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

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система підтримки прийняття рішень молодшого командного складу з використанням технології чат ботів»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Петров С.О.**

Студента групи ІН – 72 — **Дерев'янчук В.А.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-72 спеціальності "Комп'ютерні науки" денної форми навчання Дерев'янчука Владислава Андрійовича.

Тема: " Інформаційна система підтримки прийняття рішень молодшого командного складу з використанням технології чат ботів"

Затверджена наказом по СумДУ
№_____ від _____2021 р.

Зміст пояснювальної записки: 1) огляд технологій для розробки Telegram ботів 2) постановка завдання 3) опис основних підходів і методів створення веб-сервісів і ботів 4) розробка документації у вигляді діаграм і програмної частини інформаційної системи; 5) аналіз результатів розробки і роботи сервісу.

Дата видачі завдання "_____" _____2021 р.
Керівник випускної роботи _____ Петров С.О.
Завдання прийняв до виконання _____ Дерев'янчук В.А.

РЕФЕРАТ

Записка: 56 стор., 19 рис., 2 табл., 2 додатки, 13 джерела.

Об’єкт дослідження — Telegram боти у навчальному процесі і підготовці фахівців у військовій справі.

Мета роботи — розробка інформаційної системи, що складається з бота – сервісу і адміністративного інтерфейсу керування контентом робота.

Методи дослідження — метод розробки телеграм ботів, як комплексного сервісу, з використанням фреймворку Spring Boot.

Результати — Розроблена інформаційна система, що включає в себе Telegram бота і адміністративну панель. Розроблені схеми та діаграми для опису процесів і структур. Проведена перевірка працездатності застосунку. Бот побудований на основі фреймворку Spring Boot. Він дозволяє навчатися студентам та військовим дистанційно і в будь-якому місці та у будь-який час. Панель адміністратора, дозволяє редагувати існуючі та створювати нові теми для навчання, вона побудована за допомогою платформи Angular.

ЗМІСТ

ВСТУП

Соціальні мережі, месенджери, чати, різні застосунки для спілкування і отримання інформації стали невідємною частиною нашого повсякденного життя. Согодні дуже важко уявити людину, яка б не мала хочаб одного встановленого месенджеру або соціальної мережі на своєму телефоні, планшеті, ноутбуці або стаціонарному комп'ютері.

В наш час великою популярність користуються різні боти – програми які виконують роль співбесідника у чатах та месенджерах. Початковою їх метою була заміна реальної людини у переписці, але з розвитком технологій, можливості чат-ботів розширилися та набули більшого значення для людей. Зараз ці програми можуть надавати різну інформацію. Вони широко використовуються у веденні бізнесу, для приваблення нових клієнтів та інформування існуючих про різні акції і заходи компаній, для миттевих покупок товарів які пропонуються ботом. Роботи використовуются для отримання повсякденної інформації про погоду, новини у світі, квитки, людей, штрафи, стан програмного забезпечення на серверах та багато іншого. Також через них можна знаходити музику, відеоролики. Деякі плтформи навіть дозволяють створювати ігри в ботах. Тобто через функціонал ботів можна реалізувати майже будь-що, все залежить від вмінь, фантазії і цілей розробника. Майже кожен месенджер має свій функціонал для реалізації таких ботів. У кожного з них: WhatsApp, Viber, Telegram, WeChat, Line, Facebook Messenger та деяких інших є така можливість.

Оскільки більшість студентів, які навчаються в нашому університеті користуються саме месенджером Telegram, до того ж Bot Telegram API має велику функціональність та гнучкість, тому було обрано саме його для реалізації даного сервісу. Таким чином, метою данної роботи є Telegram Bot для студентів, які навчаються на кафедрі військової підготовки СумДУ, а також для всіх, хто вивчає військову справу. Ця програма має забезвечувати

студентів і курсантів, актуальною і зрозумілою інформацією на військові теми для швидкого, зручного і продуктивного навчання. Також має бути розроблений інтерфейс для управління темами та матеріалами які надаються талеграм-ботом до вивчення. Перевагою використання бота замість мобільниого застосунку полягає в тому, що немає необхідності встановлювати додаткове програмде забезпечення, яке може мати більше навантаження на пристрій.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд технологій для розробки ботів

При розробці боту необхідно обрати підхід його реалізації. Існує два основні підходи:

- Long polling
- Webhook

Вони мають різні переваги та недоліки, тому перед вибором підходу треба провести аналіз і вибрати оптимальний варіант. Webhook у веб-розробці це метод збільшення або розширення функціональності веб-сторінки або веб-застосунку за допомогою користувацьких зворотних викликів. Ці зворотні виклики можуть обслуговуватися або керуватися користувачами або веб-розробниками. Зазвичай виклик відбувається на певній події, після чого сервіс, який обслуговує їх, надсилає HTTP-запит певного формату на вказану користувачем адресу[4]. Таким чином веб-сервіс, а саме телеграм бот отримує повідомлення від серверу месенджера Telegram, оброблює запит і надсилає відповідь. Webhook є складнішим у реалізації, але має велику перевагу у тому, що навантажує сервер менше ніж інший підхід. За технологію Long polling бот звертається до сервера Telegram через короткі інтервали часу для перевірки, чи є нові повідомлення від користувачів і якщо такі запити є то сервер надсилає їх боту і вже потім бот оброблює їх.

1.2 Telegram Bot

Telegram Bot – це програма, яка автоматично, за командою або за розкладом, відповідно до заданого параметру виконує певні дії. В мессенджері Telegram чат-боти – це спеціальні аккаунти. Вони отримують від людей текстові команди оброблюють та виконують їх.

Ботів можна умовно поділити на різні види за своєю функціональністю.

- Види ботів у Telegram:

- Чат-боти
- Боти-інформатори
- Ігрові боти
- Боти-асистенти

Алгоритм роботи роботів досить простий. Повідомлення, команди і запити**,** надіслані користувачами, передаються на програмне забезпечення, запущене на серверах розробників. Посередницький анонімний сервер Telegram обробляє шифрування і здійснює зворотний зв'язок між утилітою і користувачем. Взаємодія виглядає наступним чином:

- Користувач бота віддає йому команду.
- Месенджер передає команду на сервер Telegram.
- Сервер Telegram надсилає повідомлення боту.
- Бот-сервіс оброблює запит та відповідає.
- Користувач отримує повідомлення.

Нові боти створюються за допомогою спеціальної утиліти @BotFather, що керує важливими налаштуваннями всхі ботів. Через головного бота можна:

- налаштувати уні**к**альний нікнейм
- налаштувати ім’я
- створити список доступних команд
- створити вітальне меню
- оновити токен доступу
- додати опис бота

Налаштування, які виконуються через головного бота залежать від типу бота та в яких цілях він використовується.

1.3 Telegram Bot API

Bot API – це інтерфейс на основі HTTP протоколу, створений для розробників, які прагнуть створювати ботів для месенджеру Telegram[5]. Існує багато реалізацій цього інтерфейсу для різних мов програмування, більш того

для однієї мови можна знайти більше ніж одне втілення. Прикладом такої мови може стати Java. Однією з чудових реалізацій Bot API є TelegramBots-Spring-Boot-Starter. У ній підтримується, як технологія Long polling, так і Webhook. Ця реалізація включає в себе відразу готовий до запуску веб-сервіс, що може спростити та прискорити розробку бота. Spring Boot – це фреймворк на основі мови програмування Java, який використовується для створення самостійних та готових до роботи пружинних мікросервісів.

1.4 Аналіз фреймворків для розробки ботів та веб-сервісів

Існує достатньо велика кількість фреймворків для розробки веб-сервісів, але одним з найпопулярніших для мови програмування Java є Spring Framework. Так як є реалізація інтерфейсу Telegram Bot API за допомогою TelegramBots-Spring-Boot-Starter то доцільно використовувати саме цей фреймворк. Більш того, ця реалізація має багато інших функціональностей які можуть бути використані у майбутньому для розширення та удосконалення програми, що також є дуже важливим фактором. Є можливість налаштувати відслідковування стану сервісу та запитів до нього. Також присутня підтримка технології об’єктно орієнтованого відображення та JPA інтерфейсу.

Конфігурація та налаштування змінних веб-сервісу може відбуватися дуже зручним способом, використовуючи файл конфігурації, який може мати розширення yml або properties. Також є можливість створювати декілька таких файлів для різних середовищ запуску. Наприклад, можна створити файли ініціалізації для розробки, тестування, виробництва і для будь-яких інших цілей в залежності від потреб.

1.5 Підходи побудови адміністративних клієнтів

Підходи до побудови клієнтських інтерфейсів існують різні. Можна користуватися вже готовими рішеннями або розроблювати все самостійно

підбираючи різні налаштування та технології. Також, треба звертати увагу на швидкодію інтерфейсу, яка залежить від різних чинників. Ними можуть бути:

- розміри файлів сайту
- кількість та розмір зображень
- кількість HTTP запитів
- архітектура на якій побудований інтерфейс

Чим менше розміри файлів, зображень і менша кількість запитів через мережу, тим швидше буде працювати сайт.

Архітектура сайту може також різнитися. Можна створити багатосторінковий сайт, який буде складатися з великої кількості сторінок, але такий підхід не оптимальний, особливо якщо інтерфейс великий та має багато функціональностей. Інший спосіб розробки це створити односторінковий інтерфейс. Його особливість полягає у тому, що сайт складається з однієї сторінки, а весь контент змінюється відносно дій користувача без перезавантажень. Користування таким інтерфейсом, нагадує користування настільною програмою. Користувач не помічає ніяких перезавантажень та переходів на інші сторінки, бо їх і не має і всі дії відбуваються синхронно без додаткових запитів. Використовуючи підхід SPA(single-page application) не обов’язково завантажувати весь функціонал у браузер користувача. Завантаження певних модулів може відбуватися за необхідності, що також покращує продуктивність інтерфейсу. Перевагами односторінкового інтерфейсу є:

- гнучкість
- масштабованість
- швидкість завантаження
- швидкість взаємодії
- налаштовування завантаження певних модулів
- багато технологій для реалізації

SPA краще адаптоване під мультиплатформеність: такий веб-додаток відмінно показує себе на будь-яких пристроях і браузерах[13].

1.6 Аналіз підходів зберігання даних

Для зберігання даних можна використовувати різні підходи. Дані можна розділяти та зберігати у різних місцях. Можна використовувати реляційні чи документо-орієнтовані системи керування базами даних для групування, збереження і використання інформації. Інколи, можна просто зберігати дані на сервері у файлах. Такий підхід також можна застосовувати та поєднувати з базами даних, коли інформація зі сховища містить дані про файли на сервері.

Реляційна база даних – це набір даних з певними зв'язками між ними. Дані організовуються у набори таблиць, що складаються із стовпців і рядків[6]. Таблиці містять інформацію про об'єкти, наявних у сховищі.

Реалізації популярних реляційних сховищ:

- MySQL
- Oracle Database
- MS SQL
- PostgreSQL
- IBM Db2

Документо-орієнтована база даних – це спеціально призначена для зберігання ієрархічних структур даних і зазвичай реалізована за допомогою підходу NoSQL[7].

Реалізації популярних документо-орієнтована сховищ:

- Couchbase
- Redis
- MongoDB
- IBM Cloudant
- Amazon DynamoDB

Обирати базу даних треба ретельно, так як від її можливостей буде залежити не лише швидкодія, а й масштабованість, зручність коритсування, надійність та багато інших факторів.

