Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Software Engineering and Automation

Web Programming

Laboratory work #5

Backend development

Author:
Popa Eugeniu
std. gr. FAF-202

Supervisor:
Alexei Șerșun

Chișinău 2023

1 Task

- Follow this basic tutorial to create a Telegram bot using @BotFather bot - name it using the format ‘faf20x_<your name>_<your surname>_bot‘ and save the ___token___ in a safe place
- Pick a backend framework and programming language of your choice
- Your application should use Telegram webhooks to respond to updates from chats
- Use a reverse proxy to establish a secure communication between your local web server and Telegram e.g. Ngrok
- Your bot should implement at least the following commands:
  - ‘/start‘ - to show a greeting
  - ‘/latest_news‘ with optional parameter ‘topic‘ - to search for latest news on some topic (up to 5 links)
  - ‘/save_news‘ with required parameter ‘url‘ - to add the URL to the saved news for the given user
  - ‘/saved_news‘ - to show a list of saved news for the given user

2 Results

This laboratory work was developed using Python. Flask is used to create and manage the application.

All the necessary logic for the app is implemented, including:

- basic Telegram bot with ‘/start‘
- the bot implements ‘/latest_news‘ command
- the bot implements ‘/save_news‘ command
- the bot implements ‘/saved_news‘ command

