ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ДИСТАНЦІЙНОГО НАВЧАННЯ: СЕРВЕРНА ЧАСТИНА»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Шаповалов Іван Романович
_____ «____»_____ 2023 р.
(підпис)

Науковий керівник к.пед.н., доцент, Кошова Оксана Петрівна
_____ «____»_____ 2023 р.
(підпис)

Рецензент

ПОЛТАВА – 2023

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 202__ р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка програмного забезпечення системи дистанційного навчання: серверна частина**»**
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Шаповалов Іван Романович
Затверджена наказом ректора № 144-Н від «01» вересня 2022 р.
Термін подання студентом роботи «___»_____ 202__ р.
Вихідні дані до кваліфікаційної роботи: публікації з теми, документація програмних засобів, системи дистанційного навчання.

Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

РОЗДІЛ 2. ОГЛЯД ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА АНАЛОГІЧНИХ ПРОГРАМНИХ ПРОДУКТІВ

2.1 Google Classroom

2.2 Moodle

2.3 Canvas

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Основні поняття

3.2. Опис обраної архітектури та програмних рішень

3.3. Вибір основного стеку технологій

3.4. Технології для покращення процесу розробки

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис основних модулів системи

4.2. Робота з даними

4.3. Опис роботи серверного API

4.4. Опис роботи з файлами системи

ВИСНОВКИ

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

ДОДАТОК А. ВИХІДНІ КОДИ

Перелік графічного матеріалу: 17 ілюстрацій.

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Кошова О.П.		
Інформаційний огляд	Кошова О.П.		
Теоретична частина	Кошова О.П.		
Практична частина	Кошова О.П.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 202___ р.

Здобувач вищої освіти _____ Шаповалов Іван Романович
(підпис)

Науковий керівник _____ к.пед.н., доц. Кошова О.П.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2023 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202__ р.

Погоджено
Науковий керівник _____
к.пед.н., Оксана КОШОВА
«____» _____ 202__ р.

План

кваліфікаційної роботи на тему
«Розробка програмного забезпечення системи дистанційного навчання: серверна частина»
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Шаповалов Іван Романович

Здобувач вищої освіти _____ Іван ШАПОВАЛОВ
«____» _____ 202__ р.

РЕФЕРАТ

Записка: 58 сторінок, 17 рисунків, 1 додаток, 18 літературних джерел.

СИСТЕМА ДИСТАНЦІЙНОГО НАВЧАННЯ, СЕРВЕРНА ЧАСТИНА, NESTJS, POSTGRESQL, AMAZON S3.

Мета роботи – розробка програмного забезпечення системи дистанційного навчання: серверна частина.

Об'єкт розробки – система дистанційного навчання.

Методи дослідження – фреймворк для розробки серверних додатків NestJS, систему управління базами даних PostgreSQL, Amazon S3-подібне файлове сховище, набір інструментів для документування серверного API Swagger, редактор коду Visual Studio Code, мову програмування TypeScript, систему контролю версій Git та інші допоміжні засоби та бібліотеки.

Зроблено огляд систем дистанційного навчання, виділено їх позитивні та негативні сторони. Виконано опис проектних рішень, інструментів та підходів до розробки системи дистанційного навчання. Обрані засоби та інструменти для розробки. Побудовано діаграму модулів системи. Розроблено серверну частину системи. Розроблена документація створеного серверного API.

ЗМІСТ

ВСТУП

Актуальність. Популярність впровадження Інтернет-технологій у більшість аспектів життя людини сьогодення зараз знаходиться на своєму піку. Не залишилось майже жодної ніші, куди б не зазирнув прогрес та інновації пов’язані з автоматизацією та поширенням доступності.

Не виключенням є і сучасний підхід до навчання, до якого кожного дня впроваджуються все нові і нові підходи до поліпшення та спрощення навчального процесу як для студентів, так і для викладачів. Для цього зазвичай використовуються системи управління навчанням (LMS), котрі адаптували у себе навчальні заклади усіх рівнів – від шкіл до коледжів та університетів. Дані системи надають учасникам навчального процесу великий інструментарій засобів, котрі можна легко використовувати для будь-якого типу викладання навчального матеріалу, як очною так і дистанційною.

Після початку пандемії COVID-19 майже всі навчальні заклади по всьому світу так чи інакше запровадили у себе викладання навчальних матеріалів у дистанційному форматі, що дозволило їм продовжити роботу не дивлячись на карантинні умови, запроваджені місцевою владою. Крім цього, перехід на онлайн формат навчання приніс із собою і інші переваги, такі як:

- зникнення потреби відвідувати навчальний заклад для отримання знань, що дозволило залучити більше студентів до навчання;
- використання сучасних технологій для проведення навчального процесу;
- надавання доступу до лекцій, завдань та інших важливих матеріалів за вимогою студентів;
- проведення лекцій в онлайн форматі за допомогою різних засобів спілкування на дистанції.

Мета роботи є розробка програмного забезпечення системи дистанційного навчання: серверна частина.

Об'єктом розробки є система дистанційного навчання.

Предметом розробки є програмне забезпечення системи дистанційного навчання з використанням TypeScript фреймворку NestJS для створення серверної частини додатку.

Перелік використаних методів полягає у проектуванні системи дистанційного навчання.

При реалізації проекту використано: фреймворк для розробки серверних додатків NestJS, систему управління базами даних PostgreSQL, Amazon S3-подібне файлове сховище, набір інструментів для документування серверного API Swagger, редактор коду Visual Studio Code, мову програмування TypeScript, систему контролю версій Git та інші допоміжні засоби та бібліотеки.

Пояснювальна записка складається з чотирьох розділів, а саме: постановка задачі, огляд та порівняльна характеристика аналогічних програмних продуктів, теоретичної та практичної частин. Структура побудована так, що дає змогу логічно представити матеріал та розкрити тему роботи.

Обсяг пояснювальної записки: 58 сторінок, 17 рисунків, 1 додаток, 18 літературних джерел.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Система, що розробляється, є місцем для автоматизації навчального процесу у дистанційній формі. Саме тому вона повинна покривати основні потреби студентів та викладачів при участі у навчальному процесі. Серверна частина повинна надавати клієнтській частині усе необхідне API для доступу до різних частин системи, але лише за умови коли користувач має необхідні права доступу до даних.

