ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ

ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ

ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ ІМЕНІ ІВАНА ПУЛЮЯ

Циклова комісія комп’ютерних наук

**КУРСОВА РОБОТА**

з дисципліни:

**«Розробка клієнт-серверних застосувань»**

на тему: **«Розробка RESTful вебсервісу для масажного салону»**

Студента  4  курсу групи КН-421п спеціальності 122 «Комп’ютерні науки»

Яскілка О.В.

(прізвище та ініціали)

Керівник: викладач   Слободян  Р.О.         
Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_\_ Руслан СЛОБОДЯН

(підпис)

\_\_\_\_\_\_\_\_ Ігор КАПАЦІЛА

(підпис)

м. Тернопіль – 2021

ЗМІСТ

[ВСТУП 6](#_Toc69978111)

[1 ТЕХНІЧНЕ ЗАВДАННЯ 7](#_Toc69978112)

[1.1 Найменування та область застосування 7](#_Toc69978113)

[1.2 Підстави для розробки 7](#_Toc69978114)

[1.3 Призначення розробки 7](#_Toc69978115)

[1.4 Вимоги до програмного забезпечення 7](#_Toc69978116)

[1.5 Вимоги до програмної документації 8](#_Toc69978117)

[1.6 Техніко-економічні показники 8](#_Toc69978118)

[1.7 Стадії та етапи розробки 8](#_Toc69978119)

[1.8 Порядок контролю та прийому 8](#_Toc69978120)

[2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ 9](#_Toc69978121)

[2.1 Постановка задачі на розробку програмного забезпечення 9](#_Toc69978122)

[2.2 Проєктування і опис прикладного інтерфейсу сервера 9](#_Toc69978123)

[2.3 Розробка структури бази даних сервера 9](#_Toc69978124)

[2.4 Розробка системи класів 10](#_Toc69978125)

[2.5 Опис форматів запитів до сервера та відповідей сервера 10](#_Toc69978126)

[2.6 Опис файлової структури ПЗ 11](#_Toc69978127)

[3 ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ 13](#_Toc69978128)

[ВИСНОВКИ 14](#_Toc69978129)

[ПЕРЕЛІК ПОСИЛАНЬ 15](#_Toc69978130)

[Додаток А Лістинг файлу «pom.xml» 16](#_Toc69978131)

[Додаток Б Лістинг файлу «Main.java» 17](#_Toc69978132)

[Додаток В Лістинг файлу «.java» 18](#_Toc69978133)

[Додаток Г Лістинг файлу «.java» 19](#_Toc69978134)

[Додаток Ґ Лістинг файлу «.java» 20](#_Toc69978135)

[Додаток Д Лістинг файлу «.java» 21](#_Toc69978136)

[Додаток Е Лістинг файлу «.java» 22](#_Toc69978137)

[Додаток Є Лістинг файлу «.java» 23](#_Toc69978138)

[Додаток Ж Лістинг файлу «.java» 24](#_Toc69978139)

[Додаток З Лістинг файлу «.java» 25](#_Toc69978140)

[Додаток И Лістинг файлу «.java» 26](#_Toc69978141)

[Додаток І Лістинг файлу «.java» 27](#_Toc69978142)

[Додаток Ї Лістинг файлу «.java» 28](#_Toc69978143)

[Додаток Й Лістинг файлу «.java» 29](#_Toc69978144)

[Додаток К Лістинг файлу «.java» 30](#_Toc69978145)

[Додаток Л Лістинг файлу «.java» 31](#_Toc69978146)

[Додаток М Лістинг файлу «.java» 32](#_Toc69978147)

[Додаток Н Лістинг файлу «.java» 33](#_Toc69978148)

[Додаток О Лістинг файлу «.java» 34](#_Toc69978149)

[Додаток П Лістинг файлу «.java» 35](#_Toc69978150)

[Додаток Р Лістинг файлу «.java» 36](#_Toc69978151)

[Додаток С Лістинг файлу «.java» 37](#_Toc69978152)

[Додаток Т Лістинг файлу «.java» 38](#_Toc69978153)

[Додаток У Лістинг файлу «.java» 39](#_Toc69978154)

[Додаток Ф Лістинг файлу «.java» 40](#_Toc69978155)

[Додаток Х Лістинг файлу «.java» 41](#_Toc69978156)

[Додаток Ц Лістинг файлу «.java» 42](#_Toc69978157)

[Додаток Ч Лістинг файлу «.java» 43](#_Toc69978158)

[Додаток Ш Лістинг файлу «.java» 44](#_Toc69978159)

[Додаток Щ Лістинг файлу «.java» 45](#_Toc69978160)

[Додаток Ю Лістинг файлу «.java» 46](#_Toc69978161)

[Додаток Я Лістинг файлу «configuration.properties» 47](#_Toc69978162)

ВСТУП

Java є мовою програмування та платформою обчислень, яка була вперше випущена Sun Microsystems у 1995 р. Існує безліч програм та веб-сайтів, які не працюють за відсутності встановленої Java, і з кожним днем кількість таких веб- сайтів та програм збільшується. Java відрізняється швидкістю, високим рівнем захисту та надійністю. Від портативних комп'ютерів до центрів даних, від ігрових консолей до суперкомп'ютерів, що використовуються для наукових розробок, від сотових телефонів до Інтернету - Java всюди!

Spring Boot - це фреймворк на основі Java з відкритим вихідним кодом, що використовується для створення мікросервісу. Він розроблений Pivotal Team і використовується для створення автономних та готових до використання пружин. Spring Boot надає хорошу платформу для розробників Java для розробки автономної та продуктивної весняної програми, яку ви можете просто запустити. Ви можете розпочати роботу з мінімальними налаштуваннями без необхідності повної настройки конфігурації Spring.

Hibernate — засіб відображення між [об'єктами](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) та [реляційними](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) [структурами](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) (object-relational mapping, [ORM](https://uk.wikipedia.org/wiki/ORM)) для платформи [Java](https://uk.wikipedia.org/wiki/Java). Hibernate є [вільним](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) програмним забезпеченням, яке поширюється на умовах [GNU Lesser](https://uk.wikipedia.org/wiki/GNU_Lesser_General_Public_License) [General Public License](https://uk.wikipedia.org/wiki/GNU_Lesser_General_Public_License). Hibernate надає легкий для використання каркас ([фреймворк](https://uk.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)) для відображення між об'єктно-орієнтованою моделлю даних і традиційною [реляційною базою даних](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85).

# ТЕХНІЧНЕ ЗАВДАННЯ

## Найменування та область застосування

Найменування програми – RESTful сервіс для сайту масажного салону .

Область застосування програми – для обробки данних про користувачів та послуги салону.

## Підстави для розробки

Підставами для проведення розробки являється індивідуальне завдання на курсову роботу з дисципліни «Розробка клієнт-серверних застосувань».

Найменування теми курсової роботи – «Розробка Restful API серіера для масажного салону».

Виконавець – студент групи КН-421 Яскілка Олександр Володимирович.

## Призначення розробки

Експлуатаційним призначення розробки є підвищення ефективності та зручності використання, збору та зберігання даних необхідних для роботи салону. Відповідно функціональне призначення — реалізація RESTful сервіса, що надає перечисленні вище можливості.

## Вимоги до програмного забезпечення

### Вимоги до функціональних характеристик

Очікується, що програма на одержаний запит надаватиме наступні можливості:

* повертатиме інформацію про усіх сутностей.
* повертатиме інформацію про конкретну сутність за її id.
* можливість добавлення нової сутності в базу даних.
* можливість зміни існуючої сутності за id або, у випадку її відсутності, добавити до бази даних можливість видалення існуючої сутності.

Також вбачається необхідність реалізації можливості авторизації та ідентифікації користувачів з метою надання доступу до даних. Очікується, що даних функціонал буде імплементовано шляхом видачі JWT токенів[1] та подальшого його використання для перевірки можливості клієнта здійснювати взаємодію з сервером.

### Вимоги до часових характеристик

До програмного продукту представляються наступні часові вимоги:

* виконання запиту та його обробка — до 1 секунди.

### Вимоги до надійності

Надійність забезпечується наступними засобами:

* перевірка вхідних даних;
* забезпечення стійкості від некваліфікованих дій користувача;
* перевірка наявності бази даних та необхідних таблиць.

### Умови до експлуатації

Взаємодія з програмою проводиться шляхом надсилання запитів від клієнта до сервера, тому від користувача очікується наявність вміння створювати та надсилати HTTP запити.

### Вимоги до складу і параметрів технічних засобів

Для використання програмного засобу необхідно наяність сервера з доступом до інтернету та наступними мінімальними вимогами:

* процесор — Intel Core i3;
* оперативна памʼять — 4 Гб;
* вільний дисковий простір — від 500 Мб.

### Вимоги до інформаційної і програмної сумісності

Вихідні коди програми повинні бути реалізовані засобами мови Java, версією не нижче 17. Середовище розробки, що буде використано, — IntellijIdea. Застосовано фреймворки Spring і Hibernate та бібліотеку JPA. Відповідно повинно забезпечуватися сумісність з MySQL 8.

Програмний засіб повинний бути сумісним з операційною системою Windows, версією не нижче 7.

## Вимоги до програмної документації

Після завершення розробки програмного засобу підготувати наступну документацію:

* інструкцію з інсталяції та первинного налаштування програми;
* інструкція з експлуатації;
* опис функціоналу та можливостей програми.

До програмного документації представляються наступні вимоги:

* кожний програмний модуль, клас, метод або процедурний блок повинен мати на початку стислий опис функціонального призначення, вхідних та вихідних даних, обмеження та умови використання;
* коментарі повинні бути стислими, чіткими та однозначними.

## Техніко-економічні показники

Цей продук не розраховувався на економічні ефективності та вартість.

## Стадії та етапи розробки

Розробка даного програмного забезпечення складатиметься з наступних

етапів:

* Аналіз.

Метою даного етапу є уточнення та детальний опис поставленого завдання, функціоналу, зовнішнього вигляду інтерфейсу, яким чином здійснюватиметься взаємодія з програмним засобом, як зберігатиметься інформація. Проводиться огляд та оцінка існуючих рішень, їхніх переваг та недоліків. Відбувається обʼєктна декомпозиція системи, визначається функціональна роль її складових.

