

Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій
Кафедра оптоелектроніки та інформаційних технологій

Курсова робота
Розробка серверної частини веб-сайту
«Курси домедичної допомоги»

Виконав:
студент групи ФсІ – 22
спеціальності 122 – Комп'ютерні науки
Гладун М.В.
Науковий керівник:
доц. Мостова М.Р.
19 грудня 2023 р.

Львів 2023

Анотація

Ця курсова робота націлена на розробку бекенду для веб-сайту "Курси домедичної допомоги" з використанням Node.js та фреймворків, таких як Express та Mongoose. Проект орієнтований на створення функціональної основи для користувачів, а саме: реєстрації, авторизації та забезпечення комунікації учнів та викладачів, взаємодії з уроками.

Основні ідеї програми включають реалізацію безпечного та доступного середовища, де люди без медичного досвіду можуть отримати необхідні знання та навички для надання невідкладної домедичної допомоги.

Проект використовує MongoDB для зберігання даних, забезпечує конфіденційність користувачів за допомогою бібліотеки bcrypt, яка хешує паролі, для безпечного зберігання у базі даних. Також присутня інтеграція з Email для сповіщення учасників про реєстрацію та проходження курсів. Використовуються додаткові фреймворки, такі як multer для завантаження файлів та swagger-ui для документації API.

Даний веб-сервер містить як функціонал для простого відвідувача, так і для викладачів, адже є можливість додавати уроки, їх редагувати й видаляти, переглядати записи на них тощо. Використані протоколи передачі даних, веб-запити, присутня складова взаємодії між сервером та клієнтом.

Для забезпечення надійності та правильності функціональності серверу виконано тестування API за допомогою ПЗ Insomnia, яке дозволяє перевіряти реєстрацію та авторизації користувачів, взаємодію з уроками, завантаження файлів, а також виконання різноманітних запитів до сервера, передаючи інформацію у форматі JSON й унікальні користувацькі JWT. Результати перевірок використовуються для виявлення та виправлення помилок, які виникають на різних етапах розробки.

Abstract

This coursework aims to develop the backend for the "First Aid Courses" website using Node.js and frameworks such as Express and Mongoose. The project focuses on creating a functional foundation for users, including registration, authorization, and facilitating communication between students and instructors, as well as interaction with lessons.

The main concepts of the program involve implementing a secure and accessible environment where individuals without medical experience can acquire the necessary knowledge and skills for providing emergency medical assistance.

The project utilizes MongoDB for data storage and ensures user confidentiality through the bcrypt library, which hashes passwords for secure storage in the database. Additionally, there is integration with email services to notify participants about registration and course progress. Supplementary frameworks, such as multer for file uploads and swagger-ui for API documentation, are also employed.

The web server encompasses functionality for both casual visitors and instructors. Instructors can add, edit, and delete lessons, as well as view records associated with them. The project utilizes data transfer protocols, web requests, and incorporates components for interaction between the server and client.

To ensure the reliability and correctness of the server's functionality, API testing is conducted using the Insomnia software. This tool allows for the verification of user registration and authorization, lesson interaction, file uploads, and various server requests by transmitting information in JSON format and unique user JWTs. The results of these tests are used to identify and address errors occurring at different stages of development.

Перелік умовних позначень та скорочень

ПЗ – Програмне забезпечення.

JSON – JavaScript Object Notation.

JWT – JSON Web Token.

SRP – Single Responsibility Principle.

СУБД – Система управління базою даних.

NOSQL – Not Only Structured Query Language.

ID – Identifier.

HTTP – Hypertext Transfer Protocol.

Зміст

Вступ.....	5
Розділ-1 «Постановка завдання»	6
1.1 Вхідні дані для виконання проекту	6
1.2 Очікувані результати виконання проекту	6
1.3 Архітектура рішення	7
1.4 Вибір і обґрунтування засобів та технологій використаних для виконання проекту	7
Розділ-2 «Теоретичні відомості»	9
2.1 Технології, використані для виконання проекту	9
2.2 Програмні засоби, використані для виконання проекту	15
2.3 Ключові елементи програмування на стороні сервера, використані при виконанні проекту	16
Розділ-3 «Реалізація проекту. Послідовний опис виконання проекту з документуванням коду»	19
3.1 Підключення модулів, фреймворків, бази даних, реалізація завантаження зображень на сервер та запуск веб-сервера	19
3.2 Створення моделей користувача та уроку	20
3.3 Реалізація контролера, у якому містяться усі запити для взаємодії із користувачами	23
3.4 Реалізація контролера, у якому містяться усі запити для взаємодії користувачів із уроками	30
3.5 Створення валідацій та middleware для аутентифікації користувачів за допомогою JWT	39
3.6 Реалізація Swagger-документації	42
Розділ-4 «Тестування проекту»	45
4.1 Методика тестування результатів	45
4.2 Результати тестування	45
4.3 Висновки, зроблені після тестувань	52
Розділ-5 «Демонстрація функціональних можливостей розробленого проекту»	53
5.1 Виконання запитів, пов'язаних із користувачами	53
5.2 Виконання запитів для керування уроками	56
5.3 Перегляд документації через SwaggerUi	64
5.4 Виконання запитів через створені html-форми	65
Висновки	67
6.1 Основні результати проекту	67
6.2 Шляхи покращення	68
6.3 Перспективи використання	68
Список використаних джерел	69
Додатки	70

Вступ

У теперішніх умовах війни постає проблема із низьким рівнем медичної підготовки цивільного населення й, зокрема, деяких військових. Тому виникає потреба в існуванні інтуїтивно зрозумілої та доступної кожному платформи для навчання онлайн. Курси онлайн стають надзвичайно актуальними у сучасних умовах з ряду ключових причин: доступність із будь-якої точки світу, раціональне використання часу, широкий вибір матеріалів. Також дуже важливим плюсом є безпека, оскільки люди у прифронтових зонах не мають змоги відвідувати різноманітні медичні тренінги та офлайн-заняття, тому онлайн-освіта залишається для них єдиним джерелом знань.

Сервер призначений для широкого кола користувачів, які зацікавлені у вивченні невідкладної домедичної допомоги, адже у теперішніх реаліях цими навичками повинен володіти кожен. Це можуть бути як особи без медичного досвіду, що шукають базові знання, так і фахівці медичної галузі, які прагнуть покращити свої навички або поділитись власним досвідом.

Платформа буде включати курси із базових реанімаційних заходів у дорослих та дітей. Це покрокові дії обстеження людини без свідомості та розпізнавання зупинки серця, якісні компресії грудної клітини, штучна вентиляція легень, безпечне використання дефібрилятора, надання допомоги особі при удавленні/вдушенні та забезпечення стабільного бокового положення особі без свідомості. Після завершення проходження курсу творець курсу зможе надати сертифікат, який буде підтверджувати навички, отримані під час навчання.

Збільшення рівня медичної грамотності в цивільному та військовому населенні може виявитися вирішальним фактором для врятованих життів у воєнний період.

Отже, створення платформи для вивчення основ надання домедичної допомоги онлайн є стратегічно важливою ідеєю для кожного із нас.

Розділ-1 «Постановка завдання»

1.1 Вхідні дані для виконання проекту

До виконання проекту у мене є наступні вхідні дані для взаємодії з уроками: заголовки, тексти лекцій, посилання на навчальні відео та тематичні зображення.

Також для взаємодії сервісу із користувачами потрібно мати їхні шаблонні дані: повне ім'я, електронну пошту, пароль, роль(студент/викладач) та досвід роботи(для викладачів).

Ці дані потрібно буде зберігати у базі даних й мати змогу їх додавати, редагувати та видаляти.

1.2 Очікувані результати виконання проекту

Передбачена така функціональність API:

- Аутентифікація та авторизація користувачів через JWT.
- Можливість завантаження файлів (зображень) на сервер.
- Можливість створення, оновлення, видалення та отримання інформації про уроки.
- Функціонал для отримання інформації про користувачів та їхні записи на уроки.
- Хешування паролів для забезпечення безпеки користувачів та усунення вразливостей, пов'язаних із збереженням паролів.
- Можливість зберігати моделі користувача та уроків у базі даних MongoDB.
- Можливість локального розгортання проекту.
- Можливість розширення API (додавання нових функцій та модифікація існуючих) без значних змін проекту.

- Автоматичне відправлення електронних листів при реєстрації користувача та при записі на урок.
- Наявність чіткої та докладної документації до API, включаючи Swagger-документацію.

1.3 Архітектура рішення

Проект має структуру, яка відповідає принципам SRP(Single Responsibility Principle). Згідно якого кожен модуль (контролери, моделі, валідації) відповідає за одну конкретну задачу. Наприклад, контролери обробляють HTTP-запити, моделі відповідають за представлення та збереження інформації про уроки та користувачів в базі даних, у валідаціях реалізовано перевірку коректності та валідність даних, які надходять від користувачів, зокрема вони грають важливу роль у забезпеченні безпеки введених даних.

1.4 Вибір і обґрунтування засобів та технологій використаних для виконання проекту

У цій курсовій роботі використано ряд засобів та технологій для створення високофункціонального та зручного веб-серверу:

- Express.js – мінімалістичний та гнучкий фреймворк для створення серверних додатків на платформі Node.js. Він дозволяє швидко налаштовувати серверну логіку та обробку маршрутів.
- MongoDB - документ-орієнтована база даних, що добре підходить для проектів, де структура даних може змінюватися. Mongoose — це бібліотека для Node.js, з допомогою якої може взаємодіяти бекенд та база даних.
- JWT (JSON Web Token) – бібліотека, яка забезпечує ефективний спосіб забезпечити автентифікацію та авторизацію між клієнтом та сервером за допомогою створення унікальних токенів.

- Bcrypt - шифрування паролів користувачів перед зберіганням у базі даних. Це сприяє безпеці паролів та захисту від несанкціонованого доступу.
- Multer – це middleware для фреймворка express, яка використовується для обробки файлових завантажень. Це корисно для роботи з файлами, такими як зображення, які можуть бути пов'язані з уроками чи користувачами.
- Nodemailer – бібліотека для автоматичної відправки електронних листів користувачам.
- Swagger та Swagger UI Express - документування API та надання інтерфейсу для взаємодії з API. Дозволяє зручно переглядати та тестувати веб-сервер за допомогою веб-інтерфейсу.

Розділ-2 «Теоретичні відомості»

2.1 Технології, використані для виконання проекту

JavaScript:

JavaScript — високорівнева мова програмування. Вона підтримує імперативний, функціональний, подієво-орієнтований підходи, має динамічну типізацію та застосовується для запису послідовних операцій — «сценаріїв» чи «скриптів». Такі послідовності зазвичай інтерпретуються, а не компілюються, а тому не потребують додаткових програм чи інструментів перетворення на інший рівень кодування.

Асинхронний JavaScript є ключовою концепцією, особливо в контексті веб-розробки та використання мови програмування в середовищі браузера чи на сервері з використанням Node.js. Асинхронний підхід дозволяє ефективно взаємодіяти з подіями та операціями, які займають час, наприклад, запити до баз даних, без переривання виконання інших операцій.

Мову веб-програмування JavaScript використовують переважно у Front-end розробці, проте завдяки платформі Node.js нею можна писати і Back-end. JavaScript дає змогу створювати застосунки та сайти. Крім того, вона виходить за межі браузера і може застосовуватись для написання мобільних та десктопних застосунків, вебсерверів тощо.

JavaScript вважають такою, що підходить початківцям, оскільки вона має відносно прості програмні інструкції та для запуску першого рядка коду потребує тільки браузера. Згідно з останнім рейтингом мов програмування, JavaScript залишається найпопулярнішою мовою серед українських програмістів усіх рівнів.

Node.JS:

Node.js - це вільна та відкрита платформа, яка дозволяє виконувати код JavaScript на сервері. Вона побудована на двигуні V8, який розроблений Google для використання в браузері Chrome, що забезпечує швидке та ефективне

виконання JavaScript. Однією з ключових особливостей Node.js є асинхронне програмування та подійний цикл, що дозволяє ефективно обробляти багато запитів без блокування інших операцій.

Node.js входить із пакетним менеджером npm, який спрощує управління залежностями та дозволяє розробникам легко додавати функціональність до своїх проектів. Завдяки різноманітним модулям та фреймворкам, Node.js забезпечує зручні інструменти для розширення функціональності та реагування на зміни в додатку. Node.js широко використовується для створення серверів, веб-додатків, API, а також роботи з базами даних. За допомогою різноманітних модулів та фреймворків, він надає розробникам зручні інструменти для розширення функціональності та реагування на зміни в додатку. Node.js є одним із найпопулярніших виборів для розробки веб-додатків та мікросервісів.

Express.JS:

Express.js представляє собою високопродуктивний веб-фреймворк для розробки веб-додатків та API на платформі Node.js. Його основна перевага полягає в мінімалістичному підході, надаючи лише необхідні засоби для створення ефективних рішень. Ключовий принцип – middleware, функції, які мають доступ до об'єкта запиту (req, res, next). Розробники можуть легко визначати маршрути для обробки різних URL-адрес та HTTP-методів, надаючи гнучкість та контроль над функціональністю додатку.

Також цей фреймворк забезпечує створення HTTP-сервера та його запуск, маршрутизацію шляхів, валідацію даних (Express-validator) та роботу з ресурсами.

MongoDB:

MongoDB - це СУБД, яка використовує концепцію NoSQL для зберігання та обробки даних. Така база даних дозволяє зберігати дані у форматі BSON (бінарний JSON). Це означає, що можна динамічно додавати поля до документів без строгих схем - вказує на асинхронність розробки, особливо коли додаток розвивається та змінюється з часом. Вона ефективно обробляє великі обсяги даних.

Терміни MongoDB:

- **Колекція** - це набір документів, аналог таблиці в MySQL. На відміну від таблиці, одна колекція може містити документи з різною структурою, розмірами, значеннями та зв'язками.
- **Документ** - файл у форматі BSON з даними у вигляді ключів та їх значень, аналог рядка у таблиці MySQL. На відміну від рядка, документ може зберігати інформацію зі складною структурою і є більш гнучким.
- **Поле** - це один запис у документі — ключ, якому відповідає певний набір даних, аналог стовпця у таблиці MySQL.
- **ID**(Ідентифікатор) - це обов'язковий для кожного документа, є аналогом первинного ключа в таблиці MySQL. Допомогає звертатися до даних і служить для відмінності записів один від одного. Створюється автоматично, якщо не заданий явно при створенні нового документа.

Операції із базою даних, які можна виконувати на сервері:

- **read** — читання даних з конкретної бази даних;
- **readWrite** — читання та зміна даних у конкретній базі даних;
- **find** - знаходження документів, які відповідають заданим критеріям;
- **findOne** - знаходження одного документа, який відповідає заданим критеріям;
- **count** – підрахунок кількості документів, які відповідають заданим критеріям;

- **distinct** – знаходження унікальних значення для певного поля;
- **insertOne** - додавання одного документа в колекцію;
- **updateOne** - оновлення одного документа в колекції;
- **deleteOne** - видалення одного документа в колекції;

JWT:

JSON Web Token (JWT) - це стандарт веб-безпеки для створення токенів доступу, які дозволяють передавати інформацію між сторонами як JSON об'єкт. Токени JWT використовуються для автентифікації та входу користувачів.

Структура JWT:

- **Заголовок (header)**. Хедер JWT містить інформацію про те, як обчислюється підпис JWT. Також він містить поле, яке визначає алгоритм хешування
- **Основний вміст (payload)**. Це корисна інформація, яка зберігається в JWT. Ці дані також називають JWT-заявками.
- **Підпис (signature)**. Використовується для забезпечення цілісності та автентичності даних. Основна мета підпису в JWT полягає у тому, щоб гарантувати, що дані, які містяться в токені, не були змінені та що токен був створений відповідно до очікувань.

Bcrypt:

Bcrypt - це бібліотека для хешування паролів в Node.js. Основна мета її використання - це забезпечення безпеки паролів, шляхом їх хешування та зберігання хешованого значення в базі даних замість паролю у відкритому вигляді.

Для хешування паролю використовується криптографічна функція, а також salt. Salt - це випадковий рядок, який додається до паролю перед хешуванням. Вона ускладнює хакерські атаки перебором та атаки словником.

При авторизації користувача та введенні свого паролю, він хешується тим самим алгоритмом та порівнюється зі збереженим хешем, який раніше був збережений в базі даних. Якщо отриманий хеш відповідає збереженому хешу в базі даних, то пароль введений користувачем вірний й користувач отримує доступ до свого акаунту. У протилежному випадку, якщо хеші не збігаються, повертається HTTP-статус 404 (Not Found).

Multer:

Multer - це middleware для обробки файлів у веб-додатках, які використовують Express.js, побудоване на платформі Node.js. Ця бібліотека спрощує завантаження файлів з клієнтського браузера на сервер. Multer дозволяє визначати різні параметри завантаження, такі як розмір файлу, обрані типи файлів, місце збереження тощо.

Для завантаження файлів на сервер у проєкті потрібно імпортувати модуль, вказати місце збереження завантажених файлів та реалізувати запит POST, який буде визначати шлях й у його параметрах вказується middleware Multer.

Nodemailer:

Nodemailer - це бібліотека для відправлення електронних листів в застосунках Node.js. Вона дозволяє легко налаштовувати та відправляти електронні листи з вашого сервера. Інтегрується із Gmail, Outlook та іншими поштовими службами.