В основі роботи всієї системи лежить перевірка на те чи має конкретний користувач доступ до конкретного ресурсу. Для цього користувачі використовують облікові записи, котрі містять необхідну для перевірки доступу інформацію. Така перевірка повинна проходити при кожному новому запиті до серверу, якщо доступ то запитуваного ресурсу якось обмежений. Наприклад, доступ до ендпоінту авторизації повинен мати кожний клієнт, котрий робить запит на сервер. Обліковий запис повинен мати у собі наступну інформацію про користувача:

- ідентифікатор користувача;
- ідентифікатор поточної сесії;
- ідентифікатор прив’язаного телеграм аккаунту;
- ім’я;
- прізвище;
- ім’я по-батькові;
- адреса електронної пошти;
- номер телефону (не обов’язково);
- пароль;
- роль;
- прив’язані групи;
- фото профілю;

- дата створення облікового запису;
- дата останнього внесення змін до облікового запису.

Необхідною умовою безпеки для користувача є наявність хешування паролю перед його зберіганням до бази даних. Після того, як користувач захоче повторно увійти до свого облікового запису, наданий ним пароль буде захешований та отриманий хеш буде перевірено з тим, що збережений у базі даних. Такий спосіб збереження паролю захистить його від можливих витоку даних користувачів.

Після успішного процесу авторизації користувач повинен отримати спеціальний підписаний сервером токен, який буде містити необхідну для сервера інформацію для подальшої валідації доступу до ресурсів на котрі користувач робить запити.

Ідентифікатор необхідний для ідентифікації конкретного зареєстрованого в системі користувача.

Ідентифікатор поточної сесії буде використовуватись для того, щоб валідувати чи робить користувач запит до серверу за допомогою токену, котрий був наданий йому при останньому проходженні процесу авторизації. Якщо вхід до облікового запису було виконано з будь-якого іншого пристрою, то сесія на всіх інших девайсах стає не валідною і не буде мати право для доступу до серверного API, де умовою є бути авторизованим у системі.

Ідентифікатор прив’язаного телеграм аккаунту буде вказувати на телеграм акаунт з котрого користувач використовує телеграм бота.

Адреса електронної пошти необхідна для проходження процесу авторизації та для можливості відновлення доступу для облікового запису у випадку втрати користувачем паролю.

Кожний користувач має одну з доступних у системі ролей. Такими можуть бути: студент, викладач, адміністратор. Даний параметр використовується для валідації доступу до конкретних ресурсів серверу, котрі мають обмежений доступ для деяких ролей.

Прив’язані групи необхідні користувачу для отримання поточного розкладу занять із системи, запит на котрий робиться до іншої системи, котра не прив’язана до поточної.

Дата створення та останнього редагування облікового запису необхідні для збереження останніх дій користувача з ним.

Ім’я, прізвище, ім’я по-батькові, номер телефону та фото профілю необхідні лише для відображення всередині системи для інших користувачів, будь то студенти, викладачі чи адміністратори.

Важливим умовою є те, що облікові записи користувачів може створити лише адміністратор, при чому пароль до нього повинен бути згенерований системою автоматично та надісланий за вказаною користувачем адресою електронної пошти.

Кожен курс повинен складатися з основної інформації про курс, тем, матеріалів курсу, учасників та зданих робіт учасників курсу. Матеріал курсу може бути як лекційним, так і практичним, котрі учасники курсу повинні виконати та здати викладачу на оцінювання. Матеріал повинен бути частиною конкретною теми.

Основна інформація про курс повинна бути наступною:

- ідентифікатор курсу;
- назва курсу;
- фото оформлення;
- викладач;
- учасники;
- теми;
- дата створення;
- дата останнього внесення змін;
- дата видалення.

Ідентифікатор необхідний для ідентифікації конкретного створеного в системі курсу.

Назва та оформлення курсу необхідні лише для відображення всередині системи для викладача та учасників даного курсу.

Поле викладача буде вказувати на користувача, котрий створив даний курс і в свою чергу є його викладачем та має доступ до створення тем, матеріалів, редагування основної інформації курсу, зданих учасниками практичних робіт та можливості їх оцінювання.

Поле учасників буде вказувати на поточний список учасників даного курсу, котрі мають доступ до перегляду матеріалів курсу та здачі практичних робіт викладачеві.

Поле тем буде вказувати на теми, котрі створив викладач для поточного курсу та до яких прив’язані матеріали курсу.

Дата створення та останнього редагування курсу необхідні для збереження останніх дій викладача з ним.

Дата видалення необхідна для маркування курсу як архівованого. До архівованого курсу неможливо вносити будь-які дії, він слугує лише для перегляду інформації.

Учасник курсу повинен мати наступну інформацію:

- ідентифікатор учасника;
- ідентифікатор користувача;
- ідентифікатор курсу;
- здані практичні роботи;
- дата створення;
- дата останнього внесення змін.

Ідентифікатор використовується для ідентифікації конкретного створеного в системі учасника конкретного курсу.

Ідентифікатор користувача необхідний для прив’язки конкретного користувача системи до поточного учасника курсу.

Ідентифікатор курсу необхідний для прив’язки конкретного курсу у системі до поточного учасника курсу.

Здані практичні роботи є здані поточним учасником практичні роботи до прив’язаного до нього курсу.

Дата створення та останнього редагування необхідні для збереження останніх дій викладача з поточним користувачем.

Теми курсу повинні містити у собі наступну інформацію:

- ідентифікатор теми;
- назва;
- ідентифікатор курсу;
- матеріали курсу;
- дата створення;
- дата останнього внесення змін.

Ідентифікатор використовується для ідентифікації конкретної створеної в системі теми конкретного курсу.

Назва необхідна для відображення всередині системи для викладача та учасників прив’язаного до неї курсу.

Ідентифікатор курсу необхідний для прив’язки конкретного курсу у системі до поточної теми.

Матеріали курсу є прив’язані до поточної теми матеріали курсу, котрі доступні викладачу та учасникам курсу.

Дата створення та останнього редагування необхідні для збереження останніх дій викладача з поточною темою.

Матеріал курсу повинен містити у собі наступну інформацію:

- ідентифікатор матеріалу;
- назва;

- опис;
- тип;
- прикріплені файли;
- ідентифікатор теми;
- здані практичні роботи;
- дата створення;
- дата останнього внесення змін.

Ідентифікатор використовується для ідентифікації конкретного створеного в системі матеріалу конкретного курсу.

Назва та опис необхідні лише для відображення всередині системи для викладача та учасників даного курсу.

Тип вказує на один з можливих типів поточного матеріалу, котрими можуть бути лекція або практичне завдання. Якщо типом є практичне завдання, це означає, що учасники курсу можуть здавати виконані роботи по поточному матеріалу.