* Проєктування.

Проєктування умовно можна поділити на логічне та фізичне. При першому відбувається опис модулів, пакетів, класів та їх методів, взаємозвʼязків та взаємодії між ними, таким чином, щоб спростити подальше розширення системи, зміни існуючого функціоналу; здійснюється проєктування інтерфейсу, в даному випадку представленому ендпоінтами.

Фізичне проєктування полягає в безпосередньому кодуванні програми деякому алгоритмічною мовою(у даній курсовій — Java).

* Розвиток та тестування.

Даний етап полягає в послідовній реалізації класів та методів і підключенню їх до проєкту. Одержана програма тестується, у разі необхідності вносяться відповідні правки.

## Порядок контролю та прийому

Прийом розробленого програмного продукту повинен відбуватися на обʼєкті Замовника в терміни зазначенні в індивідуальному завданні.

Для прийому роботу Виконавець зобов'язаний представити:

* робочу програму, що повністю відповідає поставленому технічну завданню;
* вихідні програмні коди, які записані разом з програмою на оптичний носій інформації.

Прийом програми повинен відбуватися перед комісією, що складається з двох людей(один з яких Замовник) у присутності студентів з академічної групи Виконавця у такій послідовності:

* доповідь Виконавцем про виконану роботу;
* демонстрація робочої програми з контрольним випробуванням програмного продукту;
* відповідь на запитання присутніх та зауваження.

# РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

## Постановка задачі на розробку програмного забезпечення

Перший етап це аналіг програми. Завданням даної курсової роботи є розробка деякого клієнт-сервервого застосунку через реалізацію RESTful сервісу, тому необхідно реалізувати деякі контролери, що відповідатимуть за обробку запитів.

Визначемо наступне:

* діючих субʼєктів;
* субʼєктів, що вводять інформацію;
* субʼєктів, що отримують інформацію;
* взаємодію програми з іншими;
* дії, що можуть бути виконаними кожним з субʼєктів.

Очевидними субʼєктами є власне програма(представляюча сервер), з якою проводитимиться взаємодія, та користувач(клієнт), що вводить інформацію та отримує деяку відповіді.

Передбачається взаємодія програми з системою керування базами даних та операційною системою. Необхідність роботи з СКБД зумовлюється потребою зберігати та обробляти інформацію. Таким чином уникається проектування організації даних на фізичному носії. У свою чергу взаємодія з ОС проявляється у вигляді надання доступу програми до деякої компʼютерної мережі.

Клієнт може виконувати будь-які запити, що описані в підрозділі 1.4.1. Також для зручності користування передбачаються запити з можливістю фільтрації сутностей.

Взаємодія субʼєктів та зовнішніх програм з сервером зображено на рисунку 2.1.

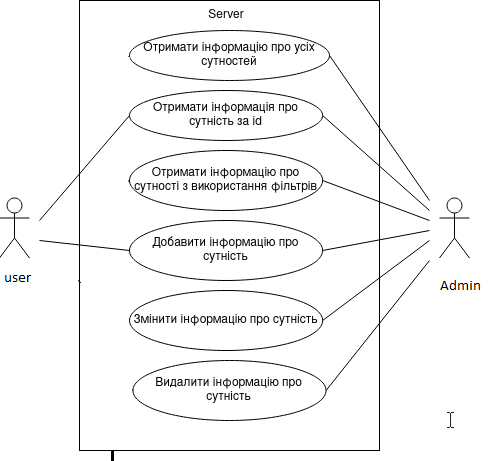


Рисунок 2.1 – Діаграма прецендетів програми

## Проєктування і опис прикладного інтерфейсу сервера

Мовою програмування, якою буде здійснено розробку програмного засобу, було визначено Java, а також ми будемо використовувати фреймворк Spring Boot[2]. Використання вище зазначений технологій спростить створення RESTful сервісу із застосуванням патерну MVC[3], що передбачає необхідність реалізації таких компонентів як Model, View та Controller. На цю роботу було звільнено від написання компоненту View, тому ми будем використовувати REST запити. Інформація буде задаватися і видаватися засобами HTTP запитів.

## Розробка структури бази даних сервера

При розробці бази даних було прийнято до уваги, все вище сказане і складено такий список таблиць

* Role;
* User;
* User\_role;
* Comment;

На рисунку 2.2 зображено ER-діаграму спроєктованої бази даних.

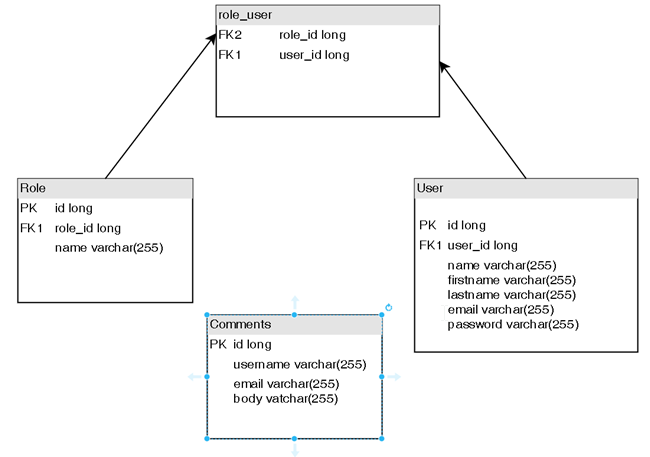


Рисунок 2.2 – ER-діаграма бази даних

Детально опишемо наявні стовбці таблиць:

Role – це таблиця ля зберігання типів ролей. Наприклад ADMIN чи USER. Сам клас скаладаєтся з поля id типу long. Foring Key role\_id типу long на таблицю role\_user. І поля name типу varchar.

Таблиця User зберігає в собі інформацію про користувачів. Складаєтся з полів id типу long, Foring Key user\_id типу long на таблицю role\_user так полів name,first,lastname,email.password типу varchar.

З метою забезпечення можливості авторизації також реалізовано таблицю user, яка включає в себе поля username та password типу varchar(255). Наяність даної таблиці зумовлюється необхідністю ідентифікації користувача та підтвердження його особистості шляхом введення паролю, що повинний бути відомий виключно серверу та власнику акаунта.

Таблиця role\_user створеня Hibernate-ом для поєднання двох минулих табиць. Складаєтся з двох зовнішніх ключів user\_id та role\_id.

Таблиця Comment створений для запису коментарів які введуть користувачі. Складаэтся з поля id типу long і полів name,email,body. Де name – це поле для збереження імені відправника,email –для зберігання e-mail, я також body – для збереження самого тесту.

## Розробка системи класів

* + 1. **Розробка JPA Entity класів**

Взаємодія сервера та бази даних здійснюється засобами JPA[4] та Hibernate[5], тому зʼявляється необхідність реалізації JPA Entity класів[6], кодування яких засновується на описах у попередньому підрозділі таблицях бази даних. Відповідно для кожної сутності буде створено представляючий її клас:

* BaseEntity;
* Comments;
* User;
* Role;

# Розробка DAO класів

Наступним етапом розробки є забезпечення можливості програми взаємодія з базою даних. Для цього створено окремий абстракний шар DAO, що надаватиме повну взаємодію з інформацією: створення, оновлення, читання, видалення.

Безпосередню роботу з базою даних, як вище зазначено, JPA та Hibernate, проте специфіка використання першої потребує створення інтерфейсів, що розширують функціонал. Таким чином було реалізовано наступні інтерфейси:

* CommentsRepository;
* RoleRepository;
* UserRepository;

# Розробка сервісів

Взаємодія з репозиторіями буде проводитися відповідними сервісами, таким чином було реалізовано наступні класи:

* UserService;(interface)
* CommentService;(interface)
* UserServiceIml;
* CommentService;

Функції інтерфейсу UserService :

* User register(User user) – метод який реалізовує регестрацію юзера.
* List<User> getAll() – метод який виводить всіх користувачів.
* User findByUsername(String username) – метод який знаходить користувачв по username.
* User findById(Long id) – метод який знаходить користувачв по id.
* void delete(Long id) – метод для видалення користувача.

Функції інтерфейсу CommentService:

* CommentDto createComment(Comments comments) – метод який реалізовує створення коментарія.
* List<Comments> getCommentsAll() – метод який виводить всі коментарії.
* Comments findByUsername(String username) – метод який знаходить коментарій по username.
* CommentDto updateComment(long commentId, CommentDto commentRequest) – метод для оновлення кментарія.
* void deleteComment(Long commentId)– метод для видалення коментаря.

# Розробка контролерів

Як у попередніх підрозділах при найменуванні контролерів будемо опиратися на створені визначені раніше сутності, таким чином одержимо наступні класи контролерів:

* AdminRetsControllerV1;
* UserRetsControllerV1;
* AuthenticationRestControllerV1;

З метою забезпечення можливості клієнту авторизації добавлено контролер AuthController, який містить єдиний метод public @ResponseBody User login(), що власне відповідає за іденктифікацію користувача.

