Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Software Engineering and Automation

Web Programming

Laboratory work #5

Backend development

Author:
Liviu Mocanu
std. gr. FAF-203

Supervisor:
Alexei Șerșun

Chișinău 2023

1 Task

Our task for this laboratory work was to:

1. Follow a tutorial to create a Telegram bot using @BotFather bot
2. Pick a backend framework and programming language of our choice
3. Our application should use Telegram webhooks to respond to updates from chats;
4. Use a reverse proxy to establish a secure communication between our local web server and Telegram;
5. Our 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.
6. Our application should send HTTP requests to some news API aggregators (e.g. Google News API, NYT API) to fetch the news;

We had no special conditions.

2 Results

I chose the following to complete this laboratory work:

- `Java` and specifically `Spring` as the backend framework
- `PostgreSQL` as the DBMS
- `Ngrok` for the reverse proxy
- `GNews` as the news API aggregator

Besides the main functionality I added some new commands I liked to see in my Telegram Bot and that is:

1. `/news_on` with required parameter 'date' (dd/MM/yyyy) and optional parameter 'topic' - to search for news on some topic for a certain date (up to 5 links);
2. `/remove_news` with required parameter 'URL' - to remove the URL from your saved news;
3. `/menu` - to show an inline menu of topics you could choose to see the latest news of (2 pages, navigation through buttons);
4. `/help` - to see the list of supported commands.

The endpoint I receive the updates through the webhook is located in my `TelegramBotController` controller:

```java
@RestController
@RequiredArgsConstructor
public class TelegramBotController {

    private final MessageService messageService;

    @PostMapping(value = "/update", consumes = "application/json")
    public void receiveUpdate(@RequestBody Update update) {
        messageService.receiveUpdate(update);
    }
}
```

Listing 1: Endpoint for receiving updates

As we can see the update goes into the MessageService's function for receiving an update:

```java
@Service
@RequiredArgsConstructor
public class MessageService {
    // ...

    public void receiveUpdate(Update update) {
        if (update.hasCallbackQuery()) {
            processCallbackQuery(update);
            return;
        }

        var msg = update.getMessage();
        var user = msg.getFrom();
        var userId = user.getId();

        var txt = msg.getText();
        if (msg.isCommand()) {
            switch (txt) {
                case "/start" -> greeting(user);
                case "/saved_news" -> retrieveNews(userId);
                case "/menu" -> menu(userId);
                case "/help" -> help(userId);
                default -> {
                    if (txt.startsWith("/latest_news")) latestNews(userId, txt);
                    else if (txt.startsWith("/news_on")) newsOn(userId, txt);
                    else if (txt.startsWith("/save_news")) saveNews(userId, txt);
                    else if (txt.startsWith("/remove_news")) removeNews(userId, txt);
                    else sendNotification(userId, "Unknown command. Check your
    spelling and try again!");
                }
            }
            return;
        }

        sendNotification(userId, "Sorry, I'm mostly a news bot. \uD83D\uDE14 " +
                "The main way of interacting with me is through commands. (try /help)
     \uD83E\uDD13");
    }

    // ...
}
```

Listing 2: MessageService manage update function

Basically I am checking for 3 things:

- whether the update has a callback query (Inline Menu functionality)

- whether the update is a command
- whether the update is just a message

In case the update is a simple text form the user, the bot replies with a request for interacting with it solely through the supported commands.

Figure 1: Bot's reply for a simple text from the User.

In order to send a message back I have the the sendMessage function that deal with the the creation of a request by receiving some params for this request and the method:

```
1
2  private void sendMessage(Params params, String method) {
3      String url = API_URL + BOT_TOKEN + method;
4
5      // Convert parameters to JSON
6      String json;
7      try {
8          json = objectMapper.writeValueAsString(params);
9      } catch (JsonProcessingException e) {
10         throw new RuntimeException("Failed to convert parameters to JSON", e);
11     }
12
13     // Create the HTTP headers
14     HttpHeaders headers = new HttpHeaders();
15     headers.setContentType(MediaType.APPLICATION_JSON);
16
17     // Create the HTTP entity (headers + body)
18     HttpEntity<String> entity = new HttpEntity<>(json, headers);
19
20     // Send the request
21     ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.POST,
       entity, String.class);
22
23     // Optional: check the response status code and/or body
24     if (!response.getStatusCode().is2xxSuccessful()) {
25         System.out.println("Failed to send message: " + response.getBody());
26     }
27 }
```

