МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему: «Інформаційна система онлайн допомоги студенту»

Завідувач
випускаючої кафедри **Довбиш А.С.**
Студент гр. ІН-62 **Матусевич М.М.**
Керівник роботи **Берест О.Б.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2020 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-62 спеціальності "Інформатика" денної форми навчання Матусевича Миколи Миколайовича.

Тема: «Інформаційна система онлайн допомоги студенту»

Затверджена наказом по СумДУ
№_____ від _____2020 р.

Зміст пояснювальної записки: 1) Інформаційний огляд; 2) Постановка задачі й формування завдань дослідження; 3) Вибір методу рішення задачі; 4) Практична реалізація; 5) Аналіз результатів.

Дата видачі завдання "_____" _____2020 р.
Керівник випускної роботи _____ Берест О.Б.
Завдання прийняв до виконання _____ Матусевич М.М.

РЕФЕРАТ

Записка: 40 стор., 5 рис., 4 табл., 4 додатки, 10 джерел.

Об'єкт дослідження — Веб-додаток у вигляді телеграм боту для отримання розкладу СумДУ.

Мета роботи — розробити веб-додаток у вигляді телеграм боту для отримання розкладу СумДУ. Додаток повинен бути зручним та зрозумілим у використанні.

Результати — проведено аналіз актуальності реалізації боту, обрані методи для виконання завдання. Розроблений алгоритм реалізовано у формі веб-додатку у вигляді телеграм боту, створеного за допомогою інструментального програмного середовища Python.

ВЕБ-ДОДАТОК, ТЕЛЕГРАМ БОТ, РОЗКЛАД, ІНФОРМАЦІЙНА ДОПОМОГА, ОНЛАЙН ДОПОМОГА СТУДЕНТУ, PYTHON.

Зміст

ВСТУП

Робота присвячена проблемі адаптації студентів в навчальному середовищі шляхом взаємодії початківця з телеграм ботом. А саме створення функції доступу до інформації для студентів СумДУ.

Ця проблема є досить актуальною в наш час, де значне спілкування відбувається засобами мережі Інтернет. Такий додаток буде являти собою не тільки вільний простір для діалогів, а й буде сприяти навчальному процесу. Студенти матимуть прямий доступ до актуального розкладу занятть.

Додаток буде мати закритий доступ.

Боти - акаунти в Telegram, створені для того, щоб автоматично обробляти і відправляти повідомлення. [2] Студенти можуть взаємодіяти з телеграм ботами за допомогою повідомлень, що відправляються через звичайні або групові чати. Логіка бота контролюється за допомогою HTTPS запитів до нашого API для пошукових роботів.

Переваги ботів Telegram над мобільними додатками, сайтами і групами в соціальних мережах:

1. Персональний підхід.

Персональний підхід користується неабиякою популярністю у користувачів. Безліч примітивних додатків та сайтів, що незрозумілі та іноді складні в інтуїтивному використанні втрачають свою актуальність. На противагу їм дедалі частіше з’являються персональні боти, що надають інформаційні послуги ефективніше. Крім того, персональне обслуговування відкриває нові можливості для просування послуги, додаткових бонусів і персональних акції, що може принести величезну віддачу.

2. Використання мінімум трафіку для текстового інтерфейсу.

Дуже актуальна перевага на сьогоднішній день. Оскільки, більшість користувачів надають перевагу мобільним пристроям при використанні інтернету.

3. Вартість розробки.

Розробка проєкту із створення телеграм боту не потребує капітальних фінансових витрат. Безперечно, це є величезною перевагою телеграм бота. Telegram створений під всі відомі операційні системи, тобто компіляція коду відбудеться на різних платформах. Також всі переваги Telegram будуть наявні в програмному додатку. Наприклад: кросплатформеність, надійність доставки повідомлень і швидкість, чати (групи і супергрупи), можливість ділитися файлами різного типу і розміру.

4. Гнучкість і швидкість відповіді.

Зміна коду в програмі не потребує багато часу для кінцевого результату, який стане помітним для студента. Миттєва реакція на будь-яку потребу з розсилкою всім відповідальним особам.

5. Без встановлення і авторизації.

Щоб скористатися ботом, студенту не потрібно завантажувати і встановлювати додаток на телефон або заповнювати форму реєстрації на сайті або в додатку. [2]

Підбиваючи підсумки переваг телеграм боту в сучасній буденності, слід також детальніше описати функціонал. Ось кілька прикладів використання ботів:

Інтеграція з іншими сервісами. Наприклад, бот може залишати коментарі або керувати «розумним будинком». Або, наприклад, відправляти вам повідомлення при здійсненні якоїсь дії (Приклади: GitHub Bot, Image Bot). [1]

Утиліти і інструменти. Наявний такий функціонал як показ погоди, перекладач тексту. Також є можливість попередження о подіях за персональним запитом. (Наприклад: бот опитувань).

Ігри для одного користувача або для групи студентів. Бот може пограти в шашки або шахи, проводити вікторини. (Приклад: Trivia bot).

Соціальні сервіси. Бот може знаходити співрозмовника, ґрунтуючись на ваших спільних інтересах і захопленнях. (Приклад: HotOrBot).

Таким чином, бот можливо створити програмно для будь-якої цілі.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Діяльність ботів в Telegram

Боти – особливі акаунти, що не потребують номера телефона при створенні. Ці акаунти грають роль інтерфейсу для сервісу, який працює на віддаленому сервері.

Варто звернути увагу на те, що для створення ботів не потрібно вивчати низькорівневі методи роботи з MTProto і шифруванням - спілкування з роботом організовано за допомогою звичайного HTTPS інтерфейсу з спрощеними методами Telegram API. Який отримав назву Bot API.

Telegram пропонує два види API для розробників. API Bot полегшує роботу при створенні програм, які використовують повідомлення Telegram для інтерфейсу. API Telegram і TDLib дозволяють створювати власні клієнтські клієнти Telegram. [5]

Також є змога додати віджети Telegram на веб-сайт.

Дизайнери можуть створити анімовані наклейки або власні теми для Telegram.

Таким чином, Telegram Bots - це спеціальні облікові записи, для встановлення яких не потрібен додатковий номер телефону. Користувачі можуть взаємодіяти з ботами двома способами:

Перший із способів – можливість надсилати повідомлення та команди ботам, відкриваючи з ними чат або додаючи їх до груп. Це корисно для чатів або ботів новин.

Другий спосбів - надсилати запити безпосередньо з поля введення, ввівши @username бота та запит. Це дозволяє надсилати вміст із вбудованих ботів безпосередньо у будь-який чат, групу чи канал.

Повідомлення, команди та запити, що надсилаються студентами, передаються до програмного забезпечення, що працює на серверах. Сервер-посередник обробляє все шифрування та зв’язок із API Telegram. Взаємодія із цим сервером відбувається через простий HTTPS-інтерфейс, який пропонує

спрощену версію API Telegram. Telegram називає цей інтерфейс нашим API Bot. [2]

1. Авторизація бота

Кожному боту при створенні дається унікальний маркер аутентифікації.