Переваги підходу SQL:

- суворі правила проєктування
- єдиний стандарт написання запитів
- повна незалежність даних
- простота і доступність для розуміння користувачем
- цілісність даних
- структурованість

Переваги підходу NoSQL:

- можливість зберігання великих обсягів неструктурованої інформації
- краще піддаються масштабуванню
- швидка розробка
- не потребують глибоких знань SQL-запитів
- швидкість обробки даних

NoSQL сховища показують себе з дуже хорошого боку в симбіозі з реляційними базами даних. Наприклад, в системах, де основний обсяг інформації зберігає SQL, а за кеш відповідає NoSQL[8].

1.7 Аналіз програмних продуктів – аналогів

Після проведення пошуку та аналізу конкурентів, виявлено, що подібних ботів майже не існує або їх функціонал не працює. Тому створення програмного забезпечення, а саме ботів для навчання є дуже актуальною темою, як для Сумського державного університету так і для інших навчальних закладів.

1.8 Постановка задачі

Розробити Telegram бота для навчання студентів та військових. Програма має надавати інформацію щодо обраної теми. Інформація повинна

включати тексти, картинки, файли та інші ресурси. Має бути реалізований функціонал завантаження файлів, навігації між темами та підтемами. Користувач повинен мати змогу отримати довідку, яка допоможе йому у користуванні ботом. Для адміністрування контенту бота повинна бути розроблена адміністративна панель, яка надаватиме змогу змінювати склад тем, які надаються ботом для вивчення. Для більш якісної розробки необхідно створити документацію у вигляді діаграм, які описуватимуть можливості та функціональності застосунку.

2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1 Вибір сховища даних

Застосунок має зберігати дані про користувачів, теми і необхідні матеріали. Тому є необхідність структурованого збереження даних. Після аналізу баз даних було обрано реляційну базу MySQL. Вона являє собою одну з найбільш поширених на сьогодні систем управління базами даних в мережі Інтернет. Дана система використовується для роботи з досить великими обсягами інформації, однак MySQL ідеально підходить як для невеликих, так і для великих інтернет-проектів. Важливою характеристикою системи є її безкоштовність[9]. Для того щоб спростити процес розробки і зменшити кількість помилок, будемо використовувати технологію ORM. Вона дозволяє зменшити витрати на побудову скрипту генерації бази даних та формувати її базуючись на моделі класу, що використовується безпосередньо в коді прогрими.

2.2 Вибір засобів програмної реалізації

Реалізацію бота можна зробити на різних мовах програмування, але в даному випадку ще необхідно використовувати сховище даних. Оскільки, бот – це веб-сервіс, який використовує інтернет-мережу для обміну даними з сервером месенджеру, то було обрано мову прогрмамування Java, яка підтримує створення сервісів. Для більш ефективної роботи бота було вирішено використовувати фреймворк Spring Boot, що дозволяє дуже просто створювати сервіси і налаштовувати їх. Використання бази даних обумовило використання бібліотеки Hibernate, яка являє собою реалізацію специфікації JPA, яка в свою чергу реалізує концецію ORM. За допомого цієї бібліотеки буде здійснюватися формування таблиць у базі даних та взаємодія з ними.

Для розробки веб сервісу будемо використовувати плагін lombock, який дозволяє позбавитися від рутинних елементів та покрашить читабельність коду.

2.3 Вибір середовищ розробки

Програма містить багато частин, тому доцільним рішенням є використання середовища розробки для мови Java. Одним з найкращих варіантів є IntelliJ IDEA, бо це програмне забезпечення є безкоштовним і включає в себе багато функцій, через які дуже зручно налаштовувати середовище та сам проєкт.

Для того щоб позбавитися рутинної роботи, такої як запуск скриптів та ручнуе додавання бібліотек можна використати автоматичну збірку проєтку. Існує дві популярні системи автоматичної збірки для Java прєктів:

- Maven
- Gradle

Apache Maven – це засіб автоматизації збірки, який спочатку використовувався для Java проєктів. XML-файл описує проєкт, його зв'язки з зовнішніми модулями і компонентами, порядок будування, папки та необхідні плагіни. Сервер із додатковими модулями та додатковими бібліотеками розміщується на серверах[12].

Gradle – це система автоматичного збирання, яка далі розвиває принципи, закладені в Apache Ant та Apache Maven і використовує предметно-орієнтовану мову DSL на основі мови Groovy замість традиційної XML-подібної форми представлення конфігурації проекту. Для визначення порядку виконання завдань Gradle використовує орієнтований ациклічний граф.

Оцінюючи описані два варіанти було обрано використовувати Gradle, так як він має більш зрозулімий формат та його підтримка буде гарантуватися сервером на якому планується розмістити застосунок.

Для реалізації клієнтського адміністративного інтерфейсу було обрано середовище розробки Visual Studio Code. Цей редактор вихідних кодів, створений Microsoft для Windows, Linux та macOS. Особливості включають підтримку налагодження, підсвічування синтаксису, інтелектуальне заповнення коду, фрагменти, рефакторинг коду та вбудований Git, що є необхідним функціоналом при прозробці веб-додатку.

2.4 Вибір підходу до розробки адміністративного клієнта

Для розробки клієнтської панелі адміністратора за технологією JSP можна використати одну з наступних технолгій:

- Angular
- React
- Vue.js

Всі наведені вишче технології мають свої переваги та недоліки. Найпотужніший з наведених є Angular, бо він перевірений часом і має багато вже готових до використання функціональностей. Ця платформа використовує TypeScript, що також є перевагою, бо це спрощує читабельність коду та зменшує кількість помилок при розробці. TypeScript – це мова програмування, яка розширює можливості мови JavaScript. Для виконання TypeScript коду його необхідно перетворити в JavaScript, а вже потім перетворувати в машинну мову. Для перетворення однії мови програмування в іншу необхідно використовувати бібліотеку Babel. Вона буде виконивати транспіляцію коду в необхідну версію JavaScript, яка буде підтримуватися всіма браузерами. Як відомо JavaScript початково призначений для виконання JS коду в браузері. Тому необхідно встановити середовище в якаму можна буде розроблювати застосунок локально. Таким середовищем є Node.js. Воно призначине саме для виконання JavaScript коду на комп’ютері, а не в браузері. Завдяки цому середовишу з’явилася можливість розроблювати сервіси на мові JavaScript.

Для використання різних технологій та інтеграцію їх у Angular-застосунок необхідно використовувати пакетний менеджер, через який здійснюється контроль різних бібліотек і пакетів, що необхідні для розробки. Найпопулярніші пакетні менеджери це Yarn і npm. Хоч npm набув великої популярності і має багато позитивних відгуків від розробників, але він поступається швидкодії Yarn, тому, будемо користуватися саме цим пакетним менеджером. Для проєкту він створює спеціальний файл в якому формуються відомості про всі бібліотеки та їх версії. Це робится для того щоб забезпечити стійкість застосунку в користуванні саме тою версією бібліотеки, яка використовувалась при розробці, бо інколи функціонал бібліотек змінюється і використання найновіших версій може призвести до помилок при збиранні застосунку.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Проєктування інформаційної системи

Перед розробкою будь-якої інформаційної системи необхідно її спроектувати. Одни з гарних підходів проектування є опис проєкту за допомогою діаграм. Також, створені діаграми використовуются як документація до застосунку, бо такий підхід надає достатній об’єм інформації для розуміння проєкту і майже кожен фахівць у галузі інформаційних технологій може їх зрозуміти, і швидко продовжити роботу.

3.2 Архітектура проєкту

При розробці веб-сервісів існує декілька варіантів реалізації архітектур. Наприклад, можна використати мікросервісну або монолітну. Так як цей застосунок складається з невеликої частини елементів і не створює значного навантаження на сервер можна використати монолітний підхід, але замість того щоб розміщувати всі компоненти на одному сервері, деякі з них будуть відділені від основного сервісу для забезпечення більш стабільного та надійного функціонування(Рисунок 3.2.1). Таким чином, сховище даних, паналь адміністратора і Telegram бот отримають окремі сервери. Альтернативним підходом є розміщення елементів у контейнерах Docker.

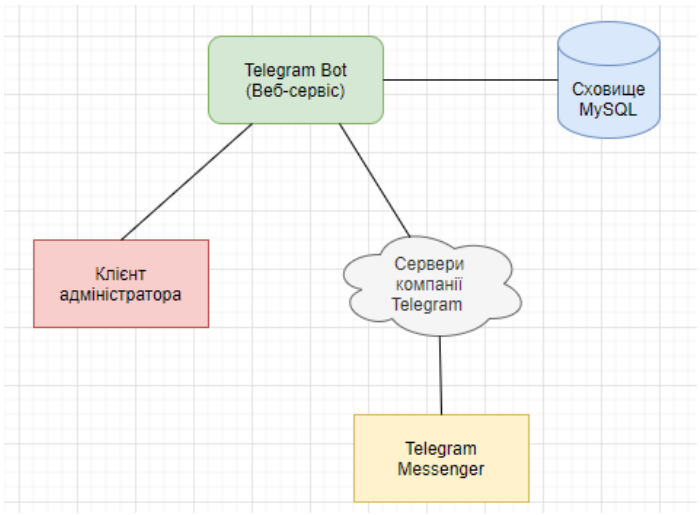

Рисунок 3.2.1– Архітектура проєкту

3.3 Варіанти користування сервісом

Діаграма варіантів використання відображає відносини, що існують між акторами і варіантами використання застосунку(Рисунок 3.3.1). Основне завдання діаграми – це дати можливість замовнику, кінцевому користувачеві і розробнику спільно обговорювати функціональність і поведінку системи. У нашому випвдку є три основні актори. Telegram користувач і клієнт – це зовнішні сутності, а саме люди які взаємодіють з сервісом. Внутрішньою сутністю є веб-сервер, який відповідає на запити користувачів, забезпечуючи їх необхідними даними.