3 Code snippets

This is the main file of the app:

```
from flask import Flask
from flask import jsonify
from flask import request
from dotenv import dotenv_values
import json
import requests
import sqlite3


app = Flask(__name__)

env_vars = dotenv_values('.env')

bot_token = env_vars.get('BOT_TOKEN')
```

```
15 newsapi_token = env_vars.get('NEWSAPI_TOKEN')
16 newsdata_token = env_vars.get('NEWSDATA_TOKEN')
17
18 URL = f'https://api.telegram.org/bot{bot_token}/'
19
20 @app.route('/', methods=['POST', 'GET'])
21 def index():
22     if request.method == 'POST':
23         r = request.get_json()
24
25         if 'message' in r and 'text' in r['message']:
26             write_json(r)
27
28             chat_id = r['message']['chat']['id']
29             message = r['message']['text']
30
31             if message == '/start':
32                 start(chat_id)
33             elif message.startswith('/latest_news'):
34                 topic = ' '.join(message.split('/latest_news ')[1:]).strip()
35                 latest_news(chat_id, topic)
36             elif message.startswith('/save_news'):
37                 news = ' '.join(message.split('/save_news ')[1:]).strip()
38                 save_news("news.db", chat_id, news)
39             elif message == '/saved_news':
40                 saved_news("news.db", chat_id)
41             else:
42                 command_not_found(chat_id)
43
44         return jsonify(r)
45
46     return '<h1>Welcome! This is laboratory work Nr.5 of the Web Programming
    university course.</h1>'
47
48 def start(chat_id):
49     bot_url = URL + 'sendMessage'
50     answer = {'chat_id': chat_id, 'text': 'Welcome! This is laboratory work Nr.5 of
    the Web Programming university course.\n\nAvailable commands:\n/start\n/
    latest_news\n/latest_news <your_topic>\n/save_news <your_URL>\n/saved_news'}
51     r = requests.post(bot_url, json=answer)
52     return r.json()
53
54 def latest_news(chat_id, topic):
55     if topic and is_not_empty(topic):
56         api_endpoint = f'https://newsapi.org/v2/everything?q="{topic}"&apiKey={
    newsapi_token}'
57
58         response = requests.get(api_endpoint)
59
60         data = json.loads(response.text)
61
62         articles = data['articles']
63         article_count = 0
64
65         parsed_articles = f'Top latest 5 news on topic "{topic}" \n\n'
66
67         for article in articles:
68             parsed_article = (f"  {article['title']}\n\n" \
69                               f"{article['description']}\n\n" \
70                               f"Read more: {article['url']}\n\n" \
```

```
                                f"Source: {article['source']['name']}\n\n\n" )
            parsed_articles += parsed_article
            article_count += 1

            if article_count == 5:
                break

        bot_url = URL + 'sendMessage'
        answer = {'chat_id': chat_id, 'text': parsed_articles}
        r = requests.post(bot_url, json=answer)
        return r.json()

    else:
        api_endpoint = f'https://newsdata.io/api/1/news?apikey={newsdata_token}&
    language=en'

        response = requests.get(api_endpoint)

        data = json.loads(response.text)

        articles = data['results']
        article_count = 0

        parsed_articles = f'Top latest 5 news \n\n'

        for article in articles:
            article_description = get_half_string(article['description'])

            parsed_article = (f"  {article['title']}\n\n" \
                              f"{article_description}\n\n" \
                              f"Read more: {article['link']}\n\n" \
                              f"Source: {article['source_id']}\n\n\n" )
            parsed_articles += parsed_article
            article_count += 1

            if article_count == 5:
                break

        bot_url = URL + 'sendMessage'
        answer = {'chat_id': chat_id, 'text': parsed_articles}
        r = requests.post(bot_url, json=answer)
        return r.json()

def create_user_table(database_name):
    conn = sqlite3.connect(database_name)
    cursor = conn.cursor()

    cursor.execute("CREATE TABLE IF NOT EXISTS saved_news(id TEXT, url TEXT)")

    conn.commit()
    conn.close()

def save_news(database_name, chat_id, url):
    if url and is_not_empty(url):
        create_user_table(database_name)
        conn = sqlite3.connect(database_name)
        cursor = conn.cursor()

        cursor.execute('SELECT * FROM saved_news WHERE id = ?', (chat_id,))
        existing_user = cursor.fetchone()
```

```
130
131         if existing_user:
132             urls = existing_user[1].split(', ')
133             if url not in urls:
134                 new_urls = existing_user[1] + ', ' + url
135                 cursor.execute('UPDATE saved_news SET url = ? WHERE id = ?', (
    new_urls, chat_id))
136         else:
137             cursor.execute('INSERT INTO saved_news (id, url) VALUES (?, ?)', (chat_id
    , url))
138
139         conn.commit()
140         conn.close()
141
142         bot_url = URL + 'sendMessage'
143         answer = {'chat_id': chat_id, 'text': "News saved to the database"}
144         r = requests.post(bot_url, json=answer)
145         return r.json()
146     else:
147         bot_url = URL + 'sendMessage'
148         answer = {'chat_id': chat_id, 'text': "Please provide a URL"}
149         r = requests.post(bot_url, json=answer)
150         return r.json()
151
152 def saved_news(database_name, chat_id):
153     create_user_table(database_name)
154
155     conn = sqlite3.connect(database_name)
156     cursor = conn.cursor()
157
158     cursor.execute('SELECT url FROM saved_news WHERE id = ?', (chat_id,))
159     urls = cursor.fetchone()
160
161     conn.close()
162
163     if urls:
164         all_parsed_news = f'Saved URLs: \n\n'
165
166         urls = urls[0].split(', ')
167
168         for url in urls:
169             parsed_news = f'  {url} \n\n'
170             all_parsed_news += parsed_news
171
172         bot_url = URL + 'sendMessage'
173         answer = {'chat_id': chat_id, 'text': all_parsed_news}
174         r = requests.post(bot_url, json=answer)
175         return r.json()
176     else:
177         bot_url = URL + 'sendMessage'
178         answer = {'chat_id': chat_id, 'text': 'No saved news'}
179         r = requests.post(bot_url, json=answer)
180         return r.json()
181
182 def command_not_found(chat_id):
183     bot_url = URL + 'sendMessage'
184     answer = {'chat_id': chat_id, 'text': 'Command not found'}
185     r = requests.post(bot_url, json=answer)
186     return r.json()
187
```

```
188 def is_not_empty(string):
189     return len(string.strip()) != 0
190 
191 def get_half_string(string):
192     half_length = len(string) // 2
193     half = string[:half_length] + "..."
194     return half
195 
196 def write_json(data, filename='answer.json'):
197     with open(filename, 'w') as f:
198         json.dump(data, f, indent=2, ensure_ascii=False)
199 
200 if __name__ == '__main__':
201     app.run()
```

- Importing Dependencies:

The Flask module is imported to create a Flask application that will handle incoming requests.

The jsonify module is imported to convert Python dictionaries to JSON responses.

The request module is imported to access the JSON payload of incoming requests.

The dotenv_values module is imported from dotenv to load environment variables from a .env file.