Прикріплені файли є списком файлів, котрий викладач прикріпляє при створенні матеріалу та який потім буде доступний для перегляду учасникам курсу.

Ідентифікатор теми необхідний для прив’язки конкретної теми курсу до поточного матеріалу.

Здані практичні роботи є списком зданих учасниками курсу практичних робіт до поточного матеріалу.

Дата створення та останнього редагування необхідні для збереження останніх дій викладача з поточним матеріалом.

Здана практична робота повинна містити наступну інформацію:

- ідентифікатор зданої практичної роботи;
- оцінка;
- ідентифікатор матеріалу;

- ідентифікатор учасника;
- файл;
- дата створення;
- дата останнього внесення змін.

Ідентифікатор використовується для ідентифікації конкретної практичної роботи в системі.

Оцінка відповідає за виставлену викладачем курсу оцінки за поточну виконану практичну роботу. Можлива оцінка повинна бути у діапазоні від 0 до 100.

Ідентифікатор матеріалу необхідний для прив’язки конкретного матеріалу курсу до поточної зданої практичної роботи.

Ідентифікатор учасника необхідний для прив’язки конкретного учасника курсу до поточної зданої практичної роботи.

Файл слугує посиланням на файл, котрий учасник курсу прикріпив при здачі поточної практичної роботи.

Дата створення та останнього редагування необхідні для збереження останніх дій викладача чи учасника курсу з поточною зданою практичною роботою.

Файл матеріалу чи зданої практичної роботи повинен містити таку інформацію:

- ідентифікатор файлу;
- посилання на файл у віддаленому сховищі;
- ідентифікатор користувача (не обов’язково);
- ідентифікатор матеріалу (не обов’язково);
- ідентифікатор зданої практичної роботи (не обов’язково);
- ім’я;
- кодування;
- ключ;

- тег сутності;
- дата створення;
- дата останнього внесення змін.

Ідентифікатор використовується для ідентифікації конкретного файлу, котрий був збережений у системі.

Посилання на файл у віддаленому сховищі містить видану сервером URL тимчасового доступу до поточного файлу для можливості користувача завантажити його собі локально.

Ідентифікатор користувача необхідний для прив’язки конкретного користувача до файлу у разі, якщо користувач системи зберіг його у своїх персональних файлах.

Ідентифікатор матеріалу необхідний для прив’язки конкретного матеріалу курсу до поточного файлу.

Ідентифікатор зданої практичної роботи необхідний для прив’язки конкретної зданої учасником курсу практичної роботи до поточного файлу.

Ім’я, кодування, ключ та тег сутності необхідні для подальшої роботи з віддаленим файловим сховищем та для можливості створювати тимчасові посилання на файли для користувачів.

Дата створення та останнього редагування необхідні для збереження останніх дій над файлом.

Особливу увагу необхідно виділити налаштуванню прав доступу до всіх компонентів курсу. Не можна допустити, щоб викладач, котрий не створив курс, або користувач, котрий не є учасником курсу, отримали доступ до його компонентів. Також не можна допустити, щоб учасники отримали доступ до налаштувань самого курсу, управління його наповненням або можливості отримання доступу до зданих практичних робіт інших учасників та можливості виставлення оцінки за неї. Дані можливості повинні бути наявні лише у викладача курсу.

РОЗДІЛ 2. ОГЛЯД ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА АНАЛОГІЧНИХ ПРОГРАМНИХ ПРОДУКТІВ

2.1 Google Classroom

Google Classroom – це безкоштовна онлайн-платформа для навчання розроблена компанією Google. Основна мета цього програмного застосунку – спростити процес обміну файлами та завданнями між викладачами та студентами. Користуватися цією платформою можна з використанням двох різних ролей – студент та викладач.

Як викладачу, користувачу доступні такі можливості:

- почати відео зустріч;
- створення та керування курсами, їх учасниками, завданнями та оцінками студентів всередині цих курсів;
- переглядати звіт по завданням студентів, як виконаним так і не виконаним;
- додавання матеріали до своїх завдань, як-от відео YouTube, опитування Google Forms, елементи з Диска Google та інше;
- надсилати прямий зворотний зв’язок студентам до кожного завдання;
- використовувати потік курсу, щоб публікувати оголошення та залучати студентів до дискусій, орієнтованих на запитання;
- запропонувати батькам підписатися на отримання електронною поштою підсумків по роботі студентів.

Студент має наступні можливості всередині додатку:

- переглядати список доступних курсів (Рисунок 2.1);
- відстежуйте роботу в класі та надсилати завдання на перевірку викладачу;
- перевіряти відгуки та оцінки до виконаних завдань, а також спілкуватися з викладачем напряму в чаті до завдання;

– ділитися ресурсами та спілкуватися в потоці класу або електронною поштою [1].

