



Geekbrains

# **Исследование возможности использования Telegram-ботов в работе профсоюзных организаций различных уровней**

Программа:  
Разработчик — Веб-разработка на Java  
Грубов Сергей Александрович

Москва  
2024

# СОДЕРЖАНИЕ

<b>Введение</b> .....	стр. 3
<b>Глава 1.</b> Теоретические основы работы с Telegram–ботами .....	стр. 5
Особенности Telegram-ботов .....	стр. 5
Основные особенности языка программирования «Java» .....	стр. 7
Основные особенности фреймворка «Spring Boot» .....	стр. 8
Использование «Maven» для автоматической сборки проектов .....	стр. 10
Среда разработки (IDE) .....	стр. 11
Процесс создания Telegram-бота .....	стр. 13
<b>Глава 2.</b> Практические инструменты и методы разработки функционала Telegram–бота для профсоюзных организаций .....	стр. 15
Регистрация Telegram-бота .....	стр. 15
Создание нового проекта «Spring Boot» .....	стр. 16
Импорт проекта «Spring Boot» в IDE .....	стр. 17
Структура проекта в IDE .....	стр. 19
Основной файл приложения .....	стр. 20
Конфигурационный класс приложения .....	стр. 20
Инициализация Telegram-бота .....	стр. 21
Определение класса для пользователей Telegram-бота .....	стр. 22
Определение интерфейса для пользователей Telegram-бота .....	стр. 23
Описание логики Telegram-бота .....	стр. 24
Логирование .....	стр. 33
Файл основных настроек «Spring Boot» .....	стр. 35
Тестирование Telegram-бота .....	стр. 36
<b>Заключение</b> .....	стр. 39
Список используемой литературы .....	стр. 41

# ВВЕДЕНИЕ

Тема работы выбрана исходя из актуальных проблематик использования современных технологий в профсоюзных организациях. В перспективных планах департамента, в котором я работаю, давно стояла задача по автоматизации процессов обращения членов профсоюзов и профсоюзных активистов. В организации есть сайт и сервисы, которые частично решают проблему обработки обращений, однако использование популярных в наше время Telegram-ботов остается у профсоюзов в проектах из-за высокой стоимости разработки и, главное, отсутствия понятных алгоритмов работы подобных систем, понимания целей и задач, которые они должны решать.

Пройденный мною курс обучения «Разработчик» по технологической специализации «Веб-разработка на Java» в «GeekBrains» позволил увидеть примерные шаги и направления для возможной реализации задач по использованию автоматизированных систем (Telegram-ботов) в моей организации. Помимо курса в «GeekBrains» я изучал различные примеры и короткие обучающие материалы по созданию Telegram-ботов.

**Цель данного проекта:** создание масштабируемого Telegram-бота, который поможет профсоюзным организациям взаимодействовать с членами профсоюзов и профсоюзными работниками по всей стране.

## **Задачи:**

1. Изучить литературу, касающуюся темы исследования.
2. Рассмотреть основные методы и инструменты, которые используются для создания Telegram-ботов.
3. Разработать Telegram-бот, который можно подключить к базе данных для сбора информации о пользователях.
4. В Telegram-боте реализовать простой и понятный для пользователей функционал.

5. Подготовить созданное приложение (Telegram-бот) для возможности запуска на сервере.
6. Провести тестирование приложения.

**Инструменты:** IntelliJ IDEA (community), Java Development Kit, Maven, фреймворк Spring Boot, система управления базами данных MySQL Workbench, мессенджер Telegram.

**Состав команды:** Грубов С.А. (Разработчик), Голубцова Л.Л. (Тестировщик), Дамбаев С.Г. (Тестировщик).

# Глава 1. Теоретические основы работы с Telegram–ботами

## Особенности Telegram-ботов

В постоянно развивающейся сфере современных коммуникаций «Telegram» превратился в платформу для личных сообщений и среду для создания интерактивных и интеллектуальных ботов. Они служат разным целям, от автоматизации задач до предоставления информации в режиме реального времени.

Telegram-боты — это автоматизированные объекты, предназначенные для взаимодействия с пользователями и предлагающие широкий спектр услуг на платформе обмена сообщениями «Telegram». Эти боты используют Telegram Bot API, предлагая разработчикам инструментарий для создания интеллектуальных, интерактивных и отзывчивых приложений.

Основные функции Telegram-ботов:

- **Отправка сообщений.** Боты могут отправлять текстовые сообщения, изображения, аудио и видео файлы, а также документы.
- **Приём сообщений.** Боты реагируют на команды, которые пользователи отправляют им, и выполняют различные действия в соответствии с этими командами.
- **Создание меню и кнопок.** Боты создают интерактивные меню и кнопки, чтобы облегчить взаимодействие с пользователем.
- **Интеграция сторонних сервисов и API.** Боты умеют интегрироваться с различными внешними сервисами и API для получения информации или выполнения определённых задач.

- **Работа с базой данных и сохранение данных.** Боты умеют сохранять информацию и данные, введенные пользователями, для дальнейшего использования.
- **Создание игр и развлечений.** Некоторые боты предоставляют различные игры и развлекательные функции.

Telegram-боты могут быть разных видов, в зависимости от целей и потребностей пользователя:

- **Командные боты.** Предназначены для выполнения определённых команд, предлагаемых пользователем.
- **Чат-боты.** Обладают возможностью участия в прямых диалогах с пользователями.
- **Торговые боты.** Работают в сфере финансовых рынков и предоставляют информацию и аналитику о рынке и инвестиционных возможностях.
- **Игровые боты.** Предлагают текстовые квесты, головоломки, игры на фотографиях или видео и многое другое.
- **Административные боты.** Используются для управления и администрирования групповыми чатами в «Telegram».

В этом проекте будут изучены интеграции разработки Telegram-бота при помощи языка программирования «Java» и фреймворка «Spring Boot».

В ходе своего обучения в «GeekBrains» я познакомился с такими известными языками программирования как «C#», «Python» и «Java». К концу обучения мною был сделан выбор в пользу углубленного изучения языка «Java». В отличие от сокращений в «Python» у «Java» более длинный, но в тоже время

понятный код, наряду с его мощностью и обширным количеством библиотек, этот язык может решить большое количество задач. Также существенным плюсом языка «Java» является его безопасность.

«Java» — строго типизированный объектно-ориентированный язык программирования общего назначения. Разработан компанией «Sun Microsystems» (в последующем приобретённой компанией «Oracle»).

## **Основные особенности языка программирования «Java»**

- **Переносимость.** Создатели реализовали принцип WORA: write once, run anywhere или «пиши один раз, запускай везде». Это значит, что написанное на «Java» приложение можно запустить на любой платформе, если на ней установлена среда исполнения «Java» (JRE, Java Runtime Environment).
- **Объектно-ориентированный подход.** «Java» основан на концепции объектов, что делает его более структурированным и модульным. Можно создавать классы и объекты, которые взаимодействуют друг с другом, чтобы решать задачи.
- **Безопасность.** «Java» обладает встроенными механизмами безопасности, которые помогают защитить программы от вредоносного кода и неправильного доступа к памяти.
- **Автоматическое управление памятью.** Разработчик создаёт объекты, а JRE с помощью сборщика мусора очищает память, когда объекты перестают использоваться.
- **Большая библиотека.** «Java» имеет обширную стандартную библиотеку, которая предлагает множество готовых решений для различных задач.

«Java» используется для написания приложений и программных кодов, которые смогут работать на различных платформах: серверах, компьютерах и ноутбуках, мобильных устройствах, приставках.

## Основные особенности фреймворка «Spring Boot»

Фреймворк (англ. framework — «каркас», «структура») — это набор инструментов, компонентов и методов, которые облегчают разработку программного обеспечения. Фреймворк по сути является готовым шаблоном для написания программы. Такие шаблоны помогают программистам быстрее и эффективнее разрабатывать приложения, предоставляя готовые решения для часто используемых задач. Фреймворки используются для решения разных задач: с их помощью можно создавать сайты, интернет-магазины, блоги, веб-приложения.

