МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

РОЗРОБКА WEB-РЕСУРСУ ЗА ДОПОМОГОЮ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

Здобувач освіти гр. ІН – 81 Олександра ПУГАЧ

Науковий керівник,
кандидат фізико-математичних наук,
доцент кафедри комп'ютерних наук Сергій ШАПОВАЛОВ

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-81 спеціальності «122 – Комп'ютерні науки» денної форми навчання Пугач Олександри Андріївни.

Тема: «РОЗРОБКА WEB-РЕСУРСУ ЗА ДОПОМОГОЮ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) огляд існуючих рішень; 2) методика вирішення поставлених задач; 3)проктування web-системи; 4) розробка web-додатку.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Сергій ШАПОВАЛОВ
Завдання прийняв до виконання _____ Олександра ПУГАЧ

РЕФЕРАТ

Записка: 41 стор., 13 рис., 9 таблиць, 1 додаток, 14 інформаційних джерел.

Об’єкт дослідження — веб-ресурс для перекладу та читання манги з можливістю легкого масштабування

Мета роботи — розробка серверної частини веб-ресурсу на базі мікросервісної архітектури.

Результати — здійснено огляд існуючих рішень. На основі аналізу недоліків та переваг ресурсів було сформовано постановку задачі. Спроектовано відповідні бази даних для збереження та обробки інформації сервісами. Розроблено API з дотриманням RESTful архітектури. Розгорнуто продукт на сервері та протестовано. Покрито технологією Apigee, за допомогою якої виправлено помилки та додано функціонал для моніторингу системи.

ВЕБ-СЕРВІС, ВЕБ-ДОДАТОК, RESTFUL API, МІКРОСЕРВІСНА АРХІТЕКТУРА

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6

ВСТУП 7

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 9

1.1 Аналіз існуючих сайтів для читання книг або манги 9

1.1.1 https://manga.in.ua/ 9

1.1.2 https://readmanga.io 10

1.2 Постановка задачі 11

2 МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ 12

2.1 Аналіз існуючих архітектурних підходів 12

2.2 RESTful API 16

3 ПРОЕКТУВАННЯ WEB-СИСТЕМИ 18

3.1 Огляд ролей та їх можливостей 18

3.1.1 Юзер 18

3.1.2 Перекладач 19

3.1.3 Модератор 19

3.1.4 Адміністратор 19

3.2 Проектування функціоналу додатку за сервісами 20

3.3 Розподіл доступу сервісного функціоналу за ролями 22

3.4 Проектування бази даних 23

4 РОЗРОБКА WEB-ДОДАТКУ 27

4.1 Серверна частина веб-додатку 27
4.1.1 Мова програмування 27
4.1.2 Spring Framework 28
4.1.3 Об’єктно орієнтоване програмування 28
4.2 Захист та безпека системи 31
4.2.1 Обмеження доступу прав користувачів за ролями 31
4.2.2 JWT токен 32
4.3 Тестування веб додатку 34
4.4 Розгортання веб додатку 35
4.5 Покриття додатку APIGEE 36
ВИСНОВКИ 39
СПИСОК ЛІТЕРАТУРИ 40
ДОДАТКИ 42
Додаток А. Лістинг програмного коду 42

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI – User interface

REST – Representational State Transfer,

API – Application Program Interface

JSON – JavaScript Object Notation

SQL – Structured Query Language

CQL – Cassandra Query Language

СУБД – система управління базами даних

ООП – об’єктно-орієнтоване програмування

MVC – Model-View-Controller

JWT – JSON Web Token

GCP – Google Cloud Platform

ВСТУП

Наразі велика кількість бізнесу потребують власних WEB-додатків розгорнутих у мережі. Коронавірус переніс майже всі процеси до Інтернету. Коли немає можливості вийти з дому для придбання якихось речей продажі можливі лише у онлайн-магазинах, тому всі компанії що працювали офлайн втрачають прибуток. У таких компаній звичайно є потреба якомога швидше розгорнути власний онлайн сервіс, але ж таких компаній мільйони і розробникам надто важко виконати потреби всіх замовників, через що потрібні швидкі інструменти та методи розробки Інтернет ресурсів.

Багато сервісів ще досі потребують розробки, а зараз під час війни – ще більше. Велика кількість українців користувалась російськими Інтеренет ресурсами і зараз відмовляється від них у користь українських продуктів. Це ж стосується і Інтернет ресурсів. Якщо ще кілька місяців тому ми мали лише російськомовні переклади манги, то зараз українці у своєму бажанні: поширювати українську мову та не використовувати мову ворога, створюють свої власні спільноти перекладачів манги на українську мову і можуть скласти гарну конкуренцію російськомовним сайтам.

Також це цікаво для бізнесу, бо ж кількість перегляду сторінок сайту дає можливість розміщувати рекламу та заробити великі кошти. Тому зараз ми створюємо власні україномовні сервіси, що будуть аналогами російським, у короткий час, щоб позбавити російський ринок прибутку зароблено з українських користувачів.

Метою роботи є створення WEB-додатку для читання та перекладу манги за максимально короткий час. Так як раніше читали книжки, зараз полюбляють читати комікси, або азіатськи комікси – мангу. Загалом платформі для читання манги містять у собі велику структуру, яка направлена не лише на читання. Так, ресурс може пропонувати такі можливості:

- Створення власної команди перекладачів, які самостійно можуть заливати мангу
- Наявність команди модераторів, які перевіряють порядок на ресурсі

- Наявність інструментів для підтримки ком’юніті читачів (чати, коментарі, обговорення).

Додаток створений у роботі має містити у собі подібні інструменти роботи ресурсу.

У роботі будуть представлені можливі методи та інструменти для швидкої розробки додатків.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз існуючих сайтів для читання книг або манги

1.1.1 https://manga.in.ua/

Переглянемо один в україномовних ресурсів для читання манги українською мовою.

Недоліки:

- Немає можливості створювати групи перекладачів, можливість додавання перекладу можливий лише з аккаунту одного користувача. Так як зазвичай декілька перекладачів, клінерів та тайперів працюють в одній команді це унеможливлює викладати нові переклади не однією людиною, а групою перекладачів, де відповідальність за додавання нової глави можуть мати декілька працівників
- Сортування манги відбувається лише за чотирма фільтрами, які не надають зручності для користувачів. Наприклад, немає сортування за датою викладання нової глави, або за популярністю манги
- Є навігація по розділам, але без наявності обкладинок манги, навіть при наведенні на назву манги, також все ж неможливо відсортувати мангу за жанром
- Сайт запам’ятовує кожен крок, що був виконаний користувачем і при спробі вийти з ресурсу кнопкою назад можна спостерігати всі відновленні кроки у зворотному порядку

Тобто на цьому ресурсі для того, щоб викласти новий переклад команда перекладачів має мати один аккаунт людини, яка буде завжди викладати нові переклади та надавати інформацію про їх колекцію перекладених глав, бо інші просто не мають доступу перевірити список або кількість манг, які команда наразі перекладає.

А користувачі можуть відібрати мангу по жанру, але не будуть бачити, яка з них недавно отримала нові глави з перекладом, або ж можуть побачити мангу, яка нещодавно отримала нові глави, але не мати змоги переглянути лише жанр, який її цікавить.

Переваги:

- Є змога викладати власний переклад
- Переклад манги відбувається лише одним аккаунтом, тобто до вашого перекладу не зможуть додати чийсь інший
- Є можливість листування та коментарів
- Зворотній зв’язок з керівниками ресурсу у Telegram

Загалом це хороший ресурс для використання, але другий пункт у перевагах може також створювати проблеми, так як у випадку, коли перекладачі перестануть перекладати ту чи іншу мангу зі своїх причин, задля відновлення перекладу потрібно буде або створювати нову мангу, або вирішувати питання на рівні розробки і, можливо, вручну додавати мангу до списку інших перекладачів.

1.1.2 https://readmanga.io

Переглянемо російськомовний ресурс

Недоліки:

- не сучасний дизайн

Лише один недолік у ресурсу говорить про добре виконану роботу над ним.

Переваги:

- декілька поєднаних ресурсів, які несуть у собі розподіл великих категорій
- можна переглядати рецензії на мангу, а не просто коментарі
- є можливість створення авторських колекцій користувачами
- є можливість підбору цікавої для користувача манги за його попередніми уподобаннями
- є кнопка випадкової манги для прочитання
- зв’язок з модераторами сайту через сам сайт
- є провідник по сайту (гайд у використанні)

- велика кількість інструментів комунікації з іншими користувачами: можливість додавати друзів та входити у різні групи
- перегляд та додавання різних закладинок до манги
- багато фільтрів для отримання списку цікавої манги, наприклад за автором, або художником

Як можна зрозуміти, маємо гігантську еко-систему з неймовірними можливостями як для користувачів, так і для перекладачів. А український аналог, досі ще є куди розвивати.

1.2 Постановка задачі

Для роботи веб-ресурсу, необхідно реалізувати наступні задачі:

1. виконати огляд інструментів для розробки та обрати оптимальні;
2. виконати проектування архітектури веб-додатка;
3. розробити сервіси роботи веб-додатку;
4. створити за розмістити відповідні до сервісів бази даних;
5. розгорнути сервіси на сервері;
6. виконати тестування сервісів;
7. покрити сервіси Apigee та обробити знайдені помилки;
8. об’єднати сервіси спільним доступом;
9. налагодити роботу моніторингу даних додатку.

Після виконання вище зазначених пунктів веб-ресурс надасть можливість моніторингу манги та користувачів, залишати відгуки на мангу, створювати команди перекладачів. А бізнес-клієнт отримає можливість аналізувати дані додатку, для розуміння подальшого розвитку продукту та встановлення нових задач.

2 МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Аналіз існуючих архітектурних підходів

Від обраної для проекту архітектури залежить подальша робота команди розробників та інші рішення для розробки додатку. Будь-який додаток, навіть, самий маленький потребує того рішення, яке дозволить спроектувати масштабований продукт для подальшого розвитку. Швидкість розробки додатку також залежить від обраної архітектури сервісу. Маємо два види найпопулярніших архітектурних рішень для WEB-сервісів: монолітну та мікросервісну. Оглянемо спочатку монолітну архітектуру.

1) Монолітна архітектура

Монолітна архітектура до недавнього часу вважалася традиційним підходом в проектуванні додатків. Саме слово «моноліт» говорить про давно усталені правила, які нікуди не поворухнулись. Але ж в наш час, це вже може насторожити, оскільки не змінюється – значить не адаптується. ІТ сфера дуже швидко розробляє нові рішення для розробки продуктів, тому всі інші рішення приймають зміни і підлаштовуються, те що монолітна архітектура не набула перевтілень, говорить про її неефективність у нашому часі.

Монолітна архітектура припускає наявність єдиної платформи для усіх компонентів додатку, що добре проілюстровано на рисунку 2.1. Це впливає на розміщення системи в одному місці, без можливості перенесення окремих частин на інші носії. Така собі централізована система, яка розсиплеться відразу повністю при виникненні проблеми хоча б десь.

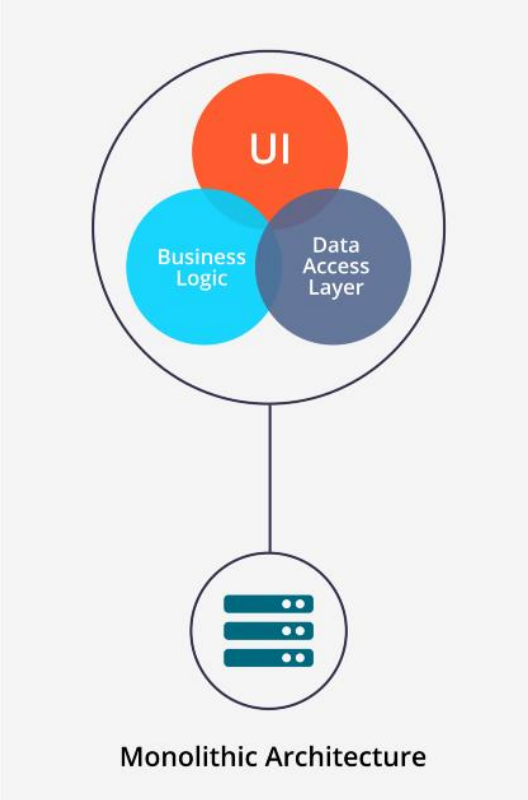

Рисунок 2.1 - Монолітна архітектура [1]

Але поговоримо про переваги монолітної архітектури:

- Програмне забезпечення, що проектувалося на основі монолітної архітектури має велику швидкодію, так як всі процеси поєднані між собою в одному місці
- Легше у розробці для невеликої команди розробників, так як не має потреби у розробці складних комунікаційних процесів між компонентами

Щодо недоліків монолітної архітектури:

- «Можливості розширення обмежені. Монолітні програми можуть бути розширені тільки повністю.»[2] Якщо якоїсь функції не закладено з початку проектування, додати її наприкінці розробки неможливо без перепису великої кількості коду;
- Через те, що всі компоненти знаходяться в одному місці виникнення помилки може зупинити роботу всього додатку;
- Незмінні у використанні інструменти. Через те, що маємо єдиний програмний код ніяких новітніх впроваджень, або більш відповідних

до поставлених задач інструментів використати неможливо. Особливо погано це впливає на довгострокові проекти, які потрібно дописувати через багато років використовуючи методи та інструменти на яких додаток і було написано.

- Не зовсім зручна для розробки великого проекту, де працює велика кількість розробників, бо кожен раз потрібно перевіряти чи ще не переписана якась логіка іншим розробником для відтворення своїх задач. Також через об’ємний код потрібно розбиратися у всіх зв’язках компонентів і ще й намагатися зрозуміти код інших розробників.

Отож маємо не зовсім зручну для новітніх рішень архітектуру. Із перерахованих вище недоліків зрозуміло чому зараз надають перевагу саме мікросервісній архітектурі.

2) Мікросервісна архітектура

Легко адаптивна архітектура, яка вирішує усі недоліки монолітної архітектури. Також дуже приємна у розробці, так як не потребує набору якихось одних інструментів, і може одночасно розроблятися великими групами розробників одночасно. Мікросервісна архітектура будується за принципом розподіленої роботи між окремими компонентами, як зображено на рисунку 2.2.