The json module is imported to work with JSON data.

The requests module is imported to make HTTP requests to external APIs.

The sqlite3 module is imported to work with an SQLite database.

- Flask App Initialization:

An instance of the Flask application is created by calling Flask(___name___) and assigning it to the variable app. The ___name___ variable represents the name of the current module.

- Loading Environment Variables:

The code uses dotenv_values('.env') to load environment variables from a .env file.

The environment variables BOT_TOKEN, NEWSAPI_TOKEN, and NEWSDATA_TOKEN are extracted from the env_vars dictionary for later use. These tokens are used for authentication with the Telegram Bot API, News API, and Newsdata API, respectively.

- Route Definition:

The main route '/' is defined with methods 'POST' and 'GET'. This means the route can handle both POST and GET requests.

When a POST request is received, the code checks if the payload contains a message with text. If so, it calls the appropriate function based on the command received.

When a GET request is received, a simple welcome message is returned as an HTML response.

- Telegram Bot Commands:

The code defines several functions that handle different commands received from the Telegram bot:

start(chat_id): This function sends a welcome message to the user identified by the chat_id parameter.

latest_news(chat_id, topic): This function retrieves the latest news articles based on the provided topic or without a topic. It makes use of two different APIs, News API and Newsdata API, depending on the availability of the topic parameter.

save_news(database_name, chat_id, url): This function saves a news article URL to the SQLite database associated with the user identified by chat_id. The URL is provided as the url parameter.

saved_news(database_name, chat_id): This function retrieves the saved news article URLs for a user from the SQLite database. It fetches the URLs stored in the database for the user identified by chat_id.

command_not_found(chat_id): This function is called when an unknown command is received. It sends a response to the user identified by chat_id indicating that the command was not found.

- SQLite Database Functions:

create_user_table(database_name): This function creates an SQLite database table called saved_news if it doesn't already exist. The table has two columns, id and url, to store user IDs and saved news article URLs, respectively.

save_news(database_name, chat_id, url): This function saves a news article URL (url) to the saved_news table in the SQLite database (database_name) associated with the user identified by chat_id. It first checks if the user already exists in the table. If so, it appends the new URL to the existing URLs for that user. If the user doesn't exist, a new row is inserted with the user ID and URL.

saved_news(database_name, chat_id): This function retrieves the saved news article URLs for a user identified by chat_id from the saved_news table in the SQLite database (database_name). It fetches the URLs associated with the user and returns them as a response.

- Helper Functions:

is_not_empty(string): This function checks if a string is not empty by stripping whitespace and checking its length. It returns True if the string is not empty and False otherwise.

get_half_string(string): This function takes a string and returns the first half of the string, truncated with an ellipsis (...). It is used to limit the length of news article descriptions.

- JSON File Writing:

write_json(data, filename='answer.json'): This function writes the received JSON data to a file with the given filename. It is not directly used in the code but can be useful for debugging purposes.

- App Execution:

Finally, the Flask application is run with app.run() if the script is executed directly. This starts the Flask development server and makes the application accessible at the specified host and port.

Overall, the code sets up a Flask web application that acts as a Telegram bot for retrieving and saving news articles. It defines routes to handle different Telegram bot commands, interacts with external APIs to fetch news articles, and uses an SQLite database to save and retrieve user-specific news URLs.

4 GitHub

`https://github.com/eugencic/web-programming/tree/main/lab5`

5 Presentation

Access this **link** to watch the video presentation.

6 Conclusion

In conclusion, during this laboratory work, I gained hands-on experience in backend development using Python. I learned how to work with the Flask framework and utilize various libraries to build a functional system.

Implementing the functionality to search for daily news on a specific topic was particularly interesting. By leveraging the capabilities of imported libraries, I was able to create a system that interacts with external APIs to retrieve and display news articles based on user input.

The initial structure of the Telegram bot, provided as a skeleton, helped me understand the main functionalities required for the task. I successfully implemented commands such as start, latest news with a specific topic, save news with a URL, and retrieve saved news.

Through this project, I gained valuable insights into the structuring of a Telegram bot and explored its available functions. The experience helped me reinforce my understanding of backend development and allowed me to apply my knowledge in a practical and engaging manner.