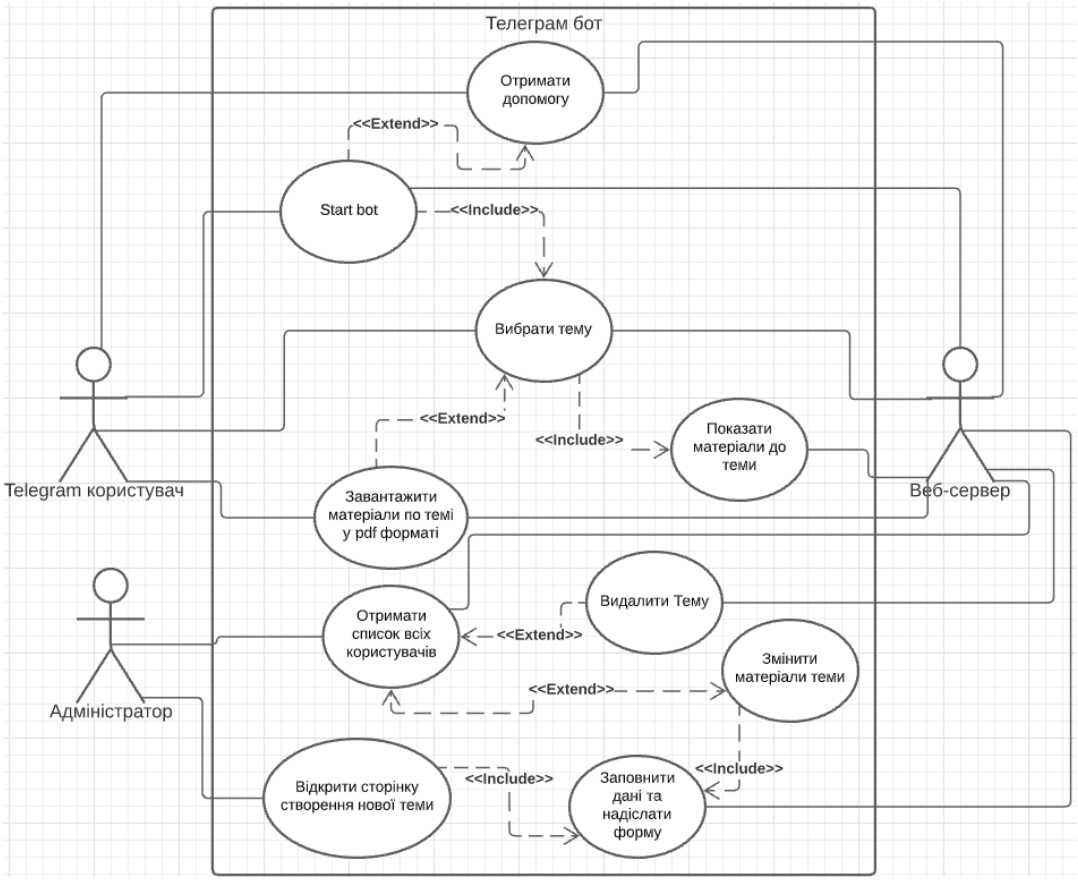

Рисунок 3.3.1– Діаграма варіантів використання

3.4 Зв’язки між сутностями бази даних

Для цього застосунку необхідна реляційна база даних, тому було вирішено побудувати схему для її подальшої реалізації.

Схема сутність-зв'язок – це різновид блок-схеми, де показано, як різні сутності пов'язані між собою всередині системи. ER-діаграми найчастіше застосовуються для проектування і налагодження реляційних баз даних в сфері освіти, дослідження, розробки програмного забезпечення та інформаційних систем. ER-моделі покладаються на стандартний набір символів, включаючи прямокутники, ромби, овали і сполучні лінії, для відображення сутностей, їх атрибутів і зав’язків[11].

У цьому сервісі необхідно зберігати дані про користувачів, теми і їх матеріали(Рисунок 3.4.1), тому необхідно розробити наступні сутності:

- User
- Topic
- Video
- Image
- File

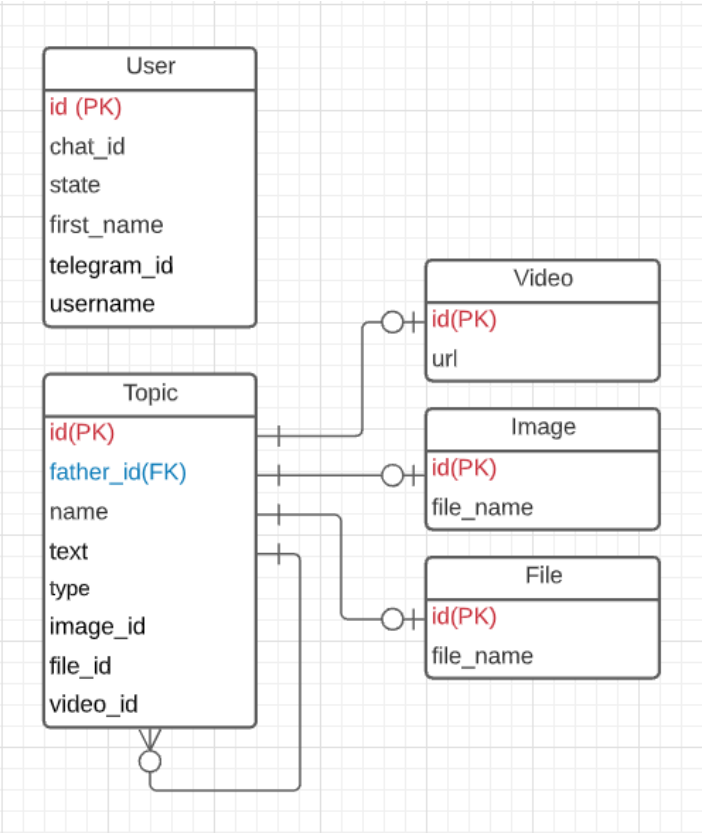

Рисунок 3.4.1– Діаграма сутність-зв’язок для сховища даних

3.5 Діаграма класів

Для кращого розуміння і швидкої реалізації основного функціоналу бота розроблюємо діаграму класів, яка значно прискорить і полегшить розробку програми.

Діаграма класів визначає типи класів системи і різного роду статичні зв'язки, які існують між ними. На діаграмах класів зображуються атрибути класів, операції класів та обмеження, які накладаються на зв'язки між класами.

Так як, було вирішено використовувати технологію об’єкто-реляційного відображення то створюємо структуру класів з якої будуть сформовані таблиці у базі даних(Рисунок 3.5.1). Кожен елемент моделі є дуже важливим, бо він матиме вплив на сховище.

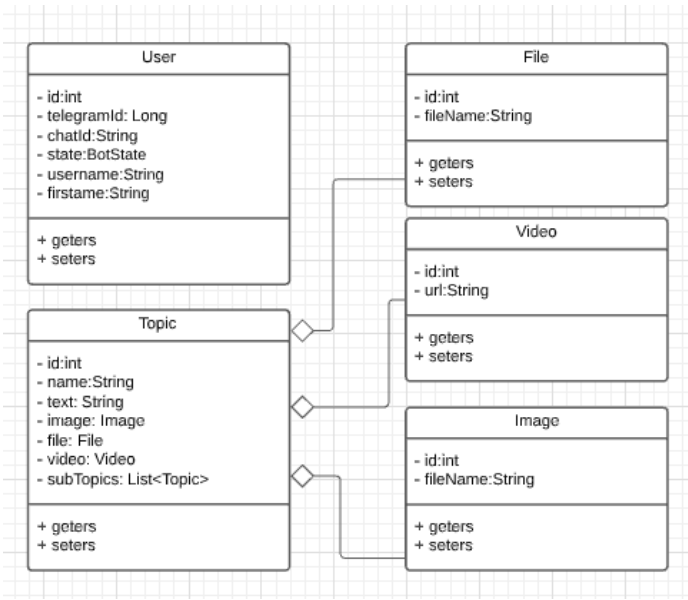

Рисунок 3.5.1– Діаграма класів моделей

Для панелі адміністратора необхідно надсилати та отримувати інформацію про моделі. Адміністратор може замінувати дані існуючих тем або створювати нові. Тому, необхідно створити спеціальні об’єкти транспортування інформації. Такий підхід необхідний для надсилання та приймання лише необхідної інформації, також він зменшує кількість даних для які передаються по мережі, що прискорює виконання сервісу. Для переведення стандартних елементів у DTO використовуються спеціальні

класи перетворювачі. Так як, сервіс передає лише дані про теми, то необхідні класи прийняття і надсилання інформації про теми, і клас який буде перетворювати дані у необхідну сутність(Рисунок 3.5.2).

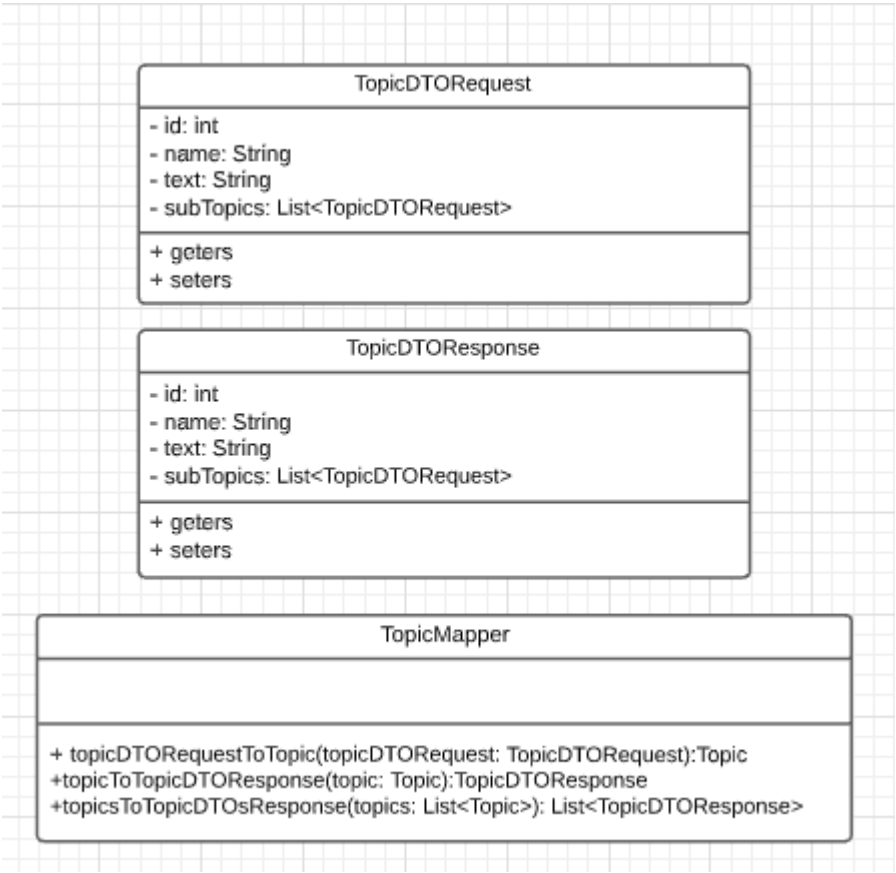

Рисунок 3.5.2 – Діаграма класів для DTO моделей

Для взаємодії сервісу з базою даних та адміністративною панеллю необхідні спеціальні класи. Сервіси відповідають за взаємодію з базою даних, тому необхідно створити для кожної сутності свій сервіс, якщо необхідно взаємодіяти безпосередньо з певною таблицею сховища. Так як, використовується інтерфейс JPA, який реалізований бібліотекою Hidernate, то будемо використовувати спеціальні інтерфейси – репозиторії які дозволяють взаємодіяти з базою без написання SQL запитів, а відразу використовувати готові методи(Рисунок 3.5.3).