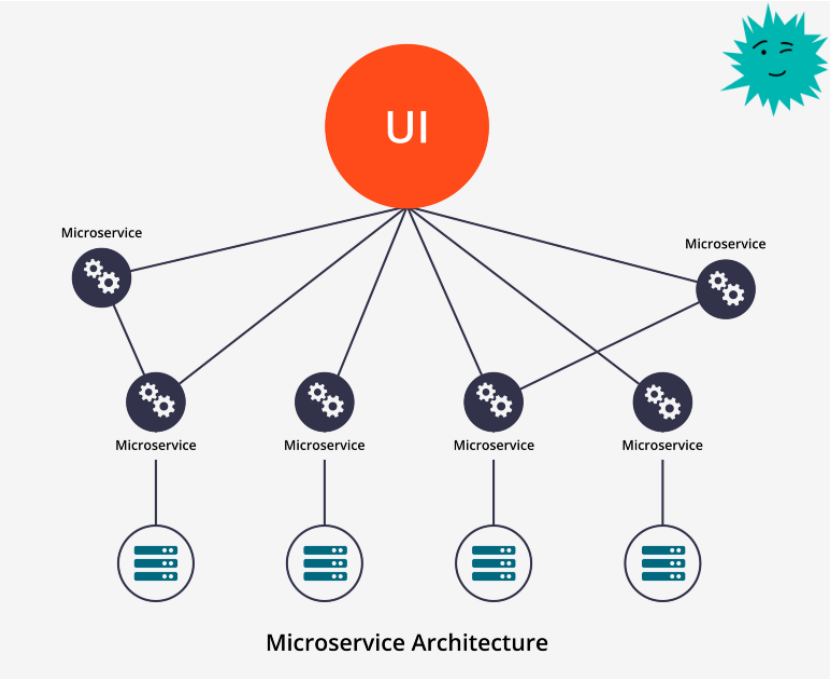

Рисунок 2.2 - Мікросервісна архітектура [1]

Переваги мікросервісної архітектури:

- Не залежить від вибору інструментів. Через те, що всі компоненти існують окремо не потребує єдиного рішення, компоненти можуть розроблятися за допомогою різних мов програмування;
- Через те, що велика кількість розробників працює одночасно маємо досить невеликий проміжок часу від початку проектування проекту та його запуском;
- Масштабованість дозволяє у будь-який час розширювати проект далі, а частини, що вже були написані можуть спокійно функціонувати так як і раніше;
- Помилка, що виникла у проекті може зупинити роботу окремого компоненту, але не всього додатку разом[3];
- Комунікація за допомогою запитів HTTP. Це дає змогу розміщувати окремі компоненти згідно з їх потребами, де потрібна швидка база даних або така, щоб могла зберігати величезну кількість інформації, де сам сервіс повинен швидко працювати. І все це можна підібрати з вигіднішим тарифним планом.

Недоліки мікросервісної архітектури:

- Велика кількість запитів, що надходять одночасно, через зв’язок сервісів з їх допомогою;
- Складність в управлінні версіями програмного забезпечення.

Судачі з аналізу мікросервісна архітектура є більш стійкою до помилок, масштабованою та надає більшу кількість інструментів для вирішення задач. Недарма сучасні веб ресурси приділяють так багато уваги саме цій архітектурі.

Вигідніше за все для розробки додатку використати мікросервісну архітектуру. Вона дозволить випустити продукт за короткий час з мінімальною кількістю функціоналу та додати інший трохи згодом.

2.2 RESTful API

«REST API (також відомий як RESTful API) — це інтерфейс програмування прикладних програм (API або веб-API), який відповідає обмеженням архітектурного стилю REST і дозволяє взаємодіяти з веб-сервісами RESTful.»[4]

REST притримується концепції CRUD. Тобто будь-яка система основана на архітектурному стилю REST вірогідніше за все має базові операції, назва кожної з яких і складає акронім CRUD, а саме: create, read. update, delete. Відповідно базовим функціям будуть відповідати так знайомі нам http запити: POST, GET, PUT/PATCH, DELETE.

Обираючи REST API ми вже зменшуємо навантаження проектування додатку, так як просто наявність базових функцій дозволяє проектувати додатковий функціонал відштовхуючись від примітивів, що значно полегшує роботу, бо працювати з чистим аркушем набагато складніше.

REST API обробляє запити за даними URL, тому при проектуванні слід звернути увагу на правильну назву ендпоінтів згідно з RESTful архітектурою. Наприклад:

- Не слід вживати у назві дієслова;
- Частіше іменники вживаються множині.

Тепер щодо загальних рекомендацій використання RESTful архітектури:

- Обов’язкове використання Status Code у обробці помилок із зрозумілим поясненням помилки;
- Загалом всі http запити слід супроводжувати зрозумілими поясненнями та статусом коду, будь це успішна операція, помилка чи введення невалідних даних;
- Використання фільтрації, сортування та розбиття на сторінки, щоб отримати запитані дані;
- Зрозуміла, чітко сформована документація.

Також при написанні коду слід звернути увагу, що архітектура підтримує як XML так і JSON, але можливості другого значно перевищують першого, через що більшість додатків віддають перевагу JSON.

3 ПРОЕКТУВАННЯ WEB-СИСТЕМИ

3.1 Огляд ролей та їх можливостей

Існують різні методики проектування продукту, . Почнемо з представлення хто і яким чином зможе використовувати додаток. Тому розподілимо можливості за такими ролями: Юзер, Перекладач, Модератор та Адміністратор.

3.1.1 Юзер

Для початку, щоб отримати саме роль Юзера потрібно зареєструватися, отож реєстрація та логін доступні користувачам без авторизації. Також відвідувачам без аккаунту можна переглядати мангу, а ось залишати коментарі/оцінки лише зареєстрованим користувачам, які вже увійшли до системи.

Юзери не мають змоги переглядати усіх користувачів, або інформацію один про одного. Таке рішення було прийняте через те, що додаток не надає можливості створювати повноцінний профіль, як у соцмережах, це непотрібний функціонал, на який буде витрачено зайвий час. Натомість додаток має надавати можливості для спілкування та розвитку ком'юніті, тому у додатку повинна бути закладена можливість для комунікації між Юзерами. Для подальших рішень додатку залишається можливим додавання друзів через id Юзерів, як наприклад у Discord. Юзери зможуть запропонувати дружбу через логін користувача, що залишив коментар, або через спільні обговорення.

Юзери можуть входити до груп перекладачів, але додавати до такої групи може лише її учасник. Додатком не передбачено можливості участі у двох або більше групах з аккаунту одного Юзера.

Також Юзер має можливість редагувати свій профіль.

3.1.2 Перекладач

Перекладач, це Юзер, якого було додано до групи перекладачів. Насправді, так як у більшості випадків перекладачі працюють одразу з тайперами та клінерами, група передбачає і їх наявність також, але так як наразі ніякого окремо доступного для цих категорій користувачів функціоналу не передбачено, їм достатньо однакового доступу, з однаковими можливостями.

Окрім функціоналу, що доступний юзеру, перед Перекладачем відкривається велика кількість різноманітних операцій з мангою та роботою з групами.

Перекладач може створювати мангу, оновлювати інформацію про неї. Але, для того, щоб Перекладач міг працювати з мангою, вона повинна бути закріплена за тією ж групою до якої входить користувач.

Отримуючи роль Перекладач від Адміністратора, а не від інших учасників групи, користувач може створити власну групу, редагувати зміни у ній. Також має змогу переглядати звіти щодо манги, яка знаходиться на перекладі у цієї команди, або ж інших.

Користувач, який має роль Перекладач може додавати до групи інших користувачів.

3.1.3 Модератор

Роль Модератор надає можливості керувати перекладом манги та групами перекладачів, також є можливість моніторингу відгуків.

Так, Модератор може видаляти мангу, групи та коментарі, які вважає правопорушенням згідно з умов користування додатком.

3.1.4 Адміністратор

Користувач, що має роль Адміністратор керує ролями – може змінювати ролі користувачів, надавати більше прав, або менше. Також Адміністратор має

доступ до акаунту користувачів. Загалом ця роль дозволяє керувати доступом до ресурсу та аккаунтами користувачів.

3.2 Проектування функціоналу додатку за сервісами

Враховуючи поставлене завдання та можливості обраних рішень, роботу додатку можна поділити на п’ять окремих сервісів. Кожен сервіс повинен мати свою базу даних. Роботу можна розподілити таким чином:

- Security service відповідає за безпеку та авторизацію у додатку. Ми можемо зареєструватися, залогінитися за допомогою додатку, також додаткові можливості типу: зміни паролю та зміни ролі теж можуть знаходитися у цьому сервісі
- User service зберігає дані про користувачів, надає можливості управління цією інформацією: збереження, видалення, редагування, видалення
- Manga service займається усім, що пов’язано із мангою, відповідно до прав користувачів за ролями можна створювати нову мангу, редагувати її;
- Team service об’єднує декілька користувачів у групи, які надалі можуть працювати з мангою, яка також належить до даної групи. Також можливе управління самими групами: редагування, створення, видалення, але найголовніше те, що можна додати нового юзера до групи;
- Review service – це базовий сервіс для коментарів або відгуків, рецензій, у подальшому його можна використати, як основу для тих же рецензій і відгуків, бо логіка з коментарями однакова. Сам сервіс працює з коментарями, приймає текст, зберігає поле свого автора та дату написання.

Таким чином, маємо спроектовані сервіси з їх ендпоінтами, які позначають функціонал даних сервісів у таблиці 3.1.

Таблиця 3.1 - Спроектований функціонал у вигляді запитів

Назва сервісу	**Ендпоінти**
Security service	POST /authenticate/register POST /authenticate/login GET / authenticate/{id} PATCH /authenticate/{Id}/changePassword PATCH /authenticate/{Id}/changeRole
User service	POST /users GET /users GET /users/{id} PATCH /users/{id} DELETE /users/{id} PATCH /users/{id}/addTeam
Manga service	POST /mangas GET /mangas GET /mangas/{id} PATCH / mangas/{id} DELETE / mangas/{id} PATCH / mangas/{id}/addTeam PATCH / mangas/{id}/completed
Team service	POST /teams GET /teams GET /teams/{id} PATCH / teams/{id} DELETE / teams/{id} GET /report GET / teams/{id}/report
Review service	POST /reviews

	GET /reviews GET /reviews/{id} PATCH / reviews/{id} DELETE / reviews/{id}

Тепер, маючи розподілену схему функціоналу між сервісами, бачимо вже зрозумілу картину роботи додатку. Розуміння того як працює система вже дозволяє починати розробку, так як функціонал кожного сервісу чітко описано в таблиці. Але це ще не завершальний етап проектування.

3.3 Розподіл доступу сервісного функціоналу за ролями

Поєднуючи у собі попередні розділи можна сформувати повноцінну роботу системи. Додавши до мікросервісного функціоналу доступ за ролями, отримаємо структуру з обмеженнями у правах, що потрібно реалізувати при написанні коду. З цією інформації можна починати розробку додатку. На рисунку 3.1 схематично зображено доступ до ендпоінтів в залежності від ролі.

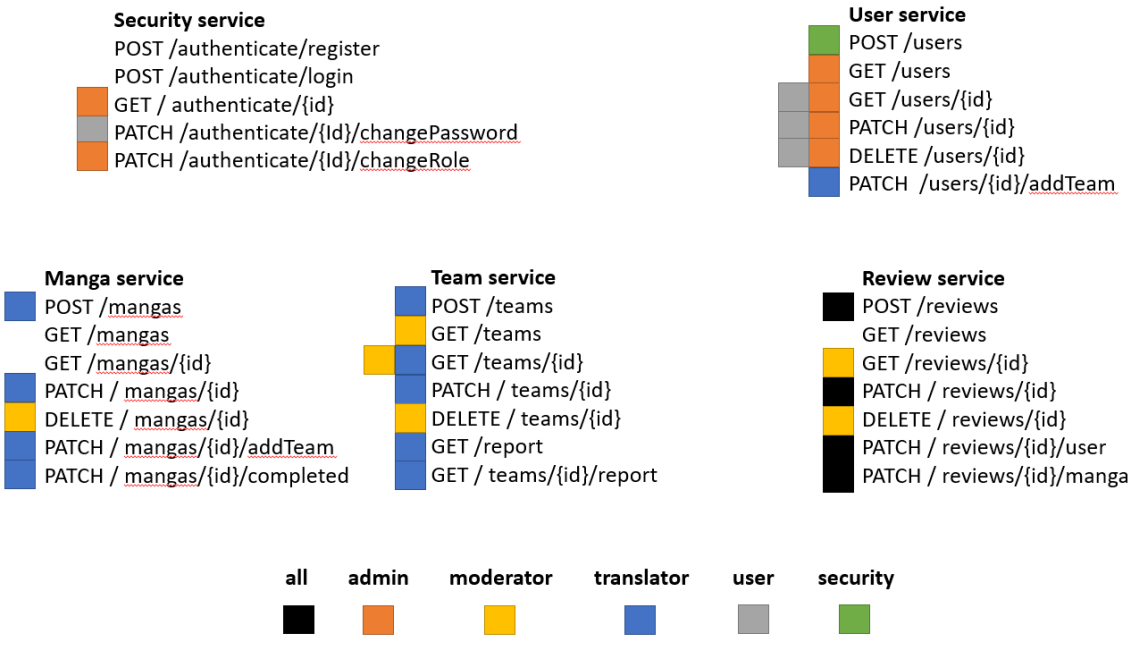

Рисунок 3.1 – Розподіл ролей додатку на всіх сервісах

З рисунку чудово видно, що адміністратори загалом керують усім, що пов’язано із користувачами, модератори слідкують за роботою команд-перекладачів, перекладачі працюють з мангою, а користувачі рецензують та читають мангу.

3.4 Проектування бази даних

Ресурс повинен мати п’ять баз даних, Security service має три таблиці, інші сервіси по одній відповідно до кожного сервісу. Security service єдиний буде мати базу даних PostgreSQL. Інші сервіси матимуть бази даних Cassandra відразу завантажені до серверів DataStax.

Спочатку розберімося із базою даних Security service, що написана мовою PostgreSQL. База даних має назву – Security та містить у собі три таблиці, вміст яких відображено у таблицях 3.2 – 3.4. Маємо три таблиці, бо нам потрібно зберігати в одній ім’я та зашифрований пароль користувачів, в другій ролі, а в третій призначення користувачу певної ролі, яке відтворене у зберіганні пари id – ролі та користувача.