В мире Java-разработки «Spring» самый популярный фреймворк для создания корпоративных приложений. «Spring» — это фреймворк для «Java», на котором пишут веб-приложения и микросервисы. А «Spring Boot» — это расширение, которое упрощает и ускоряет работу со «Spring». Оно представляет собой набор утилит, автоматизирующих настройки фреймворка. «Spring Boot» разработан для ускорения создания веб-приложений. Он отличается от своего «родителя» тем, что не требует сложной настройки и имеет ряд встроенных инструментов, упрощающих написание кода. Он отличается от базового фреймворка тем, что не требует сложной настройки и имеет ряд встроенных инструментов, упрощающих написание кода.

Некоторые особенности «Spring Boot»:

- упаковывает зависимости в стандартные starter-пакеты;
- автоматически конфигурирует приложения с помощью jar-зависимостей;
- использует JavaConfig, что позволяет отказаться от использования XML;
- не зависит от множественного импорта «Maven» и конфликтов версий;



- обеспечивает мощную пакетную обработку и управляет конечными точками RES;
- упрощает интеграцию с другими Java-фреймворками;
- локально запускает встроенные HTTP-серверы, такие как «Tomcat» и «Jetty».

Одна из особенностей «Spring Boot» — «Spring Boot Starters». Это предварительно настроенные зависимости, облегчающие работу с проектом. В зависимости от задач к приложению можно подключить один или несколько starter-пакетов.

Например, в «Spring Boot Starter Web» есть инструменты для разработки веб-приложений («Tomcat», «Jackson») и для обработки «JSON». А «Spring Boot Starter Test» содержит классы и зависимости для тестирования, такие как «Mockito» и «Spring Boot Test».

Разработчик, пишущий код на «Spring Boot», может запускать встроенные веб-серверы: «Tomcat», «Jetty» и «Undertow», не тратя время на настройку сторонних серверов. Если же ему по какой-то причине необходимо использовать свой сервер, то для этого достаточно исключить зависимости по умолчанию и выбрать подходящий starter-пакет.

Как и родительский фреймворк, «Spring Boot» позволяет работать со встроенными модулями, которые легко интегрируются в приложения, добавляя функциональность и ускоряя разработку.

Например: Spring Data — позволяет работать с различными базами данных; Spring Boot Actuator — функция для мониторинга и управления созданным приложением.

## **Использование «Maven» для автоматической сборки проектов**

В ходе реализации дипломного проекта я в значительной степени оценил возможности и свойства «Maven». Без этого инструмента было бы очень сложно собрать и запустить проект.

«Maven» — это инструмент для автоматической сборки проектов на «Java» и других языках программирования. Он помогает разработчикам правильно подключить библиотеки и фреймворки, управлять их версиями, выстроить структуру проекта и составить к нему документацию. В процессе работы приложения «Maven» вызывает компилятор и автоматически управляет зависимостями и ресурсами: Автоматическая сборка приложения важна на этапах разработки, отладки и тестирования. «Maven» помогает собрать код и ресурсы в исполняемое приложение без IDE (среды разработки).

Система сборки «Maven» отличается гибкостью:

- может использоваться в IDE — Eclipse, IntelliJ IDEA, NetBeans и других;
- не зависит от операционной системы;
- не требует установки — архив с программой можно распаковать в любой директории;
- все необходимые параметры имеют оптимальные настройки по умолчанию;
- упрощает организацию командной работы и документирование;
- запускает библиотеки для модульного тестирования;
- обеспечивает соблюдение стандартов;
- имеет огромное количество плагинов и расширений.

## Среда разработки (IDE)

IDE (Integrated Development Environment) — это интегрированная среда разработки, комплекс программных средств, используемый разработчиками для создания программного обеспечения (ПО). Есть IDE, которые предназначены для работы только с одним языком программирования. Однако большинство современных IDE позволяет работать сразу с несколькими.

IDE предназначена для упрощения разработки программы. Как правило, среды разработки имеют функционал:

В рамках работы над дипломным проектом использовалась среда разработки «IntelliJ IDEA»

«IntelliJ IDEA» — это интегрированная среда разработки, разработанная компанией «JetBrains». Она поддерживает программирование на языке «Java», на его надмножествах «Groovy» и «Kotlin», а также на языке «Scala».

Основные возможности «IntelliJ IDEA»:

- редактор кода с подсветкой синтаксиса;
- система подсказок с умным автодополнением кода;
- интеграция с системами контроля версий проекта: «Git», «SVN» и «Mercurial»;
- возможность совместной работы удаленной команды над проектом;
- поддержка дополнительных плагинов;
- встроенный профилировщик, который указывает, какие участки кода больше всего задействуют мощности процессора и оперативной памяти;
- декомпилятор двоичного кода в Java-код для его отладки;
- поддержка «GitHub» — проектами на «GitHub» можно управлять прямо из IDE.

## Особенности IntelliJ IDEA:

- интеллектуальная поддержка, которая предлагает различные средства анализа кода, помогающие разработчикам создавать качественное программное обеспечение с меньшими ошибками;
- поддержка множества инструментов для тестирования, таких как JUnit и TestNG;
- поддержка множества популярных фреймворков и библиотек.

«IntelliJ IDEA» предоставляет разработчикам множество возможностей для повышения продуктивности. Среди них — интеллектуальное автодополнение кода, мощные инструменты рефакторинга, интеграция с системами контроля версий и многое другое. Благодаря этим функциям, разработчики могут сосредоточиться на написании качественного кода, не отвлекаясь на рутинные задачи. Более того, «IntelliJ IDEA» поддерживает широкий спектр языков программирования и технологий, что делает её универсальным инструментом для разработчиков.

## Процесс создания Telegram-бота

Создание Telegram-ботов предполагает взаимодействие с «BotFather», специализированным ботом в «Telegram», предназначенным для того, чтобы помочь разработчикам генерировать уникальные токены API для соответствующих ботов. «BotFather» помогает разработчикам, проводя их через процесс создания бота, попутно предоставляя детали конфигурации.

Telegram-боты универсальны и способны выполнять различные задачи. Они могут доставлять обновления в режиме реального времени, отвечать на запросы, автоматизировать процессы и даже вести интерактивные беседы. Гибкость функциональных возможностей ботов делает их пригодными для различных приложений.

API Telegram Bot предоставляет полный набор функций. Разработчики могут отправлять сообщения и мультимедийный контент, управлять командами пользователя, устанавливать пользовательские клавиатуры и выполнять встроенные запросы. Этот API позволяет разработчикам создавать ботов, которые выходят за рамки обычных сообщений.

Telegram-боты обеспечивают двустороннюю связь. Инициирование разговоров, выдача команд и получение ответов от ботов — это все действия, которые могут выполнять пользователи. Такой интерактивный характер взаимодействия способствует привлечению пользователей и создает возможности для компаний, сообществ и отдельных лиц взаимодействовать со своей аудиторией.

Telegram уделяет приоритетное внимание безопасности, и разработчики должны придерживаться лучших практик при создании ботов. Обеспечение безопасной обработки пользовательских данных, использование HTTPS и защита токенов API являются важнейшими аспектами ответственной разработки ботов.

Для разработки Telegram-бота можно использовать различные языки программирования и фреймворки. Это включает «Java» с «Spring Boot», «Python» с «Flask» или «Django», «Node.js» с «Telegraf» и другие.

Для создания Telegram-бота необходимо получить ключ (токен) для этого необходимо использовать специальный, технический бот «BotFather» (<https://core.telegram.org/bots#botfather>). В процессе создания боту нужно будет задать название и имя пользователя name и username соответственно. Далее «BotFather» предоставит токен, он необходим для аутентификации бота.

При помощи «Spring initializer» необходимо создать новый проект и перенести в среду разработки. Далее при помощи «Maven» необходимо будет добавить все необходимые библиотеки и зависимости. Для подключения бота к базе данных потребуются установить и настроить систему управления базами данных. В ходе дипломной работы я буду использовать «MySQL Workbenche».

После реализации и запуска Telegram-бота его необходимо протестировать. Для этих целей я привлек своих коллег по работе.