Щоб реалізувати автоматичне надсилання електронних листів потрібно створити transporter для використання сервісу Gmail, де вказується email пошти, з якої будуть надсилатись листи та спеціальний згенерований 16-значний пароль для надання серверу доступу до облікового запису. Далі потрібно

визначаємо параметри листа: пошту відправника та отримувача, тему та текст листа.

Методом **transporter.sendMail** відбувається надсилання листа. Також зазвичай додається обробник помилок.

Swagger та Swagger UI Express:

Swagger - це інструмент, який спрощує процес розробки, документування та тестування API. Цей засіб надає стандартизований спосіб проектування, документування та тестування веб-сервера, і він інтегрується з фреймворками Node.js, такими як Express.

Swagger автоматично генерує інтерактивну документацію API на основі специфікації. Ця документація доступна через веб-інтерфейс, що дозволяє розробникам зрозуміти, як взаємодіяти з вашим API. Також може перевіряти вхідні запити на відповідність визначеній специфікації, гарантуючи, що дані введені правильно. Це допомагає виявити помилки на ранніх етапах розробки.

Swagger UI Express – це інтерактивний веб-сервіс, який дозволяє переглядати та тестувати API за допомогою автоматично створеного інтерфейсу.

Для підключення Swagger до проекту потрібно імпортувати усі потрібні модулі й скористатись командою:

```
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
```

У цьому випадку Swagger UI буде доступний за URL-адресою `/api-docs` на сервері.

Для встановлення кожного із модулів та фреймворків використовую команду **“npm install”**.

2.2 Програмні засоби, використані для виконання проекту

Visual Studio Code:

VSCode - це безкоштовний, відкритий редактор від Microsoft, призначений для розробки різних видів програмного забезпечення. Підтримує багато різних мов програмування, включаючи JavaScript, Python, Java, C++, і багато інших. Це робить його універсальним редактором для багатьох розробників.

У меню EXTENSIONS можна встановити пакет із компілятором для розробки на JS. Також містить зручний провідник, за допомогою якого можна керувати файлами, інформативне розміщення та підсвічування коду, що спрощує його читабельність та розуміння. У нижній частині вікна розміщена консоль, у яку можна вводити команди для керування програмою, отримувати сповіщення про помилки.

Для відстежування помилок у коді існує Debugger, за допомогою якого можна поставити точку зупинки й виконувати код покроково.

Insomnia:

Insomnia - це інструмент для розробки та тестування API. Він дозволяє розробникам легко надсилати HTTP-запити, перевіряти відповіді серверів, та налагоджувати та тестувати різноманітні типи запитів.

Ця програма дозволяє створювати різноманітні запити. Наприклад, POST, PUT, PATCH, PUT, DELETE. Також для зручності можна формувати колекції із запитів. У тілі запитів можна передавати дані у форматах JSON, XML та інші. Якщо у сервері присутня генерація JWT, то разом із запитом від певного користувача може передаватись токен. Також можна використовувати інші методи аутентифікації для тестування, такі як базова аутентифікація, OAuth...

Після відправлення запиту можна переглядати та зберігати відповіді сервера.

MongoDB Compass:

MongoDB Compass - це графічний інтерфейс користувача для роботи з базою даних MongoDB. Він надає візуальні засоби для взаємодії з MongoDB, дозволяючи розробникам та адміністраторам зручно переглядати, редагувати та аналізувати дані в базі даних.

При використанні хмарного сервісу Atlas для MongoDB, Compass забезпечує зручний спосіб підключитися до бази даних, використовуючи URL.

2.3 Ключові елементи програмування на стороні сервера, використані при виконанні проекту

Робота із JSON:

JSON(JavaScript Object Notation) – це текстовий формат обміну даними, який є важливою частиною веб-розробки та використовується для передачі даних між клієнтом та сервером.

Його структура складається із ключа та значення. Серед значень, які можна передавати є текстові рядки, числові значення, логічні значення та null, Серед значень: текстові рядки, числові значення, логічні значення, null, списки та об'єкти.

Синтаксис ключа та значення: **"name": "Ivan Petrenko"**.

Для того щоб використовувати JSON у програмі JavaScript використовують методи для його перетворення у текстовий рядок - `JSON.stringify()` та зворотню операцію(перетворення рядка JS у JSON) - `JSON.parse()`.

HTTP запити до сервера:

HTTP запити - це спосіб взаємодії клієнтів із серверами за допомогою протоколу передачі даних HTTP. Стандартний запит складається із рядка запиту, який включає метод, шлях та HTTP, заголовків та тіла.

HTTP запити (requests) поділяють на:

- GET - отримати ресурс
- POST- змінити дані ресурсу
- PUT, PATCH - замінити дані ресурсу
- DELETE - видалити ресурс
- HEAD - отримати ресурс (без тіла відповіді).

Після виконання запиту сервер нам повертає відповідь(response), яка складається із рядка відповіді, який містить HTTP, код статусу та опис статусу, заголовків та тіла.

Коди статусу поділяють на:

- 1xx - інформаційні коди
- 2xx – успіх
- 3xx – перенаправлення
- 4xx - помилка на стороні клієнта
- 5xx - помилка на стороні сервера

Автентифікація та авторизація:

Автентифікація – це певна перевірка користувача на ідентичність. Є важливою частиною безпеки програми, тому що допомагає запобігти несанкціонованому доступу до ресурсів. Автентифікація створена для того, щоб подивитись чи користувач, наприклад, зареєстрований в системі. Зазвичай складається із username та password. При неуспішній автентифікації сервер повертає код 401.

Відрізняється від авторизації тим, що авторизація оприділяє для зареєстрованого користувача його ролі та можливості. Також автентифікація відбувається в першу чергу. При неуспішній автентифікації сервер повертає код 403.

Автентифікація ділиться на **base authentication, session based authentication, та token based authentication.**

Основна ідея базової автентифікації полягає в тому, що користувач представляє ім'я користувача та пароль, які хешуються та передаються на сервер у заголовку HTTP-запиту.

Session based authentication ґрунтується на тому, що при переданні вірних персональних даних користувачем сервер створює сесію. Сесія містить унікальний id. Також у цьому типі автентифікації використовуються cookies для зберігання важливої інформації(підтримки автентифікації, стану сесії) на стороні клієнта.

Token based authentication орієнтується на наданні сервером клієнту унікального токена після введення правильних username та password. Із кожним запитом клієнта серверу передається його токен. Це спосіб дізнатись який клієнт виконав саме цей запит. Токени мають обмежений термін дії(зазвичай доки користувач не вийде із аккаунту).

Розділ-3 «Реалізація проекту. Послідовний опис виконання проекту з документуванням коду»

3.1 Підключення модулів, фреймворків, бази даних, реалізація завантаження зображень на сервер та запуск веб-сервера

В основному файлі проекту index.js імпортую такі модулі та бібліотеки: express, mongoose, bodyParser, multer, swaggerUi та swaggerDocument. Підключаю urlencodedParser й multer.

```
import express from 'express';
import mongoose from 'mongoose';
import bodyParser from 'body-parser';
import {registerValidation, loginValidation, lessonCreateValidation} from
'./validations.js';
import * as UserController from './Controllers/UserController.js';
import * as LessonController from './Controllers/LessonController.js';
import checkAuth from './utils/checkAuth.js';
import multer from 'multer';
import handleValidationErrors from './utils/handleValidationErrors.js';

import swaggerUi from 'swagger-ui-express';
import swaggerDocument from './swagger.json' assert { type: 'json' };

var urlencodedParser = bodyParser.urlencoded({extended: false});
const upload = multer({storage});
```

Також реалізую підключення до бази даних Mongo DB:

```
mongoose
  .connect(
    'mongodb+srv://miksimmiks:wqNIvgdk4tYkP5wm@cluster0.018qrom.mongodb.net/blog?re
tryWrites=true&w=majority')
  .then(() => console.log('DB OK'))
  .catch((err) => console.log('DB error', err));
```

Для завантаження файлів на сервер використовуючи бібліотеку multer створюю екземпляр diskStorage, налаштовую місце зберігання й ім'я файлу:

```
const storage = multer.diskStorage({
  destination: (_, __, cb) => {
    cb(null, 'uploads');
  },
  filename: (_, file, cb) =>{
    cb(null, file.originalname);
  },
});
```

Реалізую middleware для завантаження статичних файлів, обробки JSON та для обробки тіла запиту у форматі "text/plain". Це буде використано в подальшому при створенні html-форм для реєстрації та входу.

```
app.use('/uploads', express.static('uploads'));
app.use(express.json());
app.use(bodyParser.raw({ type: 'text/plain' }));
```

Запускаю сервер на порті 4444 за допомогою фреймворка Express.js. Додаю обробку помилок та повідомлення в консолі при успішному запуску.

```
app.listen(4444, (err) => {
  if(err) {
    return console.log(err);
  }
  console.log('Server OK');
});
```

3.2 Створення моделей користувача та уроку

У файлах `user.js` та `lesson.js` створюю моделі. Вони призначені для опису та взаємодії з даними у базі даних. Їхня основна роль полягає в тому, щоб визначати структуру та формат даних, які зберігаються у базі даних, і забезпечувати зручний інтерфейс для взаємодії з ними.

У файлі **user.js** визначаю модель користувача, яка містить такі поля: `fullName`(повне ім'я), `email`(електронна пошта), `workExperience`(досвід роботи),

selfieUrl(посилання на аватар), role(роль користувача), passwordHash(зашифрований пароль) та lessons(масив уроків, на які записаний користувач).

У цих 2-х файлах містяться такі ключові слова:

- **Enum** вказує на те, що поле role може мати лише одне значення з вказаного переліку ('student' або 'teacher').
- **Default** задає значення student за замовчуванням для поля role (якщо користувач не вказав).
- **Required** вказує, що поля fullName, email, role, passwordHash є обов'язковими для заповнення.
- **Unique** гарантує унікальність значень для поля email.
- Опція **timestamps: true** додає автоматичне створення та оновлення поля "createdAt" та "updatedAt" для відстеження часу створення та останньої модифікації запису.

```
import mongoose from 'mongoose';
const UserSchema = new mongoose.Schema({
  fullName: {
    type: String,
    required: true,
  },
  email:{
    type: String,
    required: true,
    unique: true,
  },
  workExperience:{
    type: String,
  },

  selfieUrl:{
    type: String,
  },
});
```

```

    role: {
      type: String,
      enum: ['student', 'teacher'],
      default: 'student',
      required: true,
    },

    passwordHash: {
      type: String,
      required: true,
    },
    lessons: [{
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Lesson',
    }],
  },
  timestamps: true,
},
);

export default mongoose.model('User', UserSchema);

```

У файлі **lesson.js** визначаю модель уроку, яка містить такі поля: title(заголовок уроку), text(текст уроку), videoUrl(посилання на відео), photoUrl(посилання на фото) та user(вчитель, який створив урок).

Властивість **ref: 'User'** вказує на те, що значення, що міститься в полі user, повинно відповідати ідентифікатору іншого об'єкта у колекції, яке визначене як 'User'. За допомогою цього, Mongoose розуміє, що це поле є зовнішнім ключем, яке посилається на об'єкти в колекції 'User'.

```

import mongoose from 'mongoose';

const LessonSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,

```

```

    },
    text:{
      type: String,
      required: true,
      unique: true,
    },
    videoUrl:{
      type: String,
    },
    photoUrl:{
      type: String,
    },
    user:{
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true,
    },
  },
  {
    timestamps: true,
  },
);

export default mongoose.model('Lesson', LessonSchema);

```

3.3 Реалізація контролера, у якому містяться усі запити для взаємодії із користувачами

У моєму бекенд - проєкті контролери (controllers) використовуються для обробки запитів, отриманих від ПЗ Insomnia або від html-форм. Контролери взаємодіють з моделями користувача та уроків та сервісами, виконуючи логіку, необхідну для обробки запитів.

Загалом створено 2 файли: UserController.js та LessonController.js, які містять запити для роботи із користувачами та уроками відповідно.

У UserController.js спершу підключаю необхідні для авторизації(jwt), хешування паролів(bcrypt), надсилання електронних листів(nodemailer) та роботи із раніше створеними моделями користувачів(UserModel) модулі.

```
import jwt from 'jsonwebtoken';  
import bcrypt from 'bcrypt';  
import UserModel from '../models/user.js';  
import nodemailer from 'nodemailer';
```

- Переходжу до створення запиту для реєстрації. Оскільки подальші запити будуть взаємодіяти із базою даних роблю їх асинхронними.

Використовуючи бібліотеку bcrypt, генерується "сіль" (salt) за допомогою bcrypt.genSalt(10), а потім генерується хеш паролю за допомогою bcrypt.hash(password, salt). Цей хеш паролю потім використовується для збереження в базі даних замість самого паролю, забезпечуючи безпеку.

Реалізую отримання введених користувачем даних та створення нового об'єкту користувача (doc) за допомогою моделі UserModel.

За допомогою бібліотеки jsonwebtoken створюється JWT-токен, який містить ідентифікатор користувача та його роль. Токен підписується "секретним ключем" (в моєму випадку 'secret123') та має термін дії 30 днів (expiresIn: '30d').

Створюється об'єкт transporter з використанням бібліотеки nodemailer для надсилання електронної пошти. Відправляється лист на вказану електронну адресу користувача із підтвердженням успішної реєстрації.

Відповідь сервера містить дані новоствореного користувача (без хеша паролю) та JWT-токена.

В разі виникнення помилок (наприклад, при збереженні в базу даних чи надсиланні листа), вони логуються та повертаються як відповідь сервера зі статусом 500 і повідомленням про неуспішну реєстрацію.

```

export const register = async (req, res) => {

  try{
    const password = req.body.password;
    const salt = await bcrypt.genSalt(10);
    const hash = await bcrypt.hash(password, salt);

    const doc = new UserModel({
      email:req.body.email,
      fullName:req.body.fullName,
      workExperience:req.body.workExperience,
      avatarUrl:req.body.avatarUrl,
      selfieUrl:req.body.selfieUrl,
      passwordHash: hash,
      role: req.body.role || 'teacher',
    });

    const user = await doc.save();

    const token = jwt.sign({
      _id: user._id,
      role: user.role,
    }, 'secret123',
    {
      expiresIn:'30d',
    }
    );

    const {passwordHash, ...userData} = user._doc;

    res.json({
      ... userData,
      token,
    });
  }

  const transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: 'medcourses.message@gmail.com',
      pass: 'ditgplqtpmaamqik'
    }
  });

```

```

    }
  });

  const mailOptions = {
    from: '<medcourses.message@gmail.com>',
    to: req.body.email,
    subject: 'Succesfull register',
    text: 'Вітаю, ' + req.body.fullName + '. Ви успішно зареєструвалися на сайті медкурсів. Приємного користування сервісом!'
  };

  transporter.sendMail(mailOptions, (error, info) => {
    if (error) {
      console.log(error);
    } else {
      console.log('Email sent: ' + info.response);
    }
  });

} catch(err){
  console.log(err);
  res.status(500).json({
    message: 'Неуспішна реєстрація',
  });
}
};

```

- Створюю метод для входу(авторизації) користувача в системі. Спершу метод отримує введені юзером логін та пароль, використовуючи модель користувача UserModel, шукає користувача за його електронною поштою в базі даних.

Якщо користувач не знайдений, сервер повертає відповідь зі статусом 404 та повідомленням про те, що користувача не знайдено. Якщо ж такий email існує, то використовуючи бібліотеку bcrypt, порівнюється введений користувачем пароль із збереженим у базі хешем паролю. Якщо паролі не співпадають, сервер повертає відповідь зі статусом 404 та повідомленням про неправильний пароль. Якщо ж

авторизація успішна, тоді генерується JWT-токен та відповідь сервера: дані користувача (без пароллю) та JWT-токен, який буде використаний для майбутньої взаємодії з уроками.

У випадку виникнення помилок, таких як помилки бази даних чи інші внутрішні помилки, сервер повертає відповідь із статусом 500 та повідомленням про неуспішний вхід.

```
export const login = async (req, res) => {
  try{
    const user = await UserModel.findOne({email: req.body.email});

    if (!user){
      return res.status(404).json({
        message: 'Користувач не знайдений',
      });
    }

    const isValidPass = await bcrypt.compare(req.body.password,
user._doc.passwordHash);

    if(!isValidPass){
      return res.status(404).json({
        message: 'Неправильний пароль',
      });
    }
  }
  const token = jwt.sign({
    _id: user._id,
  }, 'secret123',
  {
    expiresIn: '30d',
  }
  );

  const {passwordHash, role, ...userData} = user._doc;

  res.json({
    ... userData,
    role,
```

```

    token,
  });

} catch(err){
  console.log(err);
res.status(500).json({
message: 'Неуспішний вхід',
});
}
};

```

- Створюю метод, який отримує інформацію про поточного авторизованого користувача за його JWT. Він складається із отримання ID користувача, пошук в БД за отриманим ідентифікатором.

Якщо користувача не знайдено, сервер повертає відповідь із статусом 404 та повідомленням про те, що користувача не знайдено.

Якщо ж такий користувач існує, то відбувається відповідь сервера, яка містить дані користувача без хешу паролю.