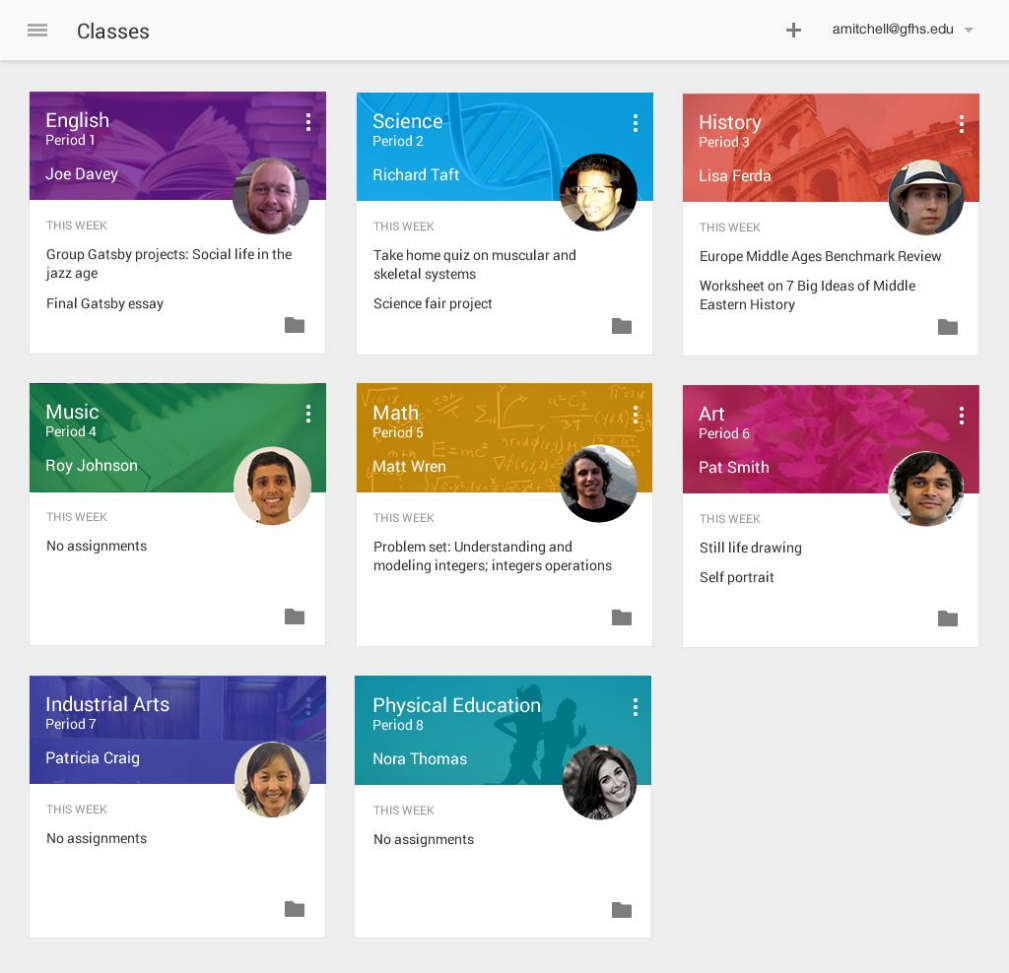

Рисунок 2.1 – список доступних курсів.

Окрім всього перерахованого, Google Classroom має доволі багато інших корисних можливостей для навчання, одна з них – звіт про оригінальність. Ця функція є вбудованим інструментом виявлення плагіату, доступ до якого мають як студенти, так і викладачі. Викладачі можуть переглядати звіт про оригінальність, що дозволяє їм перевірити академічну доброчесність поданої студентом роботи. У безкоштовній версії G Suite for Education викладачі можуть увімкнути звіт про оригінальність для 3 завдань, але мають обмежене хмарне сховище. Це обмеження знято для платної версії G Suite Enterprise for Education.

Користуватися додатком можна з будь-яких девайсів, як з персонального комп’ютера за допомогою браузера, так і через мобільні пристрої на базі операційних системи Android та IOS.

Великою перевагою Google Classroom є те, що на більшості пристроїв можна працювати в автономному режимі, завантажуючи матеріали чи завдання заздалегідь, і викладати їх онлайн, коли пристрій користувача знайде Інтернет з’єднання.

Також, будучи продуктом компанії Google, Classroom із легкістю інтегрується із іншими її програмними продуктами, такими як: Google Docs, Sheets, Slides, Sites, Planet Earth, Calendar, Gmail, а також Google Hangouts або Meet для проведення лекцій чи онлайн заходів.

2.2 Moodle

Moodle – це багатофункціональна, безпечна та масштабована система керування навчанням із відкритим вихідним кодом, яка розповсюджується за ліцензією GNU General Public License та яка легко інтегрується з іншими платформами та може бути налаштована для будь-якого обраного методу навчання. Зазвичай цей програмний продукт використовується при дистанційному або змішаному навчанні в школах, університетах та інших навчальних закладах (Рисунок 2.2).

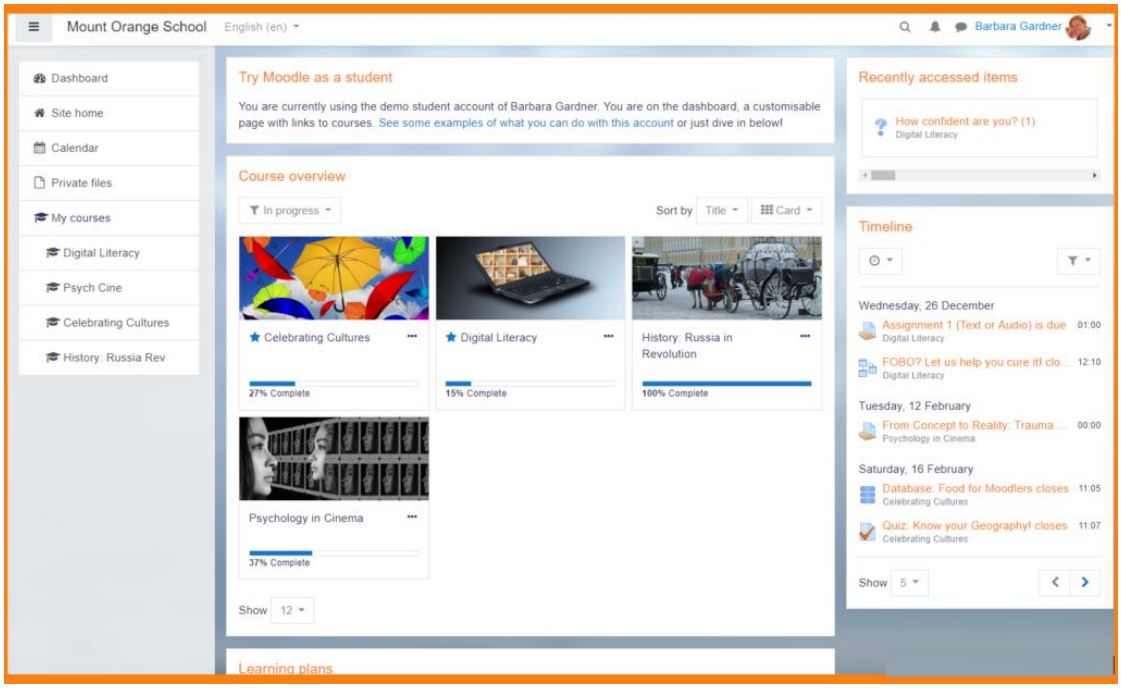

Рисунок 2.2 – Головний екран системи Moodle

Так як проект має відкриту кодову базу, будь-хто може встановити його для користування з використанням будь-якого веб-серверу (NGINX, Apache) та будь-якої реляційної СУБД (MySQL, PostgreSQL). Сертифіковані партнери Moodle також надають варіанти хостингу на своїх машинах, що дозволяє спростити процес запуску своєї платформи.