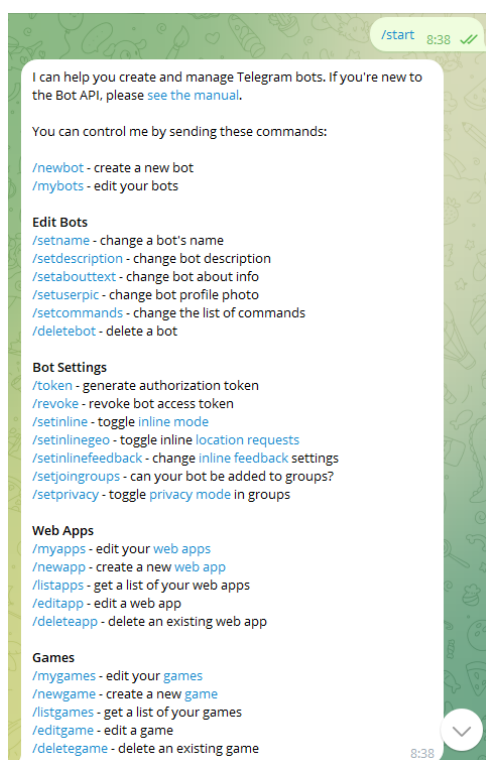
В следующей главе будет подробно описан процесс создания Telegram-бота для профсоюзных организаций.

# Глава 2. Практические инструменты и методы разработки функционала Telegram-бота для профсоюзных организаций

## Регистрация Telegram-бота

Через взаимодействие с ботом «Telegram» «BotFather» необходимо зарегистрировать нового бота, придумать ему имя и получить ключ (токен). После входа BotFather показывает список команд для взаимодействия:

*Рисунок 1. Чат-бот «BotFather».*

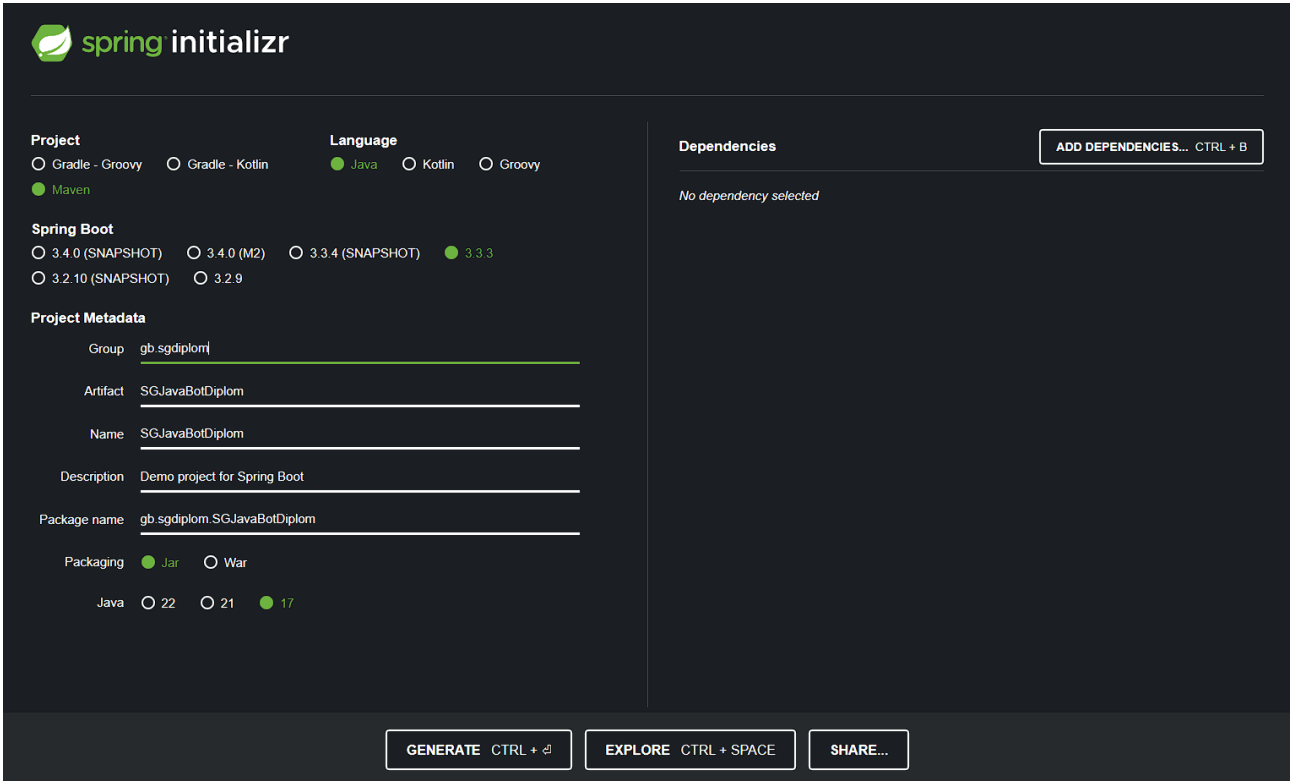


Командой /newbot создается новый бот. Для дипломного проекта имя бота определено как «SGDiplomForGBBot», имя пользователя — «@SGDiplom\_bot».

## Создание нового проекта «Spring Boot»

Для создания нового проекта «Spring Boot» я использовал «Spring initializer». В качестве настроек был выбран язык программирования «Java» версии 17, инструмент для сборки «Maven», версия «Spring Boot» 3.3.3, формат пакета «Jar»..:

Рисунок 2. Создания нового проекта «Spring Boot» в «Spring initializer».



The screenshot displays the Spring Initializr web interface with a dark theme. The interface is divided into several sections for configuring a new Spring Boot project.

- Project:** Includes radio buttons for **Gradle - Groovy**, **Gradle - Kotlin**, **Maven** (selected), and **Language** options: **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Includes radio buttons for versions: **3.4.0 (SNAPSHOT)**, **3.4.0 (M2)**, **3.3.4 (SNAPSHOT)**, **3.3.3** (selected), **3.2.10 (SNAPSHOT)**, and **3.2.9**.
- Project Metadata:** A form with input fields for:
  - Group:** gb.sgdiplo
  - Artifact:** SGJavaBotDiplom
  - Name:** SGJavaBotDiplom
  - Description:** Demo project for Spring Boot
  - Package name:** gb.sgdiplo.SGJavaBotDiplom
- Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
- Java:** Includes radio buttons for versions: **22**, **21**, and **17** (selected).
- Dependencies:** A section with a button **ADD DEPENDENCIES... CTRL + B** and the text *No dependency selected*.

At the bottom of the interface, there are three buttons: **GENERATE CTRL + G**, **EXPLORE CTRL + SPACE**, and **SHARE...**



## Импорт проекта «Spring Boot» в IDE

Для создания Telegram-бота я использовал среду разработки «IntelliJ IDEA Community edition». После сохранения и разархивирования сгенерированного «Spring initializer» проекта в файле настроек «pom.xml» необходимо добавить определенные библиотеки для работы Telegram-бота. Для проекта помимо стандартных библиотек «Spring Boot» были добавлены библиотеки «telembots» для добавления основного функционала бота, «jахb-аpi» и «mysql» для работы с базами данных, «emoji-java» для добавления имоджи и «lombok» для сокращения кода и логирования:

*Листинг 1. Файл настроек «pom.xml».*

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.3.3</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>gb.sgdiplom</groupId>
  <artifactId>SGJavaBotDiplom</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>SGJavaBotDiplom</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <java.version>17</java.version>
    <telegram.version>6.9.7.1</telegram.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.telegram</groupId>
      <artifactId>telembots</artifactId>
      <version>${telegram.version}</version>
    </dependency>