2. Оформлення запитів

Усі запити до API Telegram Bot повинні надсилатися через HTTPS і їх потрібно подавати у такій формі: https://api.telegram.org/bot <token> / METHOD_NAME. Наприклад:

https://api.telegram.org/bot123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11/getMe

Telegram підтримує методи GET та POST HTTP. Основні відмінності методів GET і POST передають данні веб-форм, які підтримують скрипт, а саме: Метод GET надсилає скрипт з усією зібраною інформацією у формі частини URL:

http://www.matusevich.com/script.php?login=admin&name=admint

Метод POST передає дані таким чином, що користувач сайту вже не бачить скриптів даних:

http://www.matusevich.com/script.php

Telegram підтримує чотири способи передачі параметрів у запитах API Bot:

Рядок запиту URL

додаток / x-www-form-urlencoded

application / json (крім завантаження файлів)

багаточастинні / форми даних (використовуються для завантаження файлів)

Відповідь містить об'єкт JSON, у якому завжди є булеве поле "добре" і може мати необов'язкове поле "Опис" рядка з описом результату, що читається людиною. Якщо "ok" дорівнює "true", запит був успішним, і результат запиту можна знайти в полі "result". У разі невдалого запиту "ОК" дорівнює помилковому, а помилка пояснюється в "описі". Поле "Цілий код коду

помилки" також повертається, але його вміст у майбутньому може бути змінено. Деякі помилки можуть також мати необов'язкове поле "параметри" типу ResponseParameters, що може допомогти автоматично впоратися з помилкою.

Усі методи в API Bot не залежать від регістру.

Усі запити повинні бути виконані за допомогою UTF-8.

1.2 Огляд подібних рішень

В наш час існує безліч ботів Telegram із різними функціями. Деякі з них несуть розважальний контент для користувачів, а деякі створюють науковий простір для вивчення матеріалу. Розглянемо саме ті боти, які є актуальними на сьогодення.

@topdf_bot – бот, що здійснює конвертацію файлів та документів у формат PDF. Підтримує такі типи документів, як: DOC, DOCX, ODT, TXT и JPG.

@driveitbot – бот, що зберігає файли на «Google Диск», що мають обсяг пам’яті до 20 МБ.

@freebooksbot, @mybookbot - боти, що здійснюють пошук книги по запиту користувача, пропонують обрати формат та зберігають вибраний файл в Dropbox.

@ytranslatebot – бот-перекладач, що працює, як в особистих чатах, так і в групових чатах Telegram. Підтримує майже всі мови світу.

@gmailbot – бот, який дозволяє користувачам Gmail не виходячи з Telegram перевіряти та відповідати на листи з електронної пошти.

@filesbot – бот, що перетворює Telegram в хмарний накопичувач. У ньому є можливість пересилати файли між користувачами месенджера, а також зберігати їх.

1.3 Постановка задачі

Слід реалізувати візуальну та серверну частину (Front-end та Back-end) веб-плагіну, із таким функціоналом.

Клієнт:

- Вибір своєї групи;
- Перегляд розкладу;
- Завантажити файл з розкладом;
- Зберегти файл із розкладом;
- Пересилання файлу із розкладом.

Адміністратор:

- Всі можливості клієнта;
- Доступ до коду та зміна функціоналу.

Сервер:

- Доступ до бази даних, можливість вносити зміни;
- Взаємодія з шаблонами веб-сайту.

2. ВИБІР МЕТОДУ РІШЕННЯ

Чатбот − комп'ютерна програма, розроблена на основі нейромереж та технологій машинного навчання, яка веде розмову за допомогою слухових або текстових методів. Чатбот використовують для досягнення якої-небудь мети (наприклад, надання потрібної інформації) або задля розваги. Чатботи часто використовуються в системах діалогу для різних практичних цілей, включаючи обслуговування клієнтів або отримання інформації. Деякі чатботи використовують складні системи обробки людської мови, але більшість використовує простіші системи.

За допомогою спеціального API сторонні розробники можуть створювати «ботів» в Telegram, спеціальні акаунти, керовані програмами. Типові боти відповідають на спеціальні команди в персональних і групових чатах, також вони можуть здійснювати пошук в інтернеті або виконувати інші завдання, застосовуються задля розваг або в бізнесі. [5]

Інтегроване середовище розробки — комплекс програмних рішень задля реалізації програмного забезпечення. Містить в собі редактор початкового коду, інструмент для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду.

Деякі середовища розробки містять компілятор, інтерпретатор або ж обидва, інші не містять жодного з них. Деякі інтегровані середовища розробки містять систему керування версіями або інструменти для полегшення розробки графічного інтерфейсу користувача (GUI) (XCode, Embarcadero Delphi). Багато сучасних IDE містять інспектор класів, інспектор об'єктів, схему ієрархії класів для полегшення об'єктно-орієнтованої розробки програмного забезпечення. [6]

Система управління базами даних (СУБД) є сукупністю програмних і лінгвістичних засобів загального або спеціального призначення, які забезпечують управління створенням і використанням баз даних.

Сучасна СУБД складається з наступних компонентів:

- ядра, що відповідає за управління даними у зовнішній і оперативної пам'яті і журналізацію;
- процесора мови БД, що забезпечує оптимізацію запитів по вилученню та зміни даних і створення машинно-незалежного виконуваного внутрішнього коду;
- підсистеми підтримки часу виконання, інтерпретує програми маніпуляції даними призначеного для користувача інтерфейсу СУБД;
- сервісних програм, призначених для забезпечення ряду додаткових можливостей обслуговування інформаційної системи.

2.1 Вибір програмної реалізації

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. [4] Вона має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. [1]

Зрозумілий синтаксис Python, динамічна обробка типів, а також те, що це інтерпретована мова, роблять її ідеальною для написання скриптів та швидкої розробки прикладних програм у багатьох галузях на більшості платформ.[3]

Інтерпретатор Python легко розширюється новими функціями та типами даних, реалізованими на C або C ++ (або інших мовах, які можна викликати з C). Python також підходить як мова розширення для налаштованих програм.[3]

Приклад коду на мові програмування Python. Генератор чисел Фібоначчі:

```
def fib():
a, b = 1, 1
while True:
yield a
```

```
a, b = b, a + b
while x in fib():
print(x)
```

2.2. Вибір інтегрованого середовища розробки (IDE)

PyCharm — інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний зневаджувач, інструмент для запуску юніт-тестів і підтримує веб-розробку на Django. [8]

Працює під такими операційними системами, як: Windows, Mac OS X та Linux.

Редактор коду PyCharm надає підтримку для таких технологій, як Python, JavaScript, CoffeeScript, TypeScript, CSS.

PyCharm надає наступні можливості для користувачів:

- Виконання статичного аналізу коду.
- Наявна зручна навігація інтерфейсу.
- Наявний рефакторинг.
- Запозичення функціоналу для веб-розробки з фреймворку Django.
- Наявний вбудований зневаджувач для Python.
- Є можливість виконання юніт-тестування.
- Використання Google App Engine.[8]

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розробка інформаційної моделі

Перед початком програмного розроблення боту необхідно створити інформаційну модель, визначивши та описавши усі необхідні складові та компоненти.