Таблиця 3.2 – Таблиця User бази даних Security сервісу

№	Поле	Пояснення	Тип
1	id	Id користувача	INT
2	name	Ім’я користувача	VARCHAR
3	password	Пароль користувача	VARCHAR

Таблиця 3.3 – Таблиця Role бази даних Security сервісу

№	Поле	Пояснення	Тип
1	id	Id ролі	INT
2	name	Назва ролі	VARCHAR

Таблиця 3.4 – Таблиця Users Roles бази даних Security сервісу

№	Поле	Пояснення	Тип
1	Roles_id	Id ролі	INT
2	Users_id	Id користувача	INT

Обрана мова для Security service не випадково. «PostgreSQL не просто реляційна, а об'єктно-реляційна СУБД. Це дає йому деякі переваги над іншими базами даних SQL з відкритим вихідним кодом, такими як MySQL, MariaDB і Firebird.»[5] PostgreSQL дуже адаптивна до різного виду рішень мова завдяки великому набору інструментів, вона підтримує майже всі існуючі типи даних. Також завдяки всьому набору функцій ця мова є зручною при створенні великих структур даних. Як для Security service, PostgreSQL є дуже надійною мовою баз даних, яка при необхідності дасть змогу відтворити будь-яку роботу таблиць. Таким чином можна, навіть, у майбутньому без перешкод сформувати складну роботу з аккаунтами користувачів без обмежень.

Інші сервіси не так важливі, як сервіс безпеки, тому для них рішення інше, а саме Cassandra. Чотири бази даних, що містять у собі по одній таблиці кожна – розміщені на серверах DataStax. Зручне зберігання даних на сервері – причина вибору цієї мови для серверів. Спроектовані бази даних у таблицях 3.5 – 3.8.

Таблиця 3.5 – База даних Users сервісу

№	Поле	Пояснення	Тип
1	id	Id користувача	INT
2	name	Ім’я користувача	VARCHAR
3	Team_id	Id команди	INT

Таблиця 3.6 – База даних Team сервісу

№	Поле	Пояснення	Тип
1	id	id	INT
2	name	Ім’я команди	VARCHAR

Таблиця 3.7 – База даних Review сервісу

№	Поле	Пояснення	Тип
1	id	id	INT
2	text	Текст відгуку	VARCHAR
3	date	Дата створення відгуку	DATE
4	Users_id	Id користувача, який написав відгук	INT
5	Mangas_id	Id манги до якої відгук написано	INT

Таблиця 3.8 – База даних Mangas сервісу

№	Поле	Пояснення	Тип
1	id	Id манги	INT
2	name	Ім’я манги	VARCHAR
	originalName	Оригінальне ім’я манги	VARCHAR
3	date	Дата створення	DATE
4	Team_id	Id команди	INT
5	completed	Статус манги по перекладу	VARCHAR

Cassandra DB на відміну від популярних реляційних баз даних набула популярності не будучи реляційною базою даних. Cassandra DB підтримує

мову CQL, яка за своєю структурою дуже схожа на мову SQL, тому проблем із переходом на цю мову у розробників не виникатиме. Натомість ця мова не має гарно розробленої гнучкості системи, як в інших мовах, наприклад тут перейменування поля або зміна його типізації неможлива. Це великий недолік, який змушує створювати нову таблицю для виправлення полів.

«CQL дозволяє користувачам організовувати дані в кластері вузлів Cassandra за допомогою:

- Keyspace;
- Таблиця;
- Розділ;
- Рядок;
- Стовпець.»[6]

Cassandra DB схожа на реляційні бази даних не лише синтаксисом, але й структурою зберігання даних та комунікацією між ними. «Простір ключів відповідає поняттю схеми бази даних (database schema) у реляційній моделі, а що у ньому сімейства стовпців – реляційної таблиці.»[7] Таке рішення використовується для проектування масштабних баз з децентралізованим зв’язком з можливістю з декількома рівнями вкладеності. Також є відмовостійкою базою даних за рахунок зберігання інформації у різних дата-центрах або розподіляються по вузлах мережі.

4 РОЗРОБКА WEB-ДОДАТКУ

4.1 Серверна частина веб-додатку

4.1.1 Мова програмування

Оскільки від обраної мови програмування залежать можливості додатку, було обрано мову програмування Java.

«Java являє собою мову програмування та платформу обчислень, яка була вперше випущена компанією Sun Microsystems у 1995 році.»[8] Строго типізована мова з підтримкою об’єктно орієнтованого методу програмування.

Це популярна мова для програмування, яка чудово підходить для написання back-end частини ресурсу. Java підтримує велику кількість авторських бібліотек, які спрощують програмування з її використанням. А строгий стиль написання коду має лише переваги для правильної побудови роботи додатку. Також велика кількість середовищ розробки підтримують плагіни автогенерації великої кількості методів, що спрощує написання коду та зберігає час розробника.

Строга типізація також є плюсом при роботі у команді розробників. Так, при написанні великої кількості різних сервісів логіка їх написання буде максимально схожою, у результаті чого іншим розробникам буде значно легше орієнтуватися у коді колег. Таким чином, строга типізація може зберегти час команді розробників у спілкуванні щодо методу написання сервісів, також при пошуку різних помилок. Команда зможе легко та швидко опрацьовувати помилки будь-яких сервісів, що може значно пришвидшити роботу розробників.

Об’єктно орієнтоване програмування закладене у Java ж надає можливість масштабування додатку у подальшому. Так як ресурс потребує швидкої розробки велика кількість функціоналу може бути упущена. Після запуску продукту є вірогідність того, що додаткового функціоналу, який ресурс буде потребувати за обсягом може становити більше половини вже існуючого, тому можливість масштабування є важливим критерієм у виборі інструментів для написання додатку.

Загалом, чому Java є найкращим вибором для написання даного ресурсу:

- Java підтримує принципи ООП такі, як: інкапсуляція, абстракція, поліморфізм, наслідування
- Java є строго типізованою мовою

4.1.2 Spring Framework

«Spring описується як легкий каркас для побудови програм на Java.»[9] Spring пропонує рішення з найменшою кількістю втручання власного коду. Це допомагає швидко і коректно писати програми не тільки веб додатків, але й великих корпоративних систем.

Модуль Spring MVC відповідає проектному шаблону, який чітко розподіляє роботу між компонентами. Цей модуль нараховує три основні компоненти роботи:

- Модель – зберігає усі дані;
- Вид – представляє дані у вигляді інтерфейсу;
- Контролер – опрацьовує запити користувача.

Вже існуючий шаблон повністю диктує правила розробки сервісу, навіть недосвідчений розробник не допустить помилок у логіки сервісу. А принцип «не переписуй, а розширяй» не надає доступу до зміни існуючих модулів. Таким чином розробник буде працювати лише з вже існуючими рішеннями використовуючи їх для розвитку поставлених задач.

4.1.3 Об’єктно орієнтоване програмування

ООП – методологія програмування який базується на програмуванні за допомогою об’єктів, які собою являють звичайні об’єкти з повсякденного життя. Об’єкт має дані і методи для роботи з ними. Методи описано в класі, екземпляром якого і є об’єкт.

Методологія налічує чотири принципи, за якими працює:

- Інкапсуляція – приховує непотрібну розробнику логіку і дозволяє використовувати лише передбачений функціонал для роботи з недоступними даними;
- Поліморфізм – дає можливість повторного використання рішення або методу для роботи з даними;
- Абстракція – працює на класах, де описує, яким має бути об’єкт, але не вдається у подробиці його роботи ;
- Наслідування – передача об’єктам властивостей батьківського класу його нащадкам, таким чином утворюється розширений функціонал у об’єкта.

«Основні завдання ООП — структурувати код, підвищити його читабельність та прискорити розуміння логіки програми. Непрямо виконуються й інші завдання: наприклад, підвищується безпека коду та скорочується його дублювання.»[11]

4.1.4 Swagger

Всі хто розробляв веб додатки мали досвід роботи з Postman і знайомлячись зі Swagger можна подумати начебто це таж сама колекція із запитами. Але Swagger інтегрується на рівні кожного сервісу і описується у програмі власноруч, або ж автоматично за допомогою спеціального генератору. Виходячи з того, що Swagger впроваджується самими розробниками, складно повірити, що це просто ще один метод для тестування додатку.

Swagger – це повноцінна документація, яка зручно налаштована за допомогою UI інтерфейсу. Також цей інструмент використовують для того, щоб об’єднати всі сервіси разом, мати доступ не просто до одного сервісу, а працювати з проектом в цілому.

У додатку інтегровано Swagger до кожного сервісу. Переглянувши рисунки 4.1 – 4.5 та таблицю 3.1 можна перевірити реалізацію спроектованої логіки сервісів.

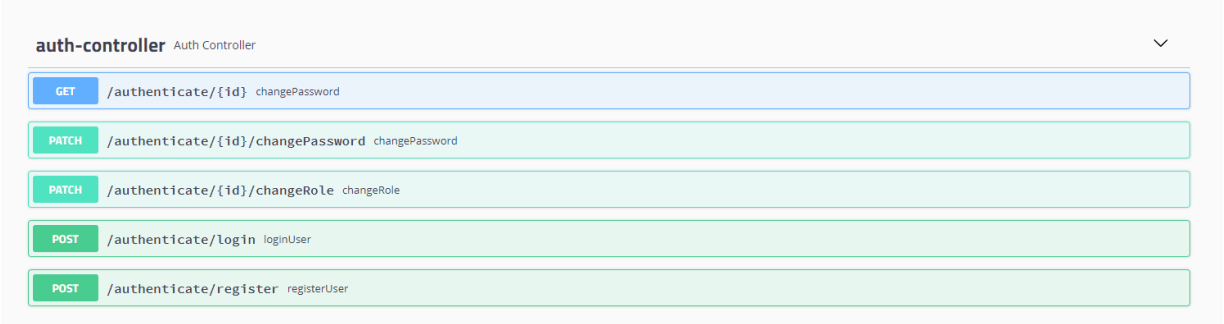

Рисунок 4.1 - Документація Swagger до Security service

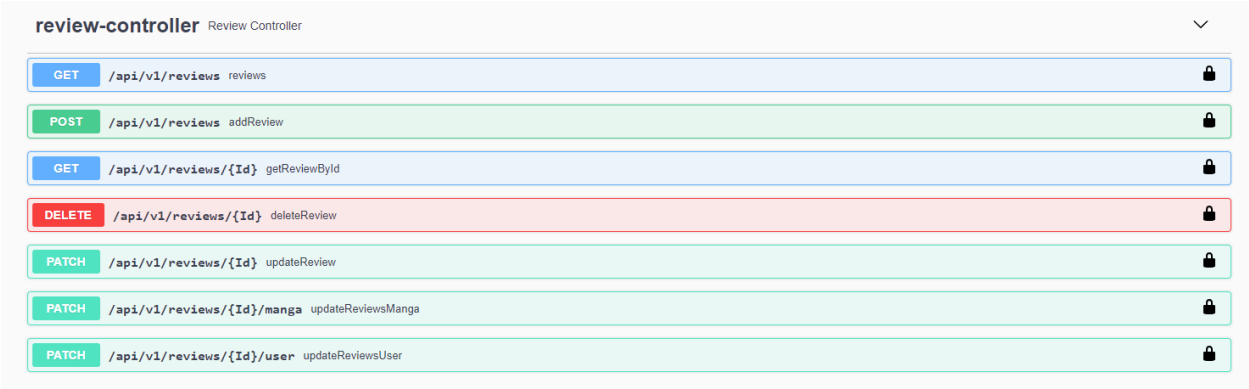

Рисунок 4.2 - Документація Swagger до Review service

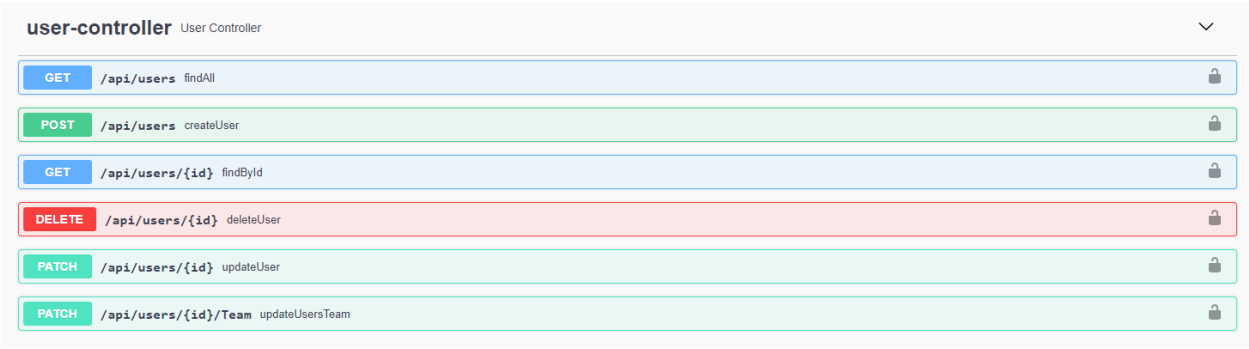

Рисунок 4.3 - Документація Swagger до User service

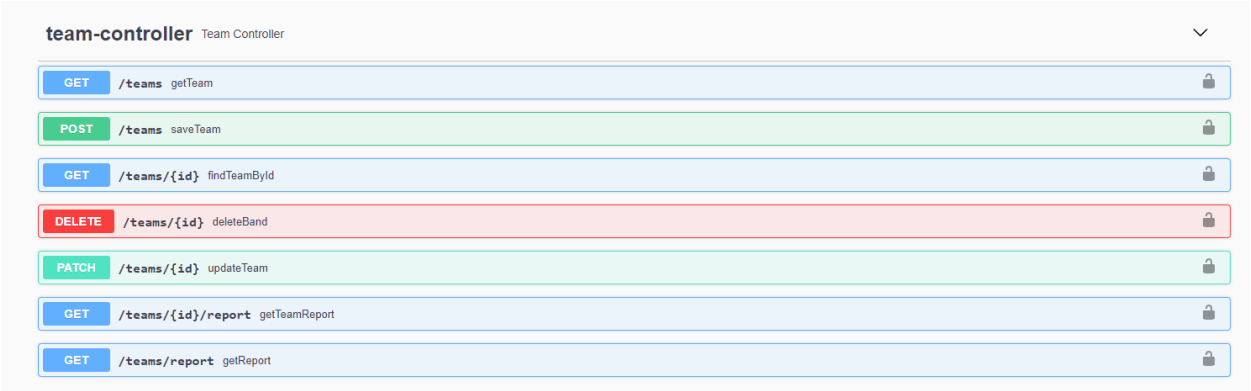

Рисунок 4.4 - Документація Swagger до Team service

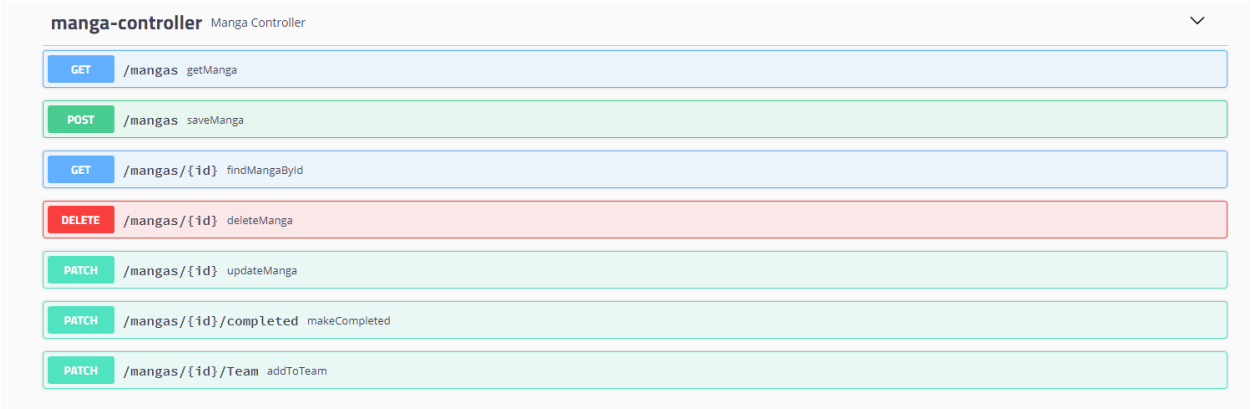

Рисунок 4.5 - Документація Swagger до Mangaservice

4.2 Захист та безпека системи

Система має захищений доступ до певних даних та функцій за домогою технології JWT токену та обмеження доступу користувачам за ролями.