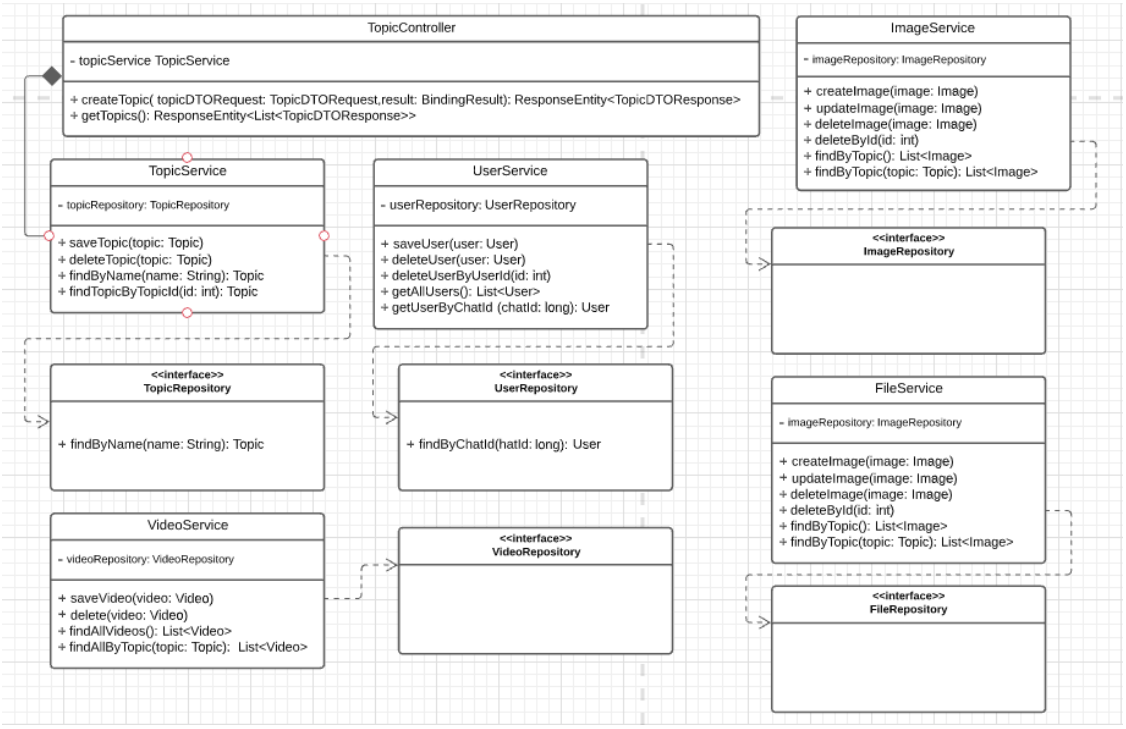

Рисунок 3.5.3 – Діаграма класів для сервісів і контролера

Головними класами даного сервісу є клас запуску Java-застосунку, який містить метод main і клас бота в якому описуються основні властивості та функціональності робота. SumduKVPBot – це і є клас бота, який імплементує інтерфейс TelegramLongPoollingBot, і отримує необхідні реалізації Telegram Bot API(Рисунок 3.5.4).

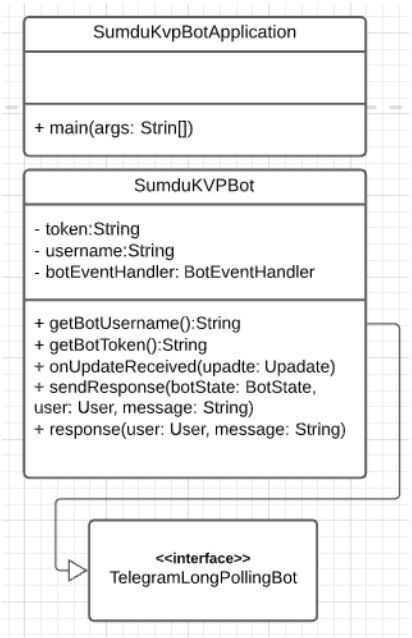

Рисунок 3.5.4 – Діаграма головних класів

3.6 Діаграми послідовностей

Діаграми послідовностей використовуються для уточнення діаграм прецедентів, більш детального опису логіки сценаріїв використання. Діаграми послідовностей зазвичай містять об'єкти, які взаємодіють в рамках сценарію, повідомлення, якими вони обмінюються, і які повертаються результати, пов'язані з повідомленнями.

При запиті тем з бота, користувачем Telegram месенджеру виконуються послідовні кроки. Коли користувач обирає тему і надсилає її, месенджер на телефоні або комп’ютері реагує на дії людині і надсилає повідомлення на сервери компанії Telegram, після обробки цього повідомлення, воно надсилається боту, сервіс оброблює надіслані дані зберігає дані користувача, за необхідності, знаходить необхідну тему у базі і надсилає її у відповідь на запит. Після чого, сервери компанії Telegram знову оброблюють повідомлення і надсилають його користувачу(Рисунок 3.6.1).

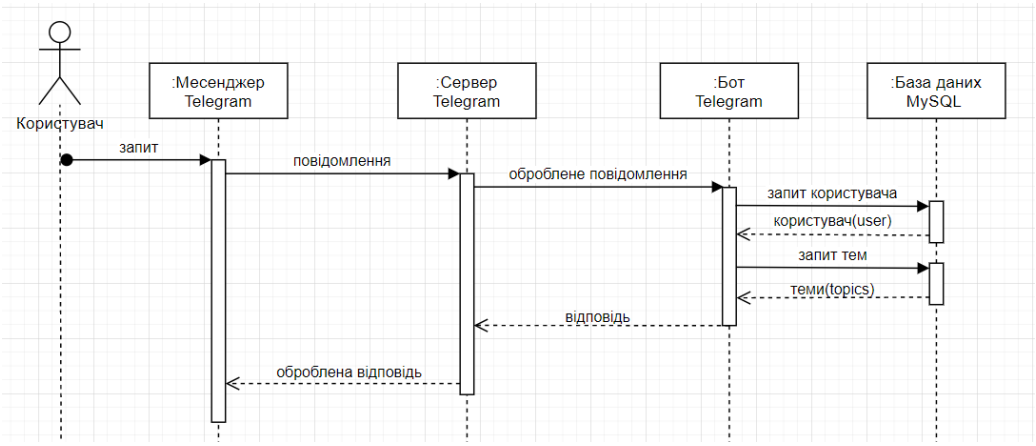

Рисунок 3.6.1 – Діаграма послідовностей для запиту тем користувачем бота

Для збереження нової теми адміністратор повинен заповнити форму та надіслати її. Після коректного заповнення, дані проходять підготовку на клієнті та надсилаються боту. Бот зі свого боку оброблює отримані дані і зберігає у сховище(Рисунок 3.6.2).

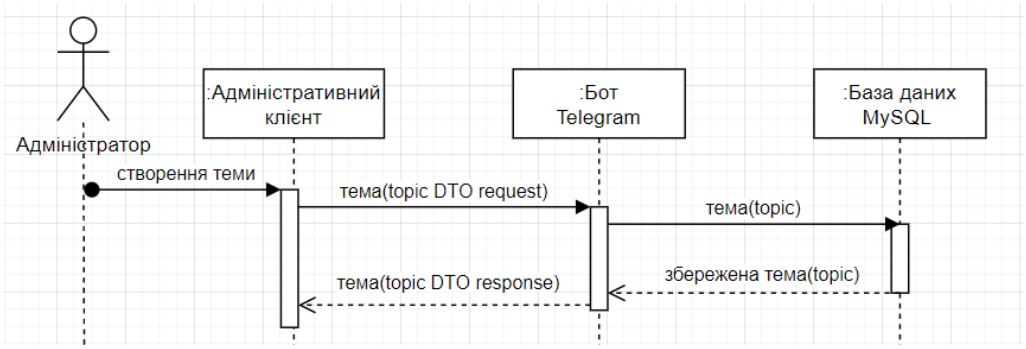

Рисунок 3.6.2 – Діаграма послідовностей для створення нової теми

Для видалення теми, адміністратор ініціює видалення теми, натискаючи на кнопку видалення. Після натискання відбувається запит до бота, який містить ідентифікатор теми. Бот приймає запит та робить запит на видалення теми до бази даних, якщо операція успішна сервер повертає статус успішного запиту(Рисунок 3.6.3).

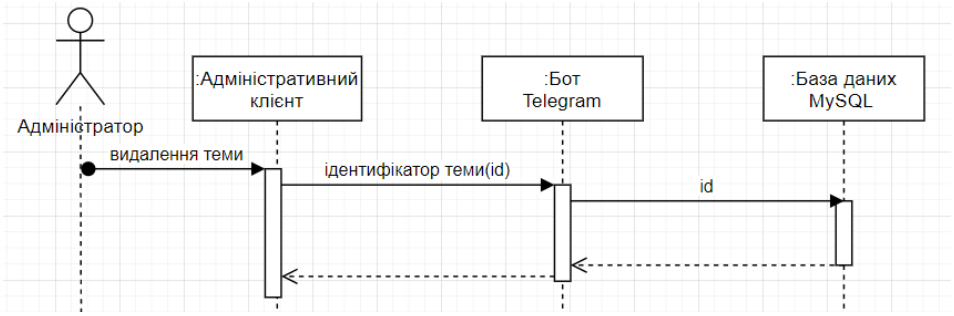

Рисунок 3.6.3 – Діаграма послідовностей для видалення теми

Для редагування існуючої теми, адміністратор повинен заповнити форму та надіслати її. Після коректного заповнення, дані проходять підготовку на клієнті та надсилаються боту. Бот, у свою чергу, оброблює отримані дані і оновлює їх для певної теми у сховищі, базуючись на отриманому ідентифікаторі теми(Рисунок 3.6.4).

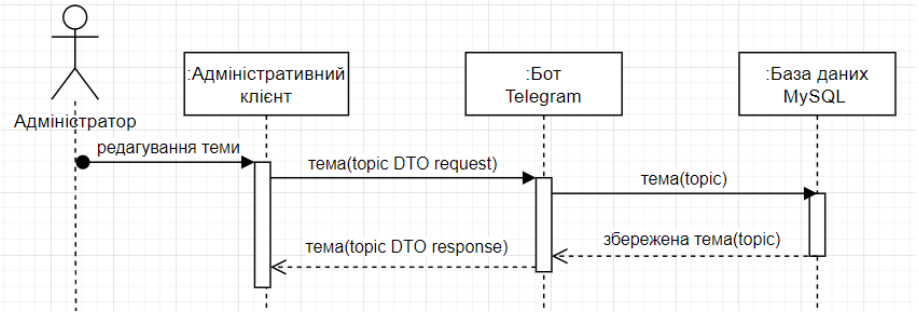

Рисунок 3.6.4 – Діаграма послідовностей для редагування теми

3.7 Панель Адміністратора

Панель адміністратора побудована за допомогою Angular платформи і містить сторінку перегляду(Рисунок 3.7.1) та редагування(Рисунок 3.7.2) тем.

Рисунок 3.7.1 – Сторінка відображення всіх тем.

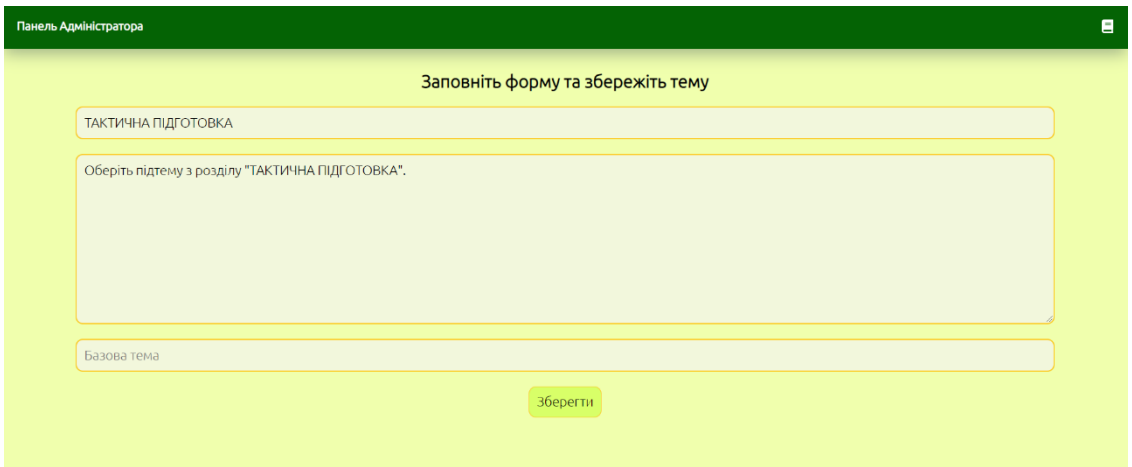

Рисунок 3.7.2 – Сторінка редагування і створення теми.