Інформаційна модель – це модель, що містить інформацію про властивості і стан об’єктів, процесів, явищ. [7]

В рамках даної дипломної роботи розробляється інформаційна модель боту, яка складається з наступних модулів: головний модуль, модуль взаємодії з базою даних навчальних занять, модуль допоміжних інструментів, скрипт для оновлення бази даних навчального розкладу, модуль з необхідною конфігурацією проекту. Також з додатком пов'язані: файл бази даних навчальних занять.

Говорячи про базу даних навчальних занять варто відзначити, що використовується всього одна таблиця, що має наступну структуру:

- ідентифікатор групи;
- день тижня;
- номер заняття;
- номер аудиторії;
- назва заняття;
- номер корпусу;
- тип заняття;
- ПІБ викладача.

Що стосується модуля взаємодії з базою даних навчальних занять, то він містить ряд функцій, що забезпечують доступ до розкладу конкретної групи на певний день тижня і дозволяють оновлювати список занять певної групи.

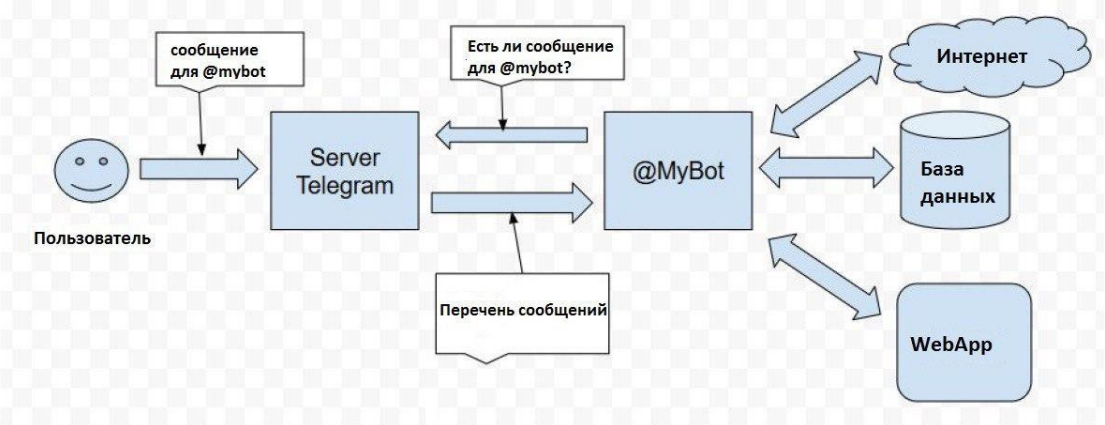

Рис. 3.1 – Діаграма користувачів

На рисунку 3.1 зображена схема реалізації, згідно якої, клієнт має деякі свої права, а сервер працює с базою даних та оброблює запити згідно дій користувача.

3.2. Програмна реалізація

Завантажуємо Python. Зробити це можна на офіційному сайті. Після установки Python'a встановлюємо програмний продукт PyCharm.

Bot Father

У пошуку telegram знаходимо Bot Farher'a і створюємо свого бота за допомогою команди /newbot. Потім вводимо ім'я і юзернейм. Зверніть увагу, що юзернейм повинен закінчуватися на bot!

Код

Відкриваємо PyCharm і створюємо новий проєкт. Після створення проєкту, створимо файл, в якому буде код.

Python пропонує досить сильну бібліотеку логування в стандартній бібліотеці. Багато програмістів використовують оператор print для лікування багів. Використання логу є більш чистим методом, якщо ви не хочете переглядати весь свій код, щоб видалити всі оператори print.

Лог , який я використовую в проєкті:

```
# -*- coding: utf-8 -*-
from __future__ import print_function

import threading
import time
import re
import sys
import six

import os
import pickle

import logging

logger = logging.getLogger('TeleBot')
formatter = logging.Formatter(
    '%(asctime)s (%(filename)s:%(lineno)d %(threadName)s) %(levelname)s - %(name)s: "%(message)s"'
)

console_output_handler = logging.StreamHandler(sys.stderr)
console_output_handler.setFormatter(formatter)
logger.addHandler(console_output_handler)

logger.setLevel(logging.ERROR)
```

Клас Handler для перемикання логів.

```
class Handler:
    """
    Class for (next step|reply) handlers
    """
    def __init__(self, callback, *args, **kwargs):
        self.callback = callback
        self.args = args
        self.kwargs = kwargs

    def __getitem__(self, item):
        return getattr(self, item)
```

Клас Saver для зберігання логів.

```
class Saver:
    """
    Class for saving (next step|reply) handlers
    """
    def __init__(self, handlers, filename, delay):
        self.handlers = handlers
        self.filename = filename
        self.delay = delay
        self.timer = threading.Timer(delay, self.save_handlers)

    def start_save_timer(self):
        if not self.timer.is_alive():
            if self.delay <= 0:
                self.save_handlers()
            else:
                self.timer = threading.Timer(self.delay, self.save_handlers)
                self.timer.start()

    def save_handlers(self):
        self.dump_handlers(self.handlers, self.filename)

    def load_handlers(self, filename, del_file_after_loading=True):
        tmp = self.return_load_handlers(filename, del_file_after_loading=del_file_after_loading)
        if tmp is not None:
            self.handlers.update(tmp)

    @staticmethod
    def dump_handlers(handlers, filename, file_mode="wb"):
        dirs = filename.rsplit('/', maxsplit=1)[0]
        os.makedirs(dirs, exist_ok=True)

        with open(filename + ".tmp", file_mode) as file:
            pickle.dump(handlers, file)

        if os.path.isfile(filename):
```

```
            os.remove(filename)

        os.rename(filename + ".tmp", filename)

    @staticmethod
    def return_load_handlers(filename, del_file_after_loading=True):
        if os.path.isfile(filename) and os.path.getsize(filename) > 0:
            with open(filename, "rb") as file:
                handlers = pickle.load(file)

            if del_file_after_loading:
                os.remove(filename)

            return handlers
```

Тепер потрібно створити змінну bot.

```
bot = telebot.TeleBot('ваш токен')
```

Змінна створюється за допомогою конструктора класу TeleBot, параметром якого виступає ваш токен, результатом якого буде об'єкт класу TeleBot

```
def __init__(self, token, threaded=True, skip_pending=False, num_threads=2):
    """
    :param token: bot API token
    :return: Telebot object.
    """

    self.token = token
    self.update_listener = []
    self.skip_pending = skip_pending

    self.__stop_polling = threading.Event()
    self.last_update_id = 0
    self.exc_info = None
```

```
    # key: message_id, value: handler list
    self.reply_handlers = {}

    # key: chat_id, value: handler list
    self.next_step_handlers = {}

    self.next_step_saver = None
    self.reply_saver = None

    self.message_handlers = []
    self.edited_message_handlers = []
    self.channel_post_handlers = []
    self.edited_channel_post_handlers = []
    self.inline_handlers = []
    self.chosen_inline_handlers = []
    self.callback_query_handlers = []
    self.shipping_query_handlers = []
    self.pre_checkout_query_handlers = []

    self.threaded = threaded
    if self.threaded:
        self.worker_pool = util.ThreadPool(num_threads=num_threads)
```

Напишемо декоратор bot.message_handler (), за допомогою якого бот буде реагувати на команду /start. Для цього в круглих дужках пишемо commands = ['start']. В результаті:

```
@bot.message_handler(commands=['start'])
```

Опишемо функцію message_handler. Декоратор обробки повідомлень.