4.2.1 Обмеження доступу прав користувачів за ролями

Додаток має чотири ролі:

- ROLE_USER
- ROLE_INTERPRETER
- ROLE_MODERATOR
- ROLE_ADMIN

Без авторизація, користувач, який не має ні одну із цих ролей має найбільші обмеження у використанні додатку, він може лише отримувати

інформацію про мангу на сайті. Якщо ж такий користувач зареєструється та авторизується він зможе залишати коментарі. Користувач, який має роль ROLE_INTERPRETER вже може працювати з мангою, з її даними – обробляти, видаляти та взагалі створювати. ROLE_MODERATOR керує даними пов’язаними з роботою користувачів, він може редагувати всі створені дані та видаляти їх. ROLE_ADMIN же взагалі надає ролі іншим користувачам, тобто керує системою доступу.

4.2.2 JWT токен

JSON Web Token – один із способів контролю доступу за допомогою створення токену. Цей токен використовує формат JSON і працює з веб орієнтованими додатками. Так як більшість додатків використовує REST архітектуру для проектування свої продуктів, JWT токен має таку ж популярність, бо у парі вони працюють дуже ефективно. Токен передається у запитах, наприклад в заголовку авторизації, і сервіс отримуючи запит, зчитує його інформацію і отримує токен.

JWT просто хешує дані, а не приховує їх, тому він потрібен лише для перевірки: чи авторизовано користувача. Розкодувати дані можно легко на онлайн сервісах, але маючи секретний код, який знають Security service та сервіс до якого ми маємо отримати доступ. А підбираючи цей секретний ключ, злочинець просто не встигне отримати дані. Велика кількість проектів використовують одразу два JWT токени – access та refresh. Суть у тому, що на основі одного токену створюється інший і access токен, який використовують для авторизації має короткий час існування.

На рисунку 4.6 зображено схему за якою токен генерується на сервісі, де користувач авторизується, а токен відправляється вже на сервіс, який перевіряє чи має користувач доступ, після чого виконує запит, або відмовляє за недоліком прав.

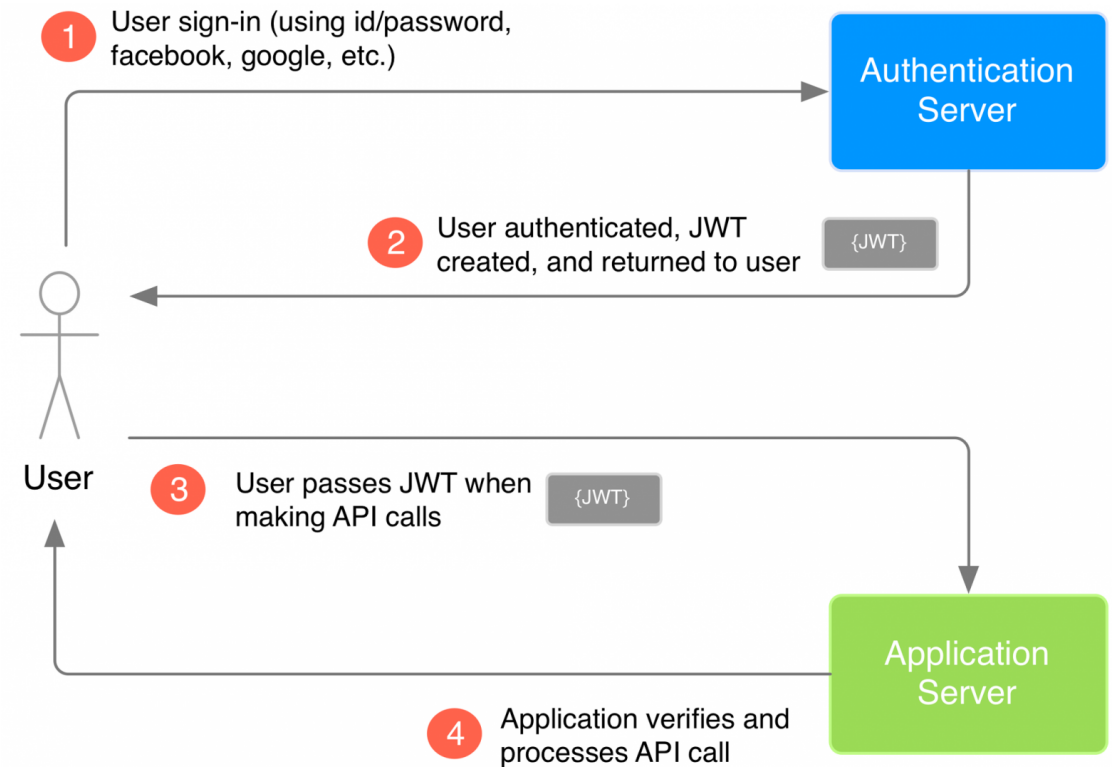

Рисунок 4.6 - Схема передачі JWT токену в клієнт-серверному додатку[12]

На рисунку 4.7 зображена реалізована авторизація за JWT токеном за допомогою Swagger.

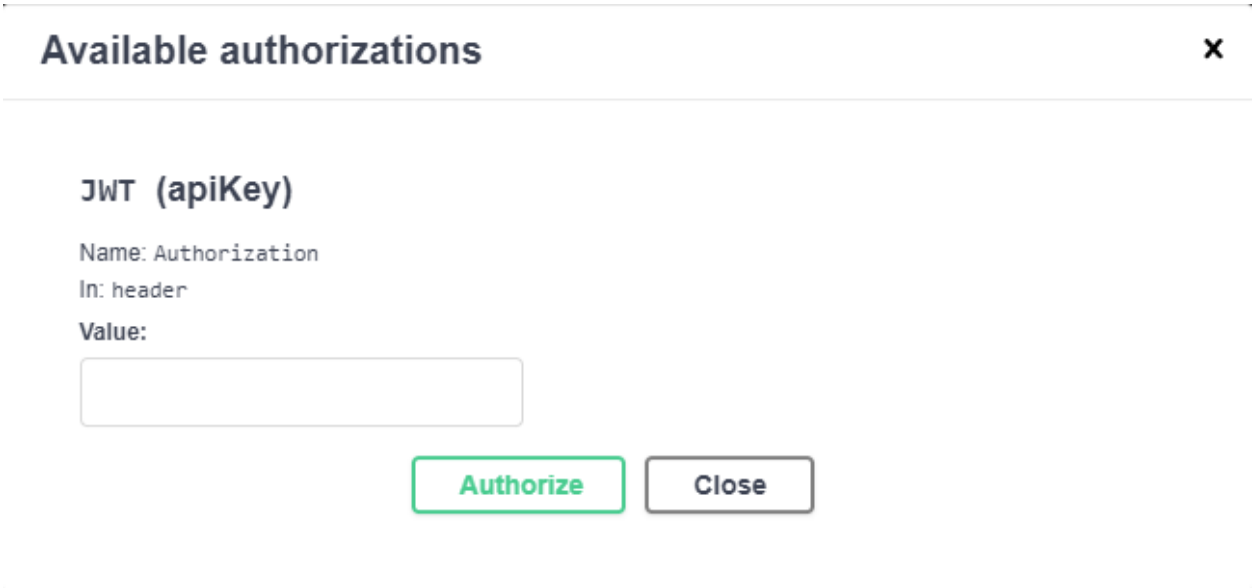

Рисунок 4.7 - Авторизація за JWT токеном у Swagger

JWT токен складається із трьох частин:

- header — заголовок,
- payload — корисне навантаження,

- signature — підпис.

На рисунку 4.8 закодований та розкодований токен, що сгенерував Security service. Всі три компоненти виділені кольорами та відокремлені крапками. У payload міститься інформація про id користувача, його логін та роль.

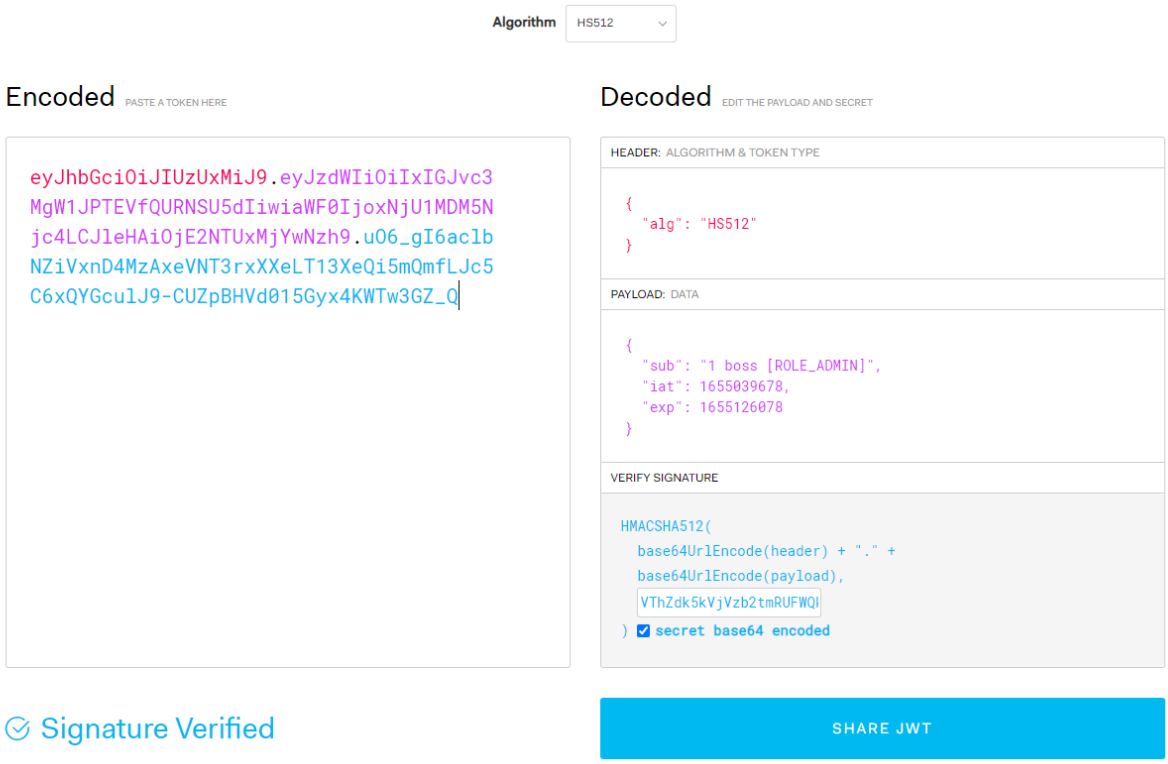

Рисунок 4.8 - JWT токен, що сгенерував Security service

Для того щоб отримати закодований рядок за алгоритмом створення JWT токену потрібно:

- закодувати header та payload;
- об’єднати отримані строки, розділивши їх крапкою;
- захешувати за допомогою вказаного алгоритму цю строку на основі секретного ключа.

4.3 Тестування веб додатку

Тестування це важлива частина розробки будь-якого проекту. Кожна програма проходить цей етап не зважаючи на кількість написаного коду,

функціонал, або професіональність розробників. Виявлення помилок перед релізом не дає можливості користувачам працювати з несправним продуктом.

Для того, щоб облегшити мануальне тестування використовувався додаток k6. З його допомогою були утворені сценарії, які виконували всі запити, вводили дані. Таким чином тестування можна проводити по декілька разів не витрачаючи час на те, щоб працювати власноруч.

Загалом k6 – це інструмент для навантажувального тестування. Але ресурси цього додатку дозволяють за допомогою UI записати всі запити на Java Script у сценарій, який потім можна використовувати для того, щоб кожного разу не відтворювати одних і тих операцій, наприклад введення даних.

4.4 Розгортання веб додатку

Для розгортання проекту використовувався сервіс Heroku. Також використовувалась технологія контейнеризації Docker.

«Docker – це платформа для розробки, доставки та запуску контейнерних програм. Docker дозволяє створювати контейнери, автоматизувати їх запуск та розгортання.»[13] Контейнер – це ізольоване середовище, яке може мати свої настройки в залежності від додатку, що упаковується. Таким чином в середині контейнеру знаходяться всі потрібні для запуску додатку ресурси. Тому, можна не хвилюватися про навколишнє середовище, контейнер запустить додаток будь-де незалежно від умов.

Контейнери захищають від помилок при розгортанні, так як цей процес максимально автоматизовано.

Платформа ж Heroku має якісну підтримку контейнерів, особливо легко працювати з нею використовуючи Docker. Платформа повністю синхронізована з Docker і для розгортання контейнеру потрібно лише вказати у Dockerfile декілька налаштувань:

- FROM, вказує на те, який базовий image використовувати;
- ADD, додає з’єднання з базою даних (у випадку manga додатку);

- COPY, копіює файли до контейнеру, у додатку – це файл точки входу програми;
- ENTRYPOINT, інструкція до запуску контейнера.

Розгорнуті сервіси доступні разом із під'єднаним Swagger за посиланнями:

- https://manga-teams-service-diploma/teams/swagger-ui.html#/
- https://manga-mangas-service-diploma/mangas/swagger-ui.html#/
- https://manga-users-service-diploma/api/users/swagger-ui/
- https://manga-reviews-service-diploma/api/v1/reviews/swagger-ui/
- https:// manga-security-service-diploma/swagger-ui.html#/

4.5 Покриття додатку APIGEE

Apigee – це платформа від Google яка використовується для розгортання, масштабування сервісів.