У цьому середовищі використовуються модулі, які поділяють елементи за їх логікою, а також компоненти – це елементи, які відповідають за створення контенту веб-сторінки.

Було розроблено три компоненти:

- app.component – головний компонент, який містить у собі інші.
- generate-topic.component – компонент створення і редагування тем.
- topic-browser.component – компонент відображення всіх існуючих тем.

Кожен компонент має три основні файли:

- html – для розмітки.
- scss – для стилізації.
- ts – для логіки.

Імена цих файлів відповідають імені компонента, але мають різне розширення.

Застосунок має два сервіси для виконання операцій обробки даних використовується topic-info.service сервіс, а для надсилання HTTP запитів topic.service.

Також, створено два модулі. Головним модулем є app.module, він містить головні налаштування всього застосунку. Другим модулем є app-routing.module, він відповідає за маршрутизацію компонент і відображення певної компоненти за певних умов. Застосунок містить більше елементів, які також дуже важливі для його роботи(Рисунок 3.7.3).

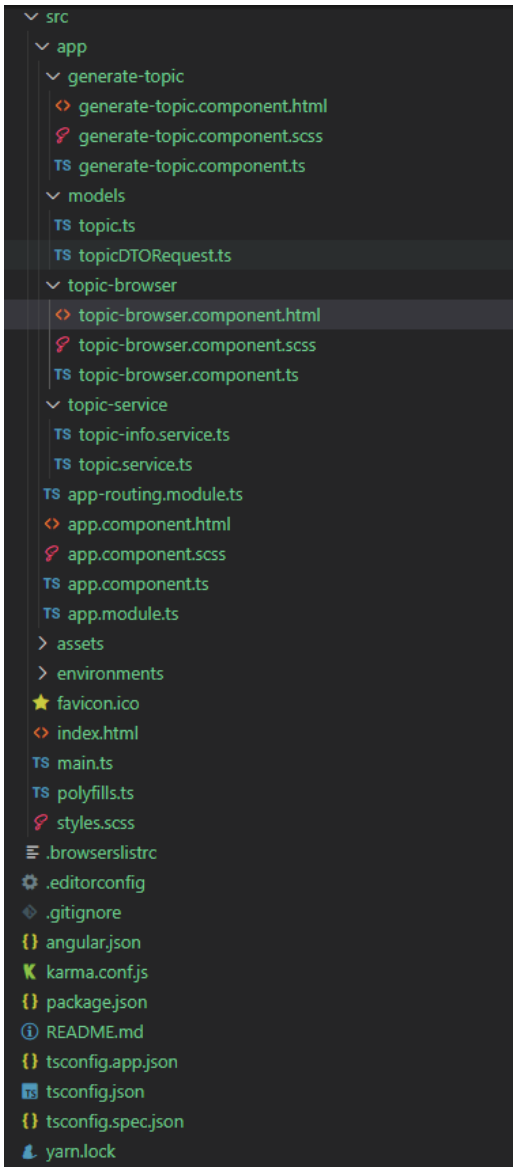

Рисунок 3.7.3 – Структура папок та файлів Angular застосунку

3.8 Автоматична збірка проєкту

Для автоматичної збірки проєкту використовується Gradle. Ця система дозволяє автоматично збирати проєкт, завантажуючи необхідні бібліотеки з репозиторію, виконувти операції генерації, копіювання, очищення та інші.

Розглянемо основні елемнти файлу build.gradle.

Плагіни, що використовуються у програмі розташовуються у блоці plugins.

```
    plugins {
id 'org.springframework.boot' version '2.4.4'
id 'io.spring.dependency-management' version '1.0.11.RELEASE'
```

```
    id 'java'
}
```

У блоці repositories вказується репозиторій з якого будуть завантажуватися бібліотеки вказані у блоці dependencies.

```
     repositories {
    mavenCentral()
}
```

У блоці dependencies розташовуються різні бібліотеки які, використовуються для розробки.

```
dependencies {
    implementation 'org.telegram:telegrambots-spring-boot-
starter:5.1.1'
    implementation 'org.springframework.boot:spring-boot-starter-
web'
    implementation 'org.springframework.boot:spring-boot-starter-
data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-
validation'

    implementation 'mysql:mysql-connector-java:8.0.23'

    implementation 'log4j:log4j:1.2.17'

    testImplementation 'org.springframework.boot:spring-boot-
starter-test'

    compileOnly 'org.projectlombok:lombok:1.18.20'
    annotationProcessor 'org.projectlombok:lombok:1.18.20'
}
```

3.9 Об’єктно реляційне відображення

Для об’єкто реляційного відображення і взаємодії з базою використовується бібліотека Hibernate.

Розглянемо на прикладі класу User, підхід моделювання табліці через модель. Якщо прибрати всі анотації з класу, то отримаємо звичайний Java клас. Тому основним підходом до об’єктро реляційного моделювання є анотації які і відповідають за те яка таблиця буде створеня. Також можна використовувати

XML файл для опису структури бази даних. Опишемо властивості Анотацій у таблиці(Таблиця 3.9.1).

Таблиця 3.9.1 – Опис анотацій класу для об’єктно-реляційного відображення

Анотація	Опис
@Entity	Позначає, що цей клас має використовуватись як сутність у базі даних
@Table	Позначає, що це таблиця, а параметр name вказує на ім’я таблиці в сховиші.
@Id	Вказує, що поле буде використовуватиь як первинний ключ
@GeneratedValue	Вказує на те, яким чином буде генеруватися значення.
@Column	Позначає, що поле це колонка, хоча всі поля автоматично стануть стовпчиками у таблиці. Параметр name відповідає за ім’я колонки у базі, unique за унікальність значення, nullable за можливість комірки приймати значення null.

```
@Data
@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", unique = true, nullable = false)
    private Long id;

    @Column(name = "telegram_id", unique = true, nullable =
```

```
false)
    private Long telegramId;

    @Column(name = "chat_id", unique = true, nullable = false)
    private Long chatId;

    @Column(name = "state", nullable = false)
    private String state;

    @Column(name = "username", unique = true)
    private String username;

    @Column(name = "first_name")
    private String firstName;
}
```

3.10 Опис основних елементів сервісу на Java

Вхідною точкою програми є функція main, яка розташована у класі SumduKvpBotApplication. Цей клас розташований на верхньому рівні відносно всіх інших класів, для того, щоб Spring Boot зміг знайти та налаштувати всі залежностіза технологією Dependency injection.

Метод corsConfigurer() відповідає за переналаштування біна, який відповідає за доступ до сервер з інших хостів.

```
      @SpringBootApplication
public class SumduKvpBotApplication {

    public static void main(String[] args) {
        SpringApplication.run(SumduKvpBotApplication.class,
args);
    }
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/**").allowedMethods("OPTIONS", "GET",
"POST", "PUT", "DELETE")
                    .allowedOrigins("http://localhost:4200");
            }
        };
    }

}
```

Розглянемо деякі елементи бота необхідні для його коректної роботи.

Клас SumduKVPBot наслідує TelegramLongPollingBot, який реалізує API ботів за технологією Long Polling. Важливими методами є getBotUsername, getBotToken, onUpdateReceived. Два перші повинні повертати правелні унікальні значення, які надаються при генерації бота через головного робота Telegram. Токен повинен зберігатися у секреті, для запобігання несанкціонованого доступу до бота. Тому конфігурація значень які повертаються з цих методів відбувається через конфігураційний файл або через змінні запуску. Метод onUpdateReceived відповідає за отримання нових повідомлень від користувачів.

```
public class SumduKVPBot extends TelegramLongPollingBot {

    private static final String DEFAULT_ERROR_RESPONSE = "Щось
пішло не так, спробуйте ще раз.";

    @Value("${bot.token}")
    private String token;

    @Value("${bot.username}")
    private String username;

    private BotEventHandler botEventHandler;

    @Autowired
    public void setBotEventHandler(BotEventHandler
botEventHandler) {
        this.botEventHandler = botEventHandler;
    }
    @Override
    public String getBotUsername() {
        return username;
    }

    @Override
    public String getBotToken() {
        return token;
    }

    @SneakyThrows
    @Override
    public void onUpdateReceived(Update update) {
        var message = update.getMessage();
```

```
        if (!botEventHandler.isRequestValid(message)) {
            execute(new
SendMessage(message.getChatId().toString(),
DEFAULT_ERROR_RESPONSE));
        }
        var user = botEventHandler.getUser(message);
        var botState =
botEventHandler.getBotState(message.getText());
        sendResponse(botState, user, message.getText());
    }
     }
```

Контролер для тем відповідає за зв'язок між інтерфейсом адміністратора та ботом. Тому розглянемо деякі елементи класу TopicController. Всі методи у цьому класі виконують функції обробників запитів. Опишемо властивості Анотацій у таблиці(Таблиця 3.10.1).

Таблиця 3.10.1 – Опис анотацій класу контролера

Анотація	Опис
@RestController	Вказує, що цей клас є REST контролером.
@RequestMapping("/topics")	Вказує, що запити зі шляхом "/topics" потрапляють у цей контролер.
@Autowired	Відповідає за впровадження залежностей, а саме отримання об’єктів, що виступають параметрами контролера.
@GetMapping, @PostMapping, @PutMapping, @DeleteMapping	Вказує який вид HTTP запитів може оброблювати метод класу.
@RequestBody	Відповідає за автоматичне перетворення тіла запиту в DTO об’єкт.
@Valid	Перевіряє валідність отриманих даних.

```
@RestController
@RequestMapping("/topics")
public class TopicController {

    private TopicService topicService;

    @Autowired
    public TopicController(TopicService topicService) {
        this.topicService = topicService;
    }

    @GetMapping
    public ResponseEntity<List<TopicDTOResponse>> getTopics() {
        return new ResponseEntity<>(topicService.findAll(),
HttpStatus.CREATED);
    }

    @PostMapping
    public ResponseEntity<TopicDTOResponse> createTopic(@Valid
@RequestBody TopicDTORequest topicDTORequest,

BindingResult result) throws Exception {
        if (result.hasErrors()) {
            return new ResponseEntity(result.getAllErrors(),
HttpStatus.BAD_REQUEST);
        }
        return new
ResponseEntity<>(topicService.create(topicDTORequest),
HttpStatus.CREATED);
    }

    @PutMapping("/{id}")
    public ResponseEntity<TopicDTOResponse>
updateTopic(@PathVariable("id") Integer id,
                                                    @Valid
@RequestBody TopicDTORequest topicDTORequest,

BindingResult result) throws Exception {
        if (result.hasErrors()) {
            return new ResponseEntity(result.getAllErrors(),
HttpStatus.BAD_REQUEST);
        }
        return new
ResponseEntity<>(topicService.update(topicDTORequest, id),
HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity deleteTopicById(@PathVariable("id")
Integer id) {
```

```
            topicService.deleteById(id);
            return new ResponseEntity(HttpStatus.OK);
        }
}
```

3.11 Приклад роботи бота

Для демонстрації роботи бота, зробимо декілька скріншотів його роботи.