Цей декоратор можна використовувати для прикраси функцій, які повинні обробляти певні типи повідомлень.

Всі обробники повідомлень тестуються у порядку, до якого вони були додані.

Приклад:

```
    bot = TeleBot ('TOKEN')
# Обробляє всі повідомлення, текст яких відповідає regexp.
```

```
@ bot.message_handler (regexp = 'someregexp')
def command_help (повідомлення):
    bot.send_message (message.chat.id, "Хтось телефонував на допомогу?")
```

```
# Обробляти всі надіслані документи типу "текст / звичайний".
@ bot.message_handler (func = lambda message: message.document.mime_type == 'текст / звичайна', content_types = ['документ'])
def command_handle_document (повідомлення):
    bot.send_message (message.chat.id, "Документ отримано, сер!")
```

```
def message_handler(self, commands=None, regexp=None, func=None, content_types=['text'], **kwargs):
    """
        :param regexp: Optional regular expression.
    :param func: Optional lambda function. The lambda receives the message to test as the first parameter. It must return True if the command should handle the message.
    :param content_types: This commands' supported content types. Must be a list. Defaults to ['text'].
    """

    def decorator(handler):
        handler_dict = self._build_handler_dict(handler,
                                                commands=commands,
                                                regexp=regexp,
                                                func=func,
                                                content_types=content_types,
                                                **kwargs)

        self.add_message_handler(handler_dict)

        return handler

    return decorator
```

Після декоратора створюємо функцію start_message, яка буде приймати параметр message (назва функції може бути будь-яка). Далі реалізуємо

відправку повідомлення від самого бота. У функції пропишемо bot.send_message (message.chat.id, 'Привіт, ти написав мені / start'). Результат:

```
bot = telebot.TeleBot('ваш токен')

@bot.message_handler(commands=['start'])
def start_message(message):
    bot.send_message(message.chat.id, 'Привіт, ти написав /start')

bot.polling()
```

Функція polling() створює новий потік, який викликає внутрішню функцію __retrieve_updates. Це дозволяє боту автоматично отримувати оновлення та відповідно повідомляти слухачів та обробників повідомлень.

```
def polling(self, none_stop=False, interval=0, timeout=20):
    """
        Always get updates.
    :param interval:
    :param none_stop: Do not stop polling when an ApiException occurs.
    :param timeout: Timeout in seconds for long polling.
    :return:
    """
    if self.threaded:
        self.__threaded_polling(none_stop, interval, timeout)
    else:
        self.__non_threaded_polling(none_stop, interval, timeout)
```

Функці requests є стандартом для внесення HTTP-запитів у Python. Він резюмує складність подання запитів API.

Методи HTTP, такі як GET і POST, визначають, яку дію ви намагаєтеся виконати, роблячи HTTP-запит. Окрім GET та POST, існує кілька інших поширених методів.

Один з найпоширеніших методів HTTP - це GET. Метод GET вказує на те, що ви намагаєтеся отримати або отримати дані з визначеного ресурсу. Для того, щоб зробити запит GET, виклик request.get ().

```
def get(url, params=None, **kwargs):

    kwargs.setdefault('allow_redirects', True)
    return request('get', url, params=params, **kwargs)
```

Response є потужним об’єктом для перевірки результатів запиту.

```
response = requests.get('https://google.com')
print(response.status_code)
```

Перша інформації, якиу можна отримати з відповіді - це код статусу. Код статусу інформує про статус запиту. Наприклад, статус 200 ОК означає, що запит був успішним, тоді як статус 404 НЕ ЗНАЙДЕНО означає, що шуканого ресурсу не знайдено. Існує безліч інших можливих кодів статусу, щоб дати конкретну інформацію.

Відповідь на запит GET часто містить якусь цінну інформацію, відому як корисне навантаження, в тілі повідомлення. Використовуючи атрибути та методи Response, можна переглядати корисне навантаження в самих різних форматах.

Щоб побачити вміст відповіді в байтах, використовується .content:

```
response = requests.get('https://google.com')
response.content
```

```
@property
def content(self):
    """Content of the response, in bytes."""

    if self._content is False:
        # Read the contents.
        if self._content_consumed:
            raise RuntimeError(
                'The content for this response was already consumed')

        if self.status_code == 0 or self.raw is None:
            self._content = None
        else:
            self._content = b''.join(self.iter_content(CONTENT_CHUNK_SIZE)) or b''
```

```
    self._content_consumed = True
    # don't need to release the connection; that's been handled by urllib3
    # since we exhausted the data.
    return self._content
```

.content надає доступ до необмежених байтів корисного навантаження відповіді, для того, щоб перетворити їх у рядок, використовують кодування символів, наприклад UTF-8. Це необхідно зробити, коли буде отримано доступ до .text:

```
@property
def text(self):

    # Try charset from content-type
    content = None
    encoding = self.encoding

    if not self.content:
        return str('')

    # Fallback to auto-detected encoding.
    if self.encoding is None:
        encoding = self.apparent_encoding

    # Decode unicode from given encoding.
    try:
        content = str(self.content, encoding, errors='replace')
    except (LookupError, TypeError):
        # A LookupError is raised if the encoding was not found which could
        # indicate a misspelling or similar mistake.
        #
        # A TypeError can be raised if encoding is None
        #
        # So we try blindly encoding.
        content = str(self.content, errors='replace')

    return content
```

Якщо подивитись на відповідь, то виявиться, що це насправді серіалізований вміст JSON. Щоб отримати словник, можна взяти рядок, який отримано з .text, і деріаріалізувати його за допомогою json.loads (). Однак більш простим способом виконання цього завдання є використання .json ():

```
def json(self, **kwargs):


    if not self.encoding and self.content and len(self.content) > 3:
        # No encoding set. JSON RFC 4627 section 3 states we should expect
```

```
    # UTF-8, -16 or -32. Detect which one to use; If the detection or
    # decoding fails, fall back to `self.text` (using chardet to make
    # a best guess).
    encoding = guess_json_utf(self.content)
    if encoding is not None:
        try:
            return complexjson.loads(
                self.content.decode(encoding), **kwargs
            )
        except UnicodeDecodeError:
            # Wrong UTF codec detected; usually because it's not UTF-8
            # but some other 8-bit codec.  This is an RFC violation,
            # and the server didn't bother to tell us what codec *was*
            # used.
            pass
    return complexjson.loads(self.text, **kwargs)
```

Тип повернутого значення .json () - це словник, тому можна отримати доступ до значень в об'єкті за ключем.

З кодами статусу та органами повідомлень можна багато зробити. Але якщо потрібна додаткова інформація, як-от метадані про саму відповідь, потрібно буде переглянути заголовки відповіді.

Заголовки відповідей можуть надати корисну інформацію, таку як тип вмісту корисної навантаження відповіді та обмеження часу на кешування відповіді. Щоб переглянути ці заголовки, необхідно .headers:

```
#: Case-insensitive Dictionary of Response Headers.
#: For example, ``headers['content-encoding']`` will return the
#: value of a ``'Content-Encoding'`` response header.
self.headers = CaseInsensitiveDict()
```

.headers повертає об’єкт, подібний до словника, дозволяє отримувати доступ до значень заголовка за ключем. Наприклад, щоб побачити тип вмісту корисного навантаження відповіді, можна отримати доступ до типу Content-Type.