У випадку виникнення помилок на стороні сервера, повертається відповідь із статусом 500.

```

export const getMe = async (req, res) => {
  try {
    const user = await UserModel.findById(req.userId);
    if (!user) {
      return res.status(404).json({
        message: 'Користувач не знайдений',
      });
    }
    const { passwordHash, ...userData } = user._doc;

    res.json(userData);

  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Немає доступу',
    });
  }
};

```

```
});
}
};
```

- Наступний метод є подібним до попереднього, адже також виводить дані про користувачів, але тепер вже усіх, які зареєстровані на сервері.

З використанням моделі користувача `UserModel`, викликається метод `find()` для отримання всіх користувачів, які знаходяться у базі даних MongoDB. Також міститься обробка помилок. При успіху, виводяться дані про усіх користувачів у форматі JSON.

```
export const getAllUsers = async (req, res) => {
  try {

    const users = await UserModel.find();
    res.json(users);
  } catch (err) {
    console.error(err);
    res.status(500).json({
      message: 'Не вдалося отримати користувачів',
    });
  }
};
```

- У файлі `index.js` реалізую маршрутизацію та, якщо потрібно, для деяких запитів додаю валідації й перевірку авторизації.

```
app.post('/auth/login', urlencodedParser, loginValidation,
handleValidationErrors, UserController.login );

app.post('/auth/register', urlencodedParser, registerValidation,
handleValidationErrors, UserController.register );

app.get('/auth/me', checkAuth, UserController.getMe);

app.get('/users', UserController.getAllUsers);
```

3.4 Реалізація контролера, у якому містяться усі запити для взаємодії користувачів із уроками

- У файлі LessonController.js спершу підключаю необхідні модулі: раніше створені модель користувача та уроку, JWT(для створення персональних токенів) та nodemailer(для надсилання електронних листів).

```
import LessonModel from '../models/lesson.js';
import UserModel from '../models/user.js';
import jwt from 'jsonwebtoken';
import nodemailer from 'nodemailer';
```

- Реалізую запит для створення уроку вчителем.

Спершу отримання інформації про авторизованого користувача через JWT, шукаю його у базі даних, та якщо такий знайдений – перевіряю його роль. Якщо роль не "teacher", повертається відповідь із статусом 403 та повідомленням про те, що користувач не має дозволу на створення уроку.

Якщо користувач має роль "вчитель", створюється новий об'єкт уроку (LessonModel). Заголовок, текст, посилання на зображення, відео та фото для уроку беруться з тіла запиту. Id автора уроку записується у БД разом із матеріалами.

При виникненні помилок на стороні сервера повертається код помилки – 500.

```
export const create = async (req, res) => {
  try {
    const userId = req.userId;
    const user = await UserModel.findById(userId);
    const userRole = user ? user.role : null;
    if (userRole !== 'teacher') {
      return res.status(403).json({
        message: 'Ви не маєте дозволу на створення уроку',
      });
    }
  }
}
```

```

const doc = new LessonModel({
  title: req.body.title,
  text: req.body.text,
  imageUrl: req.body.imageUrl,
  videoUrl: req.body.videoUrl,
  photoUrl: req.body.photoUrl,
  user: userId,
});

const lesson = await doc.save();
user.lessons.push(lesson._id);
await user.save();

res.json(lesson);
} catch (err) {
  console.log(err);
  res.status(500).json({
    message: 'Не вдалось створити урок',
  });
}
};

```

- Створюю запит для отримання усіх уроків.

З використанням моделі уроку LessonModel, викликається метод find() для отримання всіх уроків, які знаходяться у базі даних MongoDB. За допомогою методу **res.json(lessons)** повертаю користувачу, який здійснив запит список усіх уроків у форматі JSON.

Якщо ж виникає помилка на стороні сервера, повертається код – 500 та повідомлення ‘Не вдалось получить уроки’.

```

export const getAll = async (req, res) => {
  try{
    const lessons = await LessonModel.find().populate('user').exec();

    res.json(lessons);
  } catch(err){
    console.log(err);
  }
};

```



```

    res.status(500).json({
      message: 'Не вдалось получить уроки',
    });
  }
};

```

- Створюю запит для отримання уроку за його ID. Спершу отримується id уроку, яке користувач вписує у шляху. Далше за допомогою методу `.findById` знаходимо такий урок у БД.

Якщо такий існує – повертаємо JSON із даними, які містяться в уроці.

Якщо Id не знайдене – повертається статус 404, якщо ж виникає помилка на стороні сервера – повертається код – 500.

```

export const getOne = async (req, res) => {
  try {
    const lessonId = req.params.id;

    const lesson = await LessonModel.findById(lessonId);

    if (!lesson) {
      return res.status(404).json({
        message: 'Урок не найден',
      });
    }

    res.json(lesson);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не удалось получить урок',
    });
  }
};

```

- Реалізовую метод для видалення уроків. Він приймає id уроку у URL та токен користувача. Спершу йде перевірка за id уроку чи існує він у БД.

Якщо такого немає – повертається помилка 404.

За допомогою JWT перевіряється чи користувач, який хоче видалити урок є його творцем. Якщо це інший користувач, то сервер повертає відповідь ‘Ви не маєте дозволу на видалення цього уроку’ із статусом 403.

Використовуючи метод `findOneAndDelete`, урок видаляється з бази даних за його ідентифікатором. Якщо урок успішно видалено, сервер відправляє відповідь із статусом 200 та об'єктом, який містить `success: true`.

У випадку виникнення помилок на стороні сервера повертається відповідь із статусом 500.

```
export const remove = async (req, res) => {
  try {
    const lessonId = req.params.id;
    const userId = req.userId;

    const lesson = await LessonModel.findById(lessonId);

    if (!lesson) {
      return res.status(404).json({
        message: 'Урок не знайдено',
      });
    }

    if (lesson.user.toString() !== userId) {
      return res.status(403).json({
        message: 'Ви не маєте дозволу на видалення цього уроку',
      });
    }

    const result = await LessonModel.findOneAndDelete({ _id: lessonId });

    if (!result) {
      return res.status(404).json({
        message: 'Урок не знайдено',
      });
    }
  }
}
```

```

    res.json({
      success: true,
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалося видалити урок',
    });
  }
};

```

- Створюю запит для оновлення даних уроку. Як і в попередньому запиті тут реалізовано перевірку за id уроку чи такий урок у БД.

Якщо такого немає – повертається помилка 404.

За допомогою JWT перевіряється чи користувач, який хоче оновити урок є його творцем. Якщо це інший користувач, то сервер повертає відповідь ‘Ви не маєте дозволу на оновлення цього уроку’ із статусом 403.

Якщо ж усі перевірки пройшли успішно, то виконується метод **updateOne** для оновлення даних у БД й повертається статус 200 із **success: true**.

При виникненні помилки на стороні сервера повертається відповідь із статусом 500.

```

export const update = async (req, res) => {
  try {
    const lessonId = req.params.id;
    const userId = req.userId;

    const lesson = await LessonModel.findById(lessonId);

    if (!lesson) {
      return res.status(404).json({
        message: 'Урок не знайдено',
      });
    }
  }
};

```

```
    });  
  }  
  
  if (lesson.user.toString() !== userId) {  
    return res.status(403).json({  
      message: 'Ви не маєте дозволу на оновлення цього уроку',  
    });  
  }  
  
  await LessonModel.updateOne(  
    {  
      _id: lessonId,  
    },  
    {  
      title: req.body.title,  
      text: req.body.text,  
      imageUrl: req.body.imageUrl,  
      videoUrl: req.body.videoUrl,  
      photoUrl: req.body.photoUrl,  
      user: userId,  
    },  
  );  
  
  res.json({  
    success: true,  
  });  
} catch (err) {  
  console.log(err);  
  res.status(500).json({  
    message: 'Не вдалося оновити урок',  
  });  
}  
};
```

- Наступним створюю запит для можливості запису користувачів на певний урок. Цей запит приймає lessonId та JWT користувача.

Виконуються пошуки у базі даних відповідного користувача та уроку за допомогою методу **.findById**.

Якщо ж урок чи користувач не знайдено повертаються помилки із кодами 404 та 400 відповідно.

Далі йде перевірка чи користувач часом вже не зареєстрований на цей урок. Якщо виявляється, що запис уже є, то повертається код 400 та повідомленням про те, що користувач вже зареєстрований.

Коли усі перевірки пройдено, то ID уроку додається до масиву lessons користувача, і його дані оновлюються в базі даних.

Аналогічно, як і при реєстрації користувача тут реалізовано надсилання електронного листа через бібліотеку nodemailer із повідомленням про успішну реєстрацію на урок та його деталями.

Після успішної реєстрації на урок повертається код статусу 200 та JSON із вмістом: **success: true**.

У випадку виникнення помилок на стороні сервера повертається відповідь із статусом 500.

```
export const registerForLesson = async (req, res) => {
  try {
    const lessonId = req.body.lessonId;
    const userId = req.userId;

    const user = await UserModel.findById(userId);
    const lesson = await LessonModel.findById(lessonId);

    if (!lesson) {
      return res.status(404).json({
        message: 'Урок не знайдено',
      });
    }
  }
}
```

```

    }

    if (user.lessons.includes(lessonId)) {
        return res.status(400).json({
            message: 'Ви вже зареєстровані на цей урок',
        });
    }

    user.lessons.push(lessonId);
    await user.save();

    sendRegistrationEmail(user.email, user.fullName, lessonId, lesson.title);

    res.json({
        success: true,
    });
} catch (err) {
    console.log(err);
    res.status(500).json({
        message: 'Не вдалося зареєструватися на урок',
    });
}
};

const sendRegistrationEmail = (toEmail, fullName, lessonId, lessonTitle) => {
    const transporter = nodemailer.createTransport({
        service: 'gmail',
        auth: {
            user: 'medcourses.message@gmail.com',
            pass: 'ditgplqtpmaamqik',
        }
    });

    const mailOptions = {
        from: '<medcourses.message@gmail.com>',
        to: toEmail,
        subject: 'Registration for a Lesson',
        text: `Вітаю, ${fullName},\n\n Ви успішно зареєструвались на курс "${lessonTitle}" (ID: ${lessonId}).\n\nПриємного навчання!`,
    };
};

```

```

transporter.sendMail(mailOptions, (error, info) => {
  if (error) {
    console.log(error);
  } else {
    console.log('Email sent: ' + info.response);
  }
});
};

```

- Останнім запитом є отримання назв усіх уроків, на які записаний користувач за його JWT.

Спершу отримується id користувача за допомогою переданого ним JWT.

Методами `findById` та `populate` отримуються дані користувача із БД та повна інформація про уроки, на які він зареєстрований.

Якщо ж користувач не знайдений – повертається статус 404.

При успішній перевірці за допомогою методу `res.json(lessonTitles)` у форматі JSON виводяться заголовки усіх уроків, на які зареєстрований користувач.

При виникненні помилки на стороні сервера повертається відповідь із статусом 500.

```

export const getUserLessons = async (req, res) => {
  try {
    const userId = req.userId;

    const user = await UserModel.findById(userId).populate('lessons');

    if (!user) {
      return res.status(404).json({
        message: 'Користувач не знайдений',
      });
    }

    const lessons = user.lessons;
  }
};

```

```
const lessonTitles = lessons.map(lesson => lesson.title);
res.json(lessonTitles);
} catch (err) {
  console.log(err);
  res.status(500).json({
    message: 'Не вдалося отримати уроки користувача',
  });
}
};
```

- У файлі index.js реалізую маршрутизацію та, якщо потрібно, для деяких запитів додаю валідації й перевірку авторизації.

```
app.get('/lessons', LessonController.getAll);
app.get('/lessons/:id', LessonController.getOne);
app.post('/lessons', checkAuth, lessonCreateValidation, handleValidationErrors,
LessonController.create);
app.delete('/lessons/:id', checkAuth, LessonController.remove);
app.patch('/lessons/:id', checkAuth, lessonCreateValidation, handleValidationErrors,
LessonController.update);
app.post('/auth/registerForLesson', checkAuth, LessonController.registerForLesson);

app.get('/user/lessons', checkAuth, LessonController.getUserLessons);
```

3.5 Створення валідацій та middleware для аутентифікації користувачів за допомогою JWT

- У файлі Validations.js реалізую набір правил валідації для обробки даних, які надходять в тілі HTTP-запитів у маршрутах для автентифікації, реєстрації та створення уроків.

Наприклад, методи **.isEmail()** перевіряє, чи введений текст є електронною адресою, **.isLength({ min: 5 })** перевіряє, чи не менший 5 символів у довжину,

.isURL() перевіряє чи введений текст є посиланням. Ключове слово **optional** означає, що це поле не є обов'язковим для заповнення.


```
import {body} from 'express-validator'

export const loginValidation = [
  body('email', 'Неправильний формат пошти').isEmail(),
  body('password', 'Пароль має бути більшим за 5 символів').isLength({ min:
5}),
];

export const registerValidation = [
  body('email', 'Неправильний формат пошти').isEmail(),
  body('password', 'Пароль має бути більшим за 5 символів').isLength({ min:
5}),
  body('fullName', 'Неправильний формат імені').isLength({ min: 2}),
  body('workExperience', 'Неправильний формат досвіду
роботи').optional().isString(),
  body('avatarUrl', 'Неправильний формат посилання').optional().isURL(),
];

export const lessonCreateValidation = [
  body('title', 'Неправильний формат заголовка').isLength({ min:
3}).isString(),
  body('text', 'Введіть текст уроку').isLength({ min: 3}).isString(),
  body('imageUrl', 'Неправильний формат посилання').optional().isString(),
  body('videoUrl', 'Неправильний формат посилання').optional().isURL(),
];
```

- У файлі CheckAuth реалізовано middleware для перевірки токена автентифікації JWT в HTTP-заголовку Authorization

Спершу ця функція перевіряє наявність та витягує токен з заголовка Authorization.

Якщо токен присутній, спробує розшифрувати його, використовуючи секретний ключ 'secret123'.

Якщо розшифрування вдається, витягує ідентифікатор користувача (_id) та роль користувача (role) і додає їх до об'єкта req для подальшого використання.

Якщо є помилка під час розшифрування або токен відсутній, повертає відповідь зі статусом 403 (Заборонено) та повідомленням про відсутність доступу.

Якщо виникають помилки при валідації (це перевіряється методом `errors.isEmpty`), то у консоль повертається масив із помилками та їхні описи.

```
import jwt from 'jsonwebtoken';

export default (req, res, next) => {
  const token = (req.headers.authorization || '').replace(/Bearer\s?/, '');

  if (token) {
    try {
      const decoded = jwt.verify(token, 'secret123');

      req.userId = decoded._id;
      req.userRole = decoded.role;
      next();
    } catch (e) {
      return res.status(403).json({
        message: 'Немає доступу',
      });
    }
  } else {
    return res.status(403).json({
      message: 'Немає доступу',
    });
  }
};
```

```
{ validationResult } from 'express-validator';
export default (req, res, next) => {
  const errors = validationResult(req);

  if(!errors.isEmpty()){
    return res.status(400).json(errors.array());
  }
}
```

```
next();
};
```

3.6 Реалізація Swagger-документації

- Імпортую бібліотеки **swaggerUi** та **swaggerDocument** які дозволяють легко інтегрувати Swagger в Express додаток та файл `swagger.json`, який містить Swagger-специфікацію.

Встановлюю `middleware` для обробки запитів, які відповідають за шляхом `/api-docs`. Використовуючи `swaggerUi.serve`, надаються ресурси Swagger UI, такі як CSS, JavaScript.

```
import swaggerUi from 'swagger-ui-express';
import swaggerDocument from './swagger.json' assert { type: 'json' };
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
```

- У файлі `swagger.json` створюю специфікацію, яка відповідає вимогам мого проекту.

Спершу, вказую загальну інформацію, таку як версію, яка використовується, заголовок, опис та протоколи передачі даних.

Наступним етапом є опис HTTP-методів, які містяться в проекті. Він включає вид методу, коротки опис операцій, теги, до яких відноситься цей маршрут, параметри, які він приймає, схему запиту та можливі відповіді сервера.

```
"swagger": "2.0",

"info": {
  "version": "1.0.0",
  "title": "Medcourses",
  "description": "Medcourses Platform"
},
"basePath": "/",
```

```

"schemes": ["http", "https"],
"consumes": ["application/json"],
"produces": ["application/json"],
"definitions": {

```

Приклад частини специфікації для запитувхodu користувача.

```

"/auth/login": {
  "post": {
    "summary": "Login",
    "tags": ["Authentication"],
    "parameters": [
      {
        "name": "body",
        "in": "body",
        "required": true,
        "schema": {
          "$ref": "#/definitions/LoginRequestBody"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "Login successful",
        "schema": {
          "$ref": "#/definitions/UserResponse"
        }
      },
      "404": {
        "description": "User not found"
      },
      "500": {
        "description": "Internal server error"
      }
    }
  }
}

```

Відкривши URL `localhost:4444/api-docs/` отримаємо зручний інтерфейс із документацією та функціоналом для тестування нашого API:

The screenshot displays the Swagger API documentation interface. At the top, there is a 'Schemes' dropdown menu set to 'HTTP'. The interface is organized into three main sections: Authentication, User, and Lessons, each with a collapse/expand arrow on the right.