    <dependency>
```

```

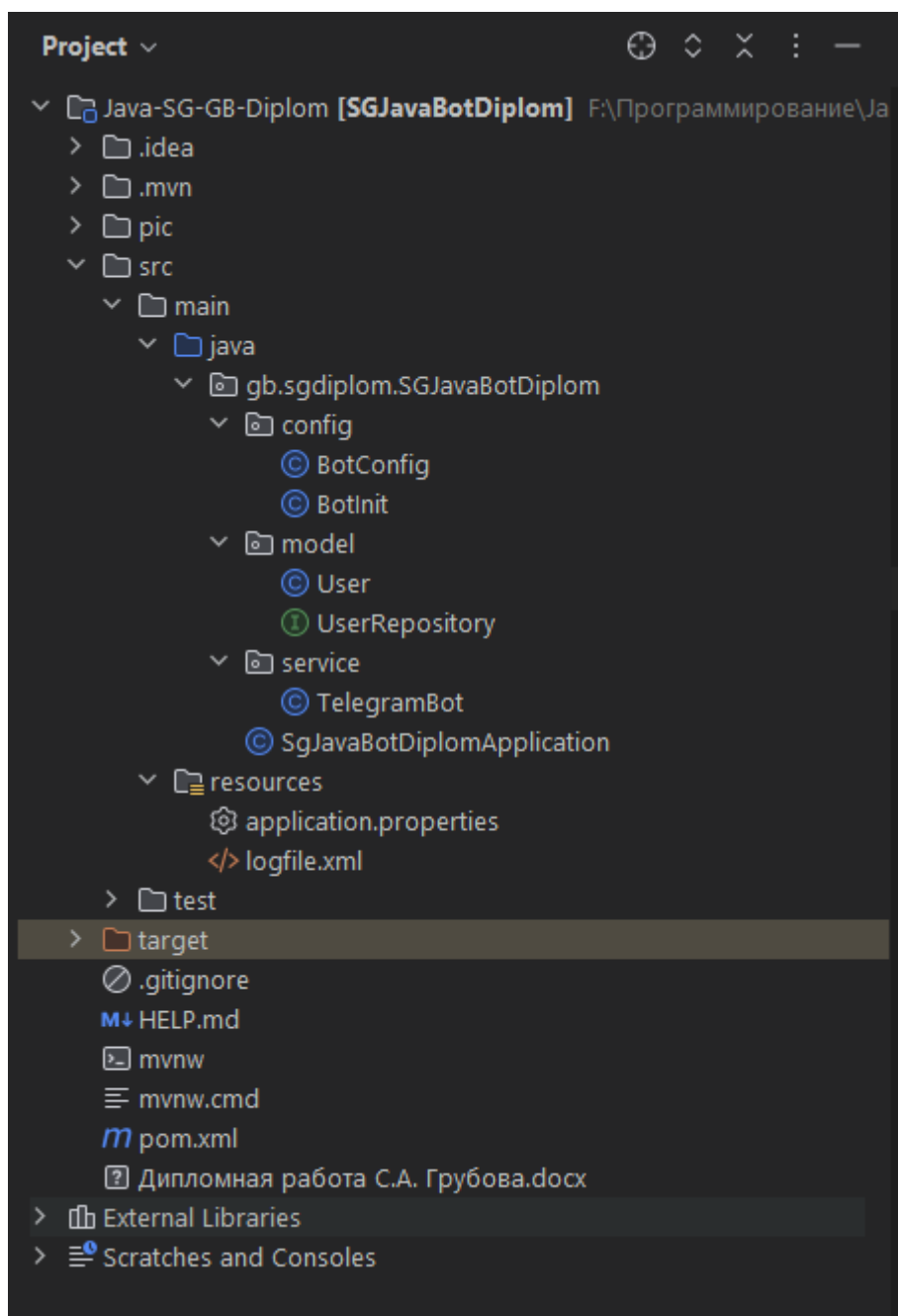
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
<!-- без javax.xml.bind приложение не может установить связь с базой данных и запуститься -->
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.4.0-b180830.0359</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
</dependency>
<!-- Добавление эмодзи в боте. Сайт с кодами: https://emojipedia.org/ -->
<dependency>
    <groupId>com.vdurmont</groupId>
    <artifactId>emoji-java</artifactId>
    <version>5.1.1</version>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <!-- делает jar файл исполняемым -->
            <configuration>
                <executable>true</executable>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

## Структура проекта в IDE

В основном пакете проекта созданы дополнительные пакеты для структурирования кода используя принципы ООП и паттерн проектирования «MVC». В основном пакете созданы дополнительные пакеты «config», «model» и «service».

Рисунок 3. Структура проекта в IDE.



## Основной файл приложения

В основном пакете в файле приложения «SgJavaBotDiplomApplication» создан одноименный класс, в котором происходит запуск приложения.

*Листинг 2. Основной класс Telegram-бота.*

```
package gb.sgdiplom.SGJavaBotDiplom;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SgJavaBotDiplomApplication {

    public static void main(String[] args) {
        SpringApplication.run(SgJavaBotDiplomApplication.class, args);
    }
}
```

## Конфигурационный класс приложения

В пакете «config» в файле «BotConfig» определены переменные, которые необходимы для запуска Telegram-бота. Аннотация «@PropertySource("application.properties")» делает ссылку на файл конфигурации «application.properties», где хранятся имя и ключ (токен) бота.

*Листинг 3. Основной класс Telegram-бота.*

```
package gb.sgdiplom.SGJavaBotDiplom.config;

import lombok.Data;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Configuration
@Data
@PropertySource("application.properties")
public class BotConfig {
    @Value("${bot.name}")
    String botName;
    @Value("${bot.token}")
    String token;
}
```

}

## Инициализация Telegram-бота

В пакете «config» в файле «BotInit» происходит инициализация бота, создается класс бота и создается сессия. Для логирования добавлена аннотация «@Slf4j».

*Листинг 4. Создание класса для инициализации Telegram-бота.*

```
package gb.sgdiplom.SGJavaBotDiplom.config;
import gb.sgdiplom.SGJavaBotDiplom.service.TelegramBot;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.TelegramBotsApi;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import org.telegram.telegrambots.updatesreceivers.DefaultBotSession;

@Slf4j
@Component
public class BotInit {
    @Autowired
    TelegramBot bot;

    @EventListener({ContextRefreshedEvent.class})
    public void init() throws TelegramApiException{
        TelegramBotsApi telegramBotsApi = new TelegramBotsApi(DefaultBotSession.class);
        try {
            telegramBotsApi.registerBot(bot);
        }
        catch (TelegramApiException e){
            log.error("Error bot:" + e.getMessage());
        }
    }
}
```

## Определение класса для пользователей Telegram-бота

В пакете «model» в файле «User» определен класс для взаимодействия с пользователями, в рамках данного проекта бот будет собирать данные о пользователях и сохранять в базу данных. Класс 5 полей «User», поле «chatId» необходимо для взаимодействия бота с конкретным пользователем, остальные поля для сбора информации.

*Листинг 5. Определение класса пользователя.*

```
package gb.sgdiplom.SGJavaBotDiplom.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Getter;

import java.sql.Timestamp;

@Getter
@Entity(name = "usersTable")
public class User {
    @Id
    private Long chatId;

    private String firstName;
    private String lastName;
    private String userName;
    private Timestamp registredAt;
    public void setChatId(Long chatId) {this.chatId = chatId;}
    public void setFirstName(String firstName) {this.firstName = firstName;}
    public void setLastName(String lastName) {this.lastName = lastName;}
    public void setUserName(String userName) {this.userName = userName;}
    public void setRegistredAt(Timestamp registredAt) {this.registredAt = registredAt;}

    @Override
    public String toString() {
        return "User{" +
            "chatId=" + chatId +
            ", firstName=" + firstName + "\" +
            ", lastName=" + lastName + "\" +
            ", userName=" + userName + "\" +
            ", registredAt=" + registredAt +
            '}'
        }
    }
```

## Определение интерфейса для пользователей Telegram-бота

В пакете «model» в файле «UserRepository» определен интерфейс для взаимодействия с объектами класса «User» и записи информации в базу данных.

*Листинг 6. Интерфейс для взаимодействия с объектами класса «User».*

```
package gb.sgdiplom.SGJavaBotDiplom.model;

import org.springframework.data.repository.CrudRepository;

public interface UserRepository extends CrudRepository<User, Long> {
}
```

## Описание логики Telegram-бота

В пакете «service» в файле «TelegramBot» определен класс, описывающий всю логику Telegram-бота. Задача бота помочь пользователям найти необходимый сервис и получить информацию. Данные о пользователях будут записываться в отдельную таблицу базы данных. Также выбор пользователей будет логироваться для последующего анализа использования бота, его доработке и усовершенствования. Класс «TelegramBot» получился самым большим и в листинге будет разделен описанием функциональных фрагментов кода.