Початковий стан бота і його привітання надають можливість користувачу ознайомитись з роботом. При введенні скісної риски з'являються команди бота(Рисунок 3.11.1).

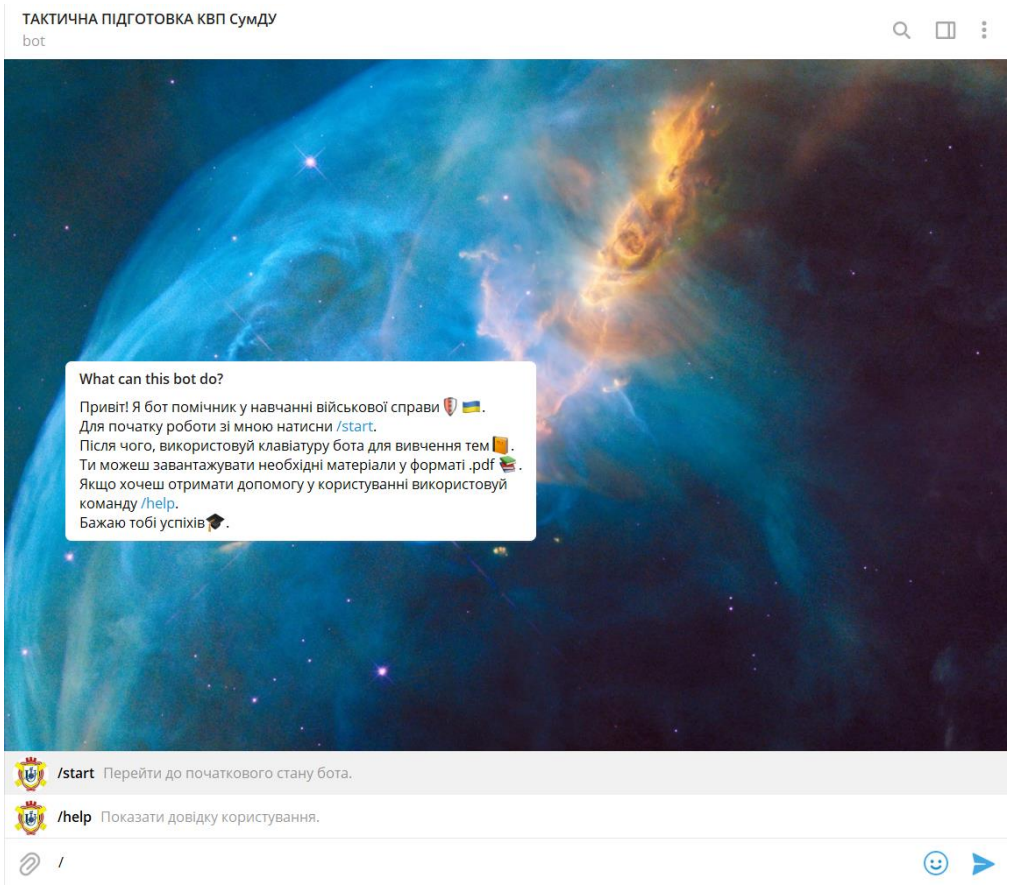

Рисунок 3.11.1 – Початковий стан бота

При навігації по темам можемо бачити матеріали, які бот надає для вивчення(Рисунок 3.11.2).

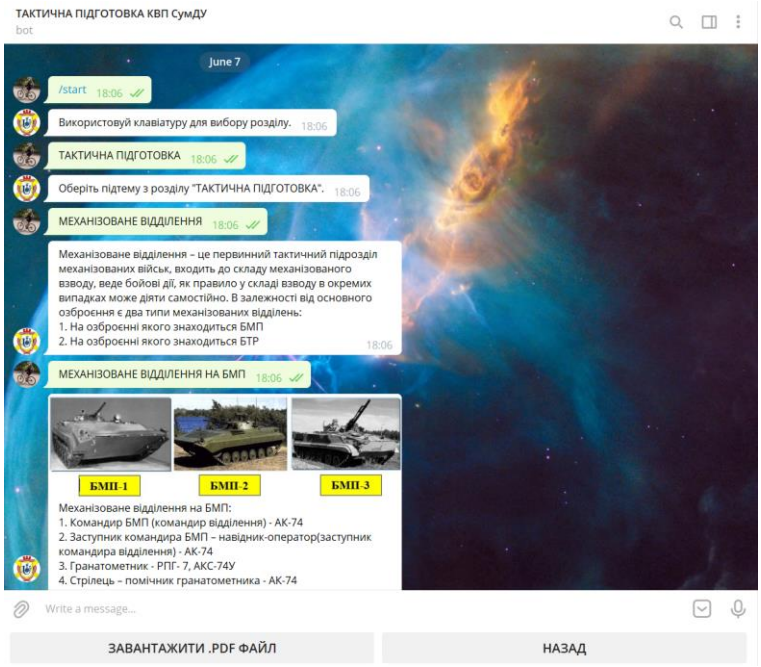

Рисунок 3.11.2 – Навігація по темах

Демонстрація завантаження pdf файлу. При натисканні на кнопку: «ЗАВАНТАЖИТИ .PDF ФАЙЛ», відбувається завантаження файлу відносно теми на якій у даний момент знаходиться користувач(Рисунок 3.11.3).

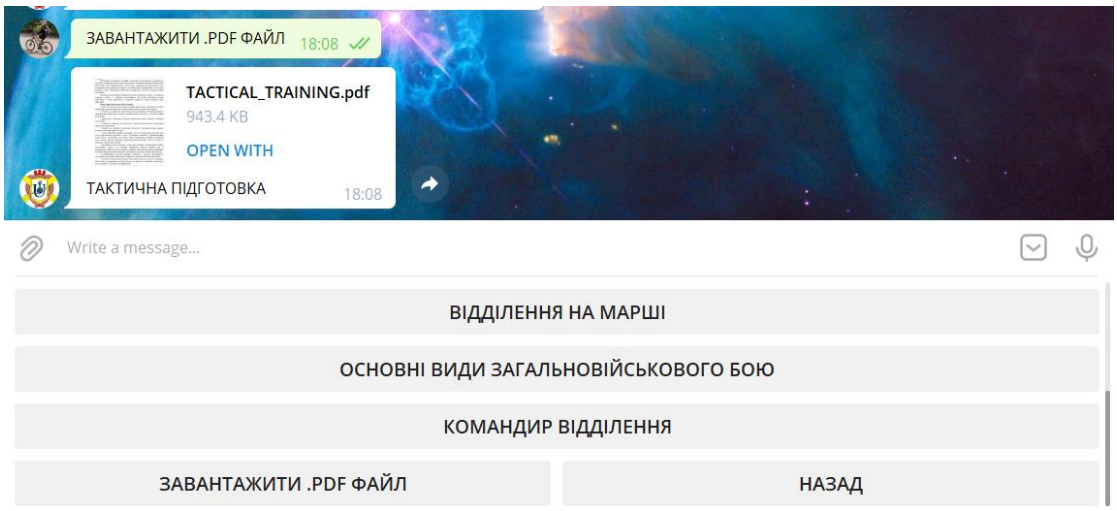

Рисунок 3.11.3 – Завантаження pdf файлу

При введенні команді /help з’являєтеся допоміжне повідомлення в якому описано як користуватися ботом та які в нього є функції(Рисунок 3.11.4).

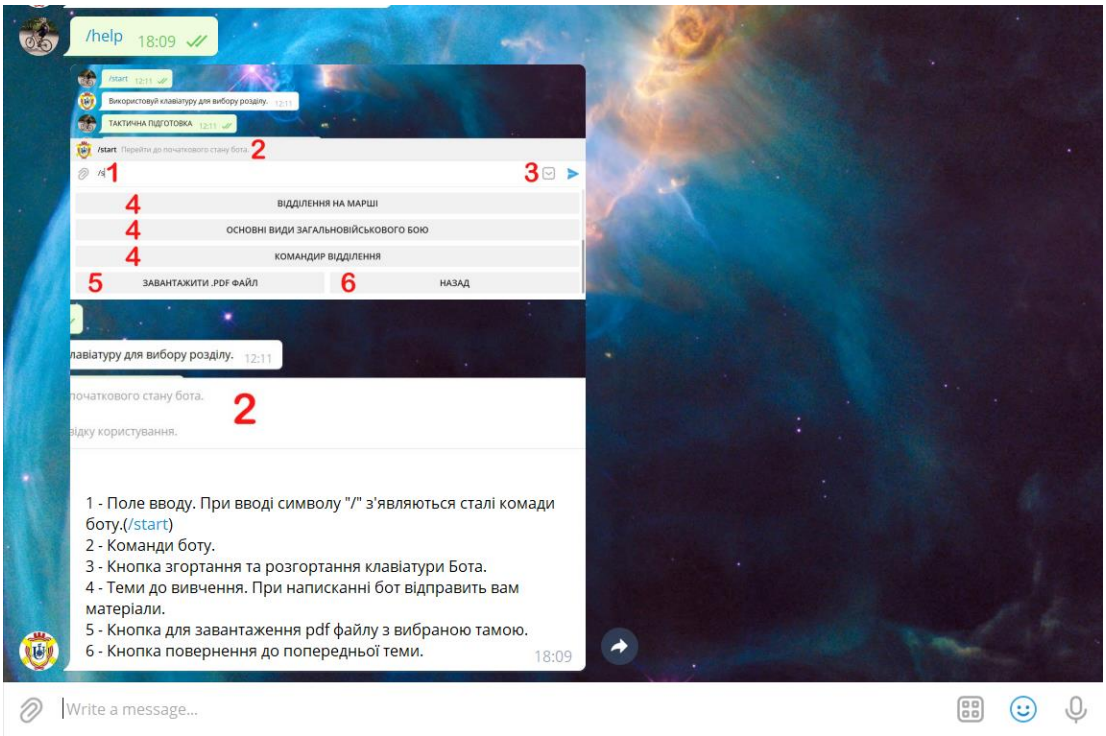

Рисунок 3.11.4 – Довідка користування ботом

Демонстрація бота на мобільному пристрої (Рисунок 3.11.5).