- Authentication**
 - POST** `/auth/register` Register a new user
 - POST** `/auth/login` Login
- User**
 - GET** `/auth/me` Get current user
 - GET** `/user/lessons` Get lessons for a user
- Lessons**
 - GET** `/lessons` Get all lessons
 - POST** `/lessons` Create a new lesson
 - GET** `/lessons/{id}` Get a specific lesson
 - DELETE** `/lessons/{id}` Delete a lesson
 - PATCH** `/lessons/{id}` Update a lesson
 - POST** `/auth/registerForLesson` Register for a lesson

Розділ-4 «Тестування проекту»

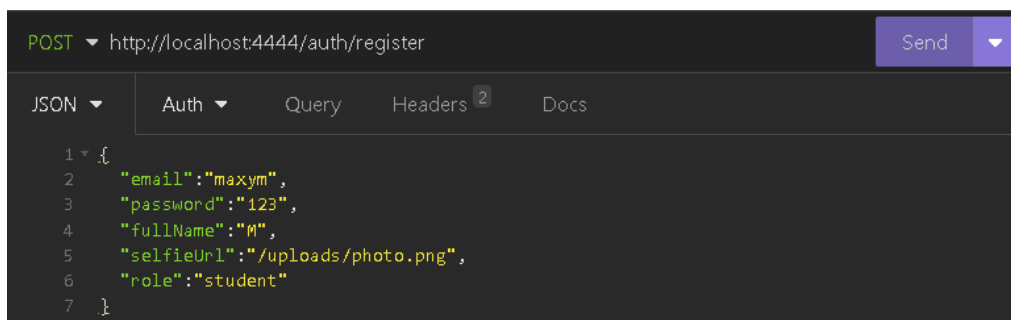
4.1 Методика тестування результатів

Тестування є важливою частиною розробки програмного забезпечення, адже дозволяє переконатися, що застосунок працює так, як очікується, і відповідає вказаним вимогам. Тестування буду проводити за допомогою ПЗ Insomnia, яке дає змогу передавати дані у тілі запитів та повертати відповідь.

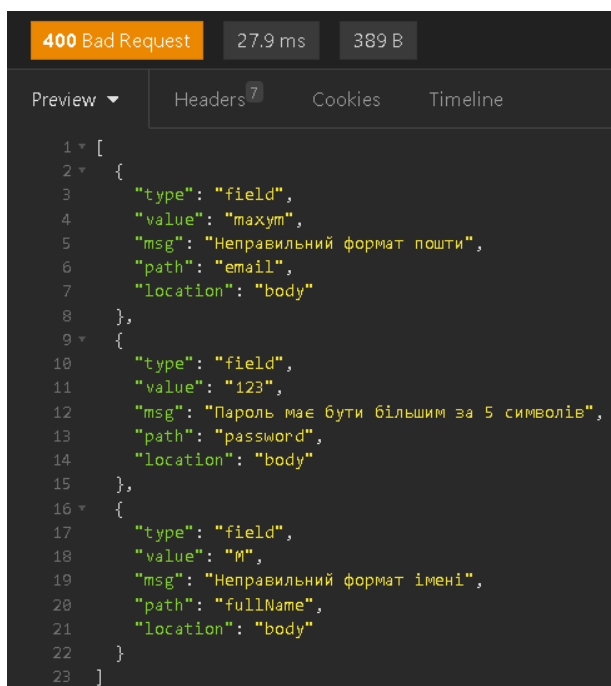
4.2 Результати тестування

- Передання некоректних даних у тілі запиту

Для цього при запиті реєстрації вводжу у полі email ім'я юзера. У полі повного імені вводжу один символ, а пароль роблю меншим, ніж 5 символів.

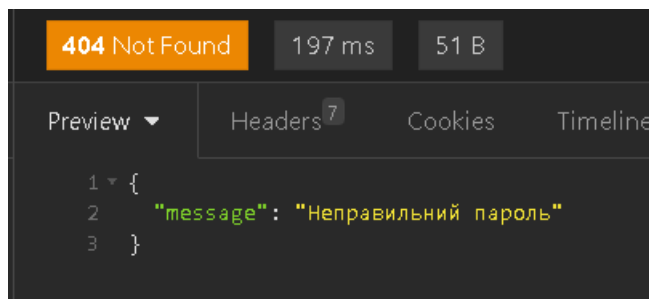
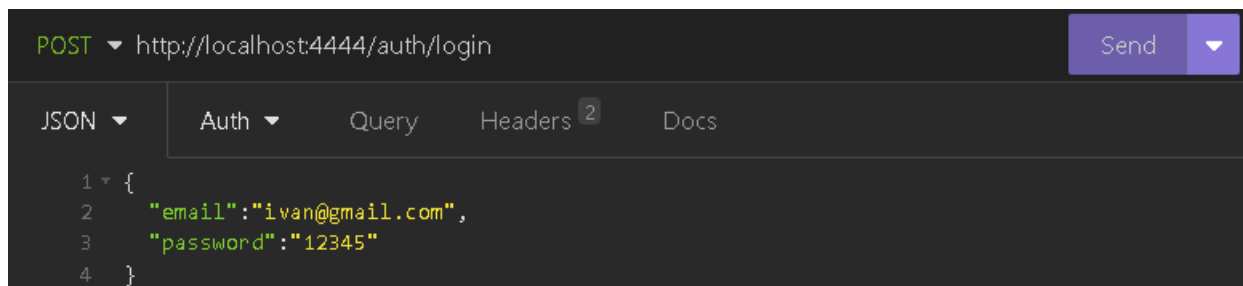
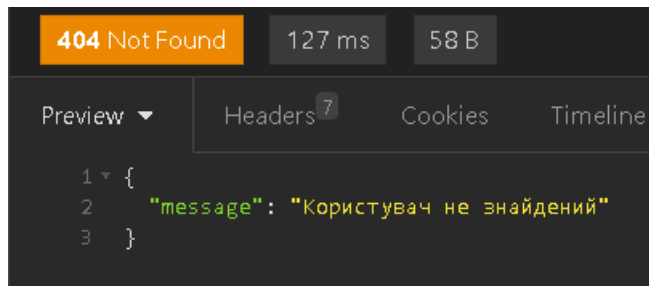
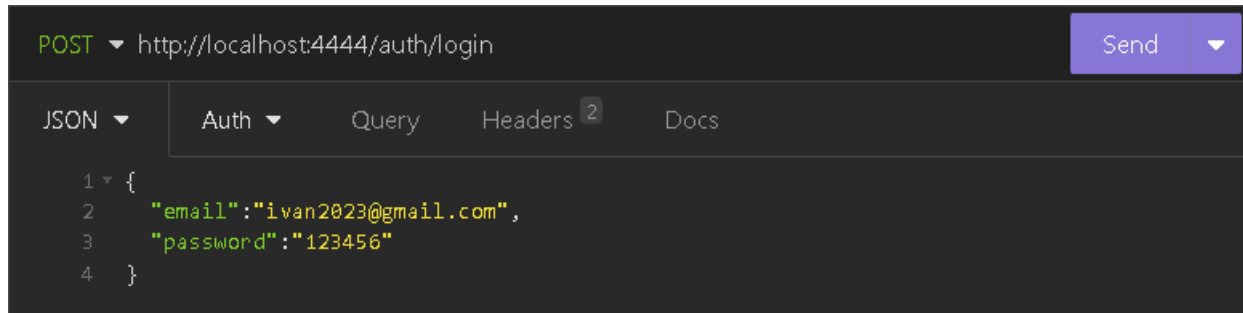


Результат: повертається код статусу 400(помилка на стороні клієнта) та JSON із валідаціями, які не пройшли перевірку.



- Спроба входу із неправильним логіном чи паролем

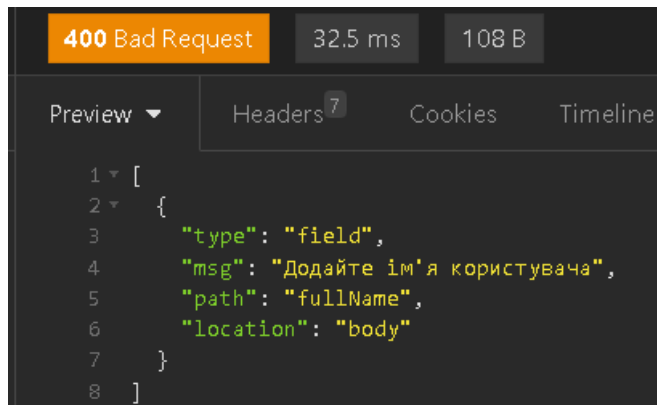
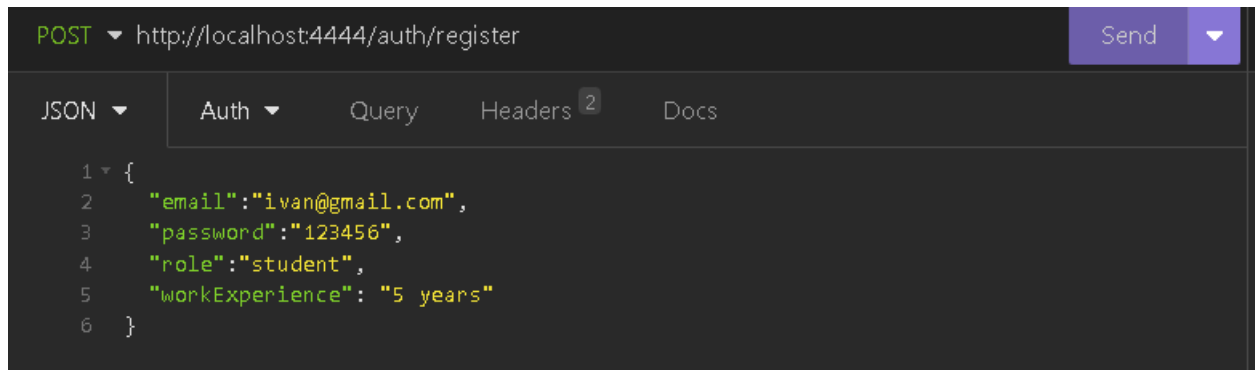
Нехай у нас є користувач Іван із логіном ivan@gmail.com та паролем 123456. Спробуємо увійти від його імені із неправильними даними:



Результати: у двох випадках повертаються коди статусу 404(not found) та JSON із відповідними повідомленнями про помилку.

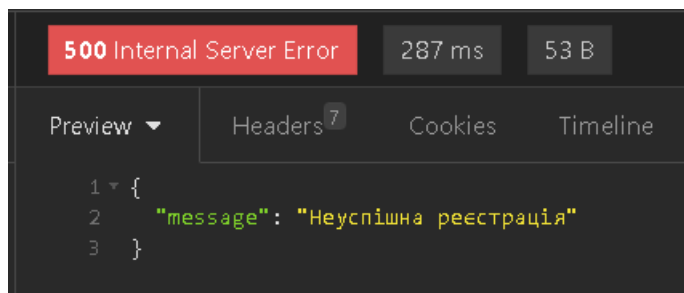
- Спроба зареєструватись без обов'язкових даних, які потрібні для реєстрації, із поштою користувача, який вже зареєстрований.

Для цього тестування ми хочемо зареєструвати певного користувача, не ввівши fullname:



В наступному тестуванні вводжу пошту користувача, яка вже використовується в БД:

При здійсненні такої операції виводиться повідомлення про помилку та код 500(помилка на стороні сервера).



Якщо заглянути у консоль, то можна побачити помилку дублюючого ключа у БД.


```

MongoServerError: E11000 duplicate key error collection: blog.users index: email_1 dup key: { email: "ivan@gmail.com" }
    at InsertOneOperation.execute (C:\Users\max_e\OneDrive\Робочий стол\Study\Coursework\node_modules\mongodb\lib\operations\insert_one.js:105:11)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
    at async executeOperationAsync (C:\Users\max_e\OneDrive\Робочий стол\Study\Coursework\node_modules\mongodb\lib\operations\insert_one.js:115:11)
    index: 0,
    code: 11000,
    keyPattern: { email: 1 },
    keyValue: { email: 'ivan@gmail.com' },
    [Symbol(errorLabels)]: Set(0) {}
  }

```

- Перевірка коректності роботи ролей користувачів. Спроба додати урок учнем.

Для цього у тіло запиту із доданням уроку передаємо токен учня:

POST http://localhost:4444/lessons Send

JSON Bearer Query Headers 2 Docs

ENABLED ☒

TOKEN eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTc3MmUwMTg4ZTQ1ZDZhMWVjNGNlIiwiaWF0IjoiMTY1MjM1MTI1In0.

PREFIX

POST http://localhost:4444/lessons Send

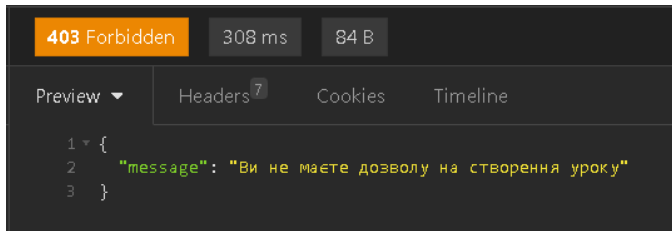
JSON Bearer Query Headers 2 Docs

```

1 {
2   "title": "Основи Першої Допомоги при Зупинці Кровотечі",
3   "text": "Існує кілька видів кровотеч, кожен з яких може вимагати специфічного підходу до зупинки. Ось кілька видів кровотеч і поради з їхньої зупинки: Артеріальна кровотеча: Характеристика: Кров іде із артерій, має яскраво-червоний колір та ритмічний пульс. Зупинка: негайно накладайте пов'язку або тиск на місце кровотечі. Використовуйте стерильний матеріал, якщо це можливо, і намагайтеся надати стискаючий тиск. Венозна кровотеча: Характеристика: Темно-червоний колір крові, струмуючою струминкою без пульсації. Зупинка: Використовуйте пов'язку або газовий тампон, намагаючись зупинити кровотечу із судин вени. Капілярна кровотеча: Характеристика: Зазвичай виникає на поверхні шкіри, кров витікає рівномірно, без пульсації. Зупинка: Застосовуйте невеликі засоби стиску, такі як биндаж чи стерильна газа, натискаючи їх на місце кровотечі. Носова кровотеча: Зупинка: Схиліть голову вперед і наперед, стискаючи область м'якого крила носа протягом 10-15 хвилин. Застосовуйте холод на задню частину шиї для вужання судин. Внутрішні кровотечі: Характеристика: Непомітні ззовні, але можуть призводити до шоку або інших симптомів. Зупинка: Викликайте екстрену медичну допомогу. Надайте підтримку потерпілому, забезпечте спокій і комфорт, не дозволяйте рухатися. У будь-якому випадку, перед зупинкою кровотечі, завжди важливо викликати медичну допомогу та намагатися уникнути контакту з кров'ю для запобігання інфекціям.",
4   "videoUrl": "https://youtu.be/112312321311211"
5 }

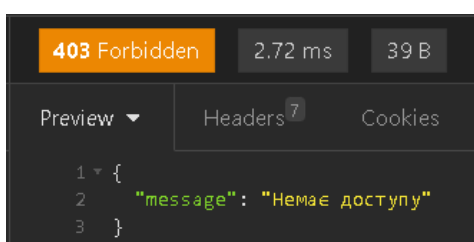
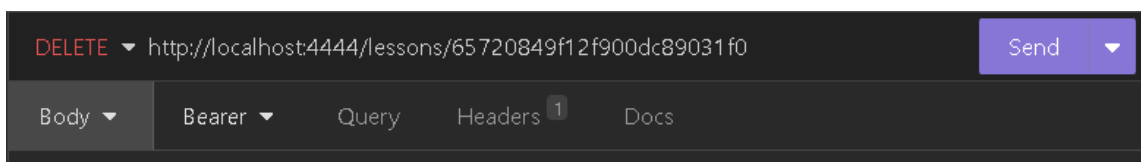
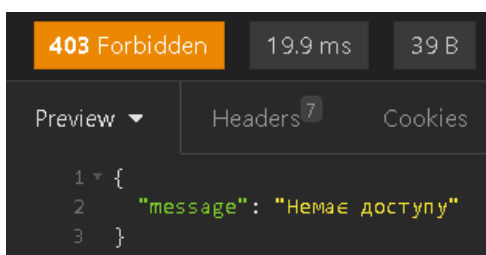
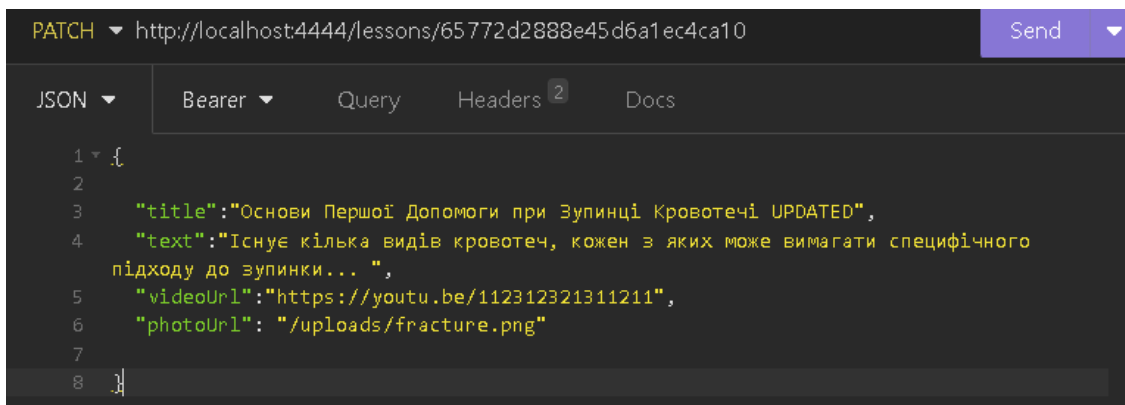
```

Результат: код статусу – 403(заборонено) та повідомлення про відсутність дозволу на здійснення даної операції.