*Листинг 7.1. Класс, описывающий логику Telegram-бота. Основное описание класса.*

```
package gb.sgdiplom.SGJavaBotDiplom.service;

import com.vdurmont.emoji.EmojiParser;
import gb.sgdiplom.SGJavaBotDiplom.config.BotConfig;
import gb.sgdiplom.SGJavaBotDiplom.model.User;
import gb.sgdiplom.SGJavaBotDiplom.model.UserRepository;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods.commands.SetMyCommands;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.updatingmessages.EditMessageText;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.commands.BotCommand;
import org.telegram.telegrambots.meta.api.objects.commands.scope.BotCommandScopeDefault;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;

@Slf4j
@Component
```



```

public class TelegramBot extends TelegramLongPollingBot {
    @Autowired
    private UserRepository userRepository;

    final BotConfig config;
    static final String HELP_TEXT = "Бот создан для дипломного проекта студента С.А. Грубова. " +
        "Команды бота вы можете посмотреть в меню слева.";

```

*Листинг 7.2. Класс, описывающий логику Telegram-бота. Конструктор.*

В данном фрагменте кода описывается основное меню бота (нижний левый угол), которое реализована через конструктор.

```

public TelegramBot(UserRepository userRepository, BotConfig config) {
    this.config = config;
    List<BotCommand> menuCommands = new ArrayList<>();
    menuCommands.add(new BotCommand("/start", "Начать общение"));
    menuCommands.add(new BotCommand("/register", "Вступить в Профсоюз"));
    menuCommands.add(new BotCommand("/law", "Получить юридическую консультацию"));
    menuCommands.add(new BotCommand("/data", "Найти свой профсоюз или профобъединение"));
    menuCommands.add(new BotCommand("/safety", "Сообщить о несчастном случае на производстве"));
    menuCommands.add(new BotCommand("/help", "Информация о боте"));
    try {
        this.execute(new SetMyCommands(menuCommands, new BotCommandScopeDefault(), null));
    } catch (TelegramApiException e) {
        log.error("Error setting bot command list: " + e.getMessage());
    }
}

@Override
public String getBotUsername() {
    return config.getBotName();
}

@Override
public String getBotToken() {
    return config.getToken();
}

@Override
public void onUpdateReceived(Update update) {

```

*Листинг 7.3. Класс, описывающий логику Telegram-бота. Ветвление.*

В данном фрагменте кода представлено ветвление исходя из выбора пользователя. В боте реализованы основное меню, кнопки выбора и кнопки выбора с текстом. Функционал меню и кнопок практически идентичен, кнопки с текстом содержат дополнительную проверку, если пользователь подтвердит выбор сервиса.

```
if (update.hasMessage() && update.getMessage().hasText()) {
    String messageText = update.getMessage().getText();
    long chatId = update.getMessage().getChatId();

    switch (messageText) {
        case "/start":
            registerUser(update.getMessage());
            startCommandReceived(chatId, update.getMessage().getChat().getFirstName());
            break;
        case "/register":
            register(chatId);
            break;
        case "Вступить в Профсоюз":
            register(chatId);
            break;
        case "/law":
            lawHelp(chatId);
            break;
        case "Получить юридическую консультацию":
            lawHelp(chatId);
            break;
        case "/data":
            dataUnions(chatId);
            break;
        case "Найти свой профсоюз или профобъединение":
            dataUnions(chatId);
            break;
        case "/safety":
            safety(chatId);
            break;
        case "Сообщить о несчастном случае на производстве":
            safety(chatId);
            break;
        case "/help":
            sendMessage(chatId, HELP_TEXT);
            break;
        default:
            sendMessage(chatId, "Command not found");
            log.info(update.getMessage().getText() + " - необработанный запрос от пользователя  
<" + update.getMessage().getChat().getFirstName() + ">");
    }
}
```

```

    }
} else if (update.hasCallbackQuery()) {
    String callbackData = update.getCallbackQuery().getData();
    long messageId = update.getCallbackQuery().getMessage().getMessageId();
    long chatId = update.getCallbackQuery().getMessage().getChatId();

    if (callbackData.equals("YES_BUTTON")) {
        EditMessageText message = new EditMessageText();
        message.setChatId(String.valueOf(chatId));
        message.setMessageId((int) messageId);
        try {
            execute(message);
        } catch (TelegramApiException e) {
            log.error("Error bot: " + e.getMessage());
        }
    } else if (callbackData.equals("NO_BUTTON")) {
        TextReceived(chatId);
        EditMessageText message = new EditMessageText();
        message.setChatId(String.valueOf(chatId));
        message.setMessageId((int) messageId);
        try {
            execute(message);
        } catch (TelegramApiException e) {
            log.error("Error bot: " + e.getMessage());
        }
    }
}
}
}

```

*Листинг 7.4. Класс, описывающий логику Telegram-бота. Описание кнопок с текстом.*

Далее в коде описываются кнопки с текстом, для каждого сервиса свое описание и ссылка на ресурс.

```

private void safety(long chatId) {
    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText("Хотите сообщить о несчастном случае на производстве?");

    InlineKeyboardMarkup markupInline = new InlineKeyboardMarkup();
    List<List<InlineKeyboardButton>> rowsInline = new ArrayList<>();
    List<InlineKeyboardButton> rowInline = new ArrayList<>();

    var yesButton = new InlineKeyboardButton();
    yesButton.setText("Да!");
    yesButton.setUrl("https://fnpr.ru/instrukciya/");

    var noButton = new InlineKeyboardButton();
}

```

```

noButton.setText("Нет");
noButton.setCallbackData("NO_BUTTON");

rowInline.add(yesButton);
rowInline.add(noButton);

rowsInline.add(rowInline);

markupInline.setKeyboard(rowsInline);
message.setReplyMarkup(markupInline);

log.info("Запрос на сообщение о несчастном случае");

try {
    execute(message);
} catch (TelegramApiException e) {
    log.error("Error bot: " + e.getMessage());
}
}

private void dataUnions(long chatId) {
    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText("Хотите найти свой профсоюз или профобъединение?");

    InlineKeyboardMarkup markupInline = new InlineKeyboardMarkup();
    List<List<InlineKeyboardButton>> rowsInline = new ArrayList<>();
    List<InlineKeyboardButton> rowInline = new ArrayList<>();

    var yesButton = new InlineKeyboardButton();
    yesButton.setText("Да!");
    yesButton.setUrl("https://fnpr.ru/structure/");

    var noButton = new InlineKeyboardButton();
    noButton.setText("Нет");
    noButton.setCallbackData("NO_BUTTON");

    rowInline.add(yesButton);
    rowInline.add(noButton);

    rowsInline.add(rowInline);

    markupInline.setKeyboard(rowsInline);
    message.setReplyMarkup(markupInline);

    log.info("Запрос на поиск профсоюза");

    try {
        execute(message);
    }
}

```

```

    } catch (TelegramApiException e) {
        log.error("Error bot: " + e.getMessage());
    }
}

```

```

private void lawHelp(long chatId) {
    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText("Хотите получить юридическую консультацию?");

    InlineKeyboardMarkup markupInline = new InlineKeyboardMarkup();
    List<List<InlineKeyboardButton>> rowsInline = new ArrayList<>();
    List<InlineKeyboardButton> rowInline = new ArrayList<>();

    var yesButton = new InlineKeyboardButton();
    yesButton.setText("Да!");
    yesButton.setUrl("https://fnpr.ru/legal/");

    var noButton = new InlineKeyboardButton();
    noButton.setText("Нет");
    noButton.setCallbackData("NO_BUTTON");

    rowInline.add(yesButton);
    rowInline.add(noButton);

    rowsInline.add(rowInline);

    markupInline.setKeyboard(rowsInline);
    message.setReplyMarkup(markupInline);

    log.info("Запрос на получение юридической консультации");

    try {
        execute(message);
    } catch (TelegramApiException e) {
        log.error("Error bot: " + e.getMessage());
    }
}

```

```

private void register(long chatId) {
    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText("Хотите вступить в профсоюз?");

    InlineKeyboardMarkup markupInline = new InlineKeyboardMarkup();
    List<List<InlineKeyboardButton>> rowsInline = new ArrayList<>();
    List<InlineKeyboardButton> rowInline = new ArrayList<>();

```

```

var yesButton = new InlineKeyboardButton();
yesButton.setText("Да!");
yesButton.setUrl("https://fnpr.ru/union/");
yesButton.setCallbackData("YES_BUTTON");

var noButton = new InlineKeyboardButton();
noButton.setText("Нет");
noButton.setCallbackData("NO_BUTTON");

rowInline.add(yesButton);
rowInline.add(noButton);

rowsInline.add(rowInline);

markupInline.setKeyboard(rowsInline);
message.setReplyMarkup(markupInline);

log.info("Запрос на вступление в профсоюз");

try {
    execute(message);
} catch (TelegramApiException e) {
    log.error("Error bot: " + e.getMessage());
}
}

```

*Листинг 7.5. Класс, описывающий логику Telegram-бота. Запись информации о пользователях в базу данных.*