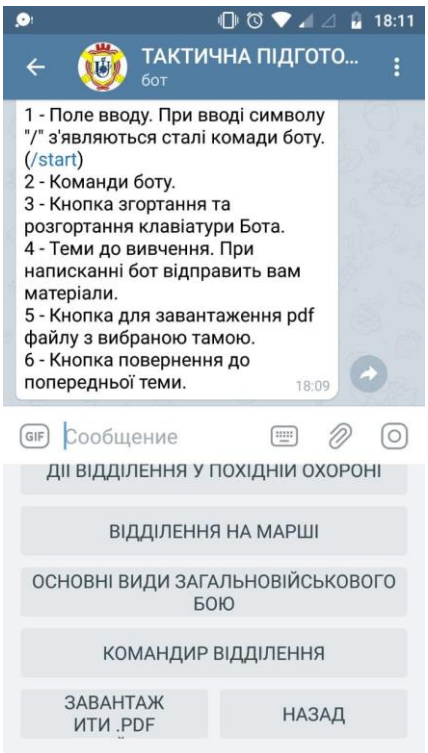

Рисунок 3.11.5 – Вигляд бота на мобільному пристрої

ВИСНОВОК

Результатом роботи є програма, а саме Telegram бот, побудований на основі фреймворку Spring Boot, що дозволяє навчатися студентам та військовим дистанційно і в бьдь-якому місці. Також було створено панель адміністратора, яка дозволяє редагувати існуючі та створювати нові теми для навчання.

При проектуванні системи було розроблено діаграми, які являють собою документацію до проєкту. Список створених діаграм:

- Діаграма архітектури проєкту
- Діаграма варіантів використання
- Діаграма класів
- Діаграма сутність-зв’язок
- Діаграма послідовності

Для розробленого веб-сервісу використовувались сучасні технологі, такі як:

- TelegramBots-Spring-Boot-Starter
- Spring Data JPA
- Hibernate
- Project Lombok
- MySQL
- Angular
- Node.js

В подальшому данимй сервіс може бути розширений та розгалуджений на менш зв’язаний підхід, який дозволить розділити функціональності по різним мікросервісам. Можна розділити логіку бота та адміністратора, який редактує і створює теми, також можна створити різні ролі для користувачів з ріним рівнем доступу до ресурсів.

СПИСОК ЛІТЕРАТУРИ

1. Блох, Джошуа Б70 Java: эффективное программирование, 3-е изд. : Пер. с англ. — СПб. : ООО “Диалектика”, 2019. — 464 с.: ил. — Парал. тит. англ.
2. Лонг Джош, Бастани Кеннет Л76 Java в облаке. Spring Boot, Spring Cloud, Cloud Foundry. – СПб.: Питер, 2019 – 624.: - (Серия «Бестсклллеры O`Reilly»).
3. Modrzyk, Nicolas Building Telegram Bots: Develop Bots in 12 Programming Languages using the Telegram Bot API, 2019
4. Webhook [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/Webhook
5. Telegram Bot API [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://core.telegram.org/bots/api/
6. Що таке SQL [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://naurok.com.ua/prezentaciya-scho-take-sql-190920.html
7. Документо-орієнтована система керування базами даних [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/Документо-орієнтована_система_керування_базами_даних
8. Сильные и слабые стороны NoSQL [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://habr.com/ru/sandbox/113232/
9. Базы данных MySQL: что такое и как с ними работать - Hostings.info [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://ru.hostings.info/schools/bazy-dannyh.html
10. Apache Maven [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/Apache_Maven

11. Что такое ER-диаграмма и как ее создать? [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.lucidchart.com/pages/ru/erd-диаграмма
12. Gradle [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/Gradle
13. Создание веб-приложений SPA и PWA [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://webevolution.ru/blog/sajti/sozdanie-web-prilozhenij-spa-pwa/

ДОДАТОК А. ПРОГРАМНИЙ КОД Telegram БОТУ

```
public class SumduKVPBot extends TelegramLongPollingBot {

    private static final String DEFAULT_ERROR_RESPONSE = "Щось
пішло не так, спробуйте ще раз.";

    @Value("${bot.token}")
    private String token;

    @Value("${bot.username}")
    private String username;

    private BotEventHandler botEventHandler;

    @Autowired
    public void setBotEventHandler(BotEventHandler
botEventHandler) {
        this.botEventHandler = botEventHandler;
    }

    @Override
    public String getBotUsername() {
        return username;
    }

    @Override
    public String getBotToken() {
        return token;
    }

    @SneakyThrows
    @Override
    public void onUpdateReceived(Update update) {
        var message = update.getMessage();
        if (!botEventHandler.isRequestValid(message)) {
            execute(new
SendMessage(message.getChatId().toString(),
DEFAULT_ERROR_RESPONSE));
        }
        var user = botEventHandler.getUser(message);
        var botState =
botEventHandler.getBotState(message.getText());
        sendResponse(botState, user, message.getText());
    }

    public void sendResponse(BotState botState, User user,
String message) throws TelegramApiException {
        switch (botState) {
            case ENTRY_POINT ->
```

```
execute(botEventHandler.getStartAnswer(user));
            case HELP_MENU ->
execute(botEventHandler.getHelp(user));
            case BACK -> response(user,
botEventHandler.stepBack(user.getState()));
            case DOWNLOAD_PDF ->
execute(botEventHandler.getFile(user));
            case SOME_TOPIC -> response(user, message);
        };
    }

    public void response(User user, String message) throws
TelegramApiException {
        var topic = botEventHandler.getTopicByMessage(message);
        if (topic == null) {
            execute(botEventHandler.getStartAnswer(user));
        } else {
            switch (topic.getType()) {
                case MESSAGE ->
execute(botEventHandler.getTopicsByFatherTopicAnswer(message,
user));
                case PHOTO ->
execute(botEventHandler.getTopicsByFatherTopicWithImageAnswer(me
ssage, user));
                case DOCUMENT ->
execute(botEventHandler.getTopicsByFatherTopicAnswer(message,
user));
                default -> execute(new
SendMessage(user.getChatId().toString(),
DEFAULT_ERROR_RESPONSE));
            };
        }
    }
}

public interface UserService {
    User save(User user);
    User update(User user);
    List<User> findAll();
    User findByUsername(String username);
    User findByTelegramId(Long telegramId);
}

public interface TopicService {
   TopicDTOResponse create(TopicDTORequest topicDTORequest)
throws Exception;
   TopicDTOResponse update(TopicDTORequest topicDTORequest,
Integer id) throws Exception;
   List<TopicDTOResponse> findAll();
   List<Topic> findByFatherId(Integer fatherId);
   List<Topic> findBasicTopics();
```

```
    Topic findTopicByName(String name);
    Topic findFatherByChild(Integer childId);
    void deleteById(Integer id);
}

@Service
public class UserServiceImpl implements UserService {

    private UserRepository userRepository;

    @Autowired
    public UserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public User save(User user) {
        return userRepository.save(user);
    }

    @Override
    public User update(User user) {
        return userRepository.save(user);
    }

    @Override
    public List<User> findAll() {
        return userRepository.findAll();
    }

    @Override
    public User findByUsername(String username) {
        var user = userRepository.findByUsername(username);
        if (user.isEmpty()) {
            return null;
        } else {
            return user.get();
        }
    }

    @Override
    public User findByTelegramId(Long telegramId) {
        return userRepository.findByTelegramId(telegramId);
    }
}

@Service
public class TopicServiceImpl implements TopicService {

    private TopicRepository topicRepository;
    private TopicMapper topicMapper;
```

```
    @Autowired
    public TopicServiceImpl(TopicRepository topicRepository,
TopicMapper topicMapper) {
        this.topicRepository = topicRepository;
        this.topicMapper = topicMapper;
    }

    @Override
    public TopicDTOResponse create(TopicDTORequest
topicDTORequest) throws Exception {
        topicRepository.saveWithFather(
                topicDTORequest.getName(),
                topicDTORequest.getText(),
                topicDTORequest.getType(),
                topicDTORequest.getFatherId());

        var topic =
topicRepository.findByName(topicDTORequest.getName());
        if (topic == null) {
            throw new Exception("Topic creating error");
        }
        return topicMapper.topicToTopicDTOResponse(topic);

    }

    @Override
    public List<TopicDTOResponse> findAll() {
        return
topicMapper.topicsToTopicDTOsResponse(topicRepository.findAll())
;
    }

    @Override
    public List<Topic> findByFatherId(Integer fatherId) {
        return topicRepository.findByFatherId(fatherId);
    }

    @Override
    public List<Topic> findBasicTopics() {
        return topicRepository.findByFatherIdIsNull();
    }

    @Override
    public Topic findTopicByName(String topicName) {
        return topicRepository.findByName(topicName);

    }

    @Override
    public Topic findFatherByChild(Integer childId) {
```

```
        return topicRepository.findFatherByChild(childId);
    }

    @Override
    public void deleteById(Integer id) {
        topicRepository.deleteById(id);
    }

    @Override
    public TopicDTOResponse update(TopicDTORequest
topicDTORequest, Integer id) throws Exception {

topicRepository.updateWithFather(topicDTORequest.getName(),
topicDTORequest.getText(),

topicDTORequest.getType(),topicDTORequest.getFatherId(), id);

        var topic =
topicRepository.findByName(topicDTORequest.getName());
        if (topic == null) {
            throw new Exception("Topic creating error");
        }
        return topicMapper.topicToTopicDTOResponse(topic);
    }

}

@Repository
public interface TopicRepository extends JpaRepository<Topic,
Integer> {

    @Query("FROM Topic WHERE father_id IS NULL")
    List<Topic> findByFatherIdIsNull();

    @Query("FROM Topic WHERE father_id = ?1")
    List<Topic> findByFatherId(Integer fatherId);

    @Query(value = "SELECT * FROM Topic WHERE id = (SELECT
father_id FROM Topic WHERE id = ?1)", nativeQuery = true)
    Topic  findFatherByChild(Integer id);

    Topic findByName(String name);

    @Modifying
    @Query(value = "insert into Topic (name, text, type,
father_id) VALUES (?1, ?2, ?3, ?4)", nativeQuery = true)
    @Transactional
    void saveWithFather(String name, String text, String type,
Integer fatherId);

    @Modifying
```

```
    @Query(value = "UPDATE Topic SET name = ?1, text = ?2, type
= ?3, father_id = ?4 WHERE id = ?5", nativeQuery = true)
    @Transactional
    void updateWithFather(String name, String text, String type,
Integer fatherId, Integer id);
}

@Data
@Entity
@Table(name = "topic")
public class Topic {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", unique = true, nullable = false)
    private Integer id;

    @Column(name = "name", unique = true, nullable = false)
    private String name;

    @Enumerated(EnumType.STRING)
    @Column(name = "type", nullable = false)
    private TopicResponseType type;

    @Lob
    @Column(name="text", nullable = false)
    private String text;

    @OneToMany(fetch=FetchType.LAZY)
    @JoinColumn(name="father_id")
    private List<Topic> subTopics;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "image_id")
    private Image image;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "file_id")
    private File file;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "video_id")
    private Video video;
}

     public class TopicMapper {

    public Topic topicDTORequestToTopic(TopicDTORequest
topicDTORequest) {
        var topic = new Topic();
        topic.setId(topicDTORequest.getId());
```

```
        topic.setName(topicDTORequest.getName());
        topic.setText(topicDTORequest.getText());

topic.setType(TopicResponseType.valueOf(topicDTORequest.getType(
).toUpperCase()));
        return topic;
    }

    public TopicDTOResponse topicToTopicDTOResponse(Topic topic)
{
        var topicDTOResponse = new TopicDTOResponse();
        topicDTOResponse.setId(topic.getId());
        topicDTOResponse.setName(topic.getName());
        topicDTOResponse.setText(topic.getText());
        return topicDTOResponse;
    }

    public List<TopicDTOResponse>
topicsToTopicDTOsResponse(List<Topic> topics) {
        return topics.stream().map(t ->
topicToTopicDTOResponse(t)).collect(Collectors.toList());
    }
}

     public class BotEventHandler {

    private TopicService topicService;
    private UserService userService;

    @Autowired
    public void setTopicService(TopicService topicService) {
        this.topicService = topicService;
    }

    @Autowired
    void setUserService(UserService userService) {
        this.userService = userService;
    }

    public boolean isRequestValid(Message message) {
        return message != null && message.hasText();
    }

    public User getUser(Message message) {
        var telegramUser = message.getFrom();
        var user =
userService.findByUsername(telegramUser.getUserName());
        if (user == null) {
            var newUser = new User();
            newUser.setUsername(telegramUser.getUserName());
            newUser.setChatId(message.getChatId());
```

```
            newUser.setFirstName(telegramUser.getFirstName());
            newUser.setState(message.getText());
            newUser.setTelegramId(telegramUser.getId());
            user = userService.save(newUser);
        } else {
            user.setTelegramId(telegramUser.getId());
            user.setUsername(telegramUser.getUserName());
            user.setChatId(message.getChatId());
            user.setFirstName(telegramUser.getFirstName());
        }
        return user;
    }

    public BotState getBotState(String strState) {
        return switch (strState.toUpperCase()) {
            case "/START" -> BotState.ENTRY_POINT;
            case "/HELP" -> BotState.HELP_MENU;
            case "ЗАВАНТАЖИТИ .PDF ФАЙЛ" ->
BotState.DOWNLOAD_PDF;
            case "НАЗАД" -> BotState.BACK;
            default -> BotState.SOME_TOPIC;
        };
    }

    public SendMessage getStartAnswer(User user) {
        updateUserData(user);
        var topics = topicService.findBasicTopics();
        var sendMessage = new
SendMessage(user.getChatId().toString(), "Використовуй
клавіатуру для вибору розділу.");
        sendMessage.enableMarkdown(true);

sendMessage.setReplyMarkup(getTopicReplyKeyboard(topics));
        return sendMessage;
    }

    public String stepBack(String lastUserState) throws
TelegramApiException {
        var topicByName =
topicService.findTopicByName(lastUserState);
        if (topicByName == null) {
            return "";
        }
        var fatherTopic =
topicService.findFatherByChild(topicByName.getId());

        return fatherTopic == null ? "" : fatherTopic.getName();
    }

    public SendMediaGroup getHelp(User user) {
        var firstPhoto = new InputMediaPhoto();
```

```
        firstPhoto.setMedia(new
File("src/main/resources/help1.png"), "help1");
        firstPhoto.setCaption("1 - Поле вводу. При вводі символу
\"/\" з'являються сталі комади боту.(/start)" +
                "\n2 - Команди боту." +
                "\n3 - Кнопка згортання та розгортання
клавіатури Бота." +
                "\n4 - Теми до вивчення. При написканні бот
відправить вам матеріали." +
                "\n5 - Кнопка для завантаження pdf файлу з
вибраною тамою." +
                "\n6 - Кнопка повернення до попередньої теми.");
        var secondPhoto = new InputMediaPhoto();

        secondPhoto.setMedia(new
File("src/main/resources/help2.png"), "help2");

        return new SendMediaGroup(user.getChatId().toString(),
Arrays.asList(firstPhoto, secondPhoto));
    }


    public SendDocument getFile(User user) {
        SendDocument sendDocument = new SendDocument();
        sendDocument.setChatId(user.getChatId().toString());
        var dbUser =
userService.findByTelegramId(user.getTelegramId());
        if (dbUser == null) {
            sendDocument.setCaption("Something goes wrong!");
            return sendDocument;
        }
        var lastTopic =
topicService.findTopicByName(dbUser.getState().toString());
        if (lastTopic == null) {
            sendDocument.setCaption("Something goes wrong!");
            return sendDocument;
        }
        sendDocument.setDocument(new InputFile(new
File("src/main/resources/" +
lastTopic.getFile().getFileName())));
        sendDocument.setCaption(lastTopic.getName());

sendDocument.setReplyMarkup(getTopicReplyKeyboardWithBackStepAnd
File(topicService.findByFatherId(lastTopic.getId())));
        return sendDocument;
    }

    public SendMessage getTopicsByFatherTopicAnswer(String
state, User user) {
        user.setState(state);
        updateUserData(user);
```

```
        var topicDTOResponse =
topicService.findTopicByName(state.toString());
        if (topicDTOResponse == null) {
            return null;
        }
        var topicDTOs =
topicService.findByFatherId(topicDTOResponse.getId());
        if (topicDTOs == null) {
            return null;
        }
        var sendMessage = new
SendMessage(user.getChatId().toString(),
topicDTOResponse.getText());
        sendMessage.enableMarkdown(true);

sendMessage.setReplyMarkup(getTopicReplyKeyboardWithBackStepAndF
ile(topicDTOs));
        return sendMessage;
    }

    public SendPhoto
getTopicsByFatherTopicWithImageAnswer(String state, User user) {
        user.setState(state);
        updateUserData(user);
        var topicDTOResponse =
topicService.findTopicByName(state.toString());
        if (topicDTOResponse == null) {
            return null;
        }
        var topicDTOs =
topicService.findByFatherId(topicDTOResponse.getId());
        if (topicDTOs == null) {
            return null;
        }
        var sendPhoto = new
SendPhoto(user.getChatId().toString(), new InputFile(new
File("src/main/resources/" +
topicDTOResponse.getImage().getFileName())));
        sendPhoto.setCaption(topicDTOResponse.getText());

sendPhoto.setReplyMarkup(getTopicReplyKeyboardWithBackStepAndFil
e(topicDTOs));

        return sendPhoto;
    }

    private ReplyKeyboardMarkup
getTopicReplyKeyboard(List<Topic> topics) {
        return buildTopicReplyKeyboard(topics);
    }
```

```
    private ReplyKeyboardMarkup
getTopicReplyKeyboardWithBackStepAndFile(List<Topic> topics) {
        var replyKeyboard = buildTopicReplyKeyboard(topics);
        var keyboard = replyKeyboard.getKeyboard();

        var keyboardRow = new KeyboardRow();
        keyboardRow.add(new
KeyboardButton(BotState.DOWNLOAD_PDF.toString()));
        keyboardRow.add(new
KeyboardButton(BotState.BACK.toString()));
        keyboard.add(keyboardRow);

        replyKeyboard.setKeyboard(keyboard);
        return replyKeyboard;
    }

    private ReplyKeyboardMarkup
buildTopicReplyKeyboard(List<Topic> topics) {
        var replyKeyboard = new ReplyKeyboardMarkup();

        replyKeyboard.setSelective(true);
        replyKeyboard.setResizeKeyboard(true);
        replyKeyboard.setOneTimeKeyboard(true);

        var keyboard = new ArrayList<KeyboardRow>();
        topics.forEach(t -> {
            var keyboardRow = new KeyboardRow();
            keyboardRow.add(new KeyboardButton(t.getName()));
            keyboard.add(keyboardRow);
        });
        replyKeyboard.setKeyboard(keyboard);
        return replyKeyboard;
    }

    private void updateUserData(User user) {
        userService.update(user);
    }

    public Topic getTopicByMessage(String message) {
        return topicService.findTopicByName(message);}
}
```