Користуватися Moodle можна з будь-якого застосунку, включаючи мобільні пристрої для яких розроблений спеціальний додаток для зручного використання всіх доступних сайтів. Він також має і додаткові функції, такі як оффлайн-доступ, і спеціальний інтерфейс, адаптований під мобільні пристрої. Цей додаток спрощує користування декількома сайтами з одного пристрою, однак користувачу необхідно перемикати сеанси для взаємодії з кожним сайтом, тому використовувати декілька сайтів одночасно неможливо, проте користувач буде мати можливість отримувати push-повідомлення та нагадування для всіх сайтів, підключених у додатку. І так як цей додаток також має відкритий вихідний код, він може бути скомпільований для використання лише одного сайту, якщо це необхідно [2].

Moodle є потужною системою для управління навчанням, котра може розширюватись, що можливо за допомогою встановлення плагінів. Офіційна бібліотека налічує близько 2100 плагінів, котрі розробляються широкою спілкою розробників для поліпшення та налаштування більш гнучкого та зручного формату онлайн навчання.

Як платформа, Moodle надає потужний інструментарій для створення і доставки онлайн-курсів із різноманітними функціями і можливостями, а наявність бази з відкритим вихідним кодом, гнучкість і широкий набір функцій роблять його одним з найпопулярніших інструментів для управління навчанням різних форматів.

2.3 Canvas

Canvas – це широко використовувана система управління навчанням (LMS) розроблена компанією Instructure, котра надає великий набір засобів для зручного управління онлайн-навчанням для освітніх закладів різних рівнів – від шкіл до вищих навчальних закладів (Рисунок 2.3).

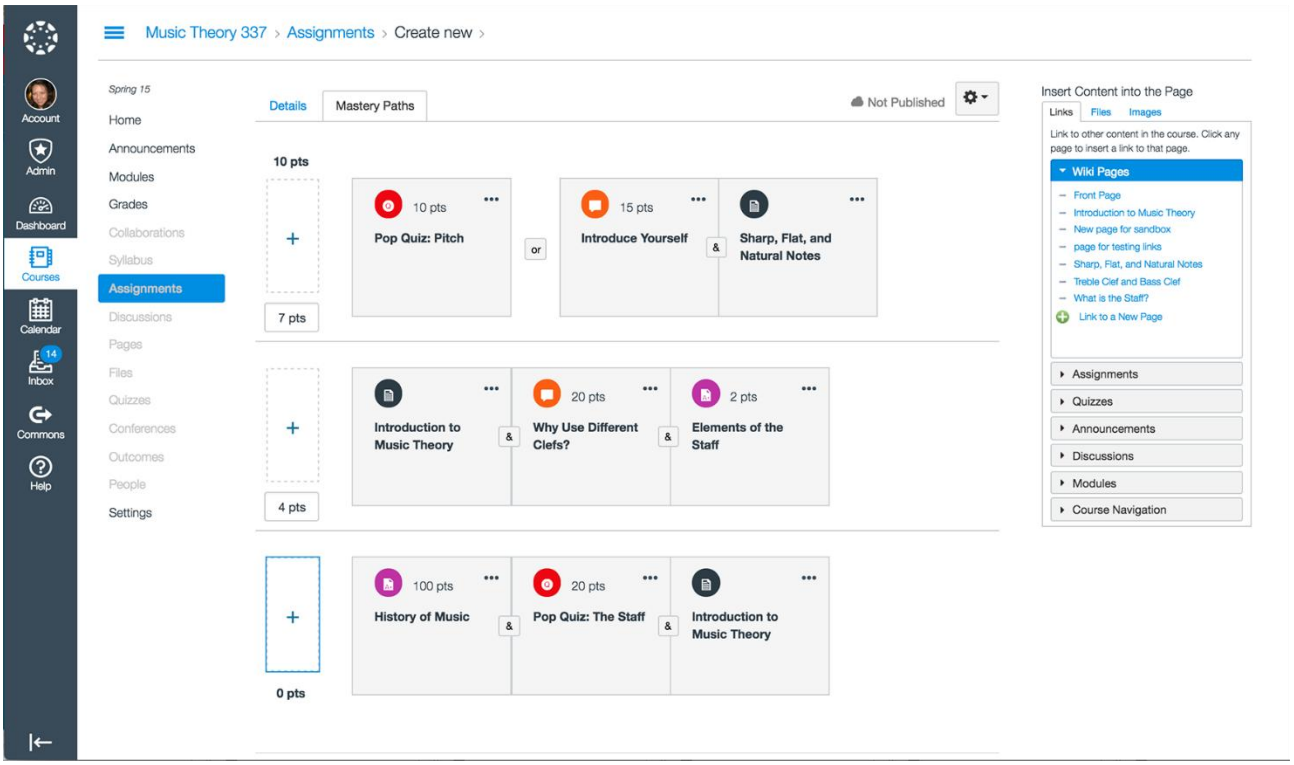

Рисунок 2.3 – Головний екран системи Canvas

Основні можливості цієї платформи:

- створення та організація курсів викладачами з додаванням різного роду матеріалів та завдань, котрі будуть доступні студентам для виконання;
- онлайн-обговорення різних ідей та тем за допомогою дошки обговорень, котра дозволяє студентам спілкуватися як між собою, так і напряму з викладачем курсу;
- виконання завдань різного виду (завантаження файлів, онлайн-тести та обговорення) студентами з подальшою перевіркою, оцінюванням та надаванням відгуків з боку викладача;
- завантаження файлів, вбудованих мультимедіа та посилань на зовнішні ресурси, як матеріали курсу з боку викладача, що дозволяє організовувати його вміст максимально зручно;
- управління оголошеннями для різних груп студентів, котрі викладач може надсилати для інформування учнів про важливі оновлення у курсі чи нагадати про важливу подію;
- надання аналітичних звітів викладачам та адміністраторам про продуктивність та залученість студентів до навчального процесу з відображенням журналу оцінок [3].

Canvas також надає можливість користування своєю платформою через мобільні додатки для пристроїв iOS та Android, що дозволяє студентам і викладачам отримувати доступ до матеріалів курсу, отримувати сповіщення, брати участь в обговореннях і надсилати завдання на ходу.

Ще однією великою перевагою даного програмного продукту є підтримка інтеграцій з широким спектром навчальних інструментів і платформ. Його можна інтегрувати із зовнішніми програмами, такими як Google Drive, Microsoft Office 365 і сторонніми інструментами навчання. Canvas також надає можливості для налаштування, що дозволяє установам адаптувати платформу до своїх конкретних потреб.