## Опис форматів запитів до сервера та відповідей сервера

Запити, що наявні в даній програмі описані у першому розділі, тому нижче буде наведених повний список запитів та можливих відповідей на них:

*Таблиця 2.1 — Requests & Responses*

|  |  |  |  |
| --- | --- | --- | --- |
| Endpoint | Method | Query params | Response |
| /api/v1/admin/ |  |  | Ендпоінт для ролі ADMIN |
| /api/v1/admin/users | GET |  | Повертає всі обʼєкт класу User |
| /api/v1/admin/comments | GET |  | Повертає всі обʼєкти класу Comment |
| /api/v1/admin/users/{id} | DELETE |  | Видаляє обʼєкт класу Comment заданим ідентифікатором id |
| /api/v1/admin/comments/{id} | GET |  | Повертає обʼєкт класу User з вказамиm Id |
| /api/v1/users/ |  |  | Ендпоінт для ролі User |
| /api/v1/users/{id} | GET |  | Повертає обʼєкт класу User з вказами Id |
| /api/v1/users/comments/{username} | GET |  | Повертає обʼєкт типу PredictedData зпараметром username |

*Продовження таблиці 2.1*

|  |  |  |  |
| --- | --- | --- | --- |
| api/v1/users/ comments | GET |  | Повертає обʼєкт класу Comment з фільтрацією для User |

|  |  |  |  |
| --- | --- | --- | --- |
| /api/v1/auth/login / | POST |  | Через headers передається  інформація про користувачі, у випадку наявності в останнього акаунта body відповіді містиме JWT токен |

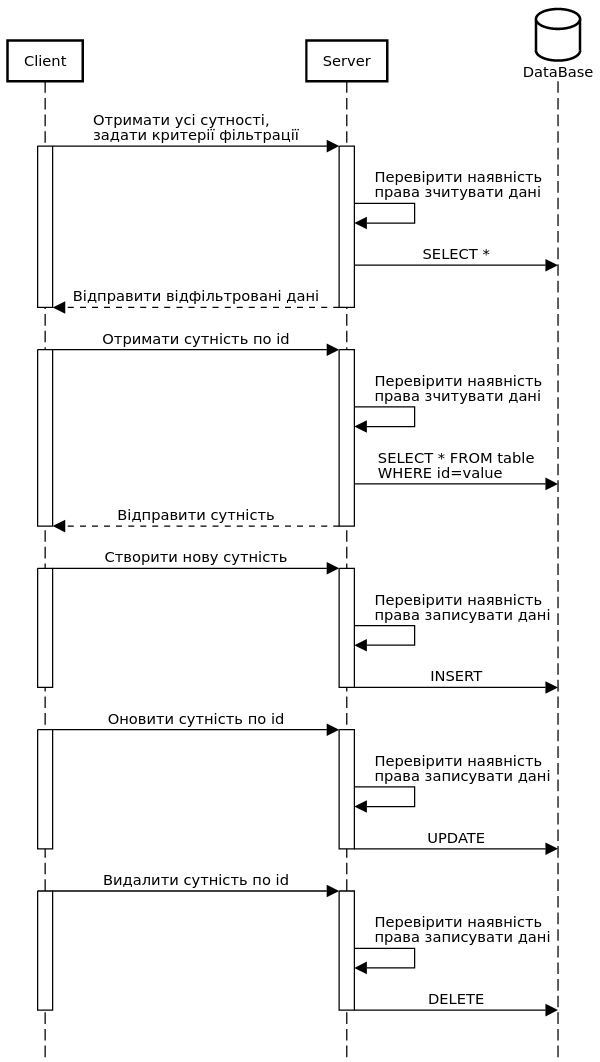


Рисунок 2.5 – Діаграма послідовностей дій

На рисунку 2.5 зображено діаграму послідовності дій, що ініціалізуються клієнтом та що відображають внутрішню діяльність сервера[7].

## Опис файлової структури ПЗ

Файлова структура програми виглядає наступним чином(див.рис. 2.6) :

* “pom.xml” — файл необхідний для роботи Maven, забезпечє збірку необхідних залежностей;
* JWTUserDetailService.java— точка входу в програму



Рисунок 2.6 – Дерево програми

В папці config створений клас для конфігурації захисту.

В папці Dto використовують для передачі даних між підсистемами програми.

В папці model створені клади сущостей.

В папці reporitory створені класи для роботи з базою данних використовуючи hibernate.

В папці rest знаходятся всі контролери.

В папці security та security/jwt знаходятся класи для роботи з spring security та в головному з jwt-токеном.

В папці service розроблені класи бізнес-логіки.

# ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ

У даному розділі курсової роботи здійснено опис проведеного з використання програмного засобу Postman[8] тестування, метою якого є виявляння не бажаної, відмінної від заданої першим розділом поведінки сервера.

Приступимо до тестування функціонал сервера. Перш за все переконаємося, що неавторизований користувач не може жодних чином взаємодіяти з програмою, окрім ендпоінту /api/v1/auth/login, проте над вказаним маршрутом також необхідно провести тестування, ціллю якого є перевірка реакції на неправильно введені дані для аутентифікації Результат зображений на рисунках 3.1 та є абсолютно задовільнимТакож можна описати розроблені за допомогою JUnit 5 модульні тести.

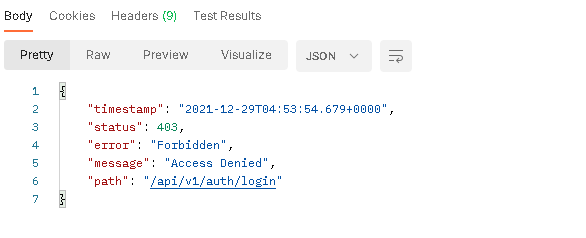


Рисунок 3.1 – Відповідь сервера при введені клієнтом неправильних даних аутентифікації

Введемо правильні імʼя користувача та пароль у вкладці Authorization. Як видно на рисунку 3.2 сервер повернув JWT токен, який нам необхідно скопіювати для забезпечення доступу до іншим ендпоінтів.

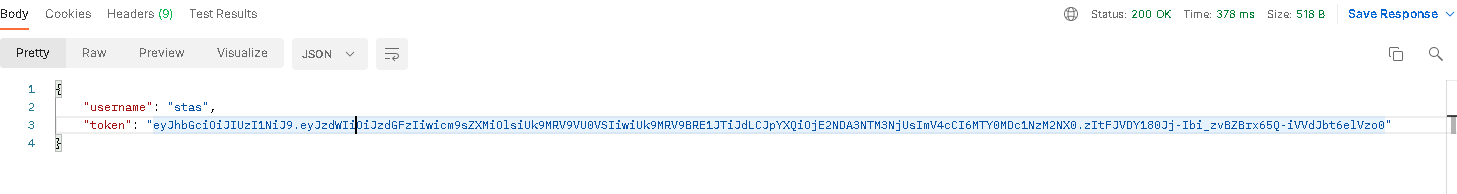


Рисунок 3.2 – Відповідь сервера на ендпоінті auth/login

Спробуємо використати отриманий раніше токін для доступу до інформації про усіх Users. Для цього додамо його до Headers при здійснені запиту. Відповідь зображено на рисунку 3.3, звідки робиться висновок про успішне надання доступу до даних на відміну від результату зображеного на рисунку 3.1.

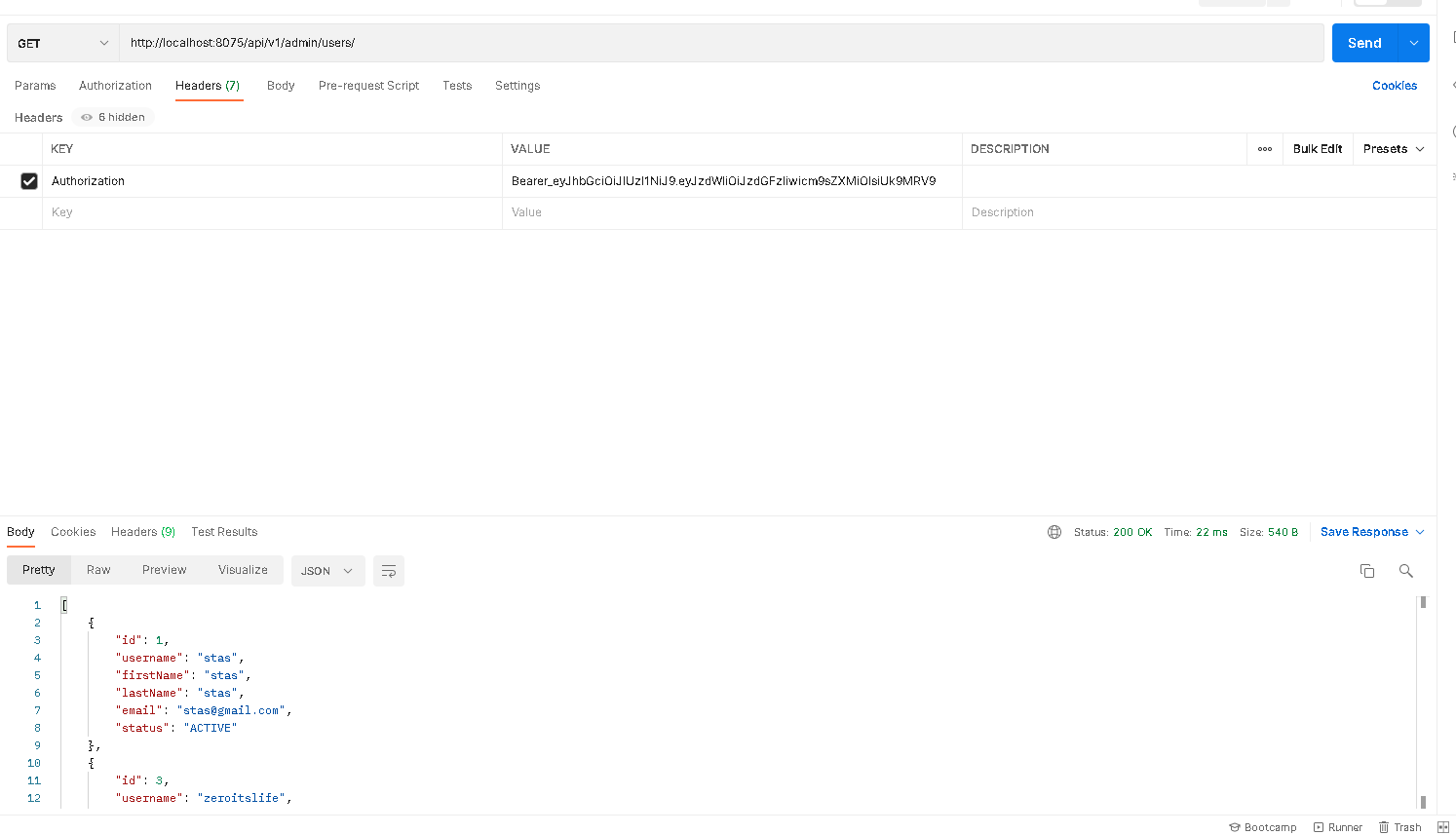


Рисунок 3.3 – Відповідь сервера на ендпоінт /users

Тепер змінимо JWT токен, щоб переконатися в тому, що програма здійснує перевірку значення токена, а не його наявність. Результат зображений на рисунку 3.5 відповідає очікуваному, що є підтвердження правильної роботи модуля безпеки, переконавшись у працездатності якого, перейдемо до тестування ендпоінтів.

