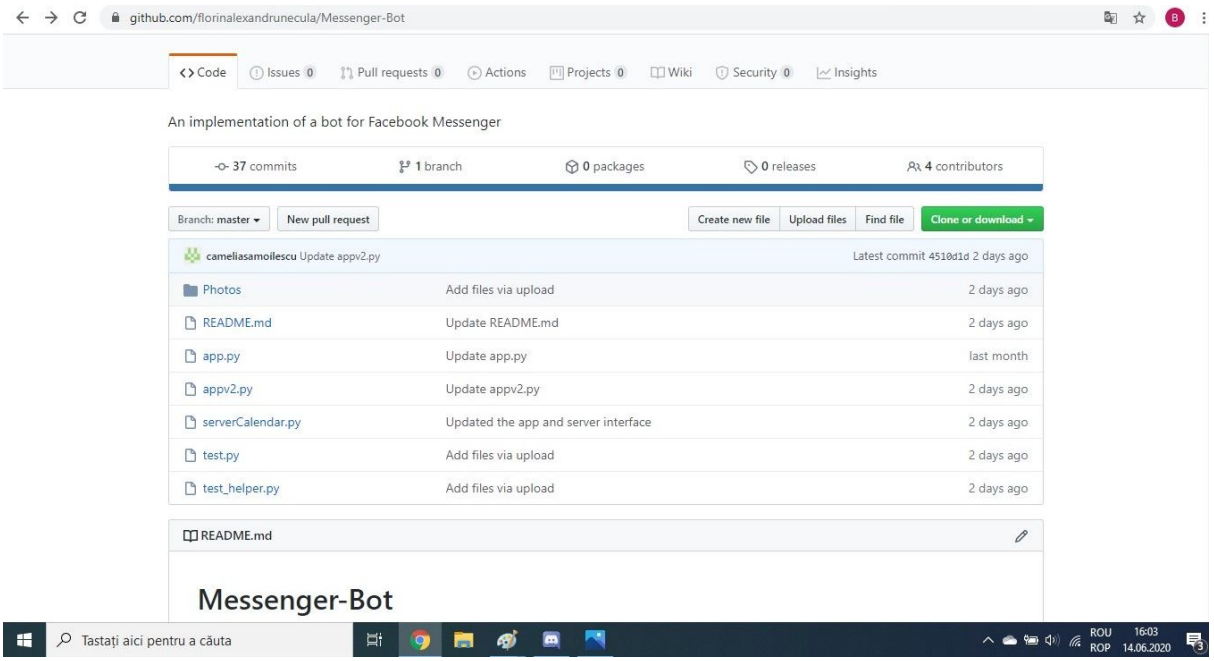
Commit-uri

Proiectul a fost pus pe Github, încă de la prima sa versiune. Acesta ne-a oferit posibilitatea de a avea acces în orice moment la codul sursă şi de a vedea modificările făcute de fiecare din echipă.

Fiecare membru a realizat noi commit-uri, ajungându-se astfel la versiunea sa finală. S-au păstrat de asemenea şi unele versiuni intermediare.

Pe Git au fost încărcate şi celelalte script-uri ajutătoare: de exemplu, cele de testare.

Link proiect: https://github.com/florinalexandrunecula/Messenger-Bot



Concluzii

Necula Florin-Alexandru: “Sunt foarte mândru de acest proiect. Este primul proiect de această magnitudine. În teorie, putem avea un group chat de peste 250 de persoane(aceasta este limita din Messenger). Trebuie doar să avem un server suficient de bun.”

Buzoi Bianca-Nicoleta: “Este ireal ce putem face cu nişte cunoştinţe de bază legate de programare. Nu am mai văzut aşa ceva până acum.”

Samoilescu Camelia: “Am învăţat atât de multe lucrând la acest proiect. Multe ore de debugging, multe ore de brainstorming, totul merită atunci când vezi că aplicaţia funcţionează cum ar trebui.”

Oanea Mălina-Ana: “La început părea dificil, însă am reuşit să învaţ foarte multe lucruri noi şi să contribui la crearea unei aplicaţii interesante.”

Link-uri către demo-ul aplicaţiei:

User1: https://www.youtube.com/watch?v=YEWAjZLx-0c
User2: https://youtu.be/LbHYLtsVhzs