- Намагання редагувати чи видалити урок не користувачем, який його створив.

Нехай існує урок, який створив вчитель Мах із ID 65772cd688e45d6a1ec4ca0c. Спробуємо змінити та видалити його вчителем John із ID 65772f7488e45d6a1ec4ca21.



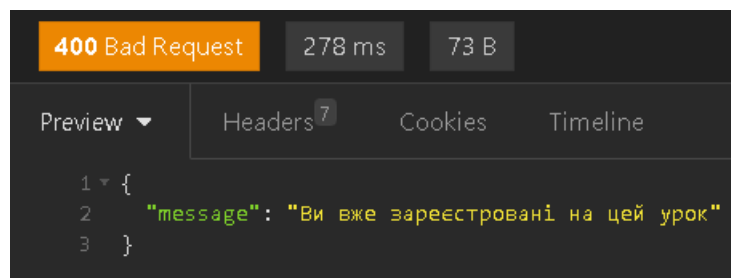
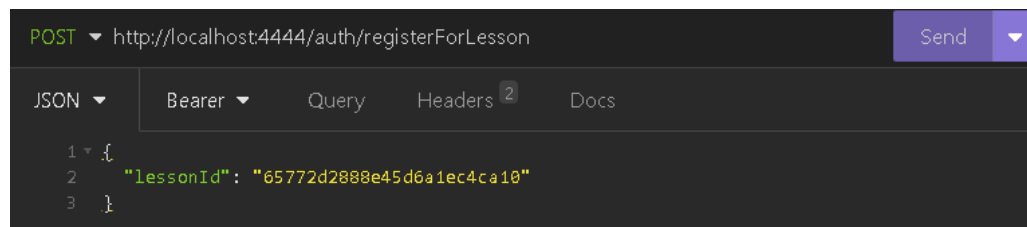
Як можна побачити у двох випадках нам повертається код статусу 403(доступ заборонений) та відповідне повідомлення.

- Спроба учнем записатись на урок, на який він уже записаний.

У нас існує студент Ivan, який записаний на урок “Основи першої допомоги при кровотечі”. ID уроку міститься у його моделі (масив lessons).

```
"lessons": [
  "65772d2888e45d6a1ec4ca10"
],
```

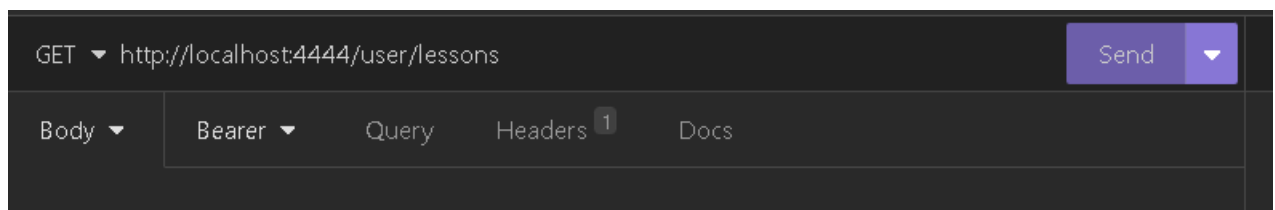
Спробуємо записатись знову на цей урок:



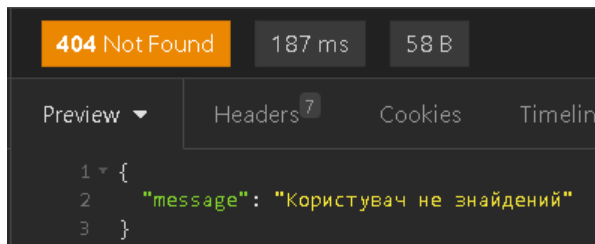
Результат: код статусу – 400(помилка на стороні клієнта) та відповідне повідомлення.

- Отримання усіх уроків, на які записаний користувач, який не існує.

Передаємо некоректний JWT користувача у тіло запита `GetUserLessons`.

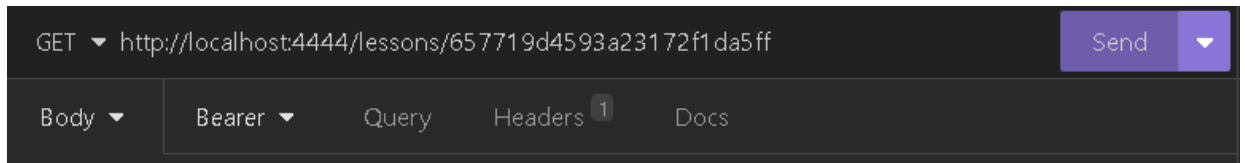


Отримуємо помилку 404(не існує) із відповідним повідомленням.

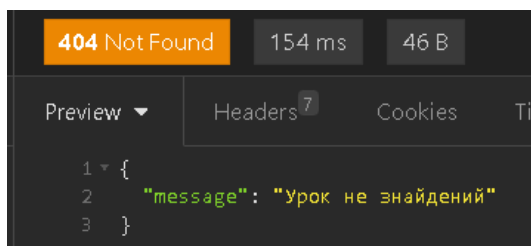


- Отримання уроку за його ID, якщо ідентифікатор некоректний.

Для цього у тілі запиту вказуємо Lesson ID, якого немає у БД.



На виході маємо помилку 404(не існує) із відповідним повідомленням.



- Тестування швидкості виконання запитів.

Виконуючи усі типи запитів можемо побачити, що швидкість виконання запиті зазвичай залежить від обсягу даних, які передаються та отримуються, інфраструктури та оптимізації сервера БД й нашого API. Більшість запитів виконуються за час не більший за 300 мілісекунд.

Реєстрація та вхід користувача:



Робота з уроками:



Виведення усіх уроків та усіх користувачів.



Останні запити займають більше часу, оскільки обробляють великий обсяг даних із MongoDB.

Отож, взаємодія нашого сервера із БД є досить швидкою та продуктивною.

4.3 Висновки, зроблені після тестувань

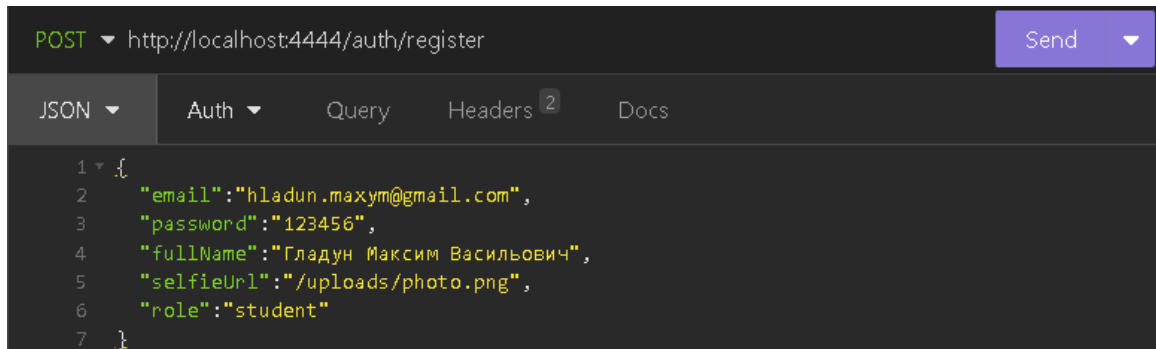
Підсумовуючи, тестування API – це один із найважливіший етапів розробки, адже він дозволяє виявити баги на ранніх етапах, заощадити час та ресурси, забезпечує безпеку сервера(захист від несанкціонованого доступу). Також сприяє уникненню некоректної роботи сервера через введення неправильних даних.

Усі запити виконуються коректно, передбачено різноманітні перевірки вхідних даних та правила доступу. Швидкість виконання запитів також оптимальна.

Розділ-5 «Демонстрація функціональних можливостей розробленого проекту»

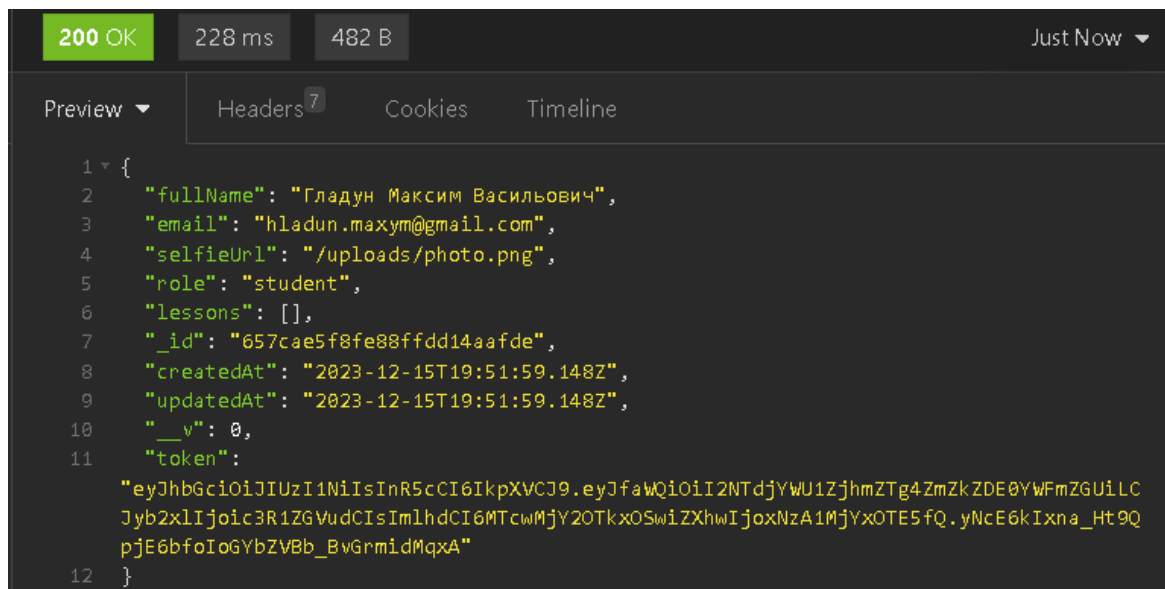
5.1 Виконання запитів, пов'язаних із користувачами

- Реєстрація користувачів:



Результати:

Повертається JSON із даними про користувача, зокрема із згенерованим JWT.



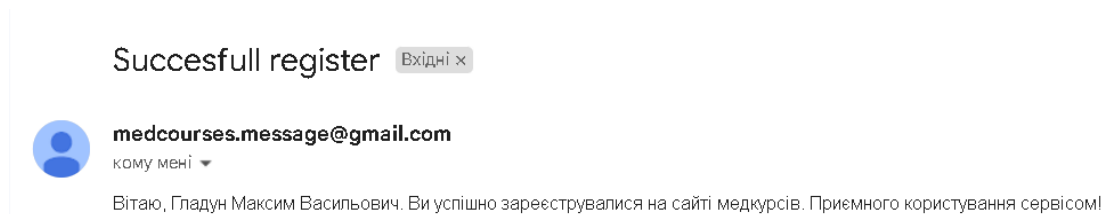
У базі даних створюється відповідний запис у таблицю із користувачами:

```

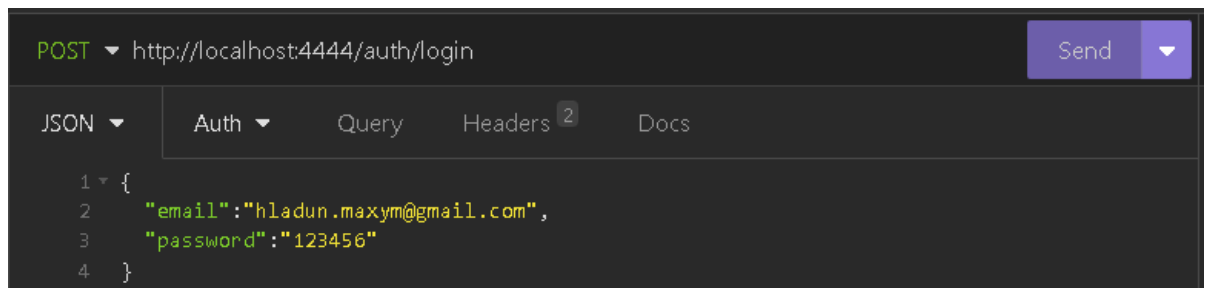
_id: ObjectId('657cae5f8fe88ffdd14aafde')
fullName: "Гладун Максим Васильович"
email: "hladun.maxym@gmail.com"
selfieUrl: "/uploads/photo.png"
role: "student"
passwordHash: "$2b$10$GKpa/vLDcrG8IL/SL8l8NulWrFYkLTGj/jMxcqeoEJWNxj5g0Fg.C"
lessons: Array (empty)
createdAt: 2023-12-15T19:51:59.148+00:00
updatedAt: 2023-12-15T19:51:59.148+00:00
__v: 0

```

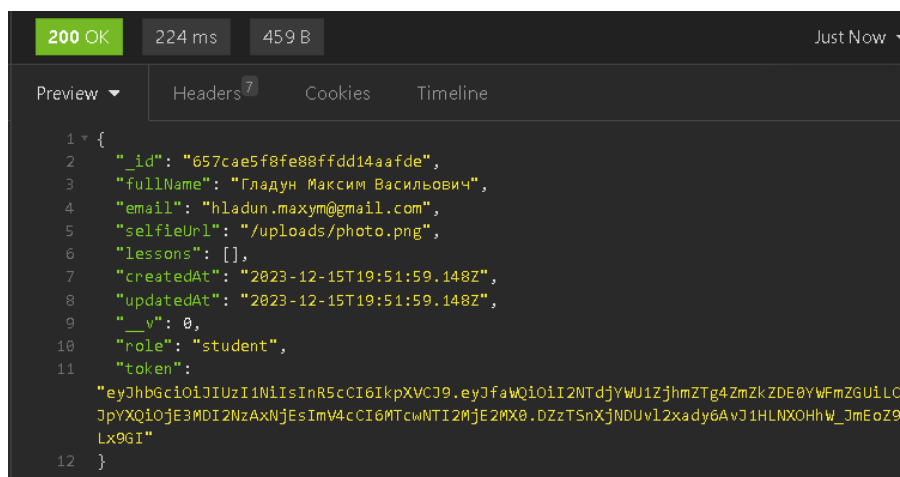
На вказаний email надсилається лист:



- Авторизація(вхід):

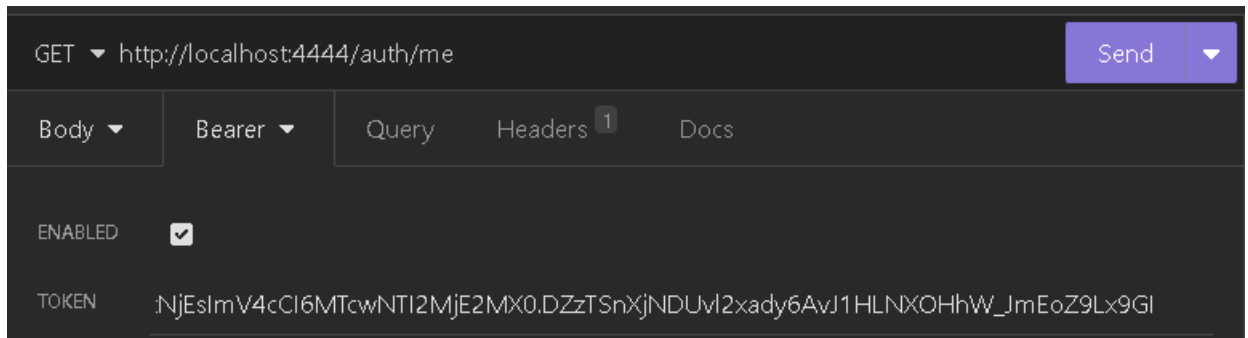


Результат: повернення інформації про користувача та генерація нового JWT.

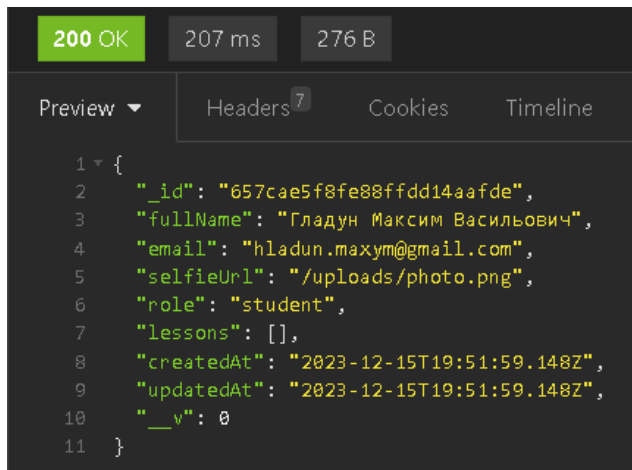


- Отримання інформація про власний профіль:

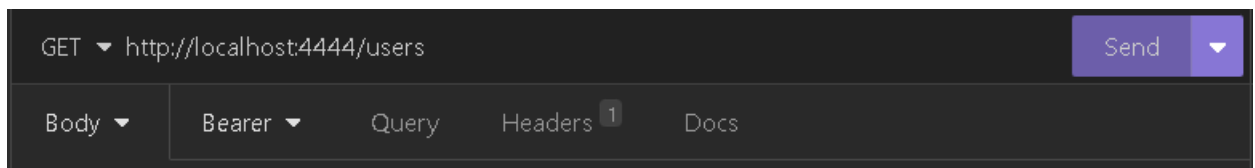
Для виконання потрібно передати JSON WEB token у тіло запиту.