Listing 3: Sending a message to User

The Params object is made to allow all the fields to be null and thus no params are added, or have only specific ones thanks to it's Builder possibilities.

```
1 @JsonInclude(JsonInclude.Include.NON_NULL)
2 @Getter
3 public class Params {
4     private String callback_query_id;
5     private Long chat_id;
6     private Integer message_id;
7     private String parse_mode;
```

```
    private String text;
    private InlineKeyboardMarkup reply_markup;
    private boolean disable_web_page_preview;

    public Params withCallbackQueryId(String callbackQueryId) {
        this.callback_query_id = callbackQueryId;
        return this;
    }

    public Params withChatId(Long chatId) {
        this.chat_id = chatId;
        return this;
    }

    public Params withMessageId(Integer msgId) {
        this.message_id = msgId;
        return this;
    }

    public Params withParseMode(String parseMode) {
        this.parse_mode = parseMode;
        return this;
    }

    public Params withText(String text) {
        this.text = text;
        return this;
    }

    public Params withInlineKeyboardMarkup(InlineKeyboardMarkup replyMarkup) {
        this.reply_markup = replyMarkup;
        return this;
    }

    public Params withWebPreview(boolean webPreview) {
        this.disable_web_page_preview = !webPreview;
        return this;
    }

    @Override
    public String toString() {
        return "Params{" +
                "callback_query_id='" + callback_query_id + '\'' +
                ", chat_id=" + chat_id +
                ", message_id=" + message_id +
                ", parse_mode='" + parse_mode + '\'' +
                ", text='" + text + '\'' +
                ", reply_markup=" + reply_markup +
                ", disable_web_page_preview=" + disable_web_page_preview +
                '}';
    }
}
```

Listing 4: Custom object for request params

In order to simplify the idea of notifying the user about an error or some additional information, the sendNotification function is in place:

```
private void sendNotification(Long userId, String text) {
    Params params = new Params()
            .withChatId(userId)
            .withText(text);
```

```
5
6       sendMessage(params, SEND);
7   }
```

Listing 5: Send notification, which is a simple text to the user

I am using 3 methods for sending a message, which are actually telegram API endpoints:

- `SEND` = "/sendMessage";
- `EDIT` = "/editMessageText";
- `ANSWER_CALLBACK` = "/answerCallbackQuery";

With this functionality in place, the Bot is already able to utilise the `` `/start` `` and `` `help` `` commands:

Figure 2: Bot's greeting including the User's first name when using /start.

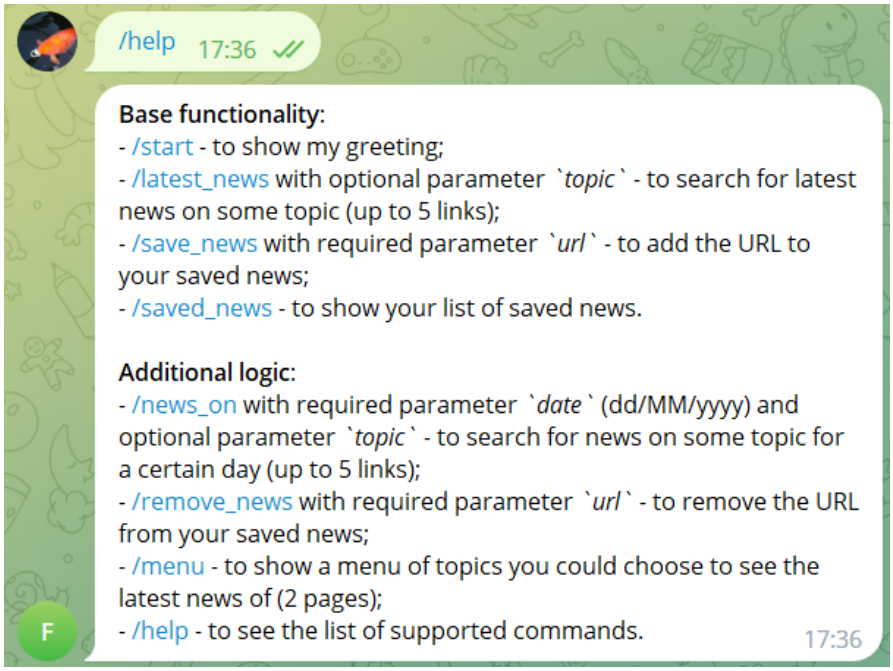


Figure 3: Bot's list of supported commands when using /help.