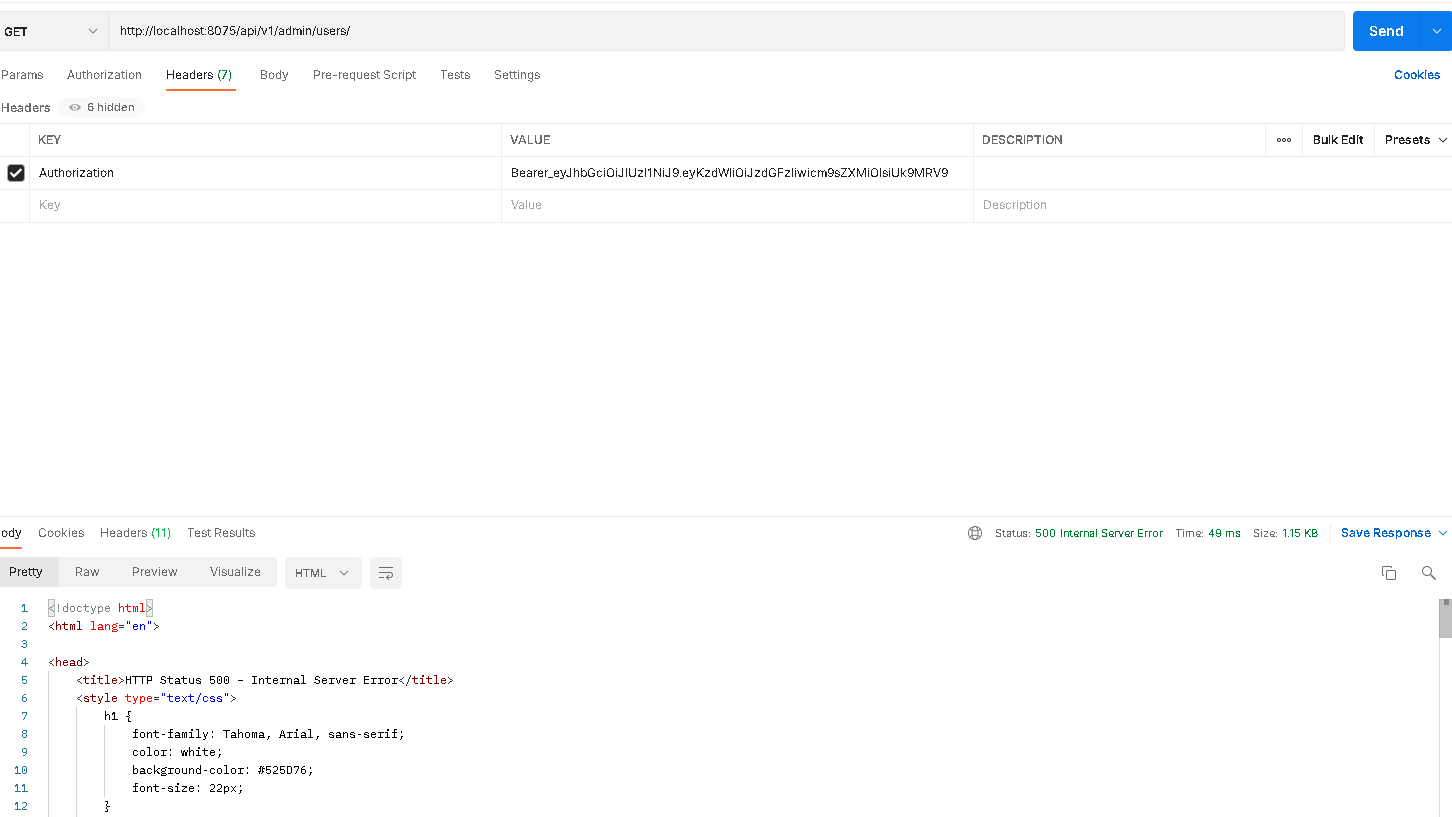


Рисунок 3.4 – Відповідь сервера на неправильний JWT токен

Тепер перевірим роботу ролей попробуєм зайти на профіль з роллю User і виконати один з ендпоїнтів результат можна бачити на рисунку 3.5. Як видно з рисунку все працює коректно.

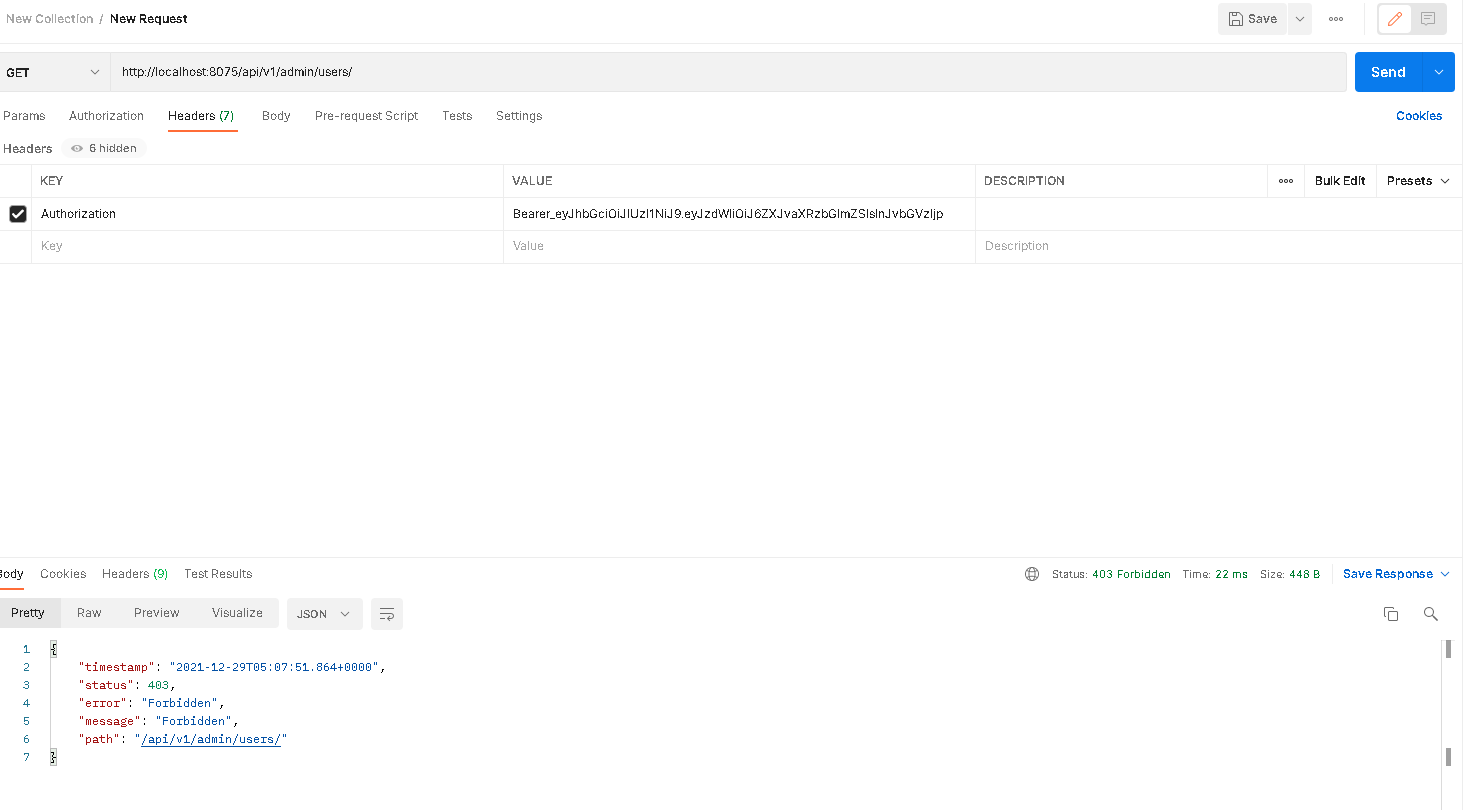


Рисунок 3.5 – Перевірка входу User користувача до Admin ендпоїнтів

Враховуючи подібність та одноманість подальших запитів, їхнє тестування не буде описано. Таким чином було протестовуно запити для усіх інших сутностей.

Результат тестування цілком задовільний, було виявлено та виправлено декілька незначних помилок.

ВИСНОВКИ

У даній курсовій роботі були описані етапи розробки клієнт-серверного застосування для зберігання та обрибки даних голосового асистенту та інформацію про ефективність його роботи. Розроблений програмний продукт відповідає усім поставлених технічним завданням.

У процесі розробки програмного засобу було закріплено та поглиблено знання з дисциплін “Розробка клієнт-серверних застосування”, “Обʼєктно- орієнтоване програмування” та “Організація баз даних та знань”, детальніше вивчено мову програмування Java, фреймворки JPA, Hibernate, Spring Boot.