За допомогою цього сервісу можна покрити додатковий функціонал, виправити помилки та з’єднати сервіси, які зберігаються на різних серверах.

Apigee складається із різних компонентів, таких як:

- Apigee сервіс, де створюється Apigee проксі
- GCP, плітформа для моніторингу, логування та аналітики.

Apigee проксі містить у собі Policy, які являють собою обгортку для вже існуючого сервісу. «Проксі-сервер API — це сервер із тонким інтерфейсом прикладних програм (API), який надає інтерфейс для існуючого сервісу або сервісів.»[14]

Policy – це настройки, які працюють із запитами та обробляють дані. Кожна із них має свій власний функціонал.

У таблиці 4.1 наведено використані Policy та їх призначення

Таблиця 4.1 – Перелік Policy Apigee та для чого вони використовуються

Іконка	Назва	Призначення
	AssignMessage policy	Встановлює змінні, може додавати у запити header та payload
	ExtractVariables policy	Дістає змінні з запитів
	KeyValueMapOperations policy	Працює із простором KVM де зберігає потрібні значення, наприклад, секретний ключ для JWT токену. Використовується для доступу до значення змінної
	RaiseFault policy	Обробляє помилки
	SOAPMessageValidation policy	Валідує значення body запитів
	MessageLogging policy	Використовується для логування до GCP
	SpikeArrest policy	Встановлює допустиме значення частоти запитів

На рисунку 4.9 зображено приклад обробки запиту із застосуванням різних Policy.

Рисунок 4.9 – Запит оброблений на платформі Apigee

Проксі-сервери можуть поєднуватись за допомогою API Product. Завдяки цій технології сервіси об’єднуються за доступом одним URL. API Product – це централізований механізм керування існуючими проксі. Він керує доступом до ендпоінтів, також надає додатковий шар безпеки встановлюючи ApiKey.

На рисунку 4.10 зображено отримані GCP логи від проксі-серверу Users

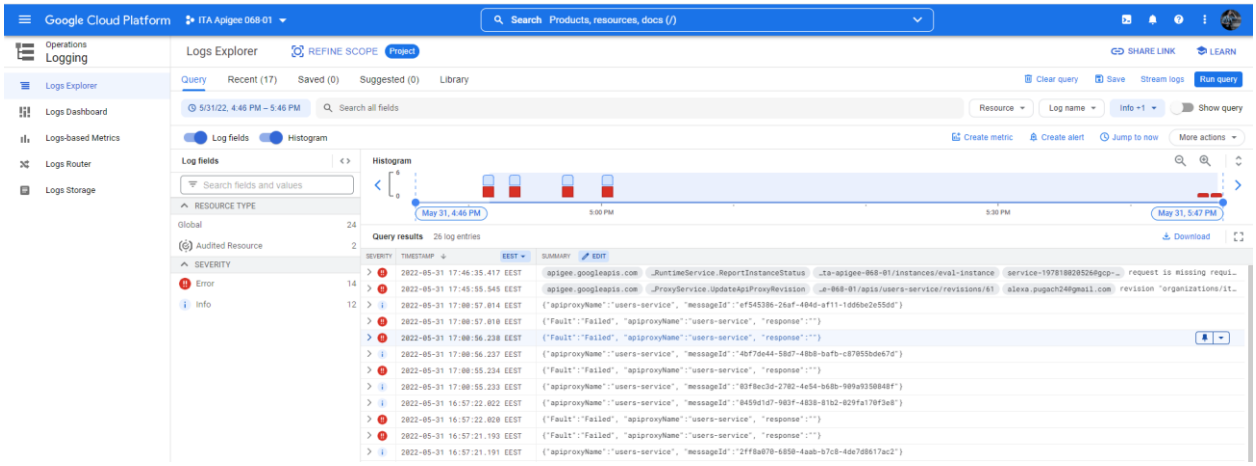

Рисунок 4.10 - Вкладка логів з GCP платформи

GCP надає можливість створення метрик, за якими далі можна отримувати сповіщення, наприклад, про перевищення кількості запитів із помилками. Також для повного аналізу є dashboards, які створюються власноруч з урахуванням потреб замовника чи власне розробників.

Доступ до API манги можливий за посиланням https://ita-apigee-068-01-manga.apigee.io/apis

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено серверну частину веб-додатку, яка покрита Apigee для можливості подальшого розвитку продукту.

Система передбачає базову роботу з об’єктами за доступом за ролями. З можливістю створення груп перекладачів, які будуть самостійно працювати із перекладом манги та розподіляти роботу між собою, як їм зручно. Бо кожен з них буде мати доступ до манги, з якою працює вся команда.

До системи закладено можливість масштабування, що було враховано при проектуванні мікросервісної архітектури. Також розгортання додатку в Apigee надає велику кількість інструментів для розширення функціоналу додатку.

Впроваджена можливість будь-якого моніторингу та керування трафіком додатку, який не допускає велике навантаження серверу та сповіщає при великих навантажень на сервер. Таким чином при роширенні сервісу можлива планова міграція ще до отримання проблем із роботою серверу.

Налаштовані метрики дозволяють у найкоротші строки знайти проблему та полагодити її. Бо коли користувач отримує проблему з додатком, метрики на основі логування передають сповіщення.

Виконано усі поставлені задачі для роботи додатку розробленого за малий проміжок часу із можливістю подальшого розширення можливостей.

СПИСОК ЛІТЕРАТУРИ

1. Микросервисы: почему это не панацея? [Електронний ресурс] // Habr, 2006 – 2022. – Режим доступу до ресурсу: https://habr.com/ru/company/vdsina/blog/553176/

2. Монолитная архитектура и микросервисная архитектура [Електронний ресурс] // russianblogs.com, 2020 – 2022. – Режим доступу до ресурсу: https://russianblogs.com/article/16941516187/

3. Мікросервісна архітектура [Електронний ресурс] / Ivan Zmerzlyi. – 2019. – Режим доступу до ресурсу: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d

4. What is a REST API? [Електронний ресурс] // Red Hat, Inc. – 2022 – Режим доступу до ресурсу: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

5. Чем PostgreSQL лучше других SQL баз данных с открытым исходным кодом [Електронний ресурс] // Habr, 2006 – 2022. – Режим доступу до ресурсу: https://habr.com/ru/post/282764/

6. Overview | Apache Cassandra Documentation [Електронний ресурс] // The Apache Software Foundation, 2009 – 2022. – Режим доступу до ресурсу: https://cassandra.apache.org/doc/latest/cassandra/architecture/overview.html

7. Что такое Apache Cassandra: архитектура, история [Електронний ресурс] // ООО «Учебный центр «Коммерсант», 2018 – 2022. – Режим доступу до ресурсу: https://www.bigdataschool.ru/wiki/cassandra

8. Что такое технология Java и каково ее применение? [Електронний ресурс] // Oracle – 2022. – Режим доступу до ресурсу: https://www.java.com/ru/download/help/whatis_java.html.

9. Spring 5 для профессионалов / Ю. Козмина, Р. Харроп, К. Шефер, К. Хо., – СПб. : ООО "Диалектика", 2019. – 1120 с.

10. Spring MVC - основные принципы для начинающих, примеры [Електронний ресурс] // Highload. – 2021 – 2022. – Режим доступу до ресурсу: https://highload.today/spring-mvc/

11. Объектно-ориентированное программирование простыми словами [Електронний ресурс] // Tproger. – 2022. – Режим доступу до ресурсу: https://tproger.ru/experts/oop-in-simple-words/

12. Пять простых шагов для понимания JSON Web Tokens (JWT) [Електронний ресурс] // Habr, 2006 – 2022. – Режим доступу до ресурсу: https://habr.com/ru/post/340146/

13. Что такое Docker: для чего он нужен и где используется [Електронний ресурс] // ООО «Селектел», 2008 – 2022. – Режим доступу до ресурсу: https://selectel.ru/blog/what-is-docker/

14. What is an API proxy? [Електронний ресурс] // TechTarget, 20019 – 2022. – Режим доступу до ресурсу: https://www.techtarget.com/searchapparchitecture/definition/API-proxy