Один поширений спосіб налаштування GET-запиту - це передавання значень через параметри рядка запиту в URL. Для цього за допомогою get () йде передача дані в парами. Наприклад, можна використовувати API пошуку GitHub для пошуку бібліотеки запитів:

```
# Search GitHub's repositories for requests
response = requests.get(
```

```
    'https://api.github.com/search/repositories',
    params={'q': 'requests+language:python'},
)

# Inspect some attributes of the `requests` repository
json_response = response.json()
repository = json_response['items'][0]
print(f'Repository name: {repository["name"]}')  # Python 3.6+
print(f'Repository description: {repository["description"]}')  # Python 3.6+
```

Передавши словник {'q': 'request + language: python'} до параметра params .get (), можна змінювати результати, що повертаються з API пошуку.

Відповідно до специфікації HTTP, POST, PUT та менш поширені запити PATCH передають свої дані через тіло повідомлення, а не через параметри в рядку запиту. Використовуючи запити, можна передати корисне навантаження параметру даних відповідної функції.

Дані приймають словник, список кортежів, байтів або файлоподібний об'єкт.

Наприклад, якщо типом вмісту запиту є application / x-www-form-urlencoded, можна надіслати дані форми у словнику:

```
requests.post('https://httpbin.org/post', data={'key':'value'})
<Response [200]>
```

Якщо необхідно надіслати дані JSON, можна скористатися параметром json. Коли йде передача данних JSON через json, запити будуть серіалізувати дані та додавати правильний заголовок Content-Type для вас.

Автентифікація допомагає службі зрозуміти, хто ви. Як правило, ви надаєте свої дані на сервер, передаючи дані через заголовок авторизації або спеціальний заголовок, визначений службою. Усі функції запиту, які ви бачили до цього моменту, містять параметр під назвою auth, який дозволяє передавати свої облікові дані.

Один із прикладів API, який вимагає автентифікації, - це Аутентифікований API користувача GitHub. Ця кінцева точка надає інформацію про автентифікований профіль користувача. Щоб зробити запит на

Аутентифікований користувальницький API, ви можете передати своє ім'я користувача та пароль GitHub в кортеж, get ():

```
from getpass import getpass
requests.get('https://api.github.com/user', auth=('username', getpass()))
```

До цього часу робота відбувалася з API-запитами високого рівня, такими як get () і post (). Ці функції - це абстракції того, що відбувається під час здійснення запитів. Вони приховують деталі реалізації, наприклад, як керуються з'єднаннями.

Під цими абстракціями знаходиться клас, який називається Сесія. Якщо вам потрібно точно налаштувати контроль над тим, як надсилаються запити або покращити ефективність запитів, можливо, буде потрібно безпосередньо використовувати екземпляр сесії.

Сеанси використовуються для збереження параметрів для запитів. Наприклад, якщо необхідно використовувати одну і ту ж аутентифікацію для кількох запитів, можна використовувати сеанс:

```
import requests
from getpass import getpass

# By using a context manager, you can ensure the resources used by
# the session will be released after use
with requests.Session() as session:
    session.auth = ('username', getpass())

    # Instead of requests.get(), you'll use session.get()
    response = session.get('https://api.github.com/user')

# You can inspect the response just like you did before
print(response.headers)
print(response.json())
```

Кожен раз, коли відбувається запит із сеансом, після його ініціалізації з обліковими записами автентифікації, облікові дані зберігатимуться.

Первинна оптимізація ефективності сеансів відбувається у вигляді стійких з'єднань. Коли додаток встановлює з'єднання з сервером за допомогою сесії, він зберігає це з'єднання навколо у пулі з'єднань. Коли програма хоче знову

підключитися до того ж сервера, вона повторно використовуватиме з'єднання з пулу, а не встановлює новий.

Тепер маємо такий функціонал:

1) Здійснювати запити, використовуючи різні методи HTTP, такі як GET, POST та PUT.
2) Налаштовувати свої запити, змінюючи заголовки, автентифікацію, рядки запитів та органи повідомлень.
3) Перевіряти дані, які ви надсилаєте на сервер, і дані, які сервер надсилає вам.

Оскільки є можливість використовувати запити, ми можемо використовувати дані з сайту розкладу СумДУ для бота, що розробляється.

```
import requests

from Groups import groupValue


def connect2Schedule(x):
    url = 'http://schedule.sumdu.edu.ua/index/pdf?id_grp=' + groupValue[x]
    myfile = requests.get(url, allow_redirects=True)
    open('./pdf/schedule.pdf', 'wb').write(myfile.content)
```

Підключившись до сайту і вказавши параметри для відображення розкладу, отримуємо файл в форматі .pdf. Користувачу потрібно буде цей файл зберігати, щоб побачити вміст файлу с розкладом, що не є зручним у користуванні. Цю проблему можна вирішити шляхом конвертації .pdf файлу у формат .png. Нижче наведено код конвертації.

```
import os
import tempfile
from pdf2image import convert_from_path


def convert2Img():
    filename = './pdf/schedule.pdf'
    with tempfile.TemporaryDirectory() as path:
        images_from_path = convert_from_path(filename, output_folder=path, last_page=1, first_page=0)

    base_filename = os.path.splitext(os.path.basename(filename))[0] + '.png'
    save_dir = './img/'
```

```
    for page in images_from_path:
        page.save(os.path.join(save_dir, base_filename), 'PNG')
```

Після цієї операції маємо можливість відправити зображення користувачу для швидкого і зручного перегляду розкладу.

```
@bot.message_handler(content_types=['text'])
def send_text(message):
    check = 0
    for x in groupName:
        if message.text.lower() == x:
            connect2Schedule(groupName.index(x))
            convert2Img()
            photo = open('./img/schedule.png', 'rb')
            bot.send_photo(message.chat.id, photo)
            check = 1
    if check == 0:
        bot.send_message(message.chat.id, 'не найдено')
```

На рисунках 3.2 -3.5 представлених нижче продемонстрована детальна робота телеграм боту.