Using the `/setcommands` provided from @BotFather, I was also able to set up a menu which allows predictions for commands:

Using `processCallbackQuery` as well as a number of `InlineKeyboardButton` paired with `InlineKeybo` I was able to have an inline menu:

```
1 private void processCallbackQuery(Update update) {
2     var query = update.getCallbackQuery();
3     var user = query.getFrom();
```

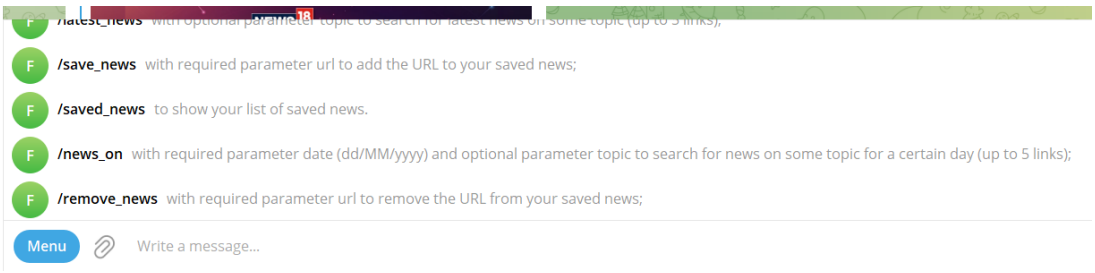


Figure 4: Menu button to see all commands

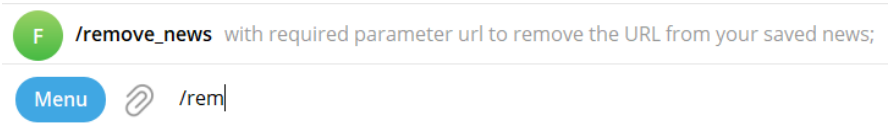
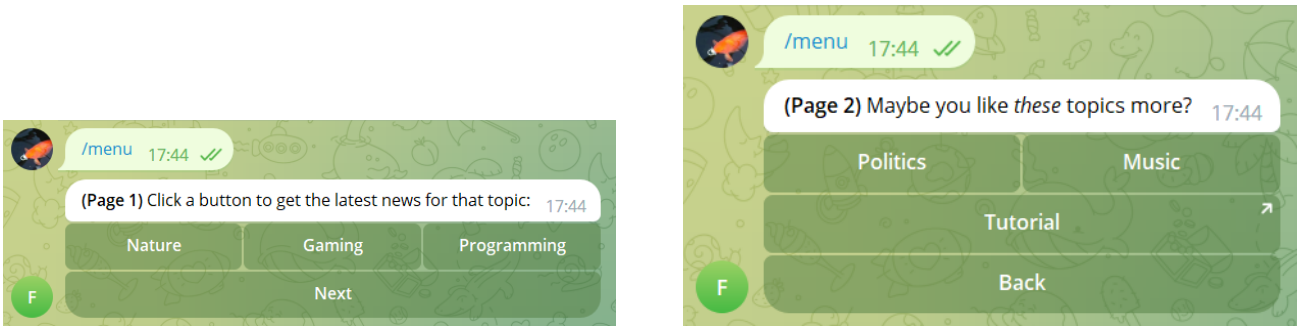


Figure 5: Predicting commands with their explanations.

```
    var data = query.getData();
    var msgId = query.getMessage().getMessageId();

    try {
        buttonTap(user.getId(), query.getId(), data, msgId);
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
}

private void buttonTap(Long chatId, String queryId, String data, int msgId) throws
    TelegramApiException {
    Params params = new Params()
            .withChatId(chatId)
            .withMessageId(msgId)
            .withParseMode("HTML");

    switch (data) {
        case "next" -> {
            params = params
                    .withText("<b>(Page 2)</b> Maybe you like <em>these</em> topics
    more?")
                    .withInlineKeyboardMarkup(keyboardM2);
            sendMessage(params, EDIT);
        }
        case "back" -> {
            params = params
                    .withText("<b>(Page 1)</b> Just choose some news already:")
                    .withInlineKeyboardMarkup(keyboardM1);
            sendMessage(params, EDIT);
        }
        case "nature" -> latestNews(chatId, "/latest_news nature");
        case "gaming" -> latestNews(chatId, "/latest_news gaming");
        case "programming" -> latestNews(chatId, "/latest_news programming");
        case "politics" -> latestNews(chatId, "/latest_news politics");
        case "music" -> latestNews(chatId, "/latest_news music");
    }

    params = new Params().withCallbackQueryId(queryId);
    sendMessage(params, ANSWER_CALLBACK);
```

```
}
```