ДОДАТКИ

Додаток А. Лістинг програмного коду

```
package com.example.jwtdemo.service;

@Slf4j
@Service
@RequiredArgsConstructor
public class UsersService {
    private final UsersRepository usersRepository;
    private final RoleRepository roleRepository;
    private final PasswordEncoder encoder;
    private final AuthenticationManager authenticationManager;
    private final JwtUtils jwtUtils;
    @Value("${user.service.url}")
    private String userURL;
    private final RestTemplate restTemplate;
    @Value("${user.app.secret}")
    private String jwtForUserService;
    @Value("${user.app.time}")
    private int jwtTimeForUserService;
    @Value("${my.app.secret}")
    private String jwtSecret;

    public Users register(SignupRequest signupRequest) {
        if (usersRepository.existsByUsername(signupRequest.getUsername())) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "User
in DB");
        }
        Users users = new Users(signupRequest.getUsername(),
encoder.encode(signupRequest.getPassword()));

users.setRoles(Set.of(roleRepository.findByName(ERole.ROLE_USER).get()));
        users.setId(createUser(signupRequest.getUsername()));
        return usersRepository.save(users);
    }

    public String getRole(Long id) {
        Optional<Users> optionalUsers1= usersRepository.findById(id);
        if (!optionalUsers1.isPresent()) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "User
with this id not found");
        }
        String role = String.valueOf(optionalUsers1.get().getRoles());
        return role;
    }

    public JwtResponse login(LoginRequest loginRequest) {
        Authentication authentication = authenticationManager
                .authenticate(new
UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
loginRequest.getPassword()));
        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = jwtUtils.generateJwtToken(authentication);
        UsersDetailsImpl usersDetails = (UsersDetailsImpl)
authentication.getPrincipal();
        List<String> roles = usersDetails.getAuthorities().stream()

.map(GrantedAuthority::getAuthority).collect(Collectors.toList());
        return new JwtResponse(jwt, usersDetails.getId(),
usersDetails.getUsername(), roles);
    }
```

```
    private Long createUser(String username) {
        try {
            ResponseEntity<UserResponse> responseEntity =
restTemplate.postForEntity(userURL,
                    new HttpEntity<>(Map.of("name", username),
createHeader(username)), UserResponse.class);
            if (Objects.requireNonNull(responseEntity.getBody()).getUserId()
== null) {
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
"Can't create user");
            }
            return responseEntity.getBody().getUserId();
        } catch (HttpClientErrorException e){
            throw new ResponseStatusException(e.getStatusCode());
        }
    }

    private HttpHeaders createHeader(String username) {
        String jwt = Jwts.builder().setSubject(username).setIssuedAt(new
Date())
                .setExpiration(new Date((new Date()).getTime() +
jwtTimeForUserService))
                .signWith(SignatureAlgorithm.HS512,
jwtForUserService).compact();
        return new HttpHeaders() {{
            set("Authorization", "Bearer " + jwt);
        }};
    }

    public Users changePassword(Long id, ChangePassword changePassword,
UserDetails details) {
        Optional<Users> optionalUsers1= usersRepository.findById(id);
        Optional<Users> optionalUsers2;
        try {
             optionalUsers2 =
usersRepository.findByUsername(details.getUsername());
        } catch (Exception e){
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
        if(!optionalUsers1.isPresent()) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "User
with this id not found");
        }
        Users users=optionalUsers1.get();
        if(!optionalUsers2.get().getId().equals(users.getId())){
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }
        if (!encoder.matches(changePassword.getOldPassword(),
users.getPassword())) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
"Current password doesn't equals your old password");
        }
        users.setPassword(encoder.encode(changePassword.getNewPassword()));
        return usersRepository.save(users);
    }

    public Users changeRole(Long id, String changeRole) {
        Optional<Users> optionalUsers1= usersRepository.findById(id);
        Optional<Users> optionalUsers2;

        try {
            Optional<Role> optionalRole1 =
roleRepository.findByName(ERole.valueOf(changeRole));
            log.info("Role user: {}", optionalRole1);
```

```
        }catch (Exception e){
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Role
with this name not found");
        }
        /*try {
            optionalUsers2 =
usersRepository.findByUsername(details.getUsername());
        } catch (Exception e){
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }*/
        if(!optionalUsers1.isPresent()) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "User
with this id not found");
        }
        Users users=optionalUsers1.get();
        /*if(!optionalUsers2.get().getId().equals(users.getId())){
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }
        if(!optionalRole1.isPresent()) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Role
with this name not found");
        }*/
        usersRepository.deleteById(id);

users.setRoles(Set.of(roleRepository.findByName(ERole.valueOf(changeRole)).ge
t()));
        return usersRepository.save(users);
    }

    public void isTokenValidAdmin(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                log.error("Unauthorized with this JWT");
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_ADMIN")) {
                return;
            } else {
                log.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            log.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }
}
package com.example.jwtdemo.service;

public class UsersDetailsImpl implements UserDetails {
    private Long id;
    private String username;
    private String password;
    private Collection<? extends GrantedAuthority> authorities;

    public UsersDetailsImpl(Long id, String username, String password,
Collection<? extends GrantedAuthority> authorities) {
        this.id = id;
```

```
        this.username = username;
        this.password = password;
        this.authorities = authorities;
    }

    public static UsersDetailsImpl build(Users users) {
        List<GrantedAuthority> authorities = users.getRoles().stream()
                .map(role -> new
SimpleGrantedAuthority(role.getName().name()))
                .collect(Collectors.toList());
        return new UsersDetailsImpl(users.getId(),
                users.getUsername(),
                users.getPassword(),
                authorities);
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    public Long getId() {
        return id;
    }

    @Override
    public String getPassword() {
        return password;
    }

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

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}

package com.example.jwtdemo.service;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    private final UsersRepository repository;

    public UserDetailsServiceImpl(UsersRepository repository) {
        this.repository = repository;
```

```
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Users users=repository.findByUsername(username).orElseThrow(()->new
UsernameNotFoundException("User not found"));
        return UsersDetailsImpl.build(users);
    }
}
package com.example.jwtdemo.domains;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Users {
    @Id
    private Long id;
    private String username;
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private String password;
    @ManyToMany(fetch = FetchType.EAGER)
    private Set<Role> roles;

    public Users(String username,String password) {
        this.username = username;
        this.password = password;
    }

}
package com.example.jwtdemo.domains;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @JsonIgnore
    private Long id;
    @Enumerated(EnumType.STRING)
    private ERole name;

    public Role(ERole name) {
        this.name = name;
    }
}
package com.example.jwtdemo.domains;

public enum ERole {
    ROLE_USER,
    ROLE_INTERPRETER,
    ROLE_MODERATOR,
    ROLE_ADMIN
}
package com.example.jwtdemo.controller;

@Slf4j
@RestController
@RequestMapping("/authenticate")
public class AuthController {
```

```
    private final UsersService usersService;

    public AuthController(UsersService usersService) {
        this.usersService = usersService;
    }

    @PostMapping("/login")
    public ResponseEntity<?> loginUser(@Valid @RequestBody LoginRequest
loginRequest) {
        log.info("Login user: {}", loginRequest);
        return ResponseEntity.ok(usersService.login(loginRequest));
    }

    @PostMapping("/register")
    public ResponseEntity<Users> registerUser(@Valid @RequestBody
SignupRequest signupRequest) {
        log.info("Register user: {}", signupRequest);
        return ResponseEntity.ok(usersService.register(signupRequest));
    }

    @GetMapping("/{id}")
    @ApiImplicitParam(name = "Authorization", value = "Access Token",
required = true, paramType = "header", example = "Bearer access_token")
    public ResponseEntity<?> changePassword(@PathVariable("id") Long id,
HttpServletRequest request) {
        usersService.isTokenValidAdmin(request);
        log.info("Searching users role with id: {}", id);
        return ResponseEntity.ok(usersService.getRole(id));
    }

    @PatchMapping("/{id}/changePassword")
    @ApiImplicitParam(name = "Authorization", value = "Access Token",
required = true, paramType = "header", example = "Bearer access_token")
    public ResponseEntity<?> changePassword(@PathVariable("id") Long id,
@RequestBody ChangePassword changePassword,
                                            @ApiIgnore
@AuthenticationPrincipal UserDetails details) {
        log.info("Change password: {}", changePassword);
        return ResponseEntity.ok(usersService.changePassword(id,
changePassword ,details));
    }

    @PatchMapping("/{id}/changeRole")
    @ApiImplicitParam(name = "Authorization", value = "Access Token",
required = true, paramType = "header", example = "Bearer access_token")
    public ResponseEntity<?> changeRole(@PathVariable("id") Long id,
@RequestBody String changeRole, HttpServletRequest request) {
        usersService.isTokenValidAdmin(request);
        log.info("Changing users role: {} role with id: {}", changeRole, id);
        return ResponseEntity.ok(usersService.changeRole(id, changeRole));
    }
}
package com.example.Users.service;

@Service
public class UserService  {

private final UserRepository userRepository;
private final RestTemplate restTemplate;
private final ServicesConnection connection;

    @Autowired
    public UserService(UserRepository userRepository,
HttpComponentsClientHttpRequestFactory factory, ServicesConnection
```

```
connection) {
        this.userRepository = userRepository;
        this.restTemplate = new RestTemplate(factory);
        this.connection = connection;
    }

    private static final Logger logger =
LoggerFactory.getLogger(UserService.class);

    @Value("${my.app.secret}")
    private String jwtSecret;

    @Value("${user.app.secret}")
    private String jwtUserSecret;

    public Optional<User> findById(Long userId){

        Optional<User> updatableUser = userRepository.findById(userId);

        if(updatableUser.isPresent()){
            return userRepository.findById(userId);
        } else {

            logger.error("Error with status code: {}", HttpStatus.NOT_FOUND);
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);

        }

    }

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

    public User createUser(User user){

        List<User> list = findAll();
        Long number;

        try {
            number = list.stream().max((o1, o2) -> (int) (o1.getUserId() -
o2.getUserId())).get().getUserId();
        } catch (NoSuchElementException e) {
            number = Long.valueOf(99);
        }

        user.setUserId(++number);
        return userRepository.save(user);


    }

    public User updateById(Long userId, User user) {

            try {

                Optional<User> updatableUser =
userRepository.findById(userId);
                User newUser = updatableUser.get();
                    if(user.getUserId() !=null)
                        newUser.setUserId(userId);
                    if(user.getName() !=null)
                        newUser.setName(user.getName());
                    if(user.getTeamId() !=null)
```

```
                        newUser.setTeamId(user.getTeamId());

                return userRepository.save(newUser);

            } catch (NoSuchElementException e) {
                logger.error("Error with exception: {}", e.getMessage());
                throw new ResponseStatusException(HttpStatus.NOT_FOUND);
            }

    }

    public ResponseEntity<Void> deleteById(Long userId){

        try {

            userRepository.deleteById(userId);
            return ResponseEntity.noContent().build();

        } catch (ResourceNotFoundException e) {

            logger.error("Error with status code: {}", e.getMessage());
            return ResponseEntity.notFound().build();

        }

    }

    public User updateTeamId(Long userId, String teamName,
HttpServletRequest request) {

        try {

            Optional<User> updatableUser = userRepository.findById(userId);
            User newUser = updatableUser.get();

            try{

                ResponseEntity<String> response =
restTemplate.exchange(connection.getUrlTeams() + teamName, HttpMethod.GET,
new HttpEntity<>(createHeaders(request.getHeader("Authorization"))),
String.class);

                try {
                    String jsonStr = new
String((Objects.requireNonNull(response.getBody())).getBytes());
                    JSONObject jsonObject = new JSONObject(jsonStr);
                    Long teamId =
Long.valueOf(String.valueOf(jsonObject.get("id")));
                    newUser.setTeamId(teamId);
                    return userRepository.save(newUser);
                } catch (JSONException e) {
                    logger.error("Error with exception: {}", e.getMessage());
                    throw new
ResponseStatusException(HttpStatus.BAD_REQUEST);
                }

            } catch (HttpClientErrorException e) {
                logger.error("Error with status code: {}",
e.getStatusCode());
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
            }

        } catch (NoSuchElementException e) {
            logger.error("Error with exception: {}", e.getMessage());
```

```
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        }

    }


    public boolean isTokenValidUser(HttpServletRequest request){

        try {

            String headerAuth = request.getHeader("Authorization");
            logger.info("Checking for the presence of a token");

            if (headerAuth!=null && headerAuth.startsWith("Bearer ")) {
                String s =
Jwts.parser().setSigningKey(jwtUserSecret).parseClaimsJws(headerAuth.substrin
g(7)).getBody().getSubject();
                logger.info("jwtToken received");
                return true;
            } else {
                logger.error("Error with status code: {}",
HttpStatus.BAD_REQUEST);
                return false;
            }

        } catch (SignatureException e){
            logger.error("Error with exception: {}", e.getMessage());
            return false;
        }

    }

    public void isTokenValidAdmin(HttpServletRequest request){

            String headerAuth = request.getHeader("Authorization");
            logger.info("Checking for the presence of a token");

        try{

            if (headerAuth!=null && headerAuth.startsWith("Bearer ")) {

                String[] s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
                logger.info("jwtToken received");
                if (s[2].contains("ROLE_ADMIN")) {
                    return;
                } else {
                    logger.error("Error with status code: {}",
HttpStatus.FORBIDDEN);
                    throw new ResponseStatusException(HttpStatus.FORBIDDEN);
                }

            } else {
                logger.error("Error with status code: {}",
HttpStatus.UNAUTHORIZED);
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }

        } catch(SignatureException e){
            logger.error("Error with exception: {}", e.getMessage());
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
```

```
    }

    public void isTokenValidInterpreter(HttpServletRequest request){

        String headerAuth = request.getHeader("Authorization");
        logger.info("Checking for the presence of a token");

        try{

            if (headerAuth!=null && headerAuth.startsWith("Bearer ")) {

                String[] s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
                logger.info("jwtToken received");
                if (s[2].contains("ROLE_INTERPRETER")) {
                    return;
                } else {
                    logger.error("Error with status code: {}",
HttpStatus.FORBIDDEN);
                    throw new ResponseStatusException(HttpStatus.FORBIDDEN);
                }

            } else {
                logger.error("Error with status code: {}",
HttpStatus.UNAUTHORIZED);
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }

        } catch(SignatureException e){
            logger.error("Error with exception: {}", e.getMessage());
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }


    }

    public void isTokenValidAdminAndUser(Long userId, HttpServletRequest
request) {

        String headerAuth = request.getHeader("Authorization");
        logger.info("Checking for the presence of a token");

        try{

            if (headerAuth!=null && headerAuth.startsWith("Bearer ")) {

                String[] s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
                logger.info("jwtToken received");
                if (s[2].contains("ROLE_ADMIN")) {
                    return;
                }else if(s[2].contains("ROLE_USER") &&
s[0].contains(String.valueOf(userId))){
                    return;
                } else {
                    logger.error("Error with status code: {}",
HttpStatus.FORBIDDEN);
                    throw new ResponseStatusException(HttpStatus.FORBIDDEN);
                }

            } else {
```

```
                logger.error("Error with status code: {}",
HttpStatus.UNAUTHORIZED);
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }

        } catch(SignatureException e){
            logger.error("Error with exception: {}", e.getMessage());
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }

    }

    private HttpHeaders createHeaders(String jwt) {

        return new HttpHeaders() {{
            set("Authorization", jwt);
        }};

    }

}
package com.example.Users.model;

@Data
@Table(value="user")
public class User {
    @Id
    @PrimaryKeyColumn(
            name = "userid",
            ordinal = 0,
            type = PrimaryKeyType.PARTITIONED)
    private Long userId;
    @Column(value="name")
    private String name;
    @Column(value="teamid")
    private Long teamId;


}
package com.example.Users.controller;

@Controller
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;
    private static final Logger logger =
LoggerFactory.getLogger(UserController.class);


    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }

    @PostMapping
    public ResponseEntity<User> createUser(@RequestBody User user,
HttpServletRequest request){

        if (!userService.isTokenValidUser(request)) {
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }

        logger.info("Creating a User");
```

```
        return ok(userService.createUser(user));

    }

    @GetMapping
    public ResponseEntity<List<User>> findAll(HttpServletRequest request){

        userService.isTokenValidAdmin(request);