Результат: відображення публічної інформації про користувача.



- Отримання інформації про усіх користувачів



Результат: відображення інформації про усіх користувачів у форматі JSON.

```

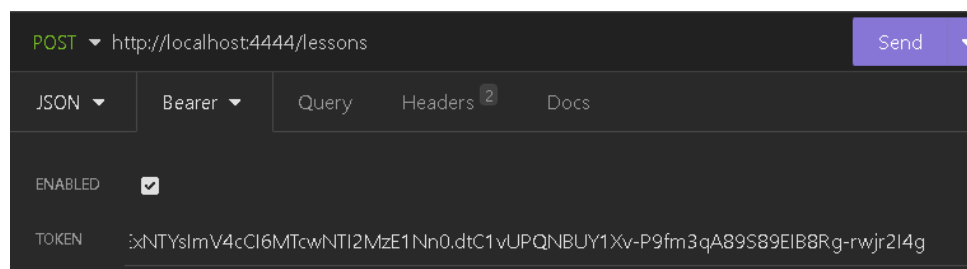
1  [
2  {
3    "_id": "657cae5f8fe88ffdd14aafde",
4    "fullName": "Гладун Максим Васильович",
5    "email": "hladun.maxym@gmail.com",
6    "selfieUrl": "/uploads/photo.png",
7    "role": "student",
8    "passwordHash":
9    "$2b$10$GKpa/vLDcrG8IL/SL8l8NulWrFYkLTGj/jMxcqeoEJWNxj5g0Fg.C",
10   "lessons": [],
11   "createdAt": "2023-12-15T19:51:59.148Z",
12   "updatedAt": "2023-12-15T19:51:59.148Z",
13   "__v": 0
14 },
15 {
16   "_id": "657cb0cb8fe88ffdd14aafe2",
17   "fullName": "Олексій Ігорович Ковальчук",
18   "email": "o.kovalchuk@gmail.com",
19   "workExperience": "15 years",
20   "selfieUrl": "/uploads/myselfie.png",
21   "role": "teacher",
22   "passwordHash":
23   "$2b$10$r7JhdFa41X8TBxPAAfbmU0JuN828qFemy9X685JzsXHJe6BrYrDAa",
24   "lessons": [],
25   "createdAt": "2023-12-15T20:02:19.038Z",
26   "updatedAt": "2023-12-15T20:02:19.038Z",
27   "__v": 0
28 },
29 {
30   "_id": "657cb0f48fe88ffdd14aafe4",
31   "fullName": "Сергій Анатолійович Мельник",
32   "email": "s.melnyk@gmail.com",
33   "workExperience": "3 years",
34   "selfieUrl": "/uploads/ph.png",
35   "role": "teacher",

```

5.2 Виконання запитів для керування уроками

- Створення уроків.

Передаю JWT лікаря Олексія Ігоровича Ковальчука.



Й додаю інформацію про урок.

```
POST http://localhost4444/lessons

JSON  Bearer  Query  Headers 2  Docs

1 {
2   "title": "Основи Першої Допомоги при Зупинці Кровотечі",
3   "text": "Існує кілька видів кровотеч, кожен з яких може вимагати специфічного
підходу до зупинки. Ось кілька видів кровотеч і поради з їхньої зупинки: Артеріальна
кровотеча: Характеристика: Кров іде із артерій, має яскраво-червоний колір та
ритмічний пульс. Зупинка: негайно накладайте пов'язку або тиск на місце кровотечі.
Використовуйте стерильний матеріал, якщо це можливо, і намагайтеся надати стискаючий
тиск. Венозна кровотеча: Характеристика: Темно-червоний колір крові, струмуючою
струминкою без пульсації. Зупинка: Використовуйте пов'язку або газовий тампон,
намагаючись зупинити кровотечу із судин вени. Капілярна кровотеча: Характеристика:
Зазвичай виникає на поверхні шкіри, кров витікає рівномірно, без пульсації. Зупинка:
Застосовуйте невеликі засоби тиску, такі як биндаж чи стерильна газа, натискуючи їх
на місце кровотечі. Носова кровотеча: Зупинка: Схиліть голову вперед і наперед,
стискаючи область м'якого крила носа протягом 10-15 хвилин. Застосовуйте холод на
задню частину шиї для вужання судин. Внутрішні кровотечі: Характеристика: Непомітні
ззовні, але можуть призводити до шоку або інших симптомів. Зупинка: Викликайте
екстрену медичну допомогу. Надajte підтримку потерпілому, забезпечте спокій і
комфорт, не дозволяйте рухатися. У будь-якому випадку, перед зупинкою кровотечі,
завжди важливо викликати медичну допомогу та намагатися уникнути контакту з кров'ю
для запобігання інфекціям. ",
4   "videoUrl": "https://youtu.be/112312321311211",
5   "photoUrl": "https://oppb.com.ua/wp-
content/uploads/2022/01/shutterstock_1894762378.jpg"
6 }
```

Результат: повернення усієї інформації про урок у форматі JSON, дати створення та оновлення й id уроку та користувача, який його створив.

```
200 OK  430 ms  2.9 KB  Just Now

Preview  Headers 7  Cookies  Timeline

1 {
2   "title": "Основи Першої Допомоги при Зупинці Кровотечі",
3   "text": "Існує кілька видів кровотеч, кожен з яких може вимагати специфічного
підходу до зупинки. Ось кілька видів кровотеч і поради з їхньої зупинки: Артеріальна
кровотеча: Характеристика: Кров іде із артерій, має яскраво-червоний колір та
ритмічний пульс. Зупинка: негайно накладайте пов'язку або тиск на місце кровотечі.
Використовуйте стерильний матеріал, якщо це можливо, і намагайтеся надати стискаючий
тиск. Венозна кровотеча: Характеристика: Темно-червоний колір крові, струмуючою
струминкою без пульсації. Зупинка: Використовуйте пов'язку або газовий тампон,
намагаючись зупинити кровотечу із судин вени. Капілярна кровотеча: Характеристика:
Зазвичай виникає на поверхні шкіри, кров витікає рівномірно, без пульсації. Зупинка:
Застосовуйте невеликі засоби тиску, такі як биндаж чи стерильна газа, натискуючи їх
на місце кровотечі. Носова кровотеча: Зупинка: Схиліть голову вперед і наперед,
стискаючи область м'якого крила носа протягом 10-15 хвилин. Застосовуйте холод на
задню частину шиї для вужання судин. Внутрішні кровотечі: Характеристика: Непомітні
ззовні, але можуть призводити до шоку або інших симптомів. Зупинка: Викликайте
екстрену медичну допомогу. Надajte підтримку потерпілому, забезпечте спокій і
комфорт, не дозволяйте рухатися. У будь-якому випадку, перед зупинкою кровотечі,
завжди важливо викликати медичну допомогу та намагатися уникнути контакту з кров'ю
для запобігання інфекціям. ",
```

```
4   "videoUrl": "https://youtu.be/112312321311211",
5   "photoUrl": "https://oppb.com.ua/wp-
content/uploads/2022/01/shutterstock_1894762378.jpg",
6   "user": "657cb0cb8fe88ffdd14aafe2",
7   "id": "657cb3c68fe88ffdd14aafeb",
8   "createdAt": "2023-12-15T20:15:02.524Z",
9   "updatedAt": "2023-12-15T20:15:02.524Z",
10  "_v": 0
11 }
```

У MongoDB створюється відповідний запис:

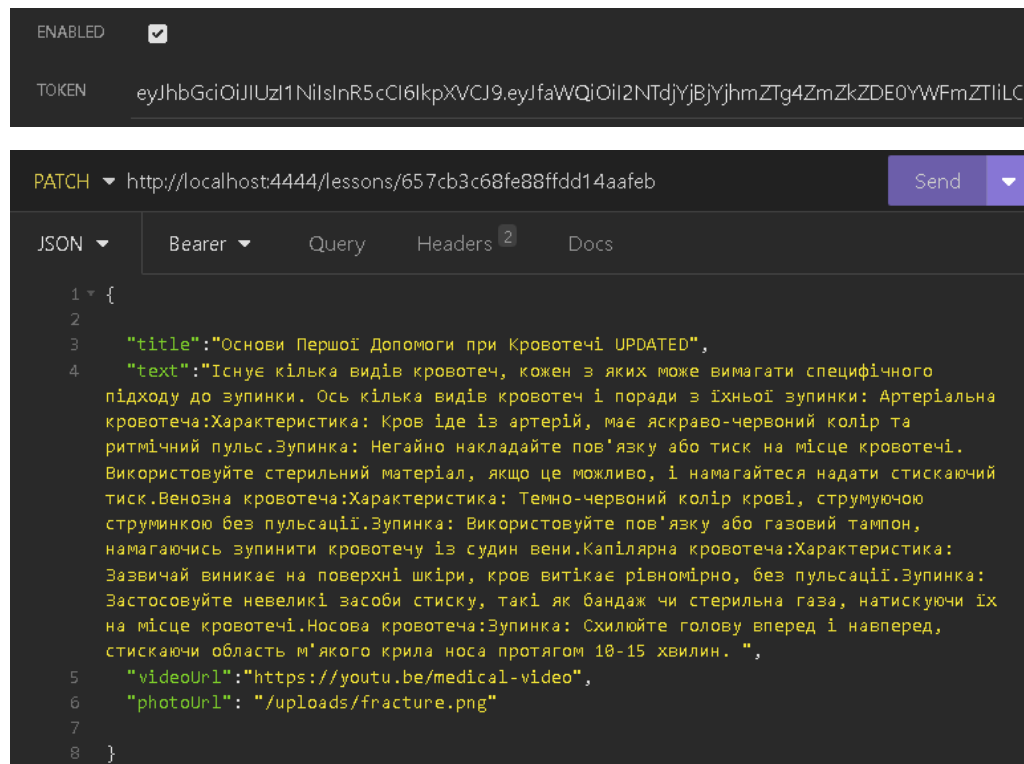
```

_id: ObjectId('657cb3c68fe88ffdd14aafeb')
title: "Основи Першої Допомоги при Зупинці Кровотечі"
text: "Існує кілька видів кровотеч, кожен з яких може вимагати специфічного п..."
videoUrl: "https://youtu.be/112312321311211"
photoUrl: "https://oppb.com.ua/wp-content/uploads/2022/01/shutterstock_1894762378..."
user: ObjectId('657cb0cb8fe88ffdd14aafe2')
createdAt: 2023-12-15T20:15:02.524+00:00
updatedAt: 2023-12-15T20:15:02.524+00:00
__v: 0

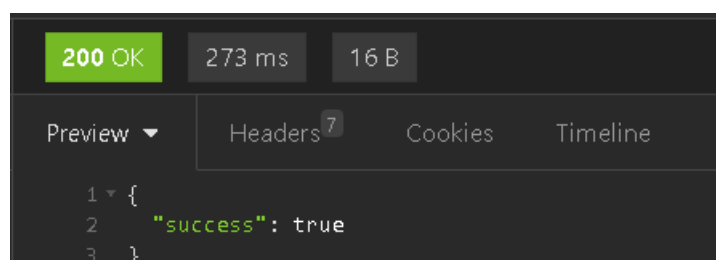
```

- Редагування уроків.

Змінюю заголовок, текст та посилання на відео уроку. Для цього передаю JWT викладача, який створив його.



Повертається результат із повідомленням про успіх у форматі JSON.

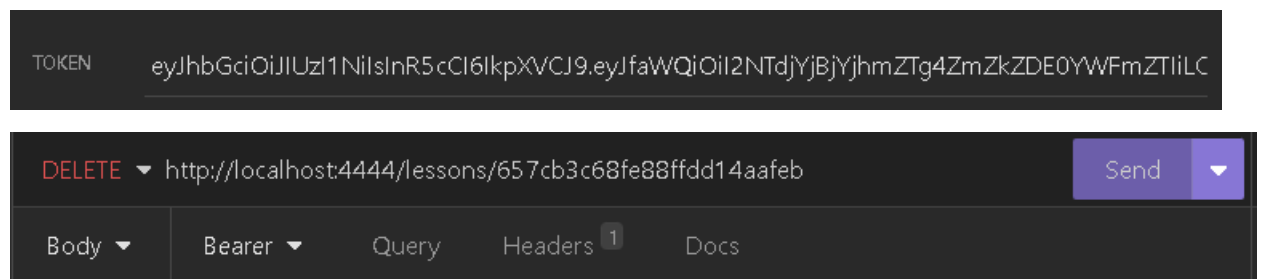


У MongoDB оновлюються поля даного уроку.

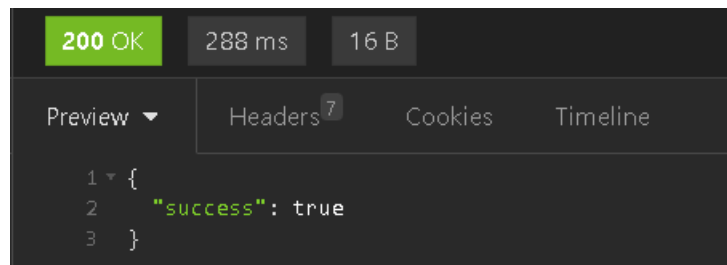
```
_id: ObjectId('657cb3c68fe88ffdd14aafefb')
title: "Основи Першої Допомоги при Кровотечі UPDATED"
text: "Існує кілька видів кровотеч, кожен з яких може вимагати специфічного п..."
videoUrl: "https://youtu.be/medical-video"
photoUrl: "/uploads/fracture.png"
user: ObjectId('657cb0cb8fe88ffdd14aafef2')
createdAt: 2023-12-15T20:15:02.524+00:00
updatedAt: 2023-12-15T20:23:13.028+00:00
__v: 0
```

- Видалення уроків.

Аналогічно до попередніх запитів передаю у заголовок запит токен вчителя, який створив даний урок та id уроку яких хочемо видалити .



У результаті повертається код статусу 200 та повідомлення про успіх.



Із бази даних зникає урок.

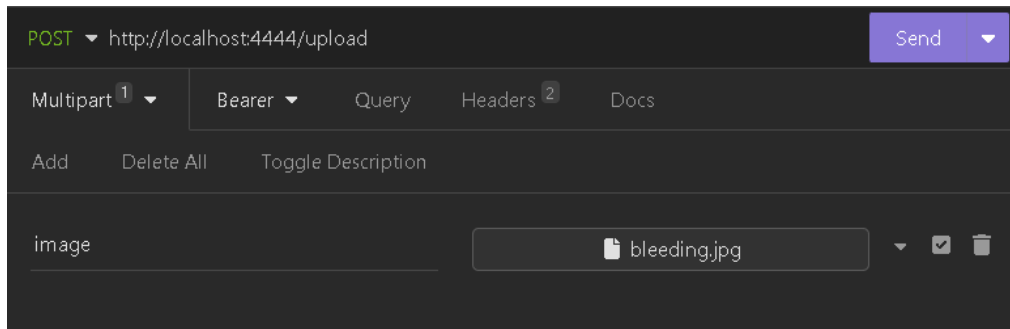


This collection has no data

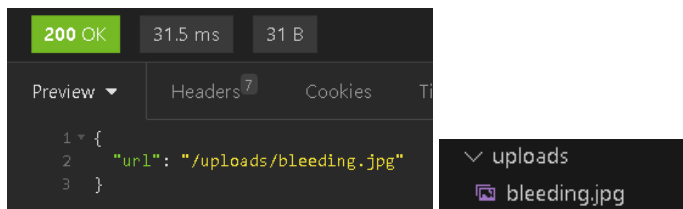
It only takes a few seconds to import data from a JSON or CSV file.

- Завантаження файлів на сервер.

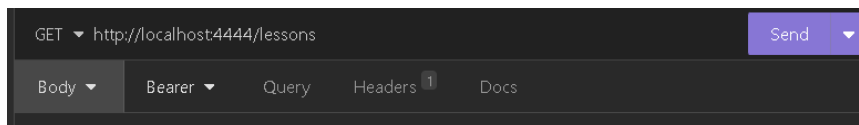
У тілі запиту передаю файл із локального сховища.