Listing 6: The processCallbackQuery function paired with buttonTap to edit the same message for Inline functionality



(a) First page of the menu

(b) Second page of the menu

Figure 6: Inline menu for quicker news search by topic

```
private void latestNews(Long chatId, String txt) {
    String topic = txt.length() > "/latest_news".length() ? txt.substring("/
    latest_news".length()).trim() : "general";

    List<Article> articles = newsService.latestNews(topic);

    if (articles.isEmpty()) {
        sendNotification(chatId, "No news on this topic were found. \uD83D\uDE22" +
                " Check your spelling or pick another topic!");
    } else {
        sendArticles(chatId, articles);
    }
}
```

Listing 7: Calling the NewsService to find the latest news for a topic

As we can see, calling latestNews with no topic will provide general news and if the topic is something that provides no news, the Bot will notify the User about it.

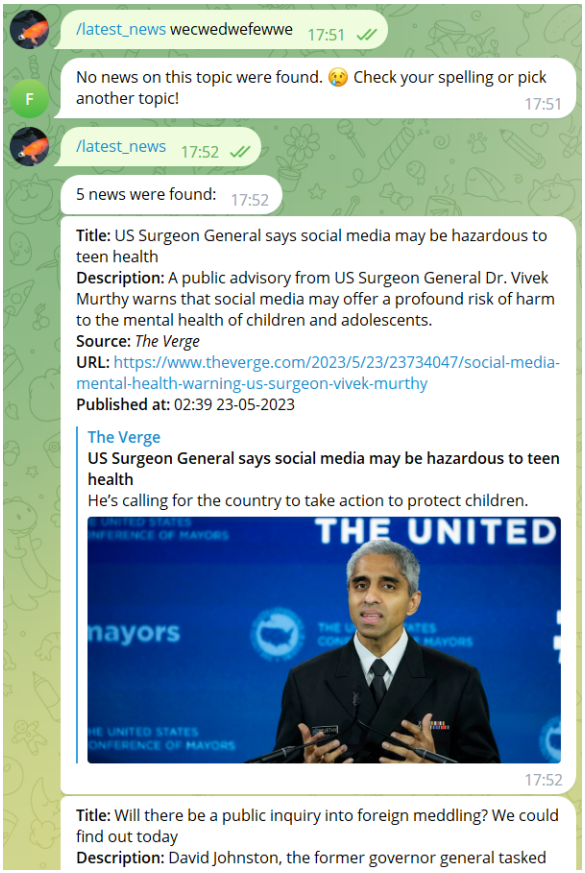
The following is the structure of an article:

```
public class Article {
    private String title;
    private String description;
    private String content;
    private String url;
    private String image;
    private String publishedAt;
    private Source source;
}
```

Listing 8: The Article class

Similarly to `/latest_news`, `/news_on` provides the User with freedom of choosing a date they want to see the news of with optionally also allowing a certain topic:

```
private void newsOn(Long userId, String txt) {
    if (txt.length() <= "/news_on".length()) {
        sendNotification(userId, "Date not provided. Command format is: /news_on DD/
    MM/YYYY [topic]");
        return;
    }
```



(a) Calling latestNews with no topic (getting general news)

(b) Calling latestNews with topic

Figure 7: Using the latestNews function

```
6
7       String dateAndTopic = txt.substring("/news_on".length()).trim();
8       String[] split = dateAndTopic.split("\\s+", 2);
9
10      String dateStr = split[0];
11      String topic = split.length > 1 ? split[1] : "general";
12
13      DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
14      LocalDate date;
15      try {
16          date = LocalDate.parse(dateStr, formatter);
17      } catch (DateTimeParseException e) {
18          sendNotification(userId, "Invalid date format. Use DD/MM/YYYY.");
19          return;
20      }
21
22      List<Article> articles = newsService.newsOn(topic, date);
23
24      if (articles.isEmpty()) {
25          sendNotification(userId, "No news on this topic and date were found. \uD83D\
    uDE22" +
26                  " Check your spelling or pick another topic/date!");
27      } else {
28          sendArticles(userId, articles);
29      }
```

```
}
```