        logger.info("Getting all Users");
        return ok(userService.findAll());

    }

    @GetMapping("/{id}")
    public ResponseEntity<?> findById(@PathVariable("id") Long id,
HttpServletRequest request){

        userService.isTokenValidAdminAndUser(id, request);
        logger.info("Getting the User with id: " +id);
        return ok(userService.findById(id));

    }

    @PatchMapping("/{id}")
    public ResponseEntity<User> updateUser(@PathVariable("id") Long id,
@RequestBody User user, HttpServletRequest request){

        userService.isTokenValidAdminAndUser(id, request);
        logger.info("Updating the User with id: " +id);
        return ok(userService.updateById(id, user));

    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable("id") Long id,
HttpServletRequest request){

        userService.isTokenValidAdminAndUser(id, request);
        logger.info("Deleting the User with id: " + id);
        return userService.deleteById(id);

    }

    @PatchMapping("/{id}/Team")
    public ResponseEntity<Object> updateUsersTeam(@PathVariable("id") Long
id,  @RequestBody String teamName, HttpServletRequest request){

        userService.isTokenValidInterpreter(request);
        logger.info("Updating the Users team with id: " +id + " to: " +
teamName);
        return ok(userService.updateTeamId(id, teamName, request));

    }


}
package com.example.teamservice.service.impl;

@Service
public class TeamServiceImpl implements TeamService {

    private final TeamRepository teamRepository;
    private final RestTemplate restTemplate;
    private final TeamClientProperties teamClientProperties;
```

```
    @Value("${my.app.secret}")
    private String jwtSecret;
    private final Logger logger =
LoggerFactory.getLogger(TeamController.class);

    public TeamServiceImpl(TeamRepository teamRepository,
HttpComponentsClientHttpRequestFactory factory, TeamClientProperties
teamClientProperties) {
        this.teamRepository = teamRepository;
        this.restTemplate = new RestTemplate(factory);
        this.teamClientProperties = teamClientProperties;
    }

    @Override
    public Team create(TeamDTO band) {
        Team team1 = teamRepository.findByName(band.getName());
        if (team1 != null) {
            logger.error("The band is in DB");
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
        List<Team> list = teamRepository.findAllTeams();
        Long l;
        try {
            l = list.stream().max((o1, o2) -> (int) (o1.getId() -
o2.getId())).get().getId();
        } catch (NoSuchElementException e) {
            l = 0L;
        }
        Team newTeam = new Team();
        newTeam.setId(++l);
        newTeam.setName(band.getName());
        return teamRepository.save(newTeam);
    }

    @Override
    public Team readById(Long id) {
        try {
            Team team = teamRepository.getTeamById(id);
            team.getName();
            return team;
        } catch (NullPointerException e) {
            logger.error("Team is not found");
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Team is
not found");
        }
    }

    @Override
    public void delete(Long id, HttpServletRequest request) {
        Team team = readById(id);
        teamRepository.deleteById(id);
    }

    @Override
    public List<Team> getAll() {
        return teamRepository.findAllTeams();
    }

    @Override
    public Map<String, List<String>> getReport(HttpServletRequest request) {
        List<Team> teams =
restTemplate.exchange(teamClientProperties.getUrlTeams(),
                HttpMethod.GET, new
HttpEntity<>(createHeaders(request.getHeader("Authorization"))), new
```

```
ParameterizedTypeReference<List<Team>>() {
                }).getBody();
        Map<String, List<String>> map = new HashMap<>();
        if (teams == null) {
            throw new NullTeamReferenceException("There are no teams");
        }
        for (Team b : teams) {
            map.put(b.getName(), getSingleReport(b.getId(), request));
        }
        return map;
    }


    @Override
    public List<String> getSingleReport(Long id, HttpServletRequest request)
{
        try {
            Team team = readById(id);
            HttpHeaders headers =
createHeaders(request.getHeader("Authorization"));
            List<User> users =
restTemplate.exchange(teamClientProperties.getUrlUsers(),
                    HttpMethod.GET, new HttpEntity<>(headers), new
ParameterizedTypeReference<List<User>>() {
                    }).getBody();
            Map<String, List<Review>> reviews =

restTemplate.exchange(teamClientProperties.getUrlReviews(),
                            HttpMethod.GET, new HttpEntity<>(headers), new
ParameterizedTypeReference<Map<String, List<Review>>>() {
                            }).getBody();
            List<Manga> mangas =

restTemplate.exchange(teamClientProperties.getUrlMangas(),
                            HttpMethod.GET, new HttpEntity<>(headers), new
ParameterizedTypeReference<List<Manga>>() {
                            }).getBody();
            List<User> listUser = users.stream().filter(o -> o.getTeamId() !=
null).filter(o -> o.getTeamId().equals(id)).collect(Collectors.toList());
            List<Review> listReview =
reviews.get("reviews").stream().filter(o -> o.getTeamId() != null).filter(o -
> o.getTeamId().equals(id)).collect(Collectors.toList());
            List<Manga> listManga = mangas.stream().filter(o -> o.getTeamId()
!= null).filter(o -> o.getTeamId().equals(id)).collect(Collectors.toList());
            List<String> s = new ArrayList<>();
            if (listUser.isEmpty()) {
                s.add("There is no users");
            } else {
                s.add(listUser.toString());
            }
            if (listReview.isEmpty()) {
                s.add("There is no reviews");
            } else {
                s.add(listReview.toString());
            }
            if (listManga.isEmpty()) {
                s.add("There is no mangas");
            } else {
                s.add(listManga.toString());
            }
            return s;
        } catch (HttpClientErrorException e) {
            throw new
ResponseStatusException(HttpStatus.valueOf(e.getRawStatusCode()));
```

```
        }
    }


    @Override
    public Object readByName(String name) {
        Team team = teamRepository.findByName(name);
        if (team == null) {
            logger.info("Team is not found");
            return Collections.emptyList();
        }
        return team;
    }

    @Override
    public Team update(Long id, TeamDTO team, String forTeamId,
HttpServletRequest request) {
        Team team1 = readById(id);
        team1.setId(id);
        if (team.getName() != null) {
            team1.setName(team.getName());
        }
        ResponseEntity<User> responseEntity;
        try {
            responseEntity =
restTemplate.exchange(teamClientProperties.getUrlUsers() + forTeamId,
                    HttpMethod.GET, new
HttpEntity<>(createHeaders(request.getHeader("Authorization"))), new
ParameterizedTypeReference<User>() {
                    });
        } catch (Exception e) {
            logger.error("Team not found");
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
        User user = responseEntity.getBody();
        Long teamId = user.getTeamId();
        if(id != teamId) {
            logger.error("You haven`t permissions");
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }
        teamRepository.update(id, team1.getName());
        return team1;
    }

    public boolean isTokenValidInterpreter(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                logger.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_INTERPRETER")) {
                return true;
            } else {
                logger.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            logger.error("Unauthorized with this JWT");
```

```
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    public boolean isTokenValidModerator(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                logger.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_MODERATOR")) {
                return true;
            } else {
                logger.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            logger.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    public boolean isTokenValidInterpreterAndModerator(HttpServletRequest
request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                logger.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_MODERATOR") ||
s[2].contains("ROLE_INTERPRETER")) {
                return true;
            } else {
                logger.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            logger.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    public String getTokenId(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        String id = " ";
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
```

```
                logger.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            id = s[0];
        }
        return id;
    }

    private HttpHeaders createHeaders(String jwt) {
        return new HttpHeaders() {{
            set("Authorization", jwt);
        }};
    }

}
package com.example.teamservice.model;

@Data
@Table("band")
public class Team {
    @Id
    @PrimaryKeyColumn(
            name = "id",
            ordinal = 0,
            type = PrimaryKeyType.PARTITIONED)
    private Long id;
    @Column(value = "name")
    @NotBlank
    @NotNull
    private String name;
}
package com.example.teamservice.controller;

@RestController
@RequestMapping("/teams")
public class TeamController {
    private final TeamService teamService;
    private final Logger logger =
LoggerFactory.getLogger(TeamController.class);

    public TeamController(TeamService teamService) {
        this.teamService = teamService;
    }

    @PostMapping
    public ResponseEntity<Team> saveTeam(@Valid @RequestBody TeamDTO team,
HttpServletRequest request) {
        teamService.isTokenValidInterpreter(request);
        logger.info("Creating new team");
        return ResponseEntity.ok(teamService.create(team));
    }

    @GetMapping
    public ResponseEntity<?> getTeam(@RequestParam(value = "teamName",
required = false) String name, HttpServletRequest request) {

        teamService.isTokenValidModerator(request);
        if (name == null) {
            logger.info("Getting all teams");
            return ResponseEntity.ok(teamService.getAll());
        } else {
            logger.info("Getting team name = {}", name);
            return ResponseEntity.ok(teamService.readByName(name));
        }
```

```
    }

    @GetMapping("/{id}")
    public ResponseEntity<Team> findTeamById(@PathVariable("id") Long id,
HttpServletRequest request) {
        teamService.isTokenValidInterpreterAndModerator(request);
        logger.info("Getting team id = {}", id);
        Team team = teamService.readById(id);
        return ResponseEntity.ok(team);
    }

    @GetMapping("/report")
    public ResponseEntity<Map<String, List<String>>>
getReport(HttpServletRequest request) {
        teamService.isTokenValidInterpreter(request);
        logger.info("Getting global report");
        return ResponseEntity.ok(teamService.getReport(request));
    }

    @GetMapping("/{id}/report")
    public ResponseEntity<List<String>> getTeamReport(@PathVariable("id")
Long id, HttpServletRequest request) {
        teamService.isTokenValidInterpreter(request);
        logger.info("Getting team report with id {}", id);
        return ResponseEntity.ok(teamService.getSingleReport(id, request));
    }


    @DeleteMapping("/{id}")
    public void deleteBand(@PathVariable("id") Long id, HttpServletRequest
request) {
        teamService.isTokenValidModerator(request);
        logger.info("Deleting team id = {}", id);
        teamService.delete(id, request);
    }

    @PatchMapping("/{id}")
    public ResponseEntity<Team> updateTeam(@PathVariable("id") Long id,
@Valid @RequestBody TeamDTO team, HttpServletRequest request) {
        teamService.isTokenValidInterpreter(request);
        String forTeamId = teamService.getTokenId(request);
        logger.info("Updating team id = {}", id);
        return ResponseEntity.ok(teamService.update(id, team, forTeamId,
request));
    }
}
package com.example.mangaservice.service;

@Slf4j
@Service
public class MangaService {
    private final MangaRepository repository;
    private final RestTemplate restTemplate;
    private final MangaClientProperties properties;
    @Value("${my.app.secret}")
    private String jwtSecret;


    public MangaService(MangaRepository repository, MangaClientProperties
properties, HttpComponentsClientHttpRequestFactory factory) {
        this.repository = repository;
        this.restTemplate = new RestTemplate(factory);
        this.properties = properties;
```

```
    }

    public Manga save(MangaDTO mangaDTO) {
        log.info("Manga before saving: {}", mangaDTO);
        Manga manga2 = repository.findByName(mangaDTO.getName());
        if (manga2 != null) {
            log.error("Manga in DB: {}", mangaDTO);
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
        List<Manga> list = repository.findAllMangas();
        Long number;
        try {
            number = list.stream().max((o1, o2) -> (int) (o1.getId() -
o2.getId())).get().getId();
        } catch (NoSuchElementException e) {
            number = 0L;
        }
        Manga manga = new Manga();
        manga.setId(++number);
        manga.setName(mangaDTO.getName());
        manga.setOriginalName(mangaDTO.getOriginalName());
        manga.setDate(mangaDTO.getDate());
        return repository.save(manga);
    }

    public Manga findById(Long id) {
        Manga manga = repository.getMangaById(id);
        if (Objects.isNull(manga)) {
            log.error("Can't found task with id: {}", id);
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        } else {
            return manga;
        }
    }

    public void delete(Long id) {
        log.info("Manga for deleting: {}", repository.getMangaById(id));
        Optional<Manga> optionalTask = repository.findById(id);
        if (!optionalTask.isPresent()) {
            log.error("Manga with id: {} not found", id);
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        }
        repository.deleteById(id);
    }


    public Object findByName(String name) {
        Manga manga = repository.findByName(name);
        if (manga == null) {
            log.info("Manga by name: {} not found", name);
            return Collections.emptyList();
        }
        log.info("Find Manga: {}. By name: {}", manga, name);
        return manga;
    }

    public Manga update(Long id, MangaForUpdateDTO mangaForUpdateDTO, String
forTeamId, HttpServletRequest request) {
        log.info("Manga with id: {}. And body: {}", id, mangaForUpdateDTO);
        Manga manga1 = findById(id);
        manga1.setId(id);
        if (mangaForUpdateDTO.getName() != null) {
            Manga manga = repository.findByName(mangaForUpdateDTO.getName());
            if (manga != null &&
```

```
!manga1.getName().equals(mangaForUpdateDTO.getName()))
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
            manga1.setName(mangaForUpdateDTO.getName());
        }
        if (mangaForUpdateDTO.getOriginalName() != null) {
            manga1.setOriginalName(mangaForUpdateDTO.getOriginalName());
        }
        if (mangaForUpdateDTO.getDate() != null) {
            manga1.setDate(mangaForUpdateDTO.getDate());
        }
        ResponseEntity<User> responseEntity;
        try {
            responseEntity = restTemplate.exchange(properties.getUrlUsers() +
forTeamId,
                    HttpMethod.GET, new
HttpEntity<>(createHeaders(request.getHeader("Authorization"))), new
ParameterizedTypeReference<User>() {
                    });
        } catch (Exception e) {
            log.error("RManga not found");
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
        User user = responseEntity.getBody();
        Long teamId = user.getTeamId();
        if(manga1.getTeamId() != teamId) {
            log.error("You haven`t permissions");
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }
        log.info("Manga which updating: {}", manga1);
        repository.update(id, manga1.getName(), manga1.getOriginalName(),
manga1.getDate());
        return manga1;
    }

    public void addMangaToTeam(Long id, String teamName, HttpServletRequest
request, String forTeamId) {
        log.info("add to Manga with id: {}. Team name: {}", id, teamName);
        ResponseEntity<Team> responseEntity;
        try {
            responseEntity = restTemplate.exchange(properties.getUrlTeams() +
teamName,
                    HttpMethod.GET, new
HttpEntity<>(createHeaders(request.getHeader("Authorization"))), new
ParameterizedTypeReference<Team>() {
                    });
        } catch (Exception e) {
            log.error("RManga not found");
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
        Team team = responseEntity.getBody();
        Long teamId = team.getId();
        ResponseEntity<User> userResponseEntity;
        try {
            userResponseEntity =
restTemplate.exchange(properties.getUrlUsers() + forTeamId,
                    HttpMethod.GET, new
HttpEntity<>(createHeaders(request.getHeader("Authorization"))), new
ParameterizedTypeReference<User>() {
                    });
        } catch (Exception e) {
            log.error("RManga not found");
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
        User user = userResponseEntity.getBody();
```

```
        Long usersTeamId = user.getTeamId();
        if(usersTeamId != teamId) {
            log.error("You haven`t permissions");
            throw new ResponseStatusException(HttpStatus.FORBIDDEN);
        }
        repository.addMangaToTeam(id, teamId);
    }

    public void makeMangaCompleted(Long id, HttpServletRequest request) {
        Optional<Manga> manga = repository.findById(id);
        if (!manga.isPresent()) {
            log.error("Manga id: {}. Not found", id);
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        }
        try {
            if (manga.get().isCompleted()) {
                log.error("Manga is completed: {}", manga.get());
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
            }
            repository.makeMangaCompleted(id);
        } catch (HttpClientErrorException e) {
            log.error("Client Error: {}", e.getMessage());
            throw new ResponseStatusException(e.getStatusCode());
        } catch (Exception e) {
            log.error("Another error: {}", e.getMessage());
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
    }

    public List<Manga> findAllMangas() {
        return repository.findAll();
    }

    public void isTokenValidAdmin(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                log.error("Unauthorized with this JWT");
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_ADMIN")) {
                return;
            } else {
                log.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            log.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    public void isTokenValidInterpreter(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
```

```
).getBody().getSubject().split("");
            } catch (Exception e) {
                log.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_INTERPRETER")) {
                return;
            } else {
                log.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            log.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    public void isTokenValidModerator(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                log.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_MODERATOR")) {
                return;
            } else {
                log.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            log.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    public void isTokenValidInterpreterAndModerator(HttpServletRequest