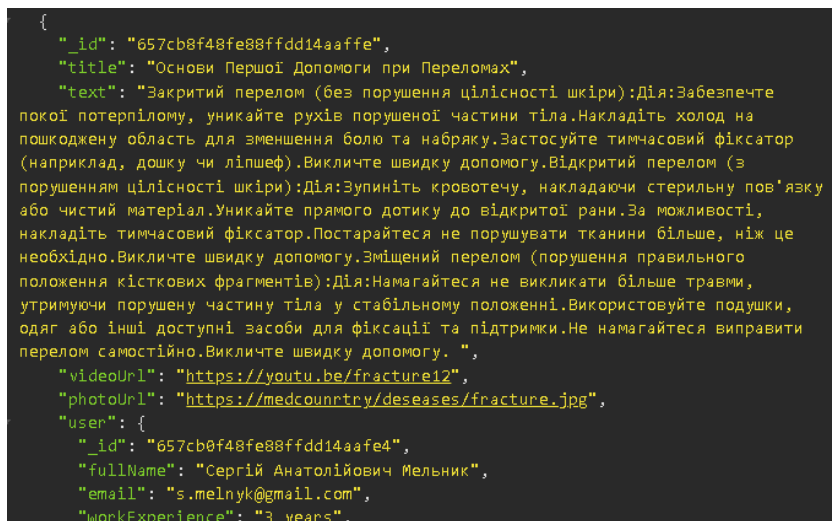
У результаті нам повертається URL цього зображення та воно з'являється у папці uploads нашого проекту.



- Отримання усіх уроків.



У результаті повертається інформація про усі уроки та користувачів які їх створили.



```

200 OK 254 ms 5.7 KB Just Now
Preview Headers Cookies Timeline
1 [
2 {
3   "_id": "657cb8158fe88ffdd14aaff3",
4   "title": "Основи Першої Допомоги при Зупинці Кровотечі",
5   "text": "Існує кілька видів кровотеч, кожен з яких може вимагати специфічного
        підходу до зупинки. Ось кілька видів кровотеч і поради з їхньої зупинки:
        Артеріальна кровотеча:Характеристика: Кров іде із артерій, має яскраво-червоний
        колір та ритмічний пульс.Зупинка: Негайно накладайте пов'язку або тиск на місце
        кровотечі. Використовуйте стерильний матеріал, якщо це можливо, і намагайтеся
        надати стискаючий тиск.Венозна кровотеча:Характеристика: Темно-червоний колір
        крові, струмуючою струминкою без пульсації.Зупинка: Використовуйте пов'язку або
        газовий тампон, намагаючись зупинити кровотечу із судин вени.Капілярна
        кровотеча:Характеристика: Зазвичай виникає на поверхні шкіри, кров витікає
        рівномірно, без пульсації.Зупинка: Застосовуйте невеликі засоби стиску, такі як
        биндаж чи стерильна газа, натискаючи їх на місце кровотечі.Носова
        кровотеча:Зупинка: Схиліть голову вперед і наперед, стискаючи область м'якого
        крила носа протягом 10-15 хвилин. Застосовуйте холод на задню частину шиї для
        вужання судин.Внутрішні кровотечі:Характеристика: Непомітні ззовні, але можуть
        призводити до шоку або інших симптомів.Зупинка: Викликайте екстрену медичну
        допомогу. Надайте підтримку потерпілому, забезпечте спокій і комфорт, не
        дозволяйте рухатися.У будь-якому випадку, перед зупинкою кровотечі, завжди важливо
        викликати медичну допомогу та намагаються уникнути контакту з кров'ю для
        запобігання інфекціям. ",
6   "videoUrl": "https://youtu.be/112312321311211",
7   "photoUrl": "https://opppb.com.ua/wp-
        content/uploads/2022/01/shutterstock_1894762378.jpg",
8   "user": {
9     "id": "657cb8158fe88ffdd14aaffe2",
10    "fullName": "Олексі́й Іро́рович Ковальчу́к",
11    "email": "o.kovalchuk@gmail.com",
12    "workExperience": "15 years",
13    "selfieUrl": "/uploads/myselfie.png",
14    "role": "teacher",

```

- Отримання уроку по його id.

Для цього у тіло запиту передаю id уроку.

```

GET http://localhost:4444/lessons/657cb8f48fe88ffdd14aaffe
Send
Body Bearer Query Headers Docs

```

Інформація аналогічного формату нам повертається, але вже тільки про один урок.

```

200 OK 137 ms 1997 B 2 Minutes Ago
Preview Headers 7 Cookies Timeline
1 {
2   "_id": "657cb8f48fe88ffdd14aaaffe",
3   "title": "Основи Першої Допомоги при Переломах",
4   "text": "Закритий перелом (без порушення цілісності шкіри):Дія:Забезпечте покої потерпілому, уникайте рухів порушеної частини тіла.Накладіть холод на пошкоджену область для зменшення болю та набряку.Застосуйте тимчасовий фіксатор (наприклад, дошку чи ліпшеф).Викличте швидку допомогу.Відкритий перелом (з порушенням цілісності шкіри):Дія:Зупиніть кровотечу, накладаючи стерильну пов'язку або чистий матеріал.Уникайте прямого дотику до відкритої рани.За можливості, накладіть тимчасовий фіксатор.Постарайтеся не порушувати тканини більше, ніж це необхідно.Викличте швидку допомогу.Зміщений перелом (порушення правильного положення кісткових фрагментів):Дія:Намагайтеся не викликати більше травми, утримуючи порушену частину тіла у стабільному положенні.Використовуйте подушки, одяг або інші доступні засоби для фіксації та підтримки.Не намагайтеся виправити перелом самостійно.Викличте швидку допомогу. ",
5   "videoUrl": "https://youtu.be/fracture12",
6   "photoUrl": "https://medcounntry/deseases/fracture.jpg",
7   "user": "657cb8f48fe88ffdd14aaafe4",
8   "createdAt": "2023-12-15T20:37:08.368Z",
9   "updatedAt": "2023-12-15T20:37:08.368Z",
10  "__v": 0
11 }

```

- Запис користувача на урок.

Передаю JWT для ідентифікація користувача й id уроку, на який потрібно записатись у тілі запиту.

```

JSON Bearer Query Headers 2 Docs
ENABLED ☒
TOKEN eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTdJYWU1ZjhmZTg4ZmZkZDE0YWVmZG

```

```

POST http://localhost:4444/auth/registerForLesson Send
JSON Bearer Query Headers 2 Docs
1 {
2   "lessonId": "65772d2888e45d6a1ec4ca10"
3 }

```

У результаті отримуємо JSON із інформацією про успіх.

```

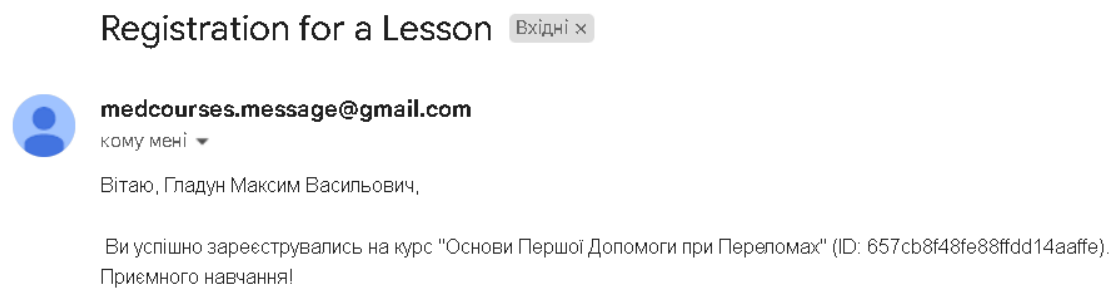
200 OK 388 ms 16 B
Preview Headers 7 Cookies Timeline
1 {
2   "success": true
3 }

```

У MongoDB у таблиці користувачів з'являється запис у масиві lessons із id уроку.

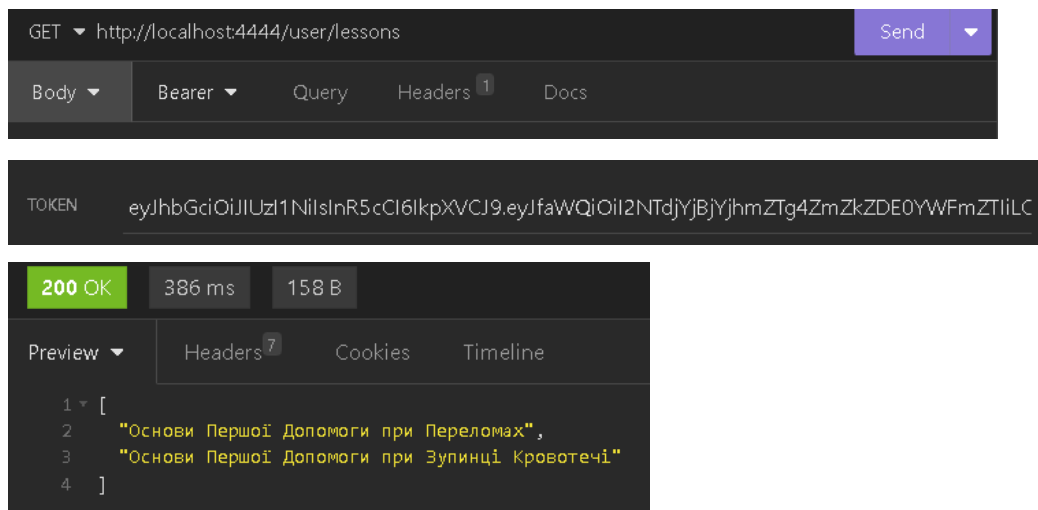
```
▼ lessons: Array (1)
  0: ObjectId('657cb8f48fe88ffdd14aaffe')
```

Користувачу на електронну пошту приходить лист про успішний запис.



- Відображення усіх уроків, на які записаний певний користувач.

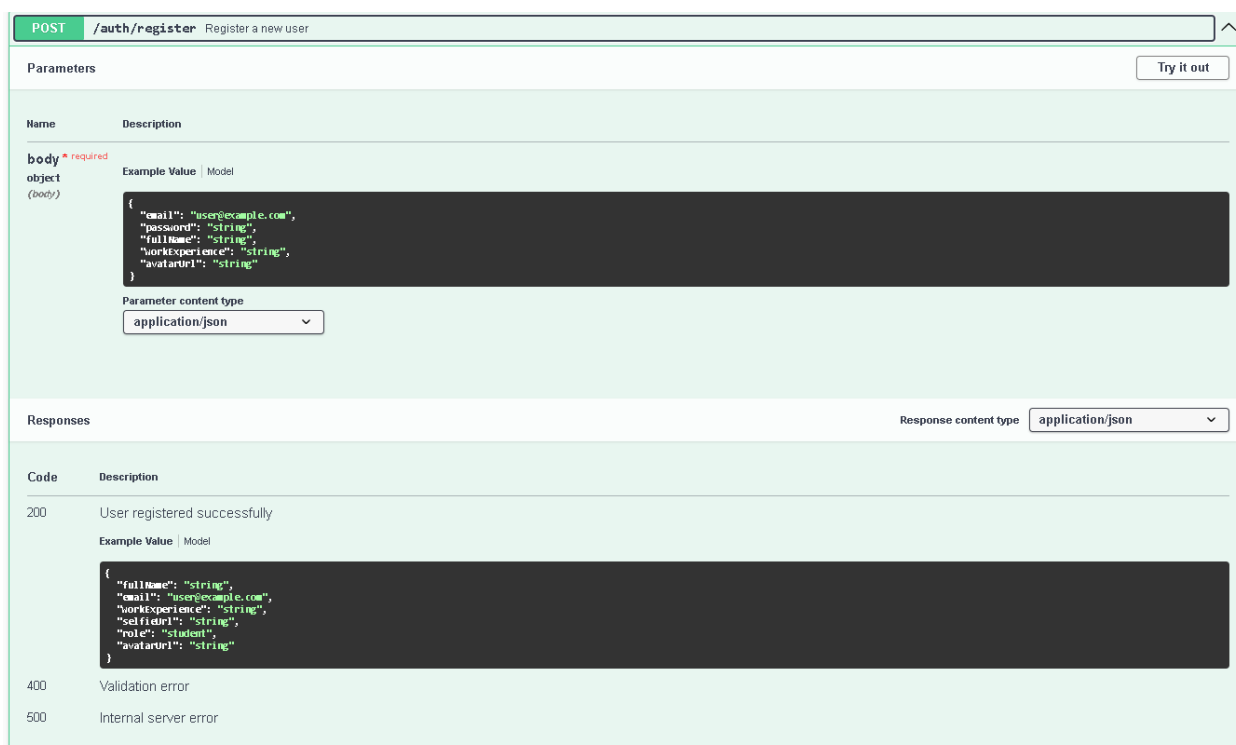
Передавши токен користувача можна отримати усі уроки, на які він записаний у форматі масиву.



5.3 Перегляд документації через SwaggerUi

Підключивши SwaggerUi та SwaggerDocument та перейшовши за URL `localhost:4444/api-docs` ми можемо переглядати документацію API, а саме усі запити, погруповані по темах, поля, які потрібно вказувати при виконання відповідних запитів та коди помилок, які можуть виникати.

Наприклад, документація для запиту реєстрації користувача:



POST /auth/register Register a new user

Parameters

Try it out

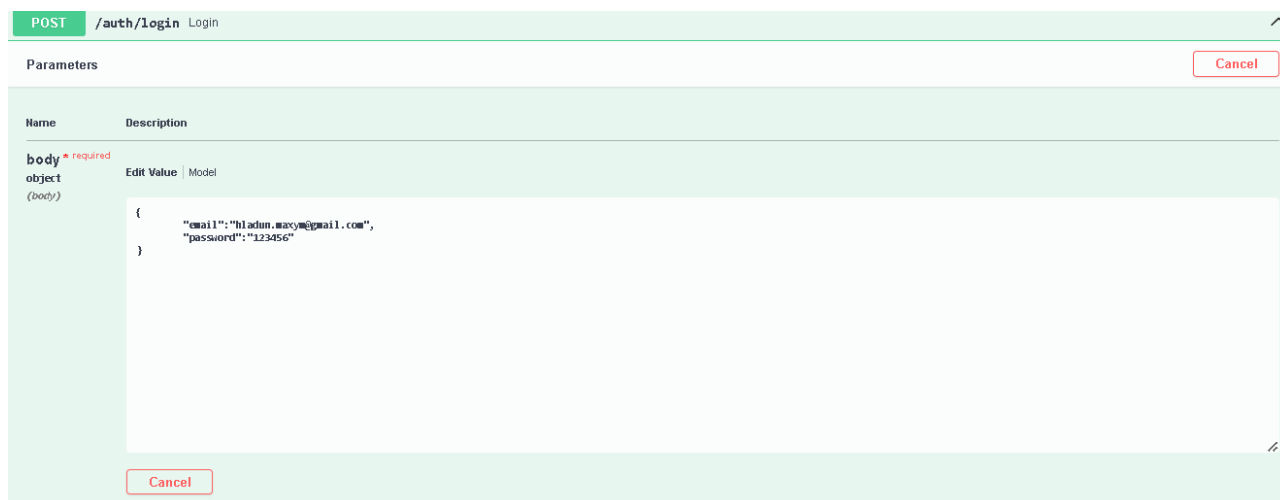
Name	Description
body * required object (body)	Example Value Model <pre>{ "email": "user@example.com", "password": "string", "fullName": "string", "workExperience": "string", "avatarUrl": "string" }</pre> Parameter content type application/json

Responses

Response content type: application/json

Code	Description
200	User registered successfully Example Value Model <pre>{ "fullName": "string", "email": "user@example.com", "workExperience": "string", "selfId": "string", "role": "Student", "avatarUrl": "string" }</pre>
400	Validation error
500	Internal server error

Також через цей інтерфейс можна виконувати відповідні запити та отримувати відповідь сервера:



POST /auth/login Login

Parameters

Cancel

Name	Description
body * required object (body)	Edit Value Model <pre>{ "email": "hladan.maxy@gmail.com", "password": "123456" }</pre>

Cancel

Для зручності реєстрації, входу та додавання уроків створив веб-сторінки для виконання цих post-запитів.

← → ↺ 127.0.0.1:5500/RegisterForm.html ☆ 🛡️ 🇺🇦 📄 🗑️ 📝 📱 🌐

Medical courses

Регистрація

Електронна пошта:

Повне ім'я:

Досвід роботи:

Роль:

Пароль:

[Зареєструватися](#) [Ви вже маєте акаунт? Увійти](#)

127.0.0.1/LoginForm.html

Medical courses

Вхід

Електронна пошта:

hiadun.maxym@gmail.com

Пароль:

Увійти

Не маєте акаунта?
Зареєструйтесь

Medical courses

Додавання уроку

JWT токен:

Заголовок:

Текст уроку:

Посилання на відео:

Посилання на фото:

Додати урок

При виконанні відповідних запитів із браузера нам будуть повертатись відповіді у вигляді JSON.

```
{ "_id": "657cae5f8fe88ffdd14aafde", "fullName": "Гладун Максим Васильович", "email": "hladun.makym@gmail.com", "selfieUrl": "/uploads/photo.png", "lessons": [{"_id": "657cb8f48fe88ffdd14aaffe", "657cb8158fe88ffdd14aaff3"}, {"createdAt": "2023-12-15T19:51:59.148Z", "updatedAt": "2023-12-15T20:53:33.356Z", "v": 2, "role": "student", "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTdjYU11ZjhmZTg4ZmZkZDE0YWFmZGUlCjprYXQiOiJlZ3M0I2NzQzNTUsImV4cCI6I6MTCwNTI2NjM1NX0.zq-MH1SEKwW4YwIL-jkZqrAKxQKrE7wDVFRArOA89VYk"}]}
```

```
{ "fullName": "Петренко Іван Миколайович", "email": "ivan.petrenko@gmail.com", "workExperience": "12 років", "role": "teacher", "lessons": [{"_id": "657cc00a8fe88ffdd14ab015", "createdAt": "2023-12-15T21:07:22.171Z", "updatedAt": "2023-12-15T21:07:22.171Z", "v": 0, "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2NTdjYU11ZjhmZTg4ZmZkZDE0YWFmZGUlCjprYXQiOiJlZ3M0I2NzQzNTUsImV4cCI6I6MTCwNTI2NjM1NX0.zq-MH1SEKwW4YwIL-jkZqrAKxQKrE7wDVFRArOA89VYk"}]}
```

Висновки

6.1 Основні результати проекту

У ході цієї курсової роботи було створено повноцінний веб-сервер для навчання, за допомогою якого люди без медичного досвіду можуть отримати необхідні знання та навички для надання невідкладної домедичної допомоги, адже їх наявність може зберегти сотні й тисячі життів.