ДОДАТОК Б. ПРОГРАМНИЙ КОД ВЕБ-СТОРІНКИ

```
@NgModule({
  declarations: [
    AppComponent,
    TopicBrowserComponent,
    GenerateTopicComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModule,
    FormsModule,
    FontAwesomeModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  public title = 'adminKVPClient';
  public faBook = faBook;
}

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

@Injectable({
  providedIn: 'root'
})
export class TopicService {
  private url = "http://localhost:5000"
  private allTopicsEndpoint = "/topics"

  constructor(private http: HttpClient) { }

  getTopics(): Observable<Topic[]> {
    return this.http.get<Topic[]>(this.url + this.allTopicsEndpoi
nt);
```

```
  }

  saveTopic(topic: TopicDTORequest): Observable<Topic> {
    return this.http.post<TopicDTORequest>(this.url + this.allTop
icsEndpoint, topic);
  }

  updateTopic(topic: TopicDTORequest): Observable<Topic> {
    return this.http.put<TopicDTORequest>(this.url + this.allTopi
csEndpoint + '/' + topic.id, topic);
  }

  deleteTopic(topicId: number): Observable<void> {
    return this.http.delete<void>(this.url + this.allTopicsEndpoi
nt + '/' + topicId);
  }
}

@Injectable({
  providedIn: 'root'
})
export class TopicInfoService {

  constructor(private topicService: TopicService) { }

  async getTopics(): Promise<Topic[]> {
    return await this.topicService.getTopics().toPromise();
  }

  async createTopic(topic: TopicDTORequest): Promise<Topic> {
    if(topic.id) {
      return await this.topicService.updateTopic(topic).toPromise
();
    } else {
      return await this.topicService.saveTopic(topic).toPromise()
;
    }
  }

  async deleteTopic(topicId: number): Promise<void> {
    await this.topicService.deleteTopic(topicId).toPromise();
  }
}


@Component({
  selector: 'app-topic-browser',
  templateUrl: './topic-browser.component.html',
  styleUrls: ['./topic-browser.component.scss']
```

```
})
export class TopicBrowserComponent implements OnInit {

  public faTrash = faTrash;
  public faPen = faPen;

  public topics: Topic[] = [];

  constructor(private topicInfoService: TopicInfoService, private
router: Router) { }

  async ngOnInit(): Promise<void> {
    await this.loadTopics();
  }

  async loadTopics() {
    this.topics = await this.topicInfoService.getTopics();
  }

  async deleteTopic(id: number) {
    await this.topicInfoService.deleteTopic(id);
    await this.loadTopics();
  }

  async editTopic(id: number) {
    this.router.navigate(['update', id])
  }


}
export interface TopicDTORequest {
    id: number;
    name: string;
    text: string;
    type: string;
    fatherId: number;
}

export interface Topic {
    id: number;
    name: string;
    text: string;
    type: string;
}


@Component({
  selector: 'app-generate-topic',
  templateUrl: './generate-topic.component.html',
```

```
    styleUrls: ['./generate-topic.component.scss']
  })
  export class GenerateTopicComponent implements OnInit {

    public showInputError = false;

    public topicName = '';
    public topicText = '';
    public fatherTopicId = '';

    public topics: Topic[] = [];
    public topic: Topic | undefined;

    constructor(private topicInfoService: TopicInfoService,  privat
e route: ActivatedRoute) { }

    async ngOnInit(): Promise<void> {
      this.topics = await this.topicInfoService.getTopics();
      this.topic = this.topics.filter((t) => t.id == this.route.sna
pshot.params['id'])?.pop()
      if (this.topic) {
        this.topicName = this.topic.name;
        this.topicText = this.topic.text;
      }
    }

    async saveTopic(): Promise<void> {
      if( this.topicName && this.topicText && this.fatherTopicId) {
        this.showInputError = false;
        await this.topicInfoService.createTopic({
          id: this.topic?.id || NaN,
          name: this.topicName,
          text: this.topicText,
          type: 'MESSAGE',
          fatherId: Number(this.fatherTopicId)
        });
      } else {
        this.showInputError = true;
      }

    }

  }
```