Було засвоєно технології проєктування та розробки клієнт-серверних застосування, засоби запезпечення взаємодії з СКБД MySQL, технологію JWT токенів, що дозволяються здійснювати аутентифікацію та авторизації користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. JSON Web Tokens - jwt.io [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/. – Дата доступу: 17.11.2021.
2. Spring Boot [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-boot. – Дата доступу: 17.11.2021.
3. MVC - MDN Web Docs Glossary Definitions of Web-related terms | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/MVC. – Дата доступу: 18.11.2021.
4. Spring Data JPA [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-data-jpa. – Дата доступу: 18.11.2021.
5. Hibernate. Everything data. - Hibernate [Електронний ресурс] – Режим доступу до ресурсу: https://hibernate.org/. – Дата доступу: 19.11.2021.
6. What is a JPA Entity? [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/cd/E16439\_01/doc.1013/e13981/undejbs003.htm. – Дата доступу: 19.11.2021.
7. Діаграма послідовностей [Електронний ресурс] – Режим доступу до ресурсу:https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0\_%D0%BF%D0%BE%D1%81%D0%BB%D1%96%D0 %B4%D0%BE%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%96 – Дата доступу: 19.11.2021.
8. Postman API Platform [Електронний ресурс] – Режим доступу до ревурсу: <https://www.postman.com/>– Дата доступу: 19.11.2021.Додаток А  
   Лістинг файлу «pom.xml»

**Додаток А  
Лістинг файлу «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 http://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>2.1.4.RELEASE</version>  
        <relativePath/> <!-- lookup parent from repository -->  
    </parent>  
    <groupId>net.proselyte</groupId>  
    <artifactId>jwtappdemo</artifactId>  
    <version>0.0.1-SNAPSHOT</version>  
    <name>JwtAppDemo</name>  
    <description>JWT based spring security REST API</description>  
  
    <properties>  
        <java.version>1.8</java.version>  
        <mysql.connetor.version>8.0.15</mysql.connetor.version>  
        <jsonwebtoken.version>0.8.0</jsonwebtoken.version>  
    </properties>  
  
    <dependencies>  
        <!-- Spring -->  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-data-jpa</artifactId>  
        </dependency>  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-security</artifactId>  
        </dependency>  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-web</artifactId>  
        </dependency>  
  
        <!-- Liquibase -->  
        <dependency>  
            <groupId>org.liquibase</groupId>  
            <artifactId>liquibase-core</artifactId>  
        </dependency>  
  
        <!-- MySQL connector -->  
        <dependency>  
            <groupId>mysql</groupId>  
            <artifactId>mysql-connector-java</artifactId>  
            <version>${mysql.connetor.version}</version>  
        </dependency>  
  
  
        <!-- JSON Web Token lib -->  
        <dependency>  
            <groupId>io.jsonwebtoken</groupId>  
            <artifactId>jjwt</artifactId>  
            <version>${jsonwebtoken.version}</version>  
        </dependency>  
  
        <!-- Lombok -->  
        <dependency>  
            <groupId>org.projectlombok</groupId>  
            <artifactId>lombok</artifactId>  
            <optional>true</optional>  
        </dependency>  
  
        <!-- Spring test -->  
        <dependency>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-test</artifactId>  
            <scope>test</scope>  
        </dependency>  
        <dependency>  
            <groupId>org.springframework.security</groupId>  
            <artifactId>spring-security-test</artifactId>  
            <scope>test</scope>  
        </dependency>  
    </dependencies>  
  
    <build>  
        <plugins>  
            <plugin>  
                <groupId>org.springframework.boot</groupId>  
                <artifactId>spring-boot-maven-plugin</artifactId>  
            </plugin>  
        </plugins>  
    </build>  
  
</project>

**Додаток Б  
Лістинг файлу «WebApplication.java»**

package net.proselyte.jwtappdemo;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class JwtAppDemoApplication {  
  
   public static void main(String[] args) {  
      SpringApplication.run(JwtAppDemoApplication.class, args);  
   }  
  
}

**Додаток В  
Лістинг файлу «SecurityConig.java»**

package net.proselyte.jwtappdemo.config;  
  
import net.proselyte.jwtappdemo.security.jwt.JwtConfigurer;  
import net.proselyte.jwtappdemo.security.jwt.JwtTokenProvider;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.config.http.SessionCreationPolicy;  
  
  
@Configuration  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
    private final JwtTokenProvider jwtTokenProvider;  
  
    private static final String ADMIN\_ENDPOINT = "/api/v1/admin/\*\*";  
    private static final String LOGIN\_ENDPOINT = "/api/v1/auth/login";  
  
    @Autowired  
    public SecurityConfig(JwtTokenProvider jwtTokenProvider) {  
        this.jwtTokenProvider = jwtTokenProvider;  
    }  
  
    @Bean  
    @Override  
    public AuthenticationManager authenticationManagerBean() throws Exception {  
        return super.authenticationManagerBean();  
    }  
  
    @Override  
    protected void configure(HttpSecurity http) throws Exception {  
        http  
                .httpBasic().disable()  
                .csrf().disable()  
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
                .and()  
                .authorizeRequests()  
                .antMatchers(LOGIN\_ENDPOINT).permitAll()  
                .antMatchers(ADMIN\_ENDPOINT).hasRole("ADMIN")  
                .anyRequest().authenticated()  
                .and()  
                .apply(new JwtConfigurer(jwtTokenProvider));  
    }  
}

**Додаток Г  
Лістинг файлу «AdminUserDto.java»**

package net.proselyte.jwtappdemo.dto;  
  
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;  
import lombok.Data;  
import net.proselyte.jwtappdemo.model.Status;  
import net.proselyte.jwtappdemo.model.User;  
  
  
  
@Data  
@JsonIgnoreProperties(ignoreUnknown = true)  
public class AdminUserDto {  
    private Long id;  
    private String username;  
    private String firstName;  
    private String lastName;  
    private String email;  
    private String status;  
  
    public User toUser() {  
        User user = new User();  
        user.setId(id);  
        user.setUsername(username);  
        user.setFirstName(firstName);  
        user.setLastName(lastName);  
        user.setEmail(email);  
        user.setStatus(Status.valueOf(status));  
        return user;  
    }  
  
    public static AdminUserDto fromUser(User user) {  
        AdminUserDto adminUserDto = new AdminUserDto();  
        adminUserDto.setId(user.getId());  
        adminUserDto.setUsername(user.getUsername());  
        adminUserDto.setFirstName(user.getFirstName());  
        adminUserDto.setLastName(user.getLastName());  
        adminUserDto.setEmail(user.getEmail());  
        adminUserDto.setStatus(user.getStatus().name());  
        return adminUserDto;  
    }  
}

}

}

**Додаток Ґ  
Лістинг файлу «AuthenticationRequestDto.java»**

package net.proselyte.jwtappdemo.dto;  
  
import lombok.Data;  
  
  
@Data  
public class AuthenticationRequestDto {  
    private String username;  
    private String password;  
}

**Додаток Д  
Лістинг файлу «CommentAdminDto.java»**

package net.proselyte.jwtappdemo.dto;  
  
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;  
import lombok.Data;  
import net.proselyte.jwtappdemo.model.Comments;  
@Data  
@JsonIgnoreProperties(ignoreUnknown = true)  
public class CommentAdminDto {  
  
    private long id;  
    private String name;  
    private String email;  
    private String body;  
  
  
  
    public Comments toComments(){  
        Comments comments = new Comments();  
        comments.setId(id);  
        comments.setName(name);  
        comments.setBody(email);  
        comments.setBody(body);  
  
  
        return comments;  
    }  
  
    public static CommentDto fromComments(Comments comments) {  
        CommentDto comments1 = new CommentDto();  
        comments1.setId(comments.getId());  
        comments1.setName(comments.getName());  
        comments1.setEmail(comments.getEmail());  
        comments1.setBody(comments.getBody());  
  
        return comments1;  
    }  
}

**Додаток Е  
Лістинг файлу «CommentDto.java»**

package net.proselyte.jwtappdemo.dto;  
  
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;  
import lombok.Data;  
import net.proselyte.jwtappdemo.model.Comments;  
  
import javax.validation.constraints.Email;  
import javax.validation.constraints.NotEmpty;  
import javax.validation.constraints.Size;  
  
@Data  
@JsonIgnoreProperties(ignoreUnknown = true)  
public class CommentDto {  
  
    private long id;  
    private String name;  
    private String email;  
    private String body;  
  
  
  
    public  Comments toComments(){  
        Comments comments = new Comments();  
        comments.setId(id);  
        comments.setName(name);  
        comments.setBody(body);  
  
  
        return comments;  
    }  
  
    public static CommentDto fromComments(Comments comments) {  
        CommentDto comments1 = new CommentDto();  
        comments1.setId(comments.getId());  
        comments1.setName(comments.getName());  
        comments1.setBody(comments.getBody());  
  
        return comments1;  
    }  
}

 package net.proselyte.jwtappdemo.dto;  
  
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;  
import lombok.Data;  
import net.proselyte.jwtappdemo.model.Comments;  
  
import javax.validation.constraints.Email;  
import javax.validation.constraints.NotEmpty;  
import javax.validation.constraints.Size;  
  
@Data  
@JsonIgnoreProperties(ignoreUnknown = true)  
public class CommentDto {  
  
    private long id;  
    private String name;  
    private String email;  
    private String body;  
  
  
  
    public  Comments toComments(){  
        Comments comments = new Comments();  
        comments.setId(id);  
        comments.setName(name);  
        comments.setBody(body);  
  
  
        return comments;  
    }  
  
    public static CommentDto fromComments(Comments comments) {  
        CommentDto comments1 = new CommentDto();  
        comments1.setId(comments.getId());  
        comments1.setName(comments.getName());  
        comments1.setBody(comments.getBody());  
  
        return comments1;  
    }  
}

**Додаток Є  
Лістинг файлу «UserDto.java»**

package net.proselyte.jwtappdemo.dto;  
  
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;  
import lombok.Data;  
import net.proselyte.jwtappdemo.model.User;  
  
@Data  
@JsonIgnoreProperties(ignoreUnknown = true)  
public class UserDto {  
    private Long id;  
    private String username;  
    private String firstName;  
    private String lastName;  
    private String email;  
  
    public User toUser(){  
        User user = new User();  
        user.setId(id);  
        user.setUsername(username);  
        user.setFirstName(firstName);  
        user.setLastName(lastName);  
        user.setEmail(email);  
  
        return user;  
    }  
  
    public static UserDto fromUser(User user) {  
        UserDto userDto = new UserDto();  
        userDto.setId(user.getId());  
        userDto.setUsername(user.getUsername());  
        userDto.setFirstName(user.getFirstName());  
        userDto.setLastName(user.getLastName());  
        userDto.setEmail(user.getEmail());  
  
        return userDto;  
    }  
}

**Додаток Ж  
Лістинг файлу «BaseEntity.java»**

package net.proselyte.jwtappdemo.model;  
  
import lombok.Data;  
import org.springframework.data.annotation.CreatedDate;  
import org.springframework.data.annotation.LastModifiedDate;  
  
import javax.persistence.Column;  
import javax.persistence.EnumType;  
import javax.persistence.Enumerated;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.MappedSuperclass;  
import java.util.Date;  
  
  
@MappedSuperclass  
@Data  
public class BaseEntity {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @CreatedDate  
    @Column(name = "created")  
    private Date created;  
  