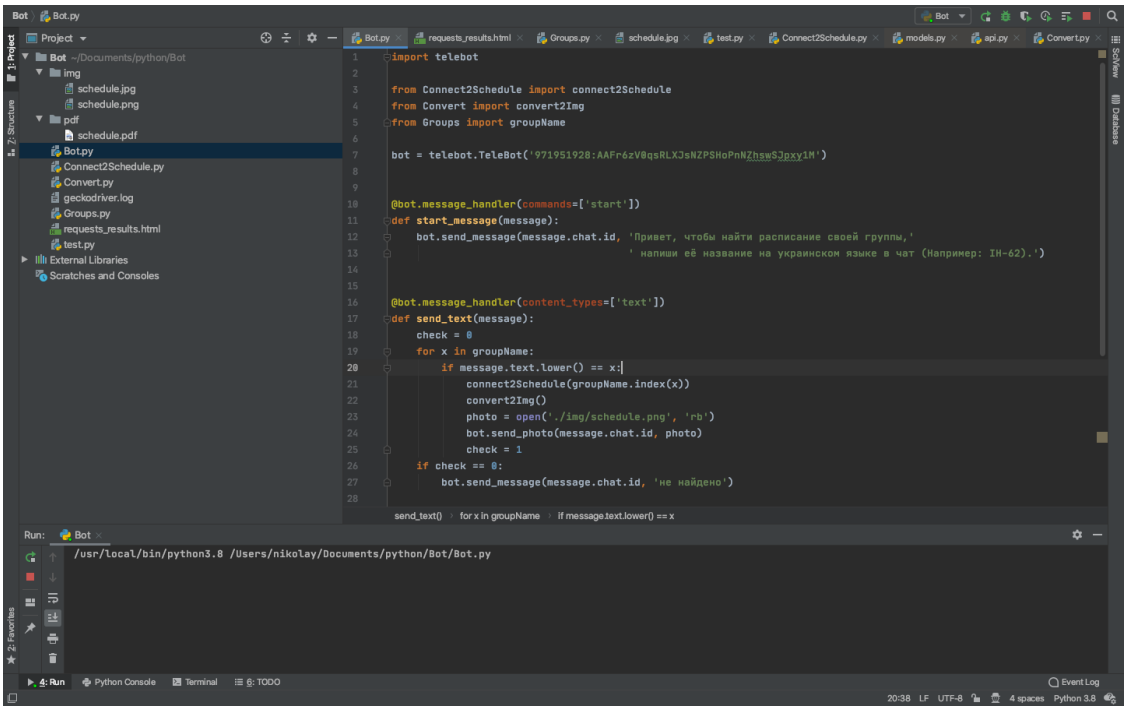

Рисунок 3.2. – Запуск в IDE PyCharm.