request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                log.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_MODERATOR") ||
s[2].contains("ROLE_INTERPRETER")) {
                return;
            } else {
                log.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            log.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
```

```
        }
    }


    public String getTokenId(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        String id = " ";
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                log.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            id = s[0];
        }
        return id;
    }

    private HttpHeaders createHeaders(String jwt) {
        return new HttpHeaders() {{
            set("Authorization", jwt);
        }};
    }
}
package com.example.mangaservice.domain;

@Data
@Table(value = "task")
public class Manga {
    @Id
    @PrimaryKeyColumn(
            name = "id",
            ordinal = 0,
            type = PrimaryKeyType.PARTITIONED)
    private Long id;
    @Column(value = "name")
    @NotBlank
    @NotNull
    private String name;
    @Column(value = "originalName")
    @NotBlank
    @NotNull
    private String originalName;
    @Column(value = "completed")
    private boolean isCompleted;
    @Column(value = "teamId")
    private Long teamId;
    @Column(value = "date")
    private Date date;
}
package com.example.mangaservice.controller;

@Slf4j
@RestController
@RequestMapping("/mangas")
public class MangaController {
    private final MangaService service;

    public MangaController(MangaService service) {
        this.service = service;
```

```
    }

    @PostMapping
    @ApiImplicitParam(name = "Authorization", value = "Access Token",
required = true, paramType = "header", example = "Bearer access_token")
    public ResponseEntity<Manga> saveManga(@Valid @RequestBody MangaDTO task,
HttpServletRequest request) {
        service.isTokenValidInterpreter(request);
        String forTeamId = service.getTokenId(request);
        log.info("Manga for saving: {}", task);
        return ResponseEntity.ok(service.save(task));
    }

    @GetMapping
    public ResponseEntity<?> getManga(@RequestParam(value = "mangaName",
required = false) String name, HttpServletRequest request) {
        if (name == null) {
            log.info("Getting all mangas");
            return ResponseEntity.ok(service.findAllMangas());
        } else {
            log.info("Searching manga with name: {}", name);
            return ResponseEntity.ok(service.findByName(name));
        }

    }

    @GetMapping("/{id}")
    public ResponseEntity<Manga> findMangaById(@PathVariable("id") Long id,
HttpServletRequest request) {
        log.info("Searching manga with id: {}", id);
        return ResponseEntity.ok(service.findById(id));
    }


    @DeleteMapping("/{id}")
    @ApiImplicitParam(name = "Authorization", value = "Access Token",
required = true, paramType = "header", example = "Bearer access_token")
    public ResponseEntity<?> deleteManga(@PathVariable("id") Long id,
HttpServletRequest request) {
        service.isTokenValidModerator(request);
        log.info("Deleting manga with id: {}", id);
        service.delete(id);
        return  ResponseEntity.noContent().build();
    }

    @PatchMapping("/{id}")
    @ApiImplicitParam(name = "Authorization", value = "Access Token",
required = true, paramType = "header", example = "Bearer access_token")
    public ResponseEntity<Manga> updateManga(@PathVariable("id") Long id,
@Valid @RequestBody MangaForUpdateDTO task, HttpServletRequest request) {
        service.isTokenValidInterpreterAndModerator(request);
        String forTeamId = service.getTokenId(request);
        log.info("Manga with id: {}. And body: {}", id, task);
        return ResponseEntity.ok(service.update(id, task, forTeamId,
request));
    }

    @PatchMapping("/{id}/Team")
    @ApiImplicitParam(name = "Authorization", value = "Access Token",
required = true, paramType = "header", example = "Bearer access_token")
    public ResponseEntity<Manga> addToTeam(@PathVariable("id") Long id,
@RequestBody TeamDTO teamDTO, HttpServletRequest request) {
        service.isTokenValidInterpreter(request);
        String forTeamId = service.getTokenId(request);
```

```
        log.info("Add manga with name {}", teamDTO);
        service.addMangaToTeam(id, teamDTO.getTeamName(), request,
forTeamId);
        return ResponseEntity.ok(service.findById(id));
    }

    @PatchMapping("/{id}/completed")
    @ApiImplicitParam(name = "Authorization", value = "Access Token",
required = true, paramType = "header", example = "Bearer access_token")
    public ResponseEntity<Manga> makeCompleted(@PathVariable("id") Long id,
HttpServletRequest request) {
        service.isTokenValidModerator(request);
        log.info("Make manga with id: {} completed", id);
        service.makeMangaCompleted(id, request);
        return ResponseEntity.ok(service.findById(id));
    }
}
package io.javabrains.reviewservice.servise;

@Service
public class ReviewServiceImpl implements ReviewService {
    @Autowired
    ReviewRepository reviewRepository;

    Logger logger = LoggerFactory.getLogger(ReviewServiceImpl.class);

    @Value("${my.app.secret}")
    private String jwtSecret;
    /*LocalDateTime now = LocalDateTime.now();
    DateTimeFormatter format = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    Date formatDateTime = Date.parse(now.format(format));*/

    LocalDate formatDateTime = LocalDate.now();

    public List<Review> getAll() {
        List<Review> reviewList = new ArrayList<Review>();
        reviewRepository.findAll().forEach(weapon -> reviewList.add(weapon));
        return reviewList;
    }

    public ReviewResponse getAllReviews() {
      List<Review> reviewList = getAll();
      ReviewResponse reviewResponse = new ReviewResponse();
      reviewResponse.setWeapons(reviewList);
      logger.info("All the weapons have been accessed");
      return reviewResponse;
    }

    public Review create(Review review)
    {
      List<Review> listOfReviews = reviewRepository.findAll();

    /*  if(review.getText() == null ||
review.getText().replaceAll("\\s+","").equals("")) {
        logger.error("You haven't specified a name for the review or name is
invalid");
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "You
haven't specified a name for the review or name is invalid");
      }*/

      if(review.getId() != null && reviewRepository.findById(review.getId())
!= null) {
        logger.error("Review with such name already exists");
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Review
```

```
with such id already exists");
      }


      if(review.getUserId() != null && review.getUserId() < 1) {
        logger.error("You specified an invalid bandId");
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "You
specified an invalid userId");
      }

      if(review.getMangaId() != null && review.getMangaId() < 0) {
        logger.error("You specified an invalid taskId");
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "You
specified an invalid mangaId");
      }

        if(review.getDate() == null) {
            review.setDate(formatDateTime);
        }

      Long num;
      try {
          num = listOfReviews.stream().max((o1, o2) -> {
              return (int) (o1.getId() - o2.getId());
          }).get().getId();
      } catch (NoSuchElementException e) {
          num = 0L;
      }
      review.setId(++num);
      return reviewRepository.save(review);
    }


    public ResponseEntity<?> getById(Long Id) {
      Optional<Review> reviewData = reviewRepository.findById(Id);
      if (reviewData.isPresent()) {
        logger.info("Returned the review with an id {}", Id);
        return new ResponseEntity<>(reviewData.get(), HttpStatus.OK);
      }
      else {
        logger.error("There is no such review with an id {}", Id);
        return new ResponseEntity<String>("There is no review with such id",
HttpStatus.NOT_FOUND);
      }
    }

    public ResponseEntity<?> updateById(Long Id, Review review) {
      Optional<Review> reviewData = reviewRepository.findById(Id);
      if(review.getText() == null && review.getUserId() == null &&
review.getMangaId() == null) {
        logger.error("You haven't provided any valid data to be changed");
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "You
haven't provided any valid data to be changed");
      }
      if (reviewData.isPresent()) {
        logger.info("Updating review with id {}", Id);
        Review _review = reviewData.get();
        if(review.getText() != null) {
          logger.info("Changing the text for the review with id {}", Id);
          if(review.getText().replaceAll("\\s+","").equals("")) {
            logger.error("You haven't specified a text for the review");
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "You
haven't specified a name for the review");
          }
```

```
          _review.setText(review.getText());
        }

        if(review.getUserId() != null) {
          logger.info("Changing the task_id for the review with id {}", Id);
          if(review.getUserId() < 0) {
            logger.error("You specified an invalid taskId");
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "You
specified an invalid taskId");
          }
          _review.setUserId(review.getUserId());
        }
        if(review.getMangaId() != null) {
          logger.info("Changing the band_id for the review with id {}", Id);
          if(review.getMangaId() < 1) {
            logger.error("You specified an invalid bandId");
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "You
specified an invalid bandId");
          }
          _review.setMangaId(review.getMangaId());
        }
        logger.info("Updated review: {}", _review);
        return new ResponseEntity<>(reviewRepository.save(_review),
HttpStatus.OK);
      }
      else {
        logger.error("There is no such review with an id {}", Id);
        return new ResponseEntity<String>("There is no review with such id",
HttpStatus.NOT_FOUND);
      }
    }

    public ResponseEntity<Object> addManga(Long Id, String mangaName,
HttpServletRequest request) {
      try {
        Optional<Review> weaponData = reviewRepository.findById(Id);
        if (weaponData.isPresent()) {
          RestTemplate restTemplate = new RestTemplate();
          String fullUrl = Links.getMangaUrl() + "?mangaName=" + mangaName;
          ResponseEntity<String> response = restTemplate.exchange(fullUrl,
HttpMethod.GET, new
HttpEntity<>(createHeaders(request.getHeader("Authorization"))),
String.class);
          try {
              String mangaJSON = new String((response.getBody()).getBytes());
              if (mangaJSON.equals("[]")) {
                logger.error("Specified mangaName is incorrect");
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
"Specified mangaName is incorrect");
              }
              JSONObject jsonObject = new JSONObject(mangaJSON);
              Long mangaId = jsonObject.getLong("id");
              logger.info("Adding review to the manga with id {}...",
mangaId);
              return new ResponseEntity<>(updateMangaId(Id, mangaId),
HttpStatus.OK);
          } catch (JSONException e) {
              logger.error("Error has occured: {}", e);
              throw new RuntimeException(e);
          }
        }
        else {
          logger.error("There is no such review with an id {}", Id);
          throw new ResponseStatusException(HttpStatus.NOT_FOUND, "There is
```

```
no review with such id");
        }
      }
      catch (HttpClientErrorException e){
        logger.error("Client error has occured: {}", e);
        throw new ResponseStatusException(e.getStatusCode());
      }
    }

    public ResponseEntity<Object> addUser(Long Id, Long usersId,
HttpServletRequest request) {
      try {
        Optional<Review> reviewData = reviewRepository.findById(Id);
        if (reviewData.isPresent()) {
          RestTemplate restTemplate = new RestTemplate();
          String fullUrl = Links.getUserUrl()  + usersId;
          ResponseEntity<String> response = restTemplate.exchange(fullUrl,
HttpMethod.GET, new
HttpEntity<>(createHeaders(request.getHeader("Authorization"))),
String.class);
          try {
              String userJSON = new String((response.getBody()).getBytes());
              if (userJSON.equals("[]")) {
                logger.error("Specified userId is incorrect");
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
"Specified userId is incorrect");
              }
              JSONObject jsonObject = new JSONObject(userJSON);
              Long userId = jsonObject.getLong("id");
              logger.info("Adding review to the user with id {}...", userId);
              return new ResponseEntity<>(updateUserId(Id, userId),
HttpStatus.OK);
          } catch (JSONException e) {
              logger.error("Error has occured: {}", e);
              throw new RuntimeException(e);
          }
        }
        else {
          logger.error("There is no such review with an id {}", Id);
          throw new ResponseStatusException(HttpStatus.NOT_FOUND, "There is
no review with such id");
        }
      }
      catch (HttpClientErrorException e){
        logger.error("Client error has occured: {}", e);
        throw new ResponseStatusException(e.getStatusCode());
      }
    }

    public Review update(Review review)
    {
      return reviewRepository.save(review);
    }

    public Object updateUserId(Long Id, Long userId) {
      Optional<Review> _review = reviewRepository.findById(Id);
      Review updatedReview = _review.get();
      if (_review.isPresent()) {
          logger.info("Weapon's user_id has been changed");
          updatedReview.setUserId(userId);
      }
      else {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND);
      }
```

```
      return reviewRepository.save(updatedReview);
    }

    public Object updateMangaId(Long Id, Long mangaId) {
      Optional<Review> _review = reviewRepository.findById(Id);
      Review updatedReview = _review.get();
      if (_review.isPresent()) {
          logger.info("Weapon's manga_id has been changed");
          updatedReview.setMangaId(mangaId);
      }
      else {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND);
      }
      return reviewRepository.save(updatedReview);
    }

    public ResponseEntity<?> delete(Long Id) {
      Optional<Review> weaponData = reviewRepository.findById(Id);
      if (weaponData.isPresent()) {
        reviewRepository.deleteById(Id);
        return new ResponseEntity<String>("Weapon was deleted",
HttpStatus.NO_CONTENT);
      }
      else {
        logger.error("There is no such review with an id {}", Id);
        return new ResponseEntity<String>("There is no review with such id",
HttpStatus.NOT_FOUND);
      }
    }

    public boolean isTokenValidAnybody(Long userId, HttpServletRequest
request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
                logger.error("User id: {}", userId);
            } catch (Exception e) {
                logger.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if ((s[2].contains("ROLE_MODERATOR") ||
s[2].contains("ROLE_INTERPRETER") || s[2].contains("ROLE_USER") ||
s[2].contains("ROLE_ADMIN")) && s[0].contains(String.valueOf(userId))) {
                return true;
            } else {
                logger.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            logger.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    public boolean isTokenValidModerator(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (headerAuth != null && headerAuth.startsWith("Bearer ")) {
            String[] s;
            try {
                s =
```

```
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(headerAuth.substring(7)
).getBody().getSubject().split("");
            } catch (Exception e) {
                logger.error("Unauthorized with this JWT");
                throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
            }
            if (s[2].contains("ROLE_MODERATOR")) {
                return true;
            } else {
                logger.error("Forbidden with this JWT");
                throw new ResponseStatusException(HttpStatus.FORBIDDEN);
            }
        } else {
            logger.error("Unauthorized with this JWT");
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED);
        }
    }

    public HttpHeaders createHeaders(String jwt) {
      return new HttpHeaders() {{
          set("Authorization", jwt);
      }};
  }
}
package io.javabrains.reviewservice.model;

@Table(value = "review")
public class Review {

    @Id
    @PrimaryKeyColumn(name = "Id", ordinal = 0, type =
PrimaryKeyType.PARTITIONED)
    private Long Id;

    @Column("text")
    @CassandraType(type = Name.TEXT)
    private String text;

    @Column("date")
    @CassandraType(type = Name.DATE)
    private LocalDate date;

    @Column("userId")
    private Long userId;

    @Column("mangaId")
    private Long mangaId;

    public Review() {
    }

    public Review(Long Id, String name,  LocalDate date, Long taskId, Long
bandId) {
        this.Id = Id;
        this.text = text;
        this.date = date;
        this.userId = userId;
        this.mangaId = mangaId;
    }

    public Long getId() {
        return Id;
    }
```

```
    public void setId(Long id) {
        Id = id;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public LocalDate getDate() {
        return date;
    }

    public void setDate(LocalDate date) {
        this.date = date;
    }

    public Long getUserId() {
        return userId;
    }

    public void setUserId(Long userId) {
        this.userId = userId;
    }

    public Long getMangaId() {
        return mangaId;
    }

    public void setMangaId(Long mangaId) {
        this.mangaId = mangaId;
    }

    @Override
    public String toString() {
        return "Id = " + Id + ", text = " + text + ", date = " + date +",
userId = " + userId + ", mangaId = " + mangaId;
    }

}

package io.javabrains.reviewservice.controller;

@RestController
@RequestMapping("/api/v1/reviews")
@CrossOrigin()
public class ReviewController {

  @Autowired
  ReviewService reviewService;

  @Autowired
  ReviewRepository reviewRepository;

  Logger logger = LoggerFactory.getLogger(ReviewController.class);

  @RequestMapping(method = RequestMethod.POST)
  public Review addReview (@RequestBody Review review, HttpServletRequest
request){
    reviewService.isTokenValidAnybody(review.getUserId(), request);
    logger.info("Saving review: {}", review);
```

```
    return reviewService.create(review);
  }

  @RequestMapping(method = RequestMethod.GET)
  public ReviewResponse reviews() {
    return reviewService.getAllReviews();
  }

  @GetMapping("/{Id}")
  public ResponseEntity<?> getReviewById(@PathVariable("Id") Long Id,
HttpServletRequest request) {
    reviewService.isTokenValidModerator(request);
    return reviewService.getById(Id);
  }

  @PatchMapping("/{Id}")
  public ResponseEntity<?> updateReview(@PathVariable("Id") Long Id,
@RequestBody Review review, HttpServletRequest request) {
    reviewService.isTokenValidAnybody(review.getUserId(), request);
    return reviewService.updateById(Id, review);
  }

  @RequestMapping(value="/{Id}", method = RequestMethod.DELETE)
  public ResponseEntity<?> deleteReview (@PathVariable("Id") Long Id,
HttpServletRequest request) {
    reviewService.isTokenValidModerator(request);
    logger.info("Deleting the review with and id {}", Id);
    return reviewService.delete(Id);
  }

  @PatchMapping("/{Id}/user")
  public ResponseEntity<Object> updateReviewsUser(@PathVariable("Id") Long
Id, @RequestBody Long userId, HttpServletRequest request) {
    reviewService.isTokenValidAnybody(userId, request);
    return reviewService.addUser(Id, userId, request);
  }

  @PatchMapping("/{Id}/manga")
  public ResponseEntity<Object> updateReviewsManga(@PathVariable("Id") Long
Id, @RequestBody String mangaName, @ApiIgnore Review review,
HttpServletRequest request) {
    reviewService.isTokenValidAnybody(review.getUserId(), request);
    return reviewService.addManga(Id, mangaName, request);
  }

}
```