    @LastModifiedDate  
    @Column(name = "updated")  
    private Date updated;  
  
    @Enumerated(EnumType.STRING)  
    @Column(name = "status")  
    private Status status;  
}

**Додаток З  
Лістинг файлу «Comments.java»**

package net.proselyte.jwtappdemo.model;  
  
import lombok.Data;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.Table;  
  
@Entity  
@Table(name = "comments")  
@Data  
public class Comments extends BaseEntity{  
  
    @Column(name = "username")  
    private String name;  
  
    @Column(name = "email")  
    private String email;  
  
    @Column(name = "body")  
    private String body;  
  
    @Override  
    public String toString() {  
        return "Comments{" +  
                "name='" + name + '\'' +  
                ", email='" + email + '\'' +  
                ", body='" + body + '\'' +  
                '}';  
    }  
}

**Додаток И  
Лістинг файлу «Role.java»**

package net.proselyte.jwtappdemo.model;  
  
import lombok.Data;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.ManyToMany;  
import javax.persistence.Table;  
import java.util.List;  
  
@Entity  
@Table(name = "roles")  
@Data  
public class Role extends BaseEntity {  
  
    @Column(name = "name")  
    private String name;  
  
    @ManyToMany(mappedBy = "roles", fetch = FetchType.LAZY)  
    private List<User> users;  
  
    @Override  
    public String toString() {  
        return "Role{" +  
                "id: " + super.getId() + ", " +  
                "name: " + name + "}";  
    }  
}

**Додаток І  
Лістинг файлу «Status.java»**

package net.proselyte.jwtappdemo.model;  
  
  
public enum Status {  
    ACTIVE, NOT\_ACTIVE, DELETED,NONE  
}

**Додаток Ї  
Лістинг файлу «User.java»**

package net.proselyte.jwtappdemo.model;  
  
import lombok.Data;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.FetchType;  
import javax.persistence.JoinColumn;  
import javax.persistence.JoinTable;  
import javax.persistence.ManyToMany;  
import javax.persistence.Table;  
import java.util.List;@Entity  
@Table(name = "users")  
@Data  
public class User extends BaseEntity {  
  
    @Column(name = "username")  
    private String username;  
  
    @Column(name = "first\_name")  
    private String firstName;  
  
    @Column(name = "last\_name")  
    private String lastName;  
  
    @Column(name = "email")  
    private String email;  
  
    @Column(name = "password")  
    private String password;  
  
    @ManyToMany(fetch = FetchType.*EAGER*)  
    @JoinTable(name = "user\_roles",  
            joinColumns = {@JoinColumn(name = "user\_id", referencedColumnName = "id")},  
            inverseJoinColumns = {@JoinColumn(name = "role\_id", referencedColumnName = "id")})  
    private List<Role> roles;  
}

**Додаток Й  
Лістинг файлу «CommentsRepository.java»**

package net.proselyte.jwtappdemo.repository;  
  
import net.proselyte.jwtappdemo.model.Comments;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Modifying;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
  
import java.util.List;  
  
public interface CommentsRepository extends JpaRepository<Comments,Long> {  
  
    @Query("SELECT a FROM Comments a WHERE a.name = :name")  
   Comments findByName(String name);  
  
  
}

**Додаток К  
Лістинг файлу «RoleRepository.java»**

package net.proselyte.jwtappdemo.repository;  
  
import net.proselyte.jwtappdemo.model.Role;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface RoleRepository extends JpaRepository<Role, Long> {  
    Role findByName(String name);  
}

**Додаток Л  
Лістинг файлу «UserRepository.java»**

package net.proselyte.jwtappdemo.repository;  
  
import net.proselyte.jwtappdemo.model.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
  
public interface UserRepository extends JpaRepository<User, Long> {  
    User findByUsername(String name);  
}

**Додаток М  
Лістинг файлу «AdminRestControllerV1.java»**

package net.proselyte.jwtappdemo.rest;  
  
import net.proselyte.jwtappdemo.dto.AdminUserDto;  
import net.proselyte.jwtappdemo.dto.CommentDto;  
import net.proselyte.jwtappdemo.model.Comments;  
import net.proselyte.jwtappdemo.model.User;  
import net.proselyte.jwtappdemo.service.UserService;  
import net.proselyte.jwtappdemo.service.impl.CommentsService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.ArrayList;  
import java.util.List;  
  
  
@RestController  
@RequestMapping(value = "/api/v1/admin/")  
public class AdminRestControllerV1 {  
  
    private final UserService userService;  
  
    @Autowired  
    private CommentsService commentsService;  
  
    @Autowired  
    public AdminRestControllerV1(UserService userService) {  
        this.userService = userService;  
  
    }  
  
    @GetMapping(value = "users/{id}")  
    public ResponseEntity<AdminUserDto> getUserById(@PathVariable(name = "id") Long id) {  
        User user = userService.findById(id);  
  
        if (user == null) {  
            return new ResponseEntity<>(HttpStatus.NO\_CONTENT);  
        }  
  
        AdminUserDto result = AdminUserDto.fromUser(user);  
  
        return new ResponseEntity<>(result, HttpStatus.OK);  
    }  
  
    @GetMapping(value = "/users")  
    public List<AdminUserDto> getUserALll(){  
        List<User> user = userService.getAll();  
        List<AdminUserDto> dtoList = new ArrayList<>();  
        for(int i = 0; i< user.size(); i++) {  
            AdminUserDto adminUserDto = AdminUserDto.fromUser(user.get(i));  
            dtoList.add(adminUserDto);  
        }  
        return dtoList;  
    }  
  
  
  
    @GetMapping(value = "/comments")  
    public List<Comments> getComment(){  
        List<Comments> comments = commentsService.getCommentsAll();  
        List<CommentDto> dtoList = new ArrayList<>();  
        for(int i = 0; i< comments.size(); i++) {  
            CommentDto commentDto = CommentDto.fromComments(comments.get(i));  
            dtoList.add(commentDto);  
        }  
        return comments;  
    }  
  
  
  
    @DeleteMapping("/api/v1/admin/comments/{id}")  
    public void delete(@PathVariable(name = "id") Long id) {  
        commentsService.deleteComment(id);  
    }  
}

**Додаток Н  
Лістинг файлу «AuthenticationRestControllerV1.xml»**

package net.proselyte.jwtappdemo.rest;  
  
import net.proselyte.jwtappdemo.dto.AuthenticationRequestDto;  
import net.proselyte.jwtappdemo.model.User;  
import net.proselyte.jwtappdemo.security.jwt.JwtTokenProvider;  
import net.proselyte.jwtappdemo.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.BadCredentialsException;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.AuthenticationException;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.HashMap;  
import java.util.Map;  
  
@RestController  
@RequestMapping(value = "/api/v1/auth/")  
public class AuthenticationRestControllerV1 {  
  
    private final AuthenticationManager authenticationManager;  
  
    private final JwtTokenProvider jwtTokenProvider;  
  
    private final UserService userService;  
  
    @Autowired  
    public AuthenticationRestControllerV1(AuthenticationManager authenticationManager, JwtTokenProvider jwtTokenProvider, UserService userService) {  
        this.authenticationManager = authenticationManager;  
        this.jwtTokenProvider = jwtTokenProvider;  
        this.userService = userService;  
    }  
  
    @PostMapping("login")  
    public ResponseEntity login(@RequestBody AuthenticationRequestDto requestDto) {  
        try {  
            String username = requestDto.getUsername();  
            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, requestDto.getPassword()));  
            User user = userService.findByUsername(username);  
  
            if (user == null) {  
                throw new UsernameNotFoundException("User with username: " + username + " not found");  
            }  
  
            String token = jwtTokenProvider.createToken(username, user.getRoles());  
  
            Map<Object, Object> response = new HashMap<>();  
            response.put("username", username);  
            response.put("token", token);  
  
            return ResponseEntity.ok(response);  
        } catch (AuthenticationException e) {  
            throw new BadCredentialsException("Invalid username or password");  
        }  
    }  
}

**Додаток О  
Лістинг файлу «UserRestControllerV1.java»**

package net.proselyte.jwtappdemo.rest;  
  
import net.proselyte.jwtappdemo.dto.CommentDto;  
import net.proselyte.jwtappdemo.dto.UserDto;  
import net.proselyte.jwtappdemo.model.Comments;  
import net.proselyte.jwtappdemo.model.User;  
import net.proselyte.jwtappdemo.service.UserService;  
import net.proselyte.jwtappdemo.service.impl.CommentsService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.ArrayList;  
import java.util.List;

@RestController  
@RequestMapping(value = "/api/v1/users/")  
public class UserRestControllerV1 {  
    private final UserService userService;  
    private final CommentsService commentsService;  
  
  
    @Autowired  
    public UserRestControllerV1(UserService userService, CommentsService commentsService) {  
        this.userService = userService;  
        this.commentsService = commentsService;  
    }  
  
    @GetMapping(value = "{id}")  
    public ResponseEntity<UserDto> getUserById(@PathVariable(name = "id") Long id){  
        User user = userService.findById(id);  
  
        if(user == null){  
            return new ResponseEntity<>(HttpStatus.NO\_CONTENT);  
        }  
  
        UserDto result = UserDto.fromUser(user);  
  
        return new ResponseEntity<>(result, HttpStatus.OK);  
    }  
  
  
    @GetMapping(value = "/comments/{username}")  
    public ResponseEntity<CommentDto> getCommentByUsername(@PathVariable String username){  
        Comments comments = commentsService.findByUsername(username);  
  
        if(comments == null){  
            return new ResponseEntity<>(HttpStatus.NO\_CONTENT);  
        }  
  
        CommentDto result = CommentDto.fromComments(comments);  
  
        return new ResponseEntity<>(result, HttpStatus.OK);  
    }  
  
    @GetMapping(value = "/comments")  
    public List<Comments> getComment(){  
        List<Comments> comments = commentsService.getCommentsAll();  
        List<CommentDto> dtoList = new ArrayList<>();  
        for(int i = 0; i< comments.size(); i++) {  
            CommentDto commentDto = CommentDto.fromComments(comments.get(i));  
        dtoList.add(commentDto);  
        }  
        return comments;  
    }  
  