Метод «registerUser» производит запись информации о пользователях в подключенную к боту базу данных.

```

private void registerUser(Message msg) {
    if (userRepository.findById(msg.getChatId()).isEmpty()) {
        var chatId = msg.getChatId();
        var chat = msg.getChat();
        User user = new User();

        user.setChatId(chatId);
        user.setFirstName(chat.getFirstName());
        user.setLastName(chat.getLastName());
        user.setUserName(chat.getUserName());
        user.setRegistredAt(new Timestamp(System.currentTimeMillis()));

        userRepository.save(user);

        log.info("Создана запись о пользователе: " + user);
    }
}

```

```

    }
}

```

*Листинг 7.6. Класс, описывающий логику Telegram-бота. Приветственное сообщение.*

Метод «startCommandReceived» отображает приветственное сообщение с обращением к пользователю по имени. Данное сообщение можно вызвать командой «/start».

```

private void startCommandReceived(long chatId, String name) {
    String answer = EmojiParser.parseToUnicode("Привет, " + name + ", рад с вами
познакомиться " + ":blush:\n" +
        "Этот бот создан для дипломного проекта студента С.А. Грубова.\n" +
        "Команды бота: \n\n" +
        "/start — Начать общение\n\n" +
        "/register — Вступить в Профсоюз\n\n" +
        "/law — Получить юридическую консультацию\n\n" +
        "/data — Найти свой профсоюз или профобъединение\n\n" +
        "/safety — Сообщить о несчастном случае на производстве\n\n" +
        "/help — Информация о боте\n\n" +
        "Также команды бота доступны в меню.");

    log.info("Ответ на запрос пользователя " + name);
    sendMessage(chatId, answer);
}

```

*Листинг 7.7. Класс, описывающий логику Telegram-бота. Текстовое сообщение.*

Метод «startCommandReceived» отображает текстовое сообщение, которое отображается если пользователь отказался переходить в конкретный сервис.

```

private void TextReceived(long chatId) {
    String answer = EmojiParser.parseToUnicode("Команды бота: \n\n" +
        "/start — Начать общение\n\n" +
        "/register — Вступить в Профсоюз\n\n" +
        "/law — Получить юридическую консультацию\n\n" +
        "/data — Найти свой профсоюз или профобъединение\n\n" +
        "/safety — Сообщить о несчастном случае на производстве\n\n" +
        "/help — Информация о боте\n\n" +
        "Также команды бота доступны в меню.");
    log.info("Ответ на запрос ");
    sendMessage(chatId, answer);
}

```

*Листинг 7.8. Класс, описывающий логику Telegram-бота. Создание кнопок.*

В данном методе описываются кнопки бота.

```
private void sendMessage(long chatId, String textToSend) {
    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText(textToSend);

    ReplyKeyboardMarkup keyboardMarkup = new ReplyKeyboardMarkup();
    List<KeyboardRow> keyboardRows = new ArrayList<>();
    KeyboardRow row = new KeyboardRow();
    row.add("Вступить в Профсоюз");

    keyboardRows.add(row);
    row = new KeyboardRow();
    row.add("Получить юридическую консультацию");

    keyboardRows.add(row);
    row = new KeyboardRow();
    row.add("Найти свой профсоюз или профобъединение");

    keyboardRows.add(row);
    row = new KeyboardRow();
    row.add("Сообщить о несчастном случае на производстве");

    keyboardRows.add(row);

    keyboardMarkup.setKeyboard(keyboardRows);

    message.setReplyMarkup(keyboardMarkup);

    try {
        execute(message);
    } catch (TelegramApiException e) {
        log.error("Error bot: " + e.getMessage());
    }
}
```



## Логирование

Для каждого сервиса бота определено логирование с уровнем INFO. Данные записываются в отдельный лог-файл. В последующем можно анализировать полученную информацию о выборе пользователей отдельных сервисов. Настройки логирования определены в файле «logfile.xml».

*Листинг 8. Файл настроек логирования*

```
<configuration>

  <property name="HOME_LOG" value="/var/log/gb-sg-diplomBot/bot.log"/>

  <appender name="FILE-ROLLING" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${HOME_LOG}</file>

    <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
      <fileNamePattern>/var/log/gb-sg-diplomBot/bot.%d{yyyy-MM-dd}.%i.log.gz</fileNamePattern>

<!-- максимальный размер файла журнала 10 MB -->
      <maxFileSize>10MB</maxFileSize>
<!-- максимальный суммарный размер файлов до начала удаления самых старых - 100 MB -->
      <totalSizeCap>100MB</totalSizeCap>
<!-- срок хранения файла — 50 дней -->
      <maxHistory>50</maxHistory>
    </rollingPolicy>

    <encoder>
      <pattern>%d %p %c{1.} [%t] %m%n</pattern>
    </encoder>
  </appender>

  <logger name="io.proj3ct.SpringDemoBot" level="debug" additivity="false">
    <appender-ref ref="FILE-ROLLING"/>
  </logger>

  <root level="error">
    <appender-ref ref="FILE-ROLLING"/>
  </root>

  <root level="info">
    <appender-ref ref="FILE-ROLLING"/>
  </root>

</configuration>
```

Логи содержат время обращения к сервисам, краткую информацию по необработанным запросам и информацию об ошибках.

*Листинг 8. Записи в логфайле*

```
2024-08-26T18:51:04.504+03:00    INFO    16832    ---    [SGJavaBotDiplom]    [legram    Executor]
g.s.SGJavaBotDiplom.service.TelegramBot : Запрос на вступление в профсоюз

2024-08-26T18:51:07.558+03:00    INFO    16832    ---    [SGJavaBotDiplom]    [legram    Executor]
g.s.SGJavaBotDiplom.service.TelegramBot : Запрос на получение юридической консультации

2024-08-26T18:51:12.550+03:00    INFO    16832    ---    [SGJavaBotDiplom]    [legram    Executor]
g.s.SGJavaBotDiplom.service.TelegramBot : Запрос на сообщение о несчастном случае

2024-08-26T18:51:14.266+03:00    INFO    16832    ---    [SGJavaBotDiplom]    [legram    Executor]
g.s.SGJavaBotDiplom.service.TelegramBot : Запрос на поиск профсоюза

2024-08-26T18:51:16.769+03:00    INFO    16832    ---    [SGJavaBotDiplom]    [legram    Executor]
g.s.SGJavaBotDiplom.service.TelegramBot : апарап - необработанный запрос от
пользователя <Сергей>
```

## Файл основных настроек «Spring Boot»

Основные настройки «Spring Boot» находятся в файле «application.properties». Настройки разделены на три части: основные, где записано название приложения и указан порт; данные для инициализации бота (имя и ключ); настройки базы данных.