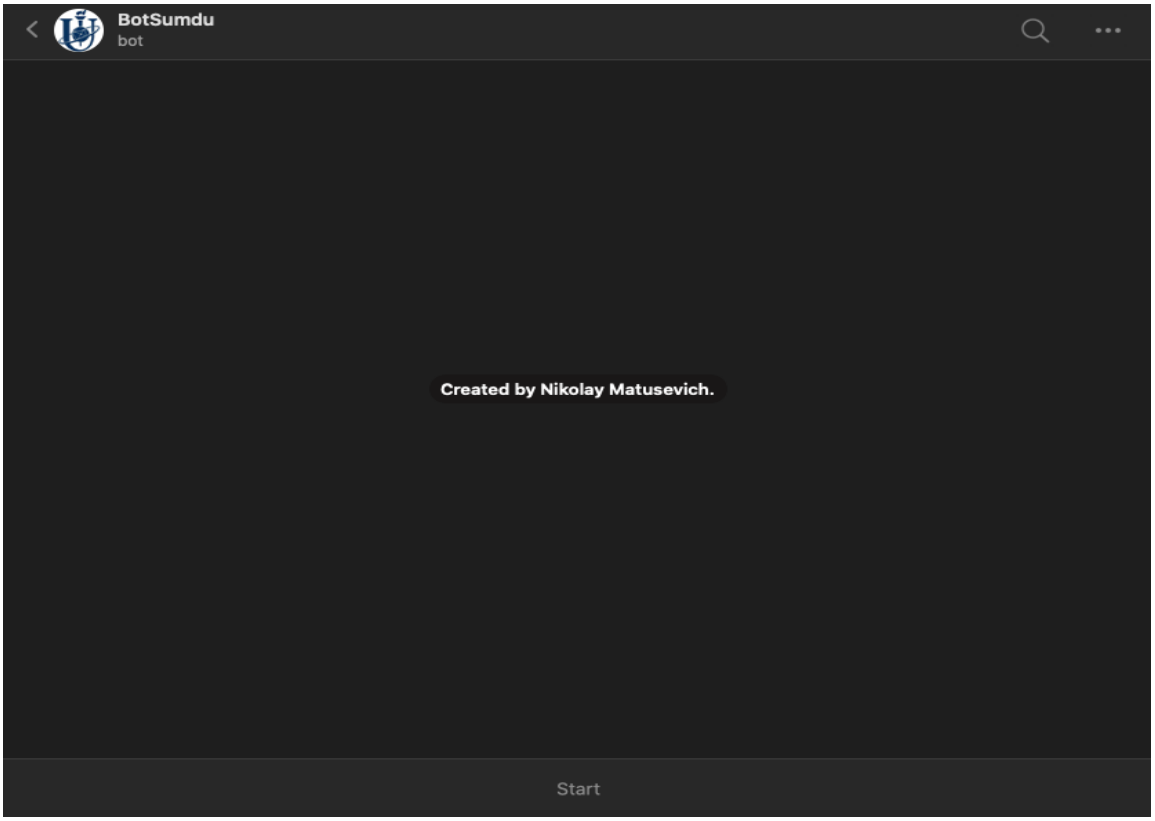

Рисунок 3.3 – Запуск бота в телеграмі

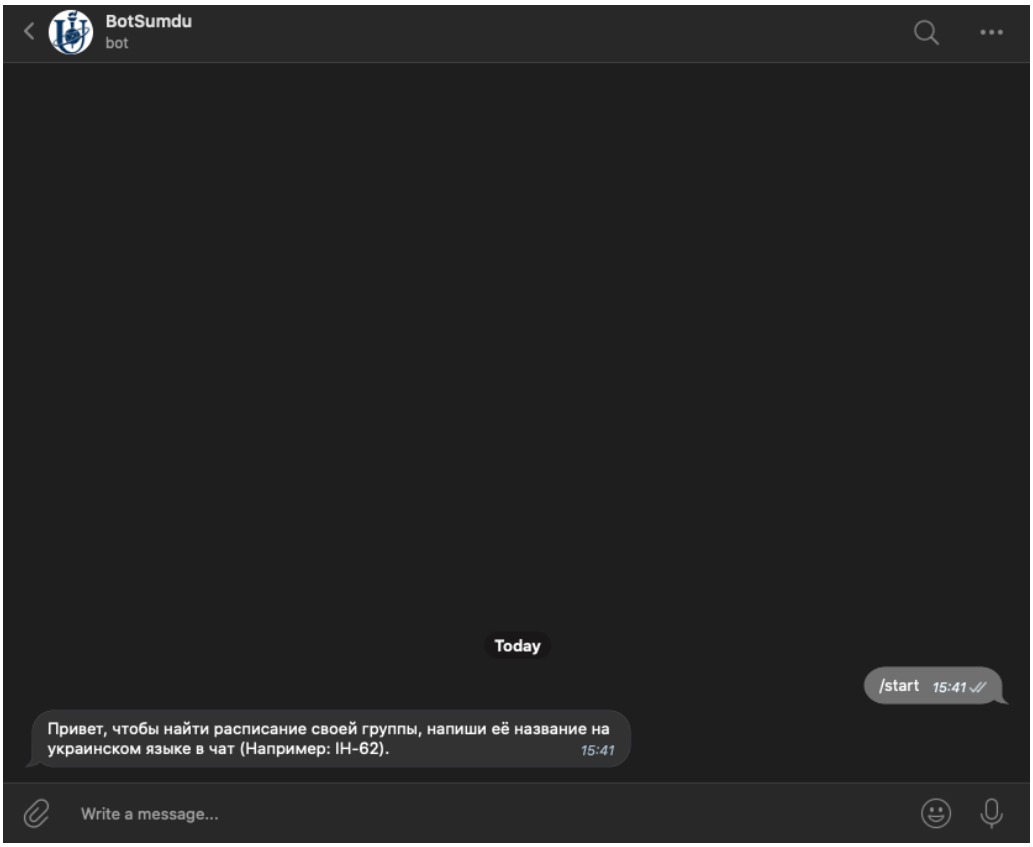

Рисунок 3.4 – Пошук групи

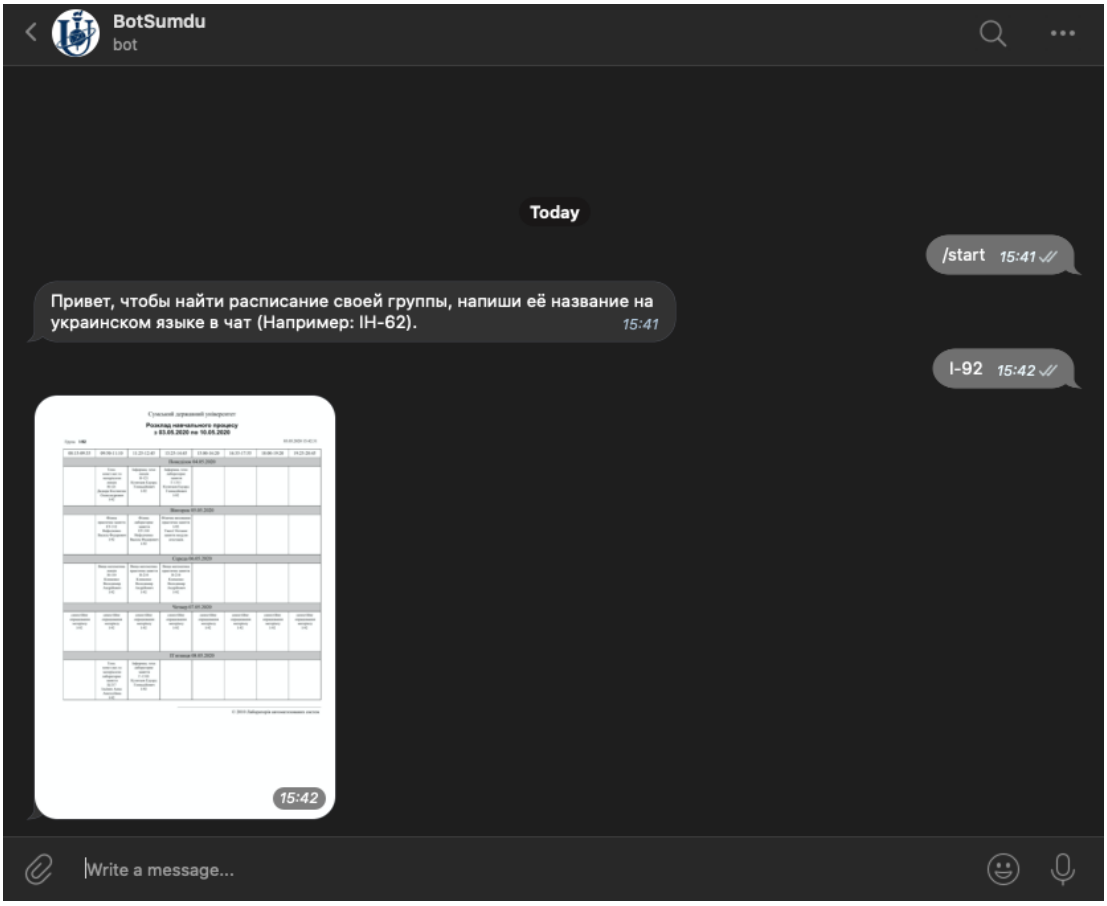

Рисунок 3.5 – Відображення розкладу

3.3. Тестування програмного додатку

Реліз жодного якісного програмного продукту неможливий без тестування. Саме такий підхід унеможливить виникненню складнощів у майбутньому користуванні.

Перший варіант, що має швидку реалізацію полягає в надбудові над Node.JS бібліотекою для реалізації Telegram ботів - node-telegram-bot-api. Опис рішення звучить наступним чином, йде перепис обробника подій на відсилку даних, після чого дані обробляються своїм методом та відсилається вручну згенерований запит.

```
describe('Telegram Test', ()=> {
const myBot = new TestBot(telegramBot);
```

```
let testChat = 0; it('should greet Masha', () => {
const telegramTest = new TelegramTest(telegramBot);
testChat++;
return telegramTest.sendUpdate(testChat, '/ping') .then((data)=> {
if (data.text === 'pong') {
return telegramTest.sendUpdate(testChat, '/start');
}
throw new Error(`Wrong answer for ping! (was ${data.text})`);
 })
 .then(data=> telegramTest.sendUpdate(testChat, data.keyboard[0][0].text))
 .then((data)=> {
 if (data.text === 'Hello, Masha!') {
 return true;
} throw new Error('Wrong greeting!');
 });
 });
 });
```

Дане рішення є оптимальним для швидкого тестування примітивних деталей та покриє більшість потреб тестування.

Проте, цього недостатньо для повного тестування такого як стрес-тестування. В такому випадку є необхідним створення індивідуального варіанту Telegram API.

```
it('should greet Masha', function testFull() {
 this.slow(400);
 this.timeout(800);
 let serverConfig = {port: 9000};
 let server = new TelegramServer(serverConfig);
let token = 'sampleToken';
let client = server.getClient(token);
let message = client.makeMessage('/start');
let telegramBot,
testBot;
return server.start()
 .then(()=> client.sendMessage(message))
 .then(()=> {
 let botOptions = {polling: true, baseApiUrl: server.ApiURL};
 telegramBot = new TelegramBot(token, botOptions);
```

```
 testBot = new TestBot(telegramBot);
 return client.getUpdates();
})
.then((updates)=> {
console.log(colors.blue(`Client received messages: ${JSON.stringify(updates.result)}`));
if (updates.result.length !== 1) { throw new Error('updates queue should contain one message!');
 }
 let keyboard = JSON.parse(updates.result[0].message.reply_markup).keyboard;
 message = client.makeMessage(keyboard[0][0].text);
 client.sendMessage(message); return client.getUpdates();
})
.then((updates)=> { console.log(colors.blue(`Client received messages:
${JSON.stringify(updates.result)}`));
if (updates.result.length !== 1) { throw new Error('updates queue should contain one message!');
}
 if (updates.result[0].message.text !== 'Hello, Masha!')
 { throw new Error('Wrong greeting message!'); } return true;
})
});
```

Даний код демонструє, що йде підняття на локальному порту сервера з подальшим налаштуванням бота на роботу із цим сервером. Потім йде надсилання повідомлень від ботів та клієнтів. Об’єкт клієнта можна отримати із серверу або є можливість реалізувати індивідуального клієнта. Сервер приймає повідомлення в стандартному JSON форматі по визначеній адресі.