  
    }

**Додаток П  
Лістинг файлу «JwtAuthenticationException.java»**

package net.proselyte.jwtappdemo.security.jwt;  
  
import org.springframework.security.core.AuthenticationException;  
  
  
public class JwtAuthenticationException extends AuthenticationException {  
    public JwtAuthenticationException(String msg, Throwable t) {  
        super(msg, t);  
    }  
  
    public JwtAuthenticationException(String msg) {  
        super(msg);  
    }  
}

**Додаток Р  
Лістинг файлу «JwtConfigurer.java»**

package net.proselyte.jwtappdemo.security.jwt;  
  
import org.springframework.security.config.annotation.SecurityConfigurerAdapter;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.web.DefaultSecurityFilterChain;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
  
public class JwtConfigurer extends SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {  
    private JwtTokenProvider jwtTokenProvider;  
  
    public JwtConfigurer(JwtTokenProvider jwtTokenProvider) {  
        this.jwtTokenProvider = jwtTokenProvider;  
    }  
  
    @Override  
    public void configure(HttpSecurity httpSecurity) throws Exception {  
        JwtTokenFilter jwtTokenFilter = new JwtTokenFilter(jwtTokenProvider);  
        httpSecurity.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);  
    }  
}

**Додаток С  
Лістинг файлу «JwtTokenFilter.java»**

package net.proselyte.jwtappdemo.security.jwt;  
  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.web.filter.GenericFilterBean;  
  
import javax.servlet.FilterChain;  
import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
import javax.servlet.http.HttpServletRequest;  
import java.io.IOException;  
  
public class JwtTokenFilter extends GenericFilterBean {  
  
    private JwtTokenProvider jwtTokenProvider;  
  
    public JwtTokenFilter(JwtTokenProvider jwtTokenProvider) {  
        this.jwtTokenProvider = jwtTokenProvider;  
    }  
  
    @Override  
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain filterChain)  
            throws IOException, ServletException {  
  
        String token = jwtTokenProvider.resolveToken((HttpServletRequest) req);  
        if (token != null && jwtTokenProvider.validateToken(token)) {  
            Authentication auth = jwtTokenProvider.getAuthentication(token);  
  
            if (auth != null) {  
                SecurityContextHolder.getContext().setAuthentication(auth);  
            }  
        }  
        filterChain.doFilter(req, res);  
    }  
  
}

**Додаток Т  
Лістинг файлу «JwtTokenProvider.java»**

package net.proselyte.jwtappdemo.security.jwt;  
  
import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.Jws;  
import io.jsonwebtoken.JwtException;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import net.proselyte.jwtappdemo.model.Role;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Component;  
  
import javax.annotation.PostConstruct;  
import javax.servlet.http.HttpServletRequest;  
import java.util.ArrayList;  
import java.util.Base64;  
import java.util.Date;  
import java.util.List;  
  
@Component  
public class JwtTokenProvider {  
  
    @Value("${jwt.token.secret}")  
    private String secret;  
  
    @Value("${jwt.token.expired}")  
    private long validityInMilliseconds;  
  
  
    @Autowired  
    private UserDetailsService userDetailsService;  
  
    @Bean  
    public BCryptPasswordEncoder passwordEncoder() {  
        BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();  
        return bCryptPasswordEncoder;  
    }  
  
    @PostConstruct  
    protected void init() {  
        secret = Base64.getEncoder().encodeToString(secret.getBytes());  
    }  
  
    public String createToken(String username, List<Role> roles) {  
  
        Claims claims = Jwts.claims().setSubject(username);  
        claims.put("roles", getRoleNames(roles));  
  
        Date now = new Date();  
        Date validity = new Date(now.getTime() + validityInMilliseconds);  
  
        return Jwts.builder()//  
                .setClaims(claims)//  
                .setIssuedAt(now)//  
                .setExpiration(validity)//  
                .signWith(SignatureAlgorithm.HS256, secret)//  
                .compact();  
    }  
  
    public Authentication getAuthentication(String token) {  
        UserDetails userDetails = this.userDetailsService.loadUserByUsername(getUsername(token));  
        return new UsernamePasswordAuthenticationToken(userDetails, "", userDetails.getAuthorities());  
    }  
  
    public String getUsername(String token) {  
        return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getSubject();  
    }  
  
    public String resolveToken(HttpServletRequest req) {  
        String bearerToken = req.getHeader("Authorization");  
        if (bearerToken != null && bearerToken.startsWith("Bearer\_")) {  
            return bearerToken.substring(7, bearerToken.length());  
        }  
        return null;  
    }  
  
    public boolean validateToken(String token) {  
        try {  
            Jws<Claims> claims = Jwts.parser().setSigningKey(secret).parseClaimsJws(token);  
  
            if (claims.getBody().getExpiration().before(new Date())) {  
                return false;  
            }  
  
            return true;  
        } catch (JwtException | IllegalArgumentException e) {  
            throw new JwtAuthenticationException("JWT token is expired or invalid");  
        }  
    }  
  
    private List<String> getRoleNames(List<Role> userRoles) {  
        List<String> result = new ArrayList<>();  
  
        userRoles.forEach(role -> {  
            result.add(role.getName());  
        });  
  
        return result;  
    }  
}

**Додаток У  
Лістинг файлу «JwtUser.java»**

package net.proselyte.jwtappdemo.security.jwt;  
  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import net.proselyte.jwtappdemo.model.User;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.util.Collection;  
import java.util.Date;  
  
public class JwtUser implements UserDetails {  
  
    private final Long id;  
    private final String username;  
    private final String firstName;  
    private final String lastName;  
    private final String password;  
    private final String email;  
    private final boolean enabled;  
    private final Date lastPasswordResetDate;  
    private final Collection<? extends GrantedAuthority> authorities;  
  
    public JwtUser(  
            Long id,  
            String username,  
            String firstName,  
            String lastName,  
            String email,  
            String password, Collection<? extends GrantedAuthority> authorities,  
            boolean enabled,  
            Date lastPasswordResetDate  
    ) {  
        this.id = id;  
        this.username = username;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.email = email;  
        this.password = password;  
        this.authorities = authorities;  
        this.enabled = enabled;  
        this.lastPasswordResetDate = lastPasswordResetDate;  
    }  
  
    @JsonIgnore  
    public Long getId() {  
        return id;  
    }  
  
    @Override  
    public String getUsername() {  
        return username;  
    }  
  
    @JsonIgnore  
    @Override  
    public boolean isAccountNonExpired() {  
        return true;  
    }  
  
    @JsonIgnore  
    @Override  
    public boolean isAccountNonLocked() {  
        return true;  
    }  
  
    @JsonIgnore  
    @Override  
    public boolean isCredentialsNonExpired() {  
        return true;  
    }  
  
    public String getFirstname() {  
        return firstName;  
    }  
  
    public String getLastname() {  
        return lastName;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
  
    @JsonIgnore  
    @Override  
    public String getPassword() {  
        return password;  
    }  
  
    @Override  
    public Collection<? extends GrantedAuthority> getAuthorities() {  
        return authorities;  
    }  
  
    @Override  
    public boolean isEnabled() {  
        return enabled;  
    }  
  
    @JsonIgnore  
    public Date getLastPasswordResetDate() {  
        return lastPasswordResetDate;  
    }  
}

**Додаток Ф  
Лістинг файлу «JwtUserFactory.java»**

package net.proselyte.jwtappdemo.security.jwt;  
  
import net.proselyte.jwtappdemo.model.Role;  
import net.proselyte.jwtappdemo.model.Status;  
import net.proselyte.jwtappdemo.model.User;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public final class JwtUserFactory {  
  
    public JwtUserFactory() {  
    }  
  
    public static JwtUser create(User user) {  
        return new JwtUser(  
                user.getId(),  
                user.getUsername(),  
                user.getFirstName(),  
                user.getLastName(),  
                user.getEmail(),  
                user.getPassword(),  
                mapToGrantedAuthorities(new ArrayList<>(user.getRoles())),  
                user.getStatus().equals(Status.ACTIVE),  
                user.getUpdated()  
        );  
    }  
  
    private static List<GrantedAuthority> mapToGrantedAuthorities(List<Role> userRoles) {  
        return userRoles.stream()  
                .map(role ->  
                        new SimpleGrantedAuthority(role.getName())  
                ).collect(Collectors.toList());  
    }  
}

**Додаток Х  
Лістинг файлу «JwtUserDetailsService.java»**

package net.proselyte.jwtappdemo.security;  
  
import lombok.extern.slf4j.Slf4j;  
import net.proselyte.jwtappdemo.model.User;  
import net.proselyte.jwtappdemo.security.jwt.JwtUser;  
import net.proselyte.jwtappdemo.security.jwt.JwtUserFactory;  
import net.proselyte.jwtappdemo.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
@Service  
@Slf4j  
public class JwtUserDetailsService implements UserDetailsService {  
  
    private final UserService userService;  
  
    @Autowired  
    public JwtUserDetailsService(UserService userService) {  
        this.userService = userService;  
    }  
  
    @Override  
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
        User user = userService.findByUsername(username);  
  
        if (user == null) {  
            throw new UsernameNotFoundException("User with username: " + username + " not found");  
        }  
  
        JwtUser jwtUser = JwtUserFactory.create(user);  
        log.info("IN loadUserByUsername - user with username: {} successfully loaded", username);  
        return jwtUser;  
    }  
}

**Додаток Ц  
Лістинг файлу «UserService.java»**

package net.proselyte.jwtappdemo.service;  
  
import net.proselyte.jwtappdemo.model.User;  
  
import java.util.List;  
  
public interface UserService {  
  
    User register(User user);  
  
    List<User> getAll();  
  
    User findByUsername(String username);  
  
    User findById(Long id);  
  
    void delete(Long id);  
}

**Додаток Ч  
Лістинг файлу «CommentService.java»**

package net.proselyte.jwtappdemo.service;  
  
import net.proselyte.jwtappdemo.dto.CommentDto;  
import net.proselyte.jwtappdemo.model.Comments;  
  
import java.util.List;  
  
public interface CommentService {  
    CommentDto createComment(CommentDto commentDto);  
  
    CommentDto createComment(Comments comments);  
  