*Листинг 9. Файл настроек «Spring Boot»*

```
# Main settings
spring.application.name=SGJavaBotDiplom
server.port=8088

# Bot settings
bot.name=SGDiplom_bot
bot.token=7265990445:AAFbzBFmKtoP9vlpZQDa8DiKg3SICL5Ou6E

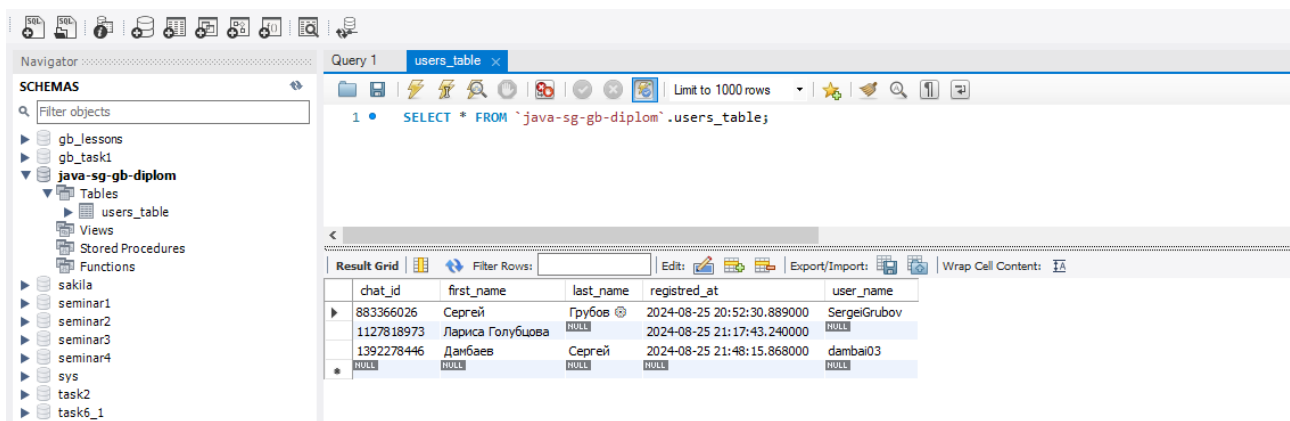
# DataBase settings
spring.datasource.url=jdbc:mysql://localhost:3306/java-sg-gb-diplom
spring.datasource.username=root
spring.datasource.password=1250
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

# Тестирование Telegram-бота

Для тестирования бота я привлек своих коллег. Бот был запущен на локальном компьютере. Ссылка на бот: [https://t.me/SGDiplom\\_bot](https://t.me/SGDiplom_bot)

Далее будут представлены иллюстрации работы бота и записи в базе данных.

Рисунок 4. Записи в базе данных после тестирования.



The screenshot shows a database interface with a sidebar on the left listing schemas like 'gb\_lessons', 'gb\_task1', and 'java-sg-gb-diplom'. The main area displays a query result for 'users\_table'.

chat_id	first_name	last_name	registered_at	user_name
883366026	Сергей	Грубов	2024-08-25 20:52:30.889000	SergeiGrubov
1127818973	Лариса	Голубцова	2024-08-25 21:17:43.240000	NULL
1392278446	Дамбаев	Сергей	2024-08-25 21:48:15.868000	dambai03
NULL	NULL	NULL	NULL	NULL

Рисунок 5. Взаимодействие с ботом. Команда «/start».

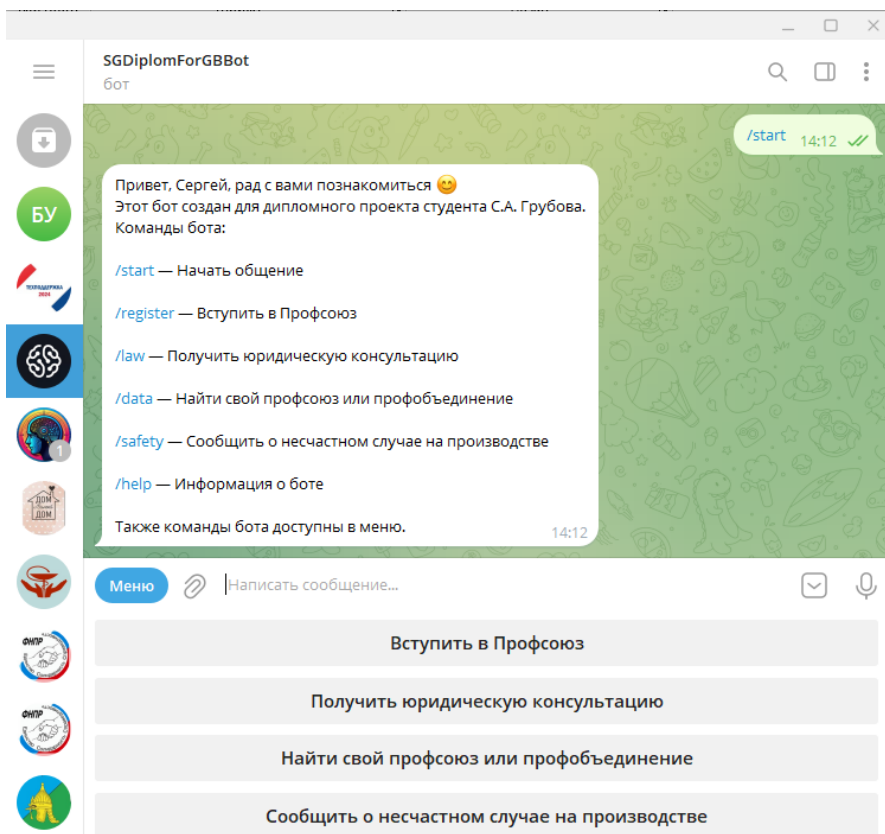


Рисунок 6. Взаимодействие с ботом. Кнопка «Вступить в профсоюз» или команда «/register».

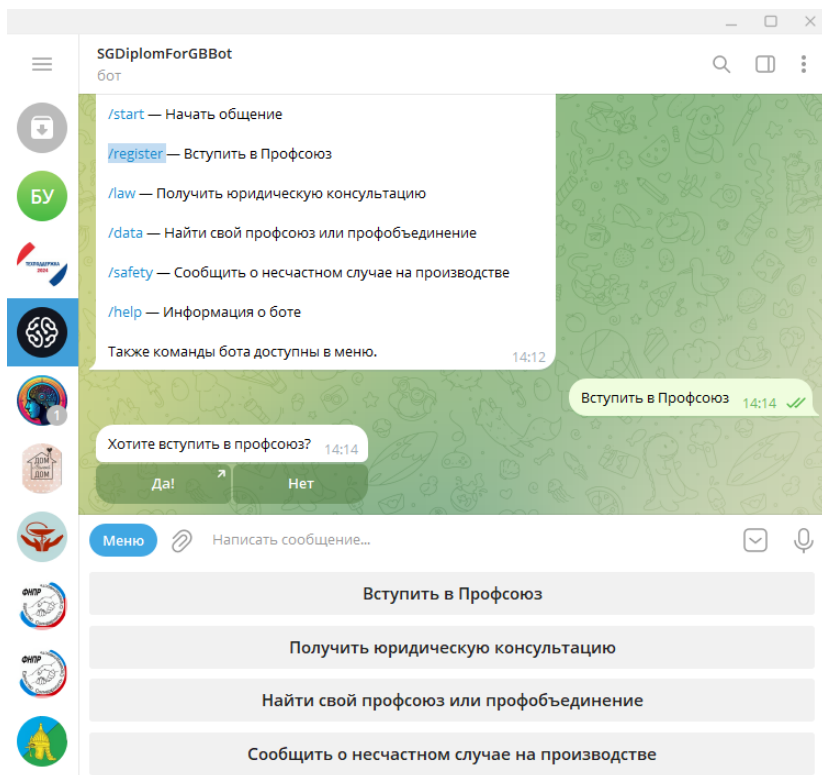


Рисунок 7. Кнопки с текстом, если выбрать «Да!».

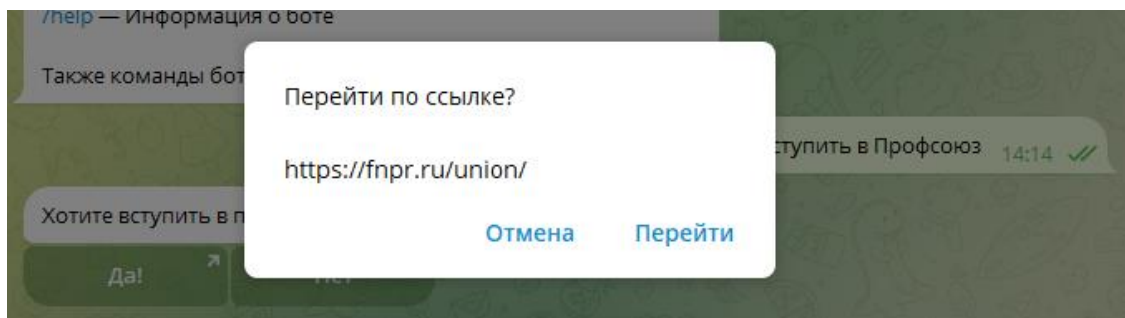


Рисунок 8. Кнопки с текстом, если выбрать «Нет».

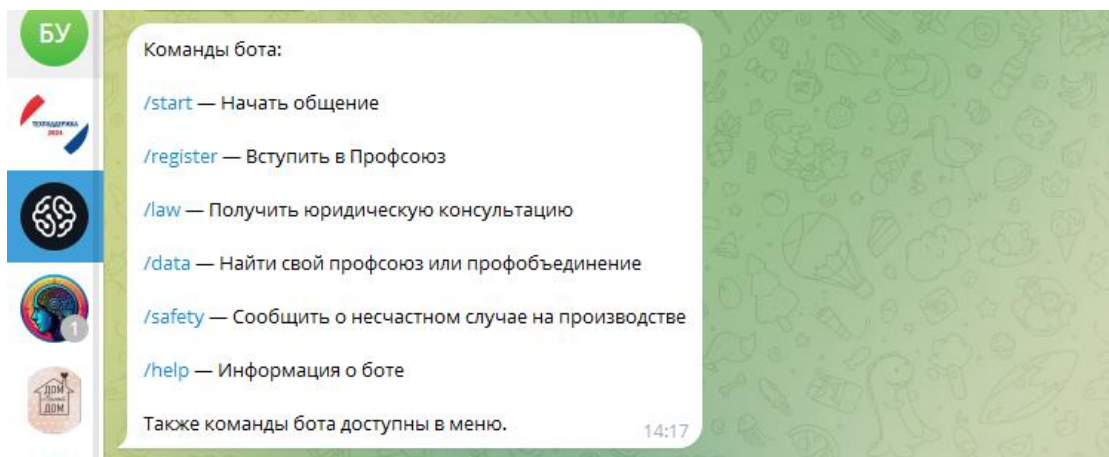
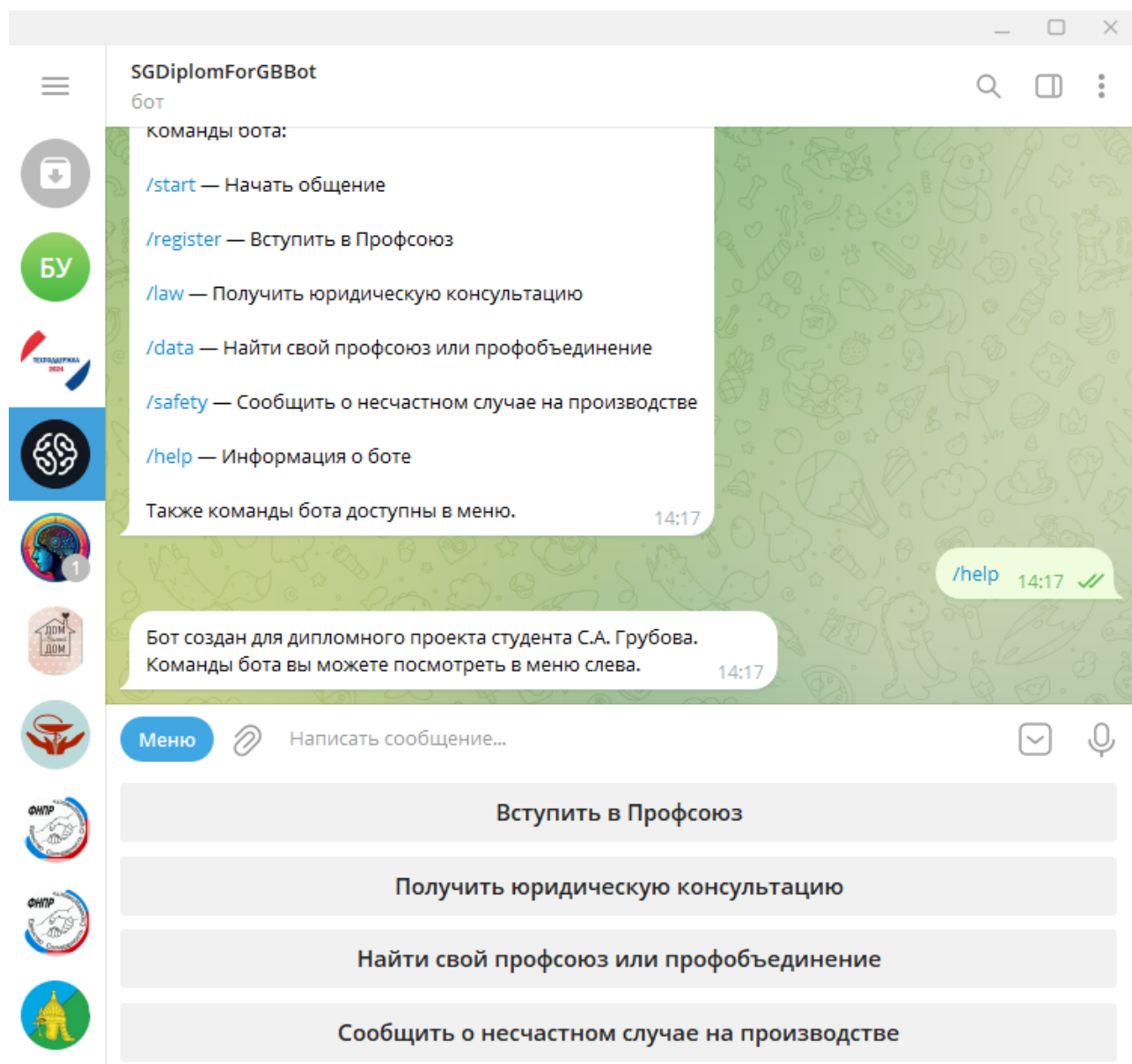


Рисунок 9. Команда «/help» отображает короткую информацию о боте и предлагает открыть меню.



# ЗАКЛЮЧЕНИЕ

В ходе проведения исследований и реализации проекта были изучены дополнительные материалы, необходимые для создания ботов. Пройденный курс обучения не охватывал такого большого объема знаний, однако он дал существенную базу знаний и практического опыта, которые позволят применять на практике реализацию сложных решений.

При изучении теоретической части проекта я столкнулся с устаревшими и неполными данными. Проект приходилось собирать по крупицам. Отдельно я бы хотел отметить инструменты, которые были использованы в ходе работы над дипломным проектом. «Spring Boot» очень мощный и не сложный фреймворк, работать с ним было одно удовольствие. Отдельно необходимо отметить значимость «Maven», этот инструмент мы рассматривали в ходе обучающего курса «GeekBrains», однако его незаменимость удалось ощутить при создании дипломного проекта. Самым сложным моментом было подключение базы данных, библиотек и зависимостей.