Listing 9: The newsOn function

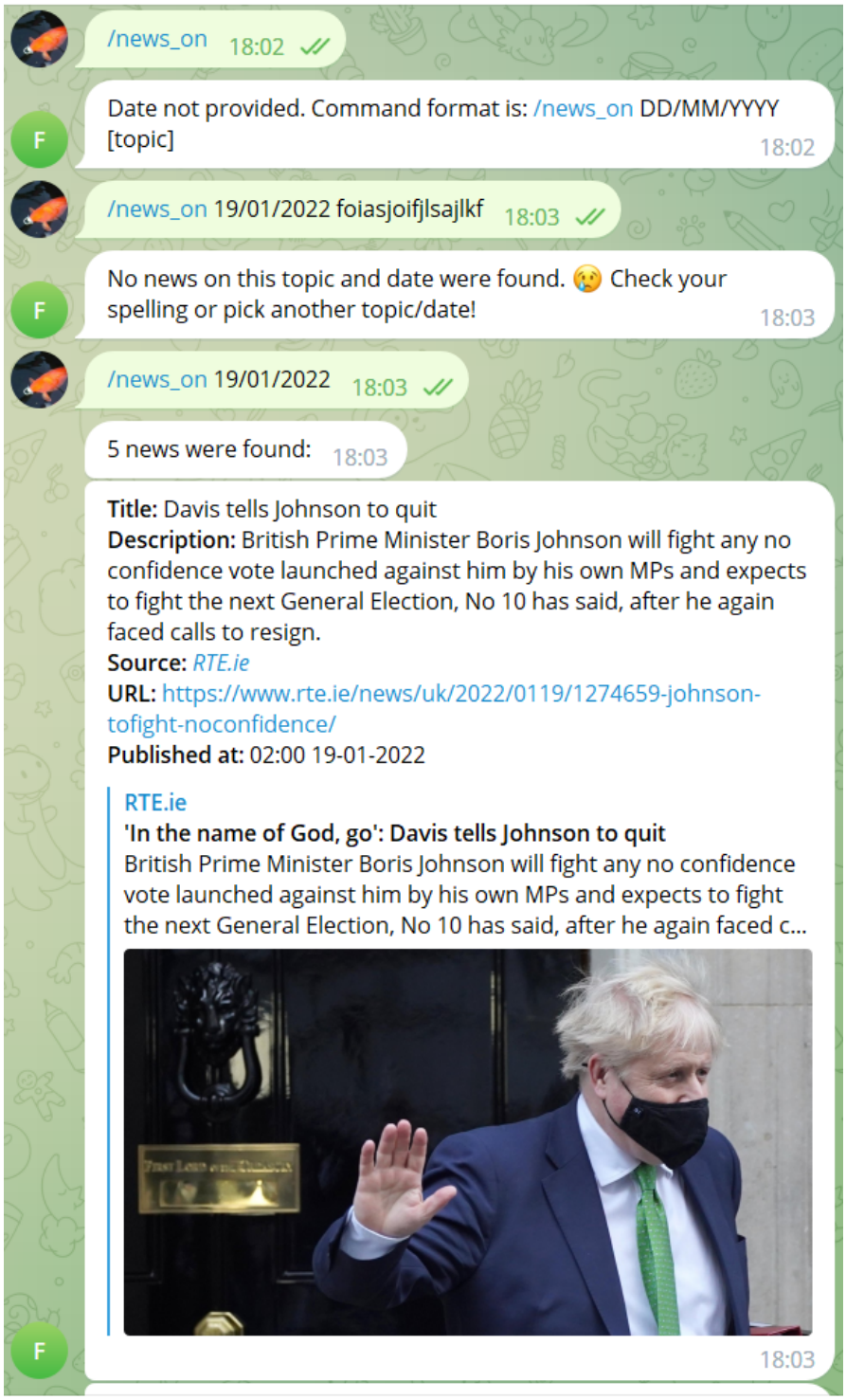


Figure 8: Finding news on a certain date with news on.