Окрім, тестування програмними засобами є тестування, що проводиться власноруч. І виявляє помилки не тільки в реалізації кодом, а й в зовнішньому вигляді та наповненні. Проведення таких превентивних заходів необхідно в першу чергу для зрозумілого та ефективного користування додатком. Мета провести тестування усіх складових програмного додатку для коректного відображення та роботи із усуненням за потреби помилок, що заважають користуванню.

Test Case ID	1	**Test Case Description**	Test connection with telegram bot		
Created By	Nikolay	**Reviewed By**	Nikolay	**Version**	1

QA Tester's Log

Tester's Name	Nikolay	**Date Tested**	июнь 1, 2020	**Test Case (Pass/Fail/ Not Executed)**	Pass

S #	**Prerequisites:**
1	Smartphone
2	Internet
3	Telegram
4	Acount for telegram

S #	**Test Data Requirement**
1	Login
2	Password
3	Name of group
4	

Test Conditions

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	download or update your telegram app	successful update/download	successful update	Pass
2	login to your account or create a new	successful login	successful login/create	Pass
3	search bot with nick **@ssumdu_bot**	successful search	successful search	Pass

3.4. Майбутні допрацювання для програмного додатку

Розклад в мобільному додатку, який не потребує додаткових завантажень безперечно є актуальною розробкою для студенства сьогодення. Зручне та мобільне використання стане допомогою для завантаження списку пар та необхідної інформації.

Проте, обмеження таким функціоналом не є доцільним. Майбутня мета розробки – об’єднати весь технічний учбовий функціонал за допомогою телеграм бота. Це дозволить угрупувати інформаційні матеріали в один електронний ресурс, що зробить його максимально корисним в допомозі студенту та викладачу.

По-перше, при розробці програмного додатку з’явилась потреба інтеграції з особовим кабінетом студента. Цей ресурс містить всю необхідну інформацію від списку групи до оцінок. Проте різноманітність сайтів на різних майданчиках викликає незручності у користуванні. Функціонал повинен задовольняти усі потреби в інформуванні.

Алгоритм додатку, що пропонується спроектовано індивідуально з урахуванням попиту студентських запитів. Вхід в особистий кабінет за допомогою телеграм бота буде можливий за коректного вводу логіна та паролю студента. Після перевірки на правильність даних користувача, студент потрапить до кабінету. Студент матиме змогу переглянути свої оцінки, групу та іншу інформацію з сайту університету не виходячи з месенджеру.

Слід зазначити, що програмне забезпечення не потребує фінансових затрат, а є цілком мобільним доповнення наявних інформаційних систем університету.

Важливим аспектом програмного додатку є безпека та захищеність персональних даних користувачів. Цей фактор забезпечить надійне користування із забезпеченням конфіденційності особистих відомостей.

4. ВИСНОВКИ

Таким чином, мною було програмно реалізовано проєкт створення телеграм боту, який розширює інформаційні можливості студентів. А саме, надання уваги проблемі адаптації студентів в навчальному середовищі шляхом взаємодії початківця з телеграм ботом.

Реалізація цього додатку можлива завдяки готовому програмному продукту Telegram. Всі переваги цього соціального каналу будуть доступні в нашому проєкті. Telegram, на перший погляд, схожий з іншими месенджерами - WhatsApp, Viber, Line. Листування, обмін фото і відео - все як у всіх. Але, на відміну від конкурентів, Telegram пропонує ряд переваг.

1. Передача файлів будь-якого формату - RAR-архів, таблиці в Excel, інструкція в PDF, MP3 і відео. За розміром файлів - обмеження до 1,5 Гб.
2. Хмарне зберігання даних.
3. Синхронізація з пристроями. Месенджер працює на IPhone, Android, WindowsPhone. Є онлайн і ПК-версії. Крім того, можна заходити одразу з декількох пристроїв: кількість одночасних сесій необмежено.
4. Миттєва доставка повідомлень і файлів.
5. Створення великих чатів.
6. Безпека. Ви можете налаштувати автоматичного видалення повідомлень і аккаунта. А ще - без побоювання вести конфіденційні бесіди: кожне повідомлення самознищується після прочитання співрозмовником.

СПИСОК ЛІТЕРАТУРИ

1. Репозиторій Python на GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/python.
2. Документація [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api.
3. Документація Python [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/.
4. Маккінлі У. Python та аналіз даних. — Переклад з англійської. — М.: ДМК Пресс, 2015. — 482 с.
5. Telegram [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Telegram.
6. Рехман, Крістофер Пол, Крістофер Р. Пол. «Платформа розробки Linux: Налаштування, використання та підтримка повного середовища програмування». 2002.
7. Інформаційна модель [Електронний ресурс] – Режим доступу до ресурсу:https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C.
8. PyCharm [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/pycharm/features/editions_comparison_matrix.html.
9. Tele documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/pycharm/features/editions_comparison_matrix.html
10. Python documentation [Електронний ресурс] – Режим доступу до ресурсу: https://devdocs.io/python~3.8/

ДОДАДОК А

ЛІСТІНГ ОСНОВНИХ ФАЙЛІВ КОДУ

Лістинг коду файлу **Bot.py**

```
import telebot

from Connect2Schedule import connect2Schedule
from Convert import convert2Img
from Groups import groupName

bot = telebot.TeleBot('971951928:AAFr6zV0qsRLXJsNZPSHoPnNZhswSJpxy1M')


@bot.message_handler(commands=['start'])
def start_message(message):
    bot.send_message(message.chat.id, 'Привет, чтобы найти расписание своей группы,'
                                      ' напиши её название на украинском языке в чат (Например: ІН-62).')


@bot.message_handler(content_types=['text'])
def send_text(message):
    check = 0
    for x in groupName:
        if message.text.lower() == x:
            connect2Schedule(groupName.index(x))
            convert2Img()
            photo = open('./img/schedule.png', 'rb')
            bot.send_photo(message.chat.id, photo)
            check = 1
    if check == 0:
        bot.send_message(message.chat.id, 'не найдено')


@bot.message_handler(content_types=['sticker'])
def sticker_id(message):
    print(message)


bot.polling()
```

Лістинг коду файлу **Connect2Schedule.py**

```
import requests

from Groups import groupValue


def connect2Schedule(x):
    url = 'http://schedule.sumdu.edu.ua/index/pdf?id_grp=' + groupValue[x]
    myfile = requests.get(url, allow_redirects=True)
    open('./pdf/schedule.pdf', 'wb').write(myfile.content)
```

Лістинг коду файлу **Convert.py**

```
import os
import tempfile
from pdf2image import convert_from_path


def convert2Img():
    filename = './pdf/schedule.pdf'
    with tempfile.TemporaryDirectory() as path:
        images_from_path = convert_from_path(filename, output_folder=path, last_page=1, first_page=0)

    base_filename = os.path.splitext(os.path.basename(filename))[0] + '.png'
    save_dir = './img/'
    for page in images_from_path:
        page.save(os.path.join(save_dir, base_filename), 'PNG')
```

Лістинг коду файлу **Groups.py**

```
groupName = ["i-82", "i-82/1", "i-82/2", "i-92", "i-92/1", "i-92/2", "i-92/3"]
groupValue = ["1000803", "1000807", "1000808", "1001434", "1001450", "1001451", "1001452"]
```