    List<Comments> getCommentsAll();  
  
  
    Comments findByUsername(String username);  
  
    CommentDto updateComment(long commentId, CommentDto commentRequest);  
  
    void deleteComment(Long commentId);  
  
  
}

**Додаток Ш  
Лістинг файлу «UserServiceImpl.java»**

package net.proselyte.jwtappdemo.service.impl;  
  
import lombok.extern.slf4j.Slf4j;  
import net.proselyte.jwtappdemo.model.Role;  
import net.proselyte.jwtappdemo.model.Status;  
import net.proselyte.jwtappdemo.model.User;  
import net.proselyte.jwtappdemo.repository.RoleRepository;  
import net.proselyte.jwtappdemo.repository.UserRepository;  
import net.proselyte.jwtappdemo.service.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.List;  
  
  
@Service  
@Slf4j  
public class UserServiceImpl implements UserService {  
  
    private final UserRepository userRepository;  
    private final RoleRepository roleRepository;  
    private final BCryptPasswordEncoder passwordEncoder;  
  
    @Autowired  
    public UserServiceImpl(UserRepository userRepository, RoleRepository roleRepository, BCryptPasswordEncoder passwordEncoder) {  
        this.userRepository = userRepository;  
        this.roleRepository = roleRepository;  
        this.passwordEncoder = passwordEncoder;  
    }  
  
    @Override  
    public User register(User user) {  
        Role roleUser = roleRepository.findByName("ROLE\_USER");  
        List<Role> userRoles = new ArrayList<>();  
        userRoles.add(roleUser);  
  
        user.setPassword(passwordEncoder.encode(user.getPassword()));  
        user.setRoles(userRoles);  
        user.setStatus(Status.ACTIVE);  
  
        User registeredUser = userRepository.save(user);  
  
        log.info("IN register - user: {} successfully registered", registeredUser);  
  
        return registeredUser;  
    }  
  
    @Override  
    public List<User> getAll() {  
        List<User> result = userRepository.findAll();  
        log.info("IN getAll - {} users found", result.size());  
        return result;  
    }  
  
    @Override  
    public User findByUsername(String username) {  
        User result = userRepository.findByUsername(username);  
        log.info("IN findByUsername - user: {} found by username: {}", result, username);  
        return result;  
    }  
  
    @Override  
    public User findById(Long id) {  
        User result = userRepository.findById(id).orElse(null);  
  
        if (result == null) {  
            log.warn("IN findById - no user found by id: {}", id);  
            return null;  
        }  
  
        log.info("IN findById - user: {} found by id: {}", result);  
        return result;  
    }  
  
    @Override  
    public void delete(Long id) {  
        userRepository.deleteById(id);  
        log.info("IN delete - user with id: {} successfully deleted");  
    }  
}

**Додаток Щ  
Лістинг файлу «CommentsService.java»**

package net.proselyte.jwtappdemo.service.impl;  
  
import lombok.extern.slf4j.Slf4j;  
import net.proselyte.jwtappdemo.dto.CommentDto;  
import net.proselyte.jwtappdemo.model.Comments;  
import net.proselyte.jwtappdemo.repository.CommentsRepository;  
import net.proselyte.jwtappdemo.service.CommentService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
@Slf4j  
public class CommentsService implements CommentService {  
  
    private final CommentsRepository commentsRepository;  
  
    @Autowired  
    public CommentsService(CommentsRepository commentsRepository) {  
        this.commentsRepository = commentsRepository;  
    }  
  
    @Override  
    public CommentDto createComment(CommentDto commentDto) {  
        return null;  
    }  
  
    @Override  
    public CommentDto createComment(Comments comments) {  
        return null;  
    }  
  
    @Override  
    public List<Comments> getCommentsAll() {  
        List<Comments> result = commentsRepository.findAll();  
        log.info("IN getAll - {} comments found", result.size());  
        return result;  
    }  
  
  
  
    @Override  
    public Comments findByUsername(String username) {  
        Comments result = commentsRepository.findByName(username);  
        log.info("IN findByUsername - user: {} found by username: {}", result, username);  
        return result;  
    }  
  
    @Override  
    public CommentDto updateComment(long commentId, CommentDto commentRequest) {  
        return null;  
    }  
  
    @Override  
    public void deleteComment(Long id) {  
        commentsRepository.deleteById(id);  
        log.info("IN delete - comment with id: {} successfully deleted");  
    }  
}

**Додаток Ю  
Лістинг файлу «db.changeloge-master.xml»**

<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog" xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance>   
                   xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  
                        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.4.xsd">  
    <include file="classpath:liquibase/db.changelog-1.0.xml"/>  
</databaseChangeLog>

**Додаток Я**

**Лістинг файлу «db.changeloge-master.xml»**

<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"  
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
                   xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  
                        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.4.xsd">  
  
  
    <changeSet id="1.0.0" author="zeroitslife">  
        <createTable tableName="users">  
            <column name="id" type="BIGINT" autoIncrement="true">  
                <constraints primaryKey="true" nullable="false"/>  
            </column>  
  
            <column name="username" type="VARCHAR(100)">  
                <constraints nullable="false" unique="true"/>  
            </column>  
  
            <column name="email" type="VARCHAR(255)">  
                <constraints nullable="false" unique="true"/>  
            </column>  
  
            <column name="first\_name" type="VARCHAR(100)">  
                <constraints nullable="false"/>  
            </column>  
  
            <column name="last\_name" type="VARCHAR(100)">  
                <constraints nullable="false"/>  
            </column>  
  
            <column name="password" type="VARCHAR(255)">  
                <constraints nullable="false"/>  
            </column>  
  
            <column name="created" type="TIMESTAMP" defaultValue="CURRENT\_TIMESTAMP()">  
                <constraints nullable="false"/>  
            </column>  
  
            <column name="updated" type="TIMESTAMP" defaultValue="CURRENT\_TIMESTAMP()">  
                <constraints nullable="false"/>  
            </column>  
  
            <column name="status" type="VARCHAR(25)" defaultValue="ACTIVE">  
                <constraints nullable="false"/>  
            </column>  
  
        </createTable>  
    </changeSet>  
  
    <changeSet id="1.0.1" author="zeroitslife">  
        <createTable tableName="roles">  
            <column name="id" type="BIGINT" autoIncrement="true">  
                <constraints primaryKey="true" nullable="false"/>  
            </column>  
  
            <column name="name" type="VARCHAR(100)">  
                <constraints nullable="false" unique="true"/>  
            </column>  
  
            <column name="created" type="TIMESTAMP" defaultValue="CURRENT\_TIMESTAMP()">  
                <constraints nullable="false"/>  
            </column>  
  
            <column name="updated" type="TIMESTAMP" defaultValue="CURRENT\_TIMESTAMP()">  
                <constraints nullable="false"/>  
            </column>  
  
            <column name="status" type="VARCHAR(25)" defaultValue="ACTIVE">  
                <constraints nullable="false"/>  
            </column>  
        </createTable>  
    </changeSet>  
  
    <changeSet id="1.0.2" author="zeroitslife">  
        <createTable tableName="user\_roles">  
            <column name="user\_id" type="BIGINT">  
            </column>  
  
            <column name="role\_id" type="BIGINT">  
            </column>  
        </createTable>  
    </changeSet>  
  
    <changeSet id="1.0.3" author="zeroitslife">  
        <addForeignKeyConstraint baseColumnNames="user\_id"  
                                 baseTableName="user\_roles"  
                                 constraintName="fk\_user\_roles\_user"  
                                 onDelete="CASCADE"  
                                 onUpdate="RESTRICT"  
                                 referencedColumnNames="id"  
                                 referencedTableName="users"/>  
    </changeSet>  
  
    <changeSet id="1.0.4" author="zeroitslife">  
        <addForeignKeyConstraint baseColumnNames="role\_id"  
                                 baseTableName="user\_roles"  
                                 constraintName="fk\_user\_roles\_roles"  
                                 onDelete="CASCADE"  
                                 onUpdate="RESTRICT"  
                                 referencedColumnNames="id"  
                                 referencedTableName="roles"/>  
    </changeSet>  
  
    <changeSet id="1.0.5" author="zeroitslife">  
        <insert tableName="roles">  
            <column name="name" value="ROLE\_USER"/>  
        </insert>  
  
        <insert tableName="roles">  
            <column name="name" value="ROLE\_ADMIN"/>  
        </insert>  
    </changeSet>  
  
    <changeSet id="1.0.6" author="zeroitslife">  
        <createTable tableName="comments">  
            <column name="id" type="BIGINT" autoIncrement="true">  
                <constraints primaryKey="true" nullable="false"/>  
            </column>  
  
            <column name="username" type="VARCHAR(100)">  
                <constraints nullable="false" unique="true"/>  
            </column>  
  
            <column name="email" type="VARCHAR(255)">  
                <constraints nullable="false" unique="true"/>  
            </column>  
  
            <column name="body" type="VARCHAR(255)">  
                <constraints nullable="false" unique="true"/>  
            </column>  
  
            <column name="created" type="TIMESTAMP" defaultValue="CURRENT\_TIMESTAMP()">  
                <constraints nullable="false"/>  
            </column>  
  
            <column name="updated" type="TIMESTAMP" defaultValue="CURRENT\_TIMESTAMP()">  
                <constraints nullable="false"/>  
            </column>  
            <column name="status" type="VARCHAR(25)" defaultValue="NONE">  
                <constraints nullable="false"/>  
            </column>  
        </createTable>  
    </changeSet>  
  
  
  
</databaseChangeLog>

**Додаток AA  
Лістинг файлу «configuration.properties»**

server.port=8075  
  
spring.datasource.url=jdbc:mysql://localhost:3306/jwtappdemo?serverTimezone=UTC  
spring.datasource.driver-class-name=com.mysql.jdbc.Driver  
spring.datasource.username=root  
spring.datasource.password=root  
  
spring.liquibase.change-log=classpath:liquibase/db.changelog-master.xml  
spring.liquibase.url=jdbc:mysql://localhost:3306/jwtappdemo?serverTimezone=UTC  
spring.liquibase.user=root  
spring.liquibase.password=root  
  
jwt.token.secret=jwtappdemo  
jwt.token.expired=3600000