Успішно реалізовано:

- ✓ JWT (JSON Web Token) для аутентифікації.
- ✓ Систему ролей (викладач, студент).
- ✓ Контроль доступу через middleware
- ✓ MongoDB як базу даних.
- ✓ Моделі User та Lesson для зберігання інформації про користувачів та уроки відповідно.
- ✓ Експрес-валідатори для перевірки коректності введених даних.
- ✓ Middleware `handlevalidationerrors.js` для обробки помилок валідації.
- ✓ Відправлення електронних листів користувачам.
- ✓ Завантаження фотографій для уроків та користувацьких аватарів.
- ✓ Документацію API через Swagger.
- ✓ CRUD-операції для уроків (створення, читання, оновлення, видалення).
- ✓ Зв'язок уроків з користувачами, можливість запису на уроки.
- ✓ Розділення проекту на модулі, для полегшення його розширення та обслуговування.

Отож, після виконання цього проекту отримав знання та навички користування із технологіями Node.js, Express.js, MongoDB та фреймворками, які використовував для додавання функціоналу. Навчився створювати документацію, необхідну для подальшого розширення проекту, надання додаткової інформації користувачам та розробникам, які будуть взаємодіяти із цим API.

6.2 Шляхи покращення

- Додання функціоналу для відгуків учнів, які пройшли курс та системи обговорення.
- Впровадження системи рейтингів для викладачів, яка відображатиме кількість учнів, їхні оцінки та активність.
- Відображення розкладу занять для людей, які проходять курс.
- Реалізація функціоналу для створення та виконання онлайн-тестів з автоматичною перевіркою.
- Розробка frontend-частини.
- Розгортання проекту на хостингах.

6.3 Перспективи використання

Веб-сервер має значний потенціал та перспективи використання серед різних груп користувачів та у галузях, таких як :

- медична(обмін досвідом);
- середня освіта (школярі та студенти);
- вища освіта (студенти та викладачі);
- корпоративна (навчання працівників компаній);
- батьки та діти (надання базових знань батькам про фізіологію та захворювання, типові для дітей).

Загалом, кожній людині, яка дбає про своє здоров'я буде корисна інформація, отримана із курсів, розміщених у цьому веб-застосунку.

Список використаних джерел

1. David Herron Node.JS Web Development: 2016. с.44-63, 112-114, 130-139.
2. Загальна інформація про MongoDB URL:
<https://www.ukraine.com.ua/uk/wiki/mongodb/overview/> (дата звернення: 03.11.2023).
3. Документація ПЗ Insomnia URL: <https://docs.insomnia.rest/insomnia/get-started> (дата звернення: 03.11.2023).
4. Express web framework (Node.js/JavaScript) URL:
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs
(дата звернення: 03.11.2023).
5. Node JS file upload URL: <https://docs.nestjs.com/techniques/file-upload> (дата звернення: 07.11.2023).
6. Node JS framework nodemailer URL: <https://nodemailer.com/> (дата звернення: 21.11.2023).
7. Swagger API documentation URL: <https://swagger.io/solutions/api-documentation/> (дата звернення: 30.11.2023).

Додатки

- **Код програми:**

Index.js:

```
import express from 'express';
import mongoose from 'mongoose';
import bodyParser from 'body-parser';
import { registerValidation, loginValidation, lessonCreateValidation } from './validations.js';
import * as UserController from './Controllers/UserController.js';
import * as LessonController from './Controllers/LessonController.js';
import checkAuth from './utils/checkAuth.js';
import multer from 'multer';
import handleValidationErrors from './utils/handleValidationErrors.js';
import swaggerUi from 'swagger-ui-express';
import swaggerDocument from './swagger.json' assert { type: 'json' };

mongoose
  .connect(
    'mongodb+srv://miksimmiks:wqNIvgdk4tYkP5wm@cluster0.018qrom.mongodb.net/blog?retryWrites=true&w=majority'
  )
  .then(() => console.log('DB OK'))
  .catch((err) => console.log('DB error', err));

const app = express();
const storage = multer.diskStorage({
  destination: (_, __, cb) => {
    cb(null, 'uploads');
  },
  filename: (_, file, cb) => {
    cb(null, file.originalname);
  },
});

var urlencodedParser = bodyParser.urlencoded({ extended: false });
const upload = multer({ storage });
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
app.use('/uploads', express.static('uploads'));
```

```

app.use(express.json());
app.use(bodyParser.raw({ type: 'text/plain' })); // body parser
app.post('/auth/login', urlencodedParser, loginValidation, handleValidationErrors, UserController.login );
app.post('/auth/register', urlencodedParser, registerValidation, handleValidationErrors,
UserController.register );
app.get('/auth/me', checkAuth, UserController.getMe);
app.get('/users', UserController.getAllUsers);
app.post('/upload', checkAuth, upload.single('image'), (req, res) => {
  res.json({
    url: `uploads/${req.file.originalname}`,
  });
});
app.get('/lessons', LessonController.getAll);
app.get('/lessons/:id', LessonController.getOne);
app.post('/lessons', checkAuth, lessonCreateValidation, handleValidationErrors, LessonController.create);
app.delete('/lessons/:id', checkAuth, LessonController.remove);
app.patch('/lessons/:id', checkAuth, lessonCreateValidation, handleValidationErrors,
LessonController.update);
app.post('/auth/registerForLesson', checkAuth, LessonController.registerForLesson);
app.get('/user/lessons', checkAuth, LessonController.getUserLessons);
app.listen(4444, (err) => {
  if(err) {
    return console.log(err);
  }
  console.log('Server OK');
});

```

Lesson.js:

```

import mongoose from 'mongoose';
const LessonSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  text: {

```



```

    type: String,
    required: true,
    unique: true,
  },
  videoUrl: {
    type: String,
  },
  photoUrl: {
    type: String,
  },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
  },
},
{
  timestamps: true,
},
);
export default mongoose.model('Lesson', LessonSchema);

```

User.js:

```

import mongoose from 'mongoose';
const UserSchema = new mongoose.Schema({
  fullName: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },

```

```

workExperience:{
  type: String,
},
selfieUrl:{
  type: String,
},
role: {
  type: String,
  enum: ['student', 'teacher'],
  default: 'student',
  required: true,
},
passwordHash: {
  type: String,
  required: true,
},
lessons: [{
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Lesson',
}],
avatarUrl: String,
},
{
  timestamps: true,
},
);
export default mongoose.model('User', UserSchema);

```

lessonController.js:

```

import LessonModel from '../models/lesson.js';
import UserModel from '../models/user.js';
import jwt from 'jsonwebtoken';
import nodemailer from 'nodemailer';
export const getAll = async (req, res) => {

```

```

try{
  const lessons = await LessonModel.find().populate('user').exec();
  res.json(lessons);
} catch(err){
  console.log(err);
  res.status(500).json({
    message: 'Не вдалось получить уроки',
  });
}
};

export const getOne = async (req, res) => {
  try {
    const lessonId = req.params.id;
    const lesson = await LessonModel.findById(lessonId);
    if (!lesson) {
      return res.status(404).json({
        message: 'Урок не найден',
      });
    }
    res.json(lesson);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалось получить урок',
    });
  }
};

export const create = async (req, res) => {
  try {
    const userId = req.userId;
    const user = await UserModel.findById(userId);
    const userRole = user ? user.role : null;

    if (userRole !== 'teacher') {

```

```

    return res.status(403).json({
      message: 'Ви не маєте дозволу на створення уроку',
    });
  }

  const doc = new LessonModel({
    title: req.body.title,
    text: req.body.text,
    imageUrl: req.body.imageUrl,
    videoUrl: req.body.videoUrl,
    photoUrl: req.body.photoUrl,
    user: userId,
  });

  const lesson = await doc.save();
  user.lessons.push(lesson._id);
  await user.save();

  res.json(lesson);
} catch (err) {
  console.log(err);
  res.status(500).json({
    message: 'Не вдалось створити урок',
  });
}
};

export const remove = async (req, res) => {
  try {
    const lessonId = req.params.id;
    const userId = req.userId;

    const lesson = await LessonModel.findById(lessonId);
    if (!lesson) {

```

```

    return res.status(404).json({
      message: 'Урок не знайдено',
    });
  }

  // Перевірка, чи аутентифікований користувач є творцем уроку
  if (lesson.user.toString() !== userId) {
    return res.status(403).json({
      message: 'Ви не маєте дозволу на видалення цього уроку',
    });
  }

  const result = await LessonModel.findOneAndDelete({ _id: lessonId });
  if (!result) {
    return res.status(404).json({
      message: 'Урок не знайдено',
    });
  }

  res.json({
    success: true,
  });
} catch (err) {
  console.log(err);
  res.status(500).json({
    message: 'Не вдалося видалити урок',
  });
}
};

export const update = async (req, res) => {
  try {
    const lessonId = req.params.id;
    const userId = req.userId;
    const lesson = await LessonModel.findById(lessonId);
    if (!lesson) {
      return res.status(404).json({

```

```

        message: 'Урок не знайдено',
    });
}
if (lesson.user.toString() !== userId) {
    return res.status(403).json({
        message: 'Ви не маєте дозволу на оновлення цього уроку',
    });
}

await LessonModel.updateOne(
    {
        _id: lessonId,
    },
    {
        title: req.body.title,
        text: req.body.text,
        imageUrl: req.body.imageUrl,
        videoUrl: req.body.videoUrl,
        photoUrl: req.body.photoUrl,
        user: userId,
    },
);

res.json({
    success: true,
});
} catch (err) {
    console.log(err);
    res.status(500).json({
        message: 'Не вдалося оновити урок',
    });
}
};

export const registerForLesson = async (req, res) => {

```

```

try {
  const lessonId = req.body.lessonId;
  const userId = req.userId;
  const user = await UserModel.findById(userId);
  const lesson = await LessonModel.findById(lessonId);
  if (!lesson) {
    return res.status(404).json({
      message: 'Урок не знайдено',
    });
  }
  if (user.lessons.includes(lessonId)) {
    return res.status(400).json({
      message: 'Ви вже зареєстровані на цей урок',
    });
  }
  user.lessons.push(lessonId);
  await user.save();
  sendRegistrationEmail(user.email, user.fullName, lessonId, lesson.title);
  res.json({
    success: true,
  });
} catch (err) {
  console.log(err);
  res.status(500).json({
    message: 'Не вдалося зареєструватися на урок',
  });
}
};

const sendRegistrationEmail = (toEmail, fullName, lessonId, lessonTitle) => {
  const transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: 'medcourses.message@gmail.com',
      pass: 'ditgplqtpmaamqik',
    },
  });

```

```

    }
  });

  const mailOptions = {
    from: '<medcourses.message@gmail.com>',
    to: toEmail,
    subject: 'Registration for a Lesson',
    text: `Вітаю, ${fullName},\n\n Ви успішно зареєструвались на курс "${lessonTitle}" (ID:
${lessonId}).\nПриємного навчання!`,
  };

  transporter.sendMail(mailOptions, (error, info) => {
    if (error) {
      console.log(error);
    } else {
      console.log('Email sent: ' + info.response);
    }
  });
};

export const getUserLessons = async (req, res) => {
  try {
    const userId = req.userId;
    const user = await UserModel.findById(userId).populate('lessons');
    if (!user) {
      return res.status(404).json({
        message: 'Користувач не знайдений',
      });
    }
    const lessons = user.lessons;
    const lessonTitles = lessons.map(lesson => lesson.title);
    res.json(lessonTitles);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалося отримати уроки користувача',
    });
  }
};

```



```
});
}
};
```

UserController.js:

```
import jwt from 'jsonwebtoken';
import bcrypt from 'bcrypt';
import UserModel from '../models/user.js';
import nodemailer from 'nodemailer';

export const register = async (req, res) => {
  try{
    const password = req.body.password;
    const salt = await bcrypt.genSalt(10);
    const hash = await bcrypt.hash(password, salt);

    const doc = new UserModel({
      email:req.body.email,
      fullName:req.body.fullName,
      workExperience:req.body.workExperience,
      avatarUrl:req.body.avatarUrl,
      selfieUrl:req.body.selfieUrl,
      passwordHash: hash,
      role: req.body.role || 'teacher',
    });

    const user = await doc.save();

    const token = jwt.sign({
      _id: user._id,
      role: user.role,
    }, 'secret123',
    {
      expiresIn:'30d',
    }
  );

  const {passwordHash, ...userData} = user._doc;
```

```

res.json({
  ... userData,
  token,
});

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'medcourses.message@gmail.com',
    pass: 'ditgplqtpmaamqik'
  }
});

const mailOptions = {
  from: '<medcourses.message@gmail.com>',
  to: req.body.email,
  subject: 'Succesfull register',
  text: 'Вітаю, ' + req.body.fullName + '. Ви успішно зареєструвалися на сайті медкурсів.  
Приємного користування сервісом!'
};

transporter.sendMail(mailOptions, (error, info) => {
  if (error) {
    console.log(error);
  } else {
    console.log('Email sent: ' + info.response);
  }
});

} catch(err){
console.log(err);
res.status(500).json({
  message: 'Неуспішна реєстрація',
});
}

};

export const login = async (req, res) => {
  try{

```

```

const user = await UserModel.findOne({email: req.body.email});
if (!user){
  return res.status(404).json({
    message: 'Користувач не знайдений',
  });
}
const isValidPass = await bcrypt.compare(req.body.password, user._doc.passwordHash);

if(!isValidPass){
  return res.status(404).json({
    message: 'Неправильний пароль',
  });
}
const token = jwt.sign({
  _id: user._id,
}, 'secret123',
{
  expiresIn: '30d',
});
const {passwordHash, role, ...userData} = user._doc;
res.json({
  ... userData,
  role,
  token,
});
} catch(err){
  console.log(err);
res.status(500).json({
message: 'Неуспішний вхід',
});
}
};

export const getMe = async (req, res) => {

```

```

try {
  const user = await UserModel.findById(req.userId);
  if (!user) {
    return res.status(404).json({
      message: 'Користувач не знайдений',
    });
  }
  const { passwordHash, ...userData } = user._doc;
  res.json(userData);
} catch (err) {
  console.log(err);
  res.status(500).json({
    message: 'Немає доступу',
  });
}
};

export const getAllUsers = async (req, res) => {
  try {
    const users = await UserModel.find();
    res.json(users);
  } catch (err) {
    console.error(err);
    res.status(500).json({
      message: 'Не вдалося отримати користувачів',
    });
  }
};

```

checkAuth.js:

```

import jwt from 'jsonwebtoken';

export default (req, res, next) => {
  const token = (req.headers.authorization || "").replace(/Bearer\s?/, "");
  if (token) {
    try {

```

```

const decoded = jwt.verify(token, 'secret123');
req.userId = decoded._id;
req.userRole = decoded.role;
next();
} catch (e) {
  return res.status(403).json({
    message: 'Немає доступу',
  });
}
} else {
  return res.status(403).json({
    message: 'Немає доступу',
  });
}
};

```

handleValidationErrors.js:

```

import { validationResult } from 'express-validator';
export default (req, res, next) => {
  const errors = validationResult(req);
  if(!errors.isEmpty()){
    return res.status(400).json(errors.array());
  }
  next();
};

```

validations.js:

```

import { body } from 'express-validator'
export const loginValidation = [
  body('email', 'Неправильний формат пошти').isEmail(),
  body('password', 'Пароль має бути більшим за 5 символів').isLength({ min: 5 }),
];
export const registerValidation = [
  body('email', 'Неправильний формат пошти').isEmail(),
  body('password', 'Пароль має бути більшим за 5 символів').isLength({ min: 5 }),
];

```

```
body('fullName', 'Неправильний формат імені').isLength({ min: 2}),
body('workExperience', 'Неправильний формат досвіду роботи').optional().isString(),
body('avatarUrl', 'Неправильний формат посилання').optional().isURL(),
];

export const lessonCreateValidation = [
  body('title', 'Неправильний формат заголовка').isLength({ min: 3 }).isString(),
  body('text', 'Введіть текст уроку').isLength({ min: 3 }).isString(),
  body('imageUrl', 'Неправильний формат посилання').optional().isString(),
  body('videoUrl', 'Неправильний формат посилання').optional().isURL(),
];
```