Цели и задачи проекта удалось полностью реализовать. В рамках тестирования были получены отзывы от коллег, которых я попросил проверить работу бота.

В перспективе определены планы по развертыванию бота на отдельном сервере моей организации и его доработки с юристами и специалистами делопроизводства. В адрес нашей организации поступает большое количество обращений от членов профсоюза, в большинстве случаев вопросы стандартные и имеют стандартные ответы. В работу с подобными обращениями можно подключить Telegram-бота.

В ближайшее время данный проект будет продемонстрирован специалистам моей организации для определения дальнейшего его развития. В ходе работы над дипломным проектом были определены некоторые векторы развития проекта. В частности для обработки запросов пользователей нужно создать сервис, в котором профильные специалисты смогут корректировать стандартные ответы

для пользователей и получать статистику по запросам. Также к боту можно подключить искусственный интеллект для обработки информации и помощи пользователям бота по отдельным категориям вопросов, которые не несут юридической ответственности, и в большинстве своем будут носить справочный характер.



## Список используемой литературы

1. Книга: Кей Хорстманн. «Java. Библиотека профессионала». Москва, Санкт-Петербург: ООО «Диалектика», 2019 год;
2. Книга: Крейг Уоллс. «Spring в действии». Москва: ДМК Пресс, 2022 год;
3. Книга: Бен Форта. «SQL за 10 минут». Москва, Санкт-Петербург: ООО «Диалектика», 2021 год;
4. Статья: «Building a Telegram bot using Spring Boot». Независимое онлайн-сообщество JCGs (Java Code Geeks). 2023 год.  
<https://examples.javacodegeeks.com/building-a-telegram-bot-using-spring-boot/>
5. Статья: «Фреймворк: как он упрощает работу и как им правильно пользоваться». Университет «Синергия». 2024 год.  
[https://synergy.ru/akademiya/programming/frejmwork\\_kak\\_on\\_uproshhaet\\_rabotu\\_i\\_kak\\_im\\_pravilno\\_polzovatsya](https://synergy.ru/akademiya/programming/frejmwork_kak_on_uproshhaet_rabotu_i_kak_im_pravilno_polzovatsya)
6. Статья: «Фреймворк». Энциклопедия «Wikipedia». 2023 год.  
<https://ru.wikipedia.org/wiki/Фреймворк>;
7. Статья: Антон Яценко. «Запускаем первое веб-приложение на Spring Boot». Skillbox. 2023 год. <https://skillbox.ru/media/code/zapuskaem-pervoe-vebprilozhenie-na-spring-boot/>