Canvas широко поширений у навчальних закладах, від шкіл до університетів, завдяки зручному інтерфейсу, широким можливостям і сильній підтримці спільноти. Його гнучкість і масштабованість роблять його придатним як для онлайн-середовища, так і для змішаного навчання.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Основні поняття

API (application programming interface) – це набір процедур, протоколів і інструментів для створення програмних додатків. Саме за допомогою API можна налаштувати роботу декількох окремих додатків між собою, будь то web додаток, мобільний додаток, чи просто програма на комп’ютері.

MVC (Model-View-Controller) – це архітектурний шаблон, який розділяє програму на три основні логічні компоненти: модель, представлення та контролер. Кожен із цих компонентів розроблено для обробки конкретних аспектів розробки програми. Модель відповідає за репрезентацію даних у системі, контролер ці дані оброблює, а представлення надає результуючим даним особливого вигляду. MVC є однією з найбільш часто використовуваних галузевих стандартних структур веб-розробки для створення масштабованих і розширюваних проектів.

DTO (Data Transfer Object) – це об'єкт, який використовується для інкапсуляції даних і передачі їх з однієї підсистеми програми в іншу. DTO найчастіше використовуються на рівні служб у програмі N-Tier для передачі даних між собою та рівнем інтерфейсу користувача. Основна перевага тут полягає в тому, що він зменшує обсяг даних, які потрібно пересилати в розподілених програмах. Вони також створюють чудові моделі в шаблоні MVC. Іншим використанням DTO може бути інкапсуляція параметрів для викликів методів. Це може бути корисно, якщо метод приймає велику кількість параметрів.

Валідатор – програмний компонент, котрий відповідає за перевірку вхідних даних перед їх подальшою обробкою та відмовляє у подальшій обробці запиту у разі невдалої валідації.

Декоратор (TypeScript) – це особливий вид оголошення, яке можна приєднати до оголошення класу, методу, засобу доступу, властивості або

параметра. Декоратори починають свій синтаксис із символу @, де вираз має обчислювати функцію, яка буде викликана під час виконання з інформацією про декоровану декларацію.

API ендпоінт – місце куди сервер отримує вхідні запити на доступ до конкретного ресурсу системи. Саме ендпоінти і слугують шляхом для доступу до серверних ресурсів за допомогою користувацьких запитів.

RESTful API - це один із багатьох можливих способів, якими програми, сервери та веб-сайти можуть обмінюватися даними та послугами. REST (Representational State Transfer) описує загальні правила представлення даних і послуг через API, щоб інші програми могли правильно запитувати та отримувати дані та служби, доступні API.

HTTP методи – метод по якому визначається доступ до ресурсу сервера. Методи, котрі використовуються найчастіше: GET (отримання інформації про ресурс від серверу), POST (додавання нової інформації), PUT (повне редагування інформації), PATCH (часткове редагування інформації), DELETE (видалення інформації).

Eager relations (TypeORM) – вид відношення між 2 таблицями у базі даних, котрий завантажується автоматично при завантаженні даних з таблиці. Дані відношення також можуть бути вкладеними, тому важливо розпоряджатися ними обережно.

Dependency injection – це техніка програмування, яка робить клас незалежним від його залежностей. Це дає вам змогу замінювати залежності, не змінюючи клас, який їх використовує. Ця техніка представляє із собою створення об’єкту класу від котрого залежать інші класи та його подальше зберігання для передачі у інші можливі залежності.

ORM (Object-Relational Mapping) – це техніка, яка дозволяє запитувати та маніпулювати даними з бази даних за допомогою об’єктно-орієнтованої парадигми. Говорячи про ORM, мається на увазі бібліотеку, яка реалізує техніку об’єктно-реляційного відображення. Бібліотека ORM – це звичайна

бібліотека, у будь-якій мові програмування, яка інкапсулює код, необхідний для обробки даних. Це дозволяє управляти даними в базі більш просто та безпечно, так як ORM в свою чергу надає і безпеку від багатьох вразливостей, котрі має SQL, наприклад, таких як SQL ін’єкції.

3.2. Опис обраної архітектури та програмних рішень

Система буде розроблена з використанням клієнт-серверної архітектури. Клієнтська частина виконана у вигляді веб-додатку з усіма описаними можливостями.

Архітектура клієнт-сервер – це одна з архітектурних шаблонів програмного забезпечення, котра є однією з найпопулярніших концепцій при створенні будь-яких веб застосунків чи мобільних додатків. Її ключовими елементами є сервер, або група серверів, котрі є централізованими системами, які розміщують, надають та керують більшістю ресурсів і послуг, які запитує клієнт. У цій моделі всі запити та послуги доставляються через мережу, і її також називають моделлю мережевих обчислень або мережею клієнт-сервер.

Клієнт-серверна архітектура зазвичай представляється як купа клієнтських пристроїв, таких як ПК, телефон, або будь-який інший смарт-девайс, котрі підключені до центрального сервера через Інтернет або іншу мережу. Клієнти надсилають запит на дані до серверу, а він у свою чергу їх приймає, оброблює та надає відповідь клієнту у вигляді пакетів даних, котрі можуть містити як і необхідні клієнту дані, так і повідомлення про помилку.

Алгоритм спілкування клієнта з сервером виглядає так:

1. клієнт надсилає свій запит до серверу для отримання необхідних йому даних;
2. сервер приймає, валідує, оброблює та відповідає клієнту про успішне виконання запиту, або про те, що сталася помилка;
3. клієнт отримує відповідь від серверу та по-своєму оброблює її, зазвичай показуючи користувачу необхідну інформацію щодо запиту.

Сервери є незалежними один від одного, як і клієнти, котрі функціонують паралельно і незалежно один від одного (Рисунок 3.1).

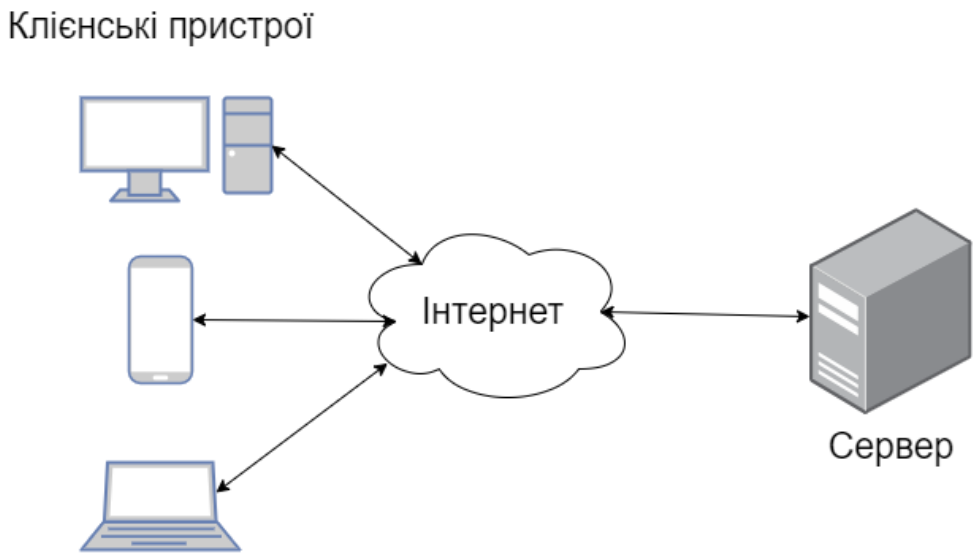

Рисунок 3.1 – Схема роботи клієнт-серверної архітектури

В цій моделі відсутня жорстка прив'язка клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів. Можлива й інша ситуація, коли клієнт звертається до різних серверів з кожним новим запитом. Зазвичай це відбувається, коли група серверів розподіляє навантаження від клієнтських запитів рівномірно між собою. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

3.3. Вибір основного стеку технологій

Серверної частини додатку була реалізована за допомогою мови програмування TypeScript. Її вибір зумовлений простотою написання коду та наявності великої кількості бібліотек і фреймворків для написання веб-додатків.

TypeScript – це безкоштовна мова програмування з відкритим вихідним кодом, розроблена та підтримується Microsoft. Це так званий супер сет JavaScript, який додає статичну типізацію до мови. При компіляції (транспіляції) на виході отримується код мовою JavaScript. За допомогою TypeScript можна розроблювати як серверні, так і клієнтські додатки [4]. Вибір даної мови програмування обумовлений її широким використанням у галузі, наявністю великої екосистеми різних бібліотек та сильної підтримки [5].

Середовищем запуску коду була обрана платформа Node.js, котра працює на основі двигуну V8, та надає можливість запускати код JavaScript, як серверний. Окрім V8, його важлива частина – бібліотека libuv, котра розроблена на мові C використовується для абстрагування неблокуючих операцій введення-виведення до узгодженого інтерфейсу на всіх підтримуваних платформах. Вона надає механізми для обробки файлової системи, DNS, мережі, дочірніх процесів, каналів, обробки сигналів, опитування та потокової передачі. Вона також містить пул потоків для асинхронних операцій, які не можна виконувати асинхронно на рівні операційної системи [6]. Вибір Node.js зумовлений тим, що ця платформа є однією з найстабільніших та більш розвинених платформ для запуску коду JavaScript на серверній частині [7]. Далі наведена схема роботи даної платформи (Рисунок 3.2) [8].

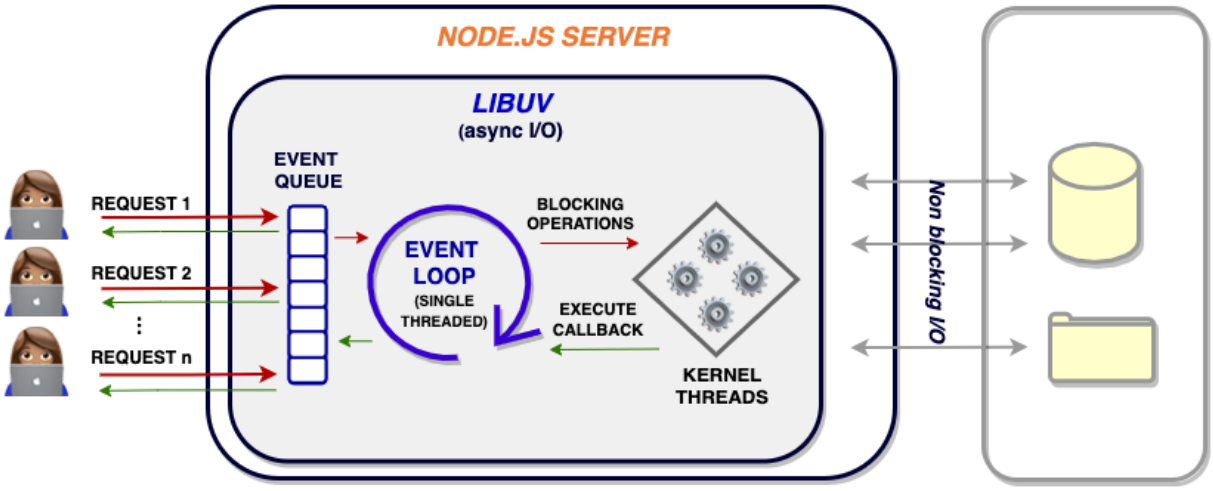

Рисунок 3.2 – Схема роботи Node.js

СУБД для серверної частини була обрана PostgreSQL – це потужна об’єктно-реляційна СУБД з відкритим вихідним кодом, активна розробка якої триває понад 35 років, що заслужило їй міцну репутацію надійності, надійності функцій і продуктивності. PostgreSQL має надійні набори функцій, включаючи Multi-Version Concurrency Control (MVCC), відновлення на певний момент часу, деталізований контроль доступу, табличні простори, асинхронну реплікацію, вкладені транзакції, резервне копіювання в режимі гарячого режиму, удосконалений оптимізатор запитів і ведення журналу наперед [9]. Він підтримує міжнародні набори символів, різні кодування символів, Unicode, а також ураховує локалізацію для сортування, чутливості до регістру та форматування. Вибір даної СУБД обумовлений тим, що вона має високу масштабованість як щодо кількості даних, якими може керувати додаток, так і щодо кількості одночасних користувачів, яких вона може прийняти, а також її високою надійністю (Рисунок 3.3).

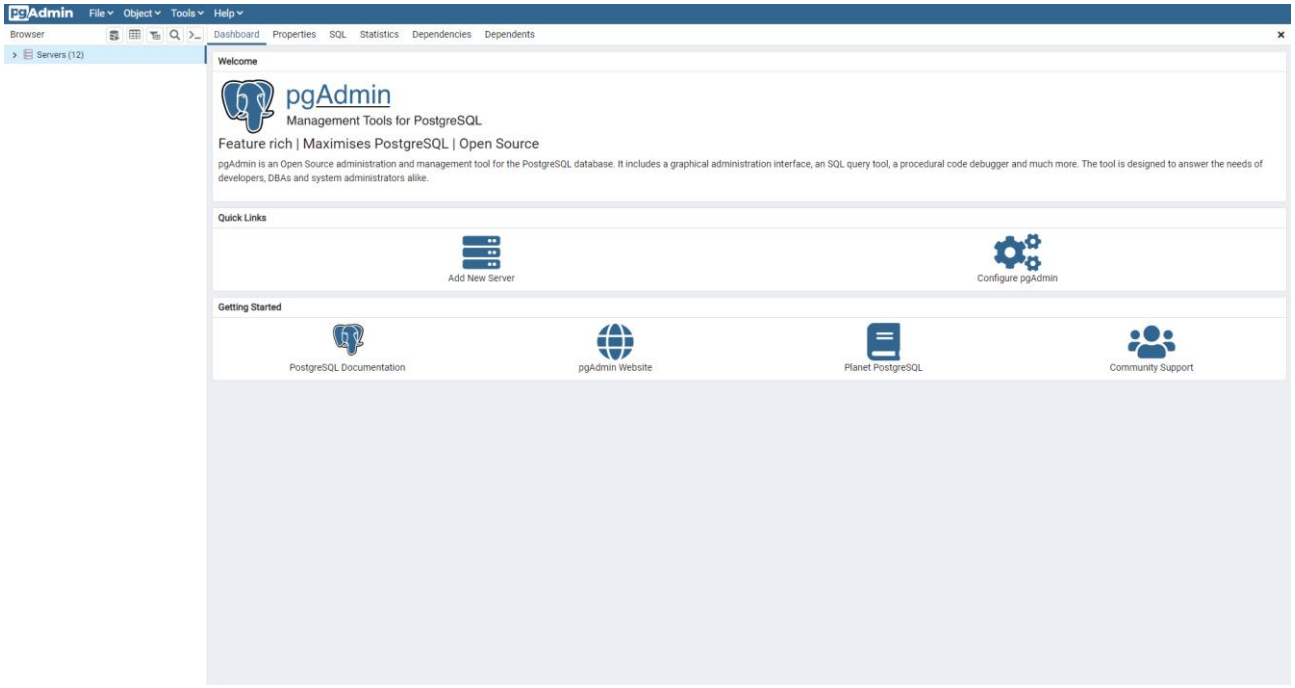

Рисунок 3.3 – Візуалізація СУБД PostgreSQL за допомогою програмного засобу pgAdmin

Фреймворком для розробки було обрано NestJS – це платформа для розробки веб-додатків з використанням паттерна MVC (Model View Controller), мови TypeScript та всіх її нововведень. Проектуючи архітектуру цього фреймворку, розробники надихалися платформою для створення користувацьких інтерфейсів Angular, що дуже добре помітно, якщо поглянути на будь-який додаток розроблений з його використанням. Цей фреймворк містить велику кількість різноманітних засобів для розробки сучасних веб-додатків, котрі будуть використані при розробці системи дистанційного навчання. Будь-який проект, розроблений за допомогою цього фреймворку, має структуру, котра складається з модулів пов’язаних між собою. Модулі можуть містити контролери, котрі представляють із себе маршрутизатори для вхідних користувацьких запитів, та сервіси, котрі займаються обробкою запитів користувачів (Рисунок 3.4). Вибір цього фреймворку обумовлений простотою написання серверного API за допомогою його засобів [10].

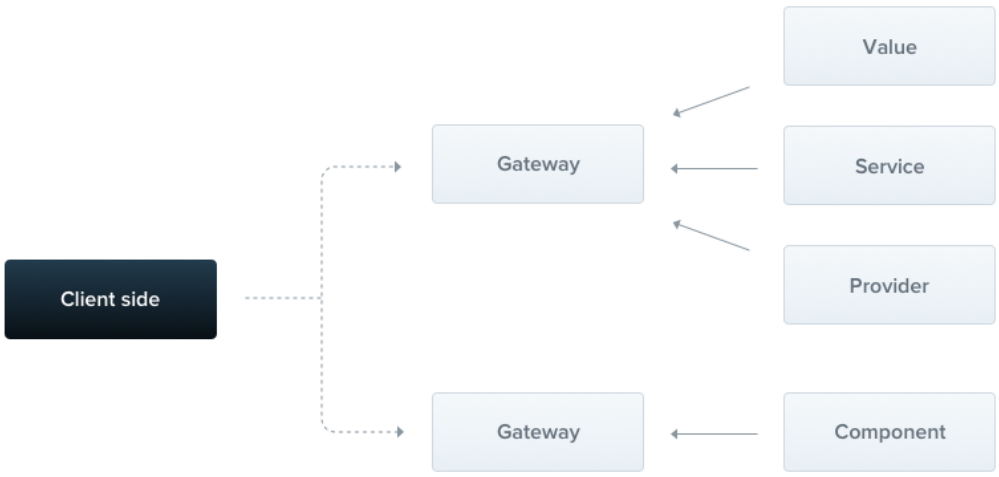

Рисунок 3.4 – Діаграма простого модулю NestJS

ORM для роботи з базою було обрано TypeORM – ORM, котра використовує можливості TypeScript для покращення процесу роботи з базою

даних. Вона підтримує усі сучасні СУБД, у тому числі PostgreSQL. Вибір цієї ORM обумовлений легкою інтеграцією з NestJS, що дозволяє з легкістю використовувати можливості TypeORM всередині сервісів [11].

Сховищем для користувацьких файлів системи було обрано Amazon Simple Storage Service (Amazon S3) – це служба зберігання об’єктів, яка пропонує найкращі в галузі масштабованість, доступність даних, безпеку та продуктивність (Рисунок 3.5). Клієнти будь-якого розміру та галузі можуть використовувати Amazon S3 для зберігання та захисту будь-яких обсягів даних для різноманітних випадків використання. Amazon S3 надає функції керування, для можливості оптимізувати, упорядковувати та налаштовувати доступ до своїх даних відповідно до конкретних бізнес-вимог, організаційних вимог і вимог відповідності. Вибір цього сховища об’єктів обумовлений простотою взаємодії з його серверним API, та його популярністю у галузі [12].

Рисунок 3.5 – Логотип сервісу Amazon S3

3.4. Технології для покращення процесу розробки

При розробці серверу для платформи дистанційного навчання також використовувалися різноманітні технології для покращення процесу написання

вихідного коду. Такі інструменти ніяк не впливають на роботу самої системи, але допомагають розробникам полегшити процес створення необхідної функціональності