When trying to save a new article, the URL is mandatory. Moreover, if the user tries to save the URL twice, they will be let know they already have it stored:

```
private void saveNews(Long userId, String txt) {
    String url;
    if (txt.length() > "/save_news".length()) url = txt.substring("/save_news".length
    ()).trim();
```

```
    else {
        sendNotification(userId, "Whoops! You didn't provide the url of the article you want to save.");
        return;
    }

    if (newsService.saveNews(userId, url)) {
        sendNotification(userId, "Successfully saved!");
    } else {
        sendNotification(userId, "You already have this article saved.");
    }
}
```

Listing 10: Save news function

Figure 9: Saving news.

When trying to remove an article, the URL is mandatory. Moreover, if the user tries to remove

an URL they did not store, the Bot will point that out:

```
private void removeNews(Long userId, String txt) {
    String url;
    if (txt.length() > "/remove_news".length()) url = txt.substring("/remove_news".
    length()).trim();
    else {
        sendNotification(userId, "Whoops! You didn't provide the url of the article
    you want to remove.");
        return;
    }

    if (newsService.removeNews(userId, url)) {
        sendNotification(userId, "Successfully removed!");
    } else {
        sendNotification(userId, "You don't have such an article saved.");
    }
}
```

Listing 11: Remove news function

When trying to retrieve the news for a user I verify whether there are any news stored and the User gets a notification for either cases, pointing out the list is empty or on the contrary how many news are stored:

```
private void retrieveNews(Long userId) {
    List<News> newsList = newsService.getUserNews(userId);

    if (newsList.isEmpty()) {
        sendNotification(userId, "Looks empty here. Try to add some news first!");
    } else {
        sendNotification(userId, "You stored " + newsList.size() + " news:");

        for (News news : newsList) {
            String message = "<b>URL:</b> " + news.getUrl();

            Params params = new Params()
                    .withChatId(userId)
                    .withText(message)
                    .withParseMode("HTML")
                    .withWebPreview(true);

            sendMessage(params, SEND);
        }
    }
}
```

Listing 12: Retrieve news for user function

I leave out the code snippets for NewsService as it mostly comprises of the communication with GNews API as well as responsible for the simple CRUD operations in regards to the database, however, for the purpose of demonstration, this is the entity stored in the db as well as the repository:

```
@Getter
@Setter
@Entity
public class News {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String url;
    private Long userId;
```

```
}

public interface NewsRepository extends JpaRepository<News, Long> {
    Optional<News> findByUserIdAndUrl(Long userId, String url);
    List<News> findAllByUserId(Long userId);
}
```

Listing 13: News entity being saved and the repository

Figure 10: Removing news.

3 Conclusion

This laboratory work for Web Programming involved creating a Telegram bot with various functionalities using different technologies and tools. I chose Java with the Spring framework as the backend, PostgreSQL as the DBMS, Ngrok as the reverse proxy, and GNews as the news API aggregator.

Following the given instructions, I successfully implemented the required commands for the Telegram bot, such as `/start`, `/latest_news`, `/save_news`, and `/saved_news`. Additionally, I took the initiative to enhance the bot's functionality by adding new commands that I found useful, including `/news_on`, `/remove_news`, `/menu` and `/help`.

Throughout this laboratory work, I had the opportunity to explore and apply my knowledge of Java and Spring, as well as work with APIs and handle HTTP requests. I successfully integrated the Telegram bot with the news API aggregator to fetch the latest news based on user requests.

Overall, this project allowed me to expand my skills and deepen my understanding of the technologies and tools involved. It was a valuable learning experience that enabled me to improve my proficiency in Java, Spring, and API integration.

References

[1] Demo of the Telegram Bot, *https://utm-my.sharepoint.com/:v:/g/personal/liviu_mocanu_isa_utm_md/EUxUNb_s3etJrPzeHKk964gB3hZGmkjdqPOLizbRmmalnA?e=OvyaKt*

[2] Telegram tutorial for creating a Bot, *https://core.telegram.org/bots/tutorial#command-logic*

Accessed on May 21, 2023.

[3] Ngrok Tutorial, *https://ngrok.com/docs*

Accessed on May 21, 2023.

[4] Setting a Telegram webhook, *https://core.telegram.org/bots/api#setwebhook*

Accessed on May 22, 2023.

[5] Telegram API methods *https://core.telegram.org/bots/api#available-methods*

Accessed on May 22, 2023.

[6] GNews API tutorial, *https://gnews.io/docs/v4#introduction*

Accessed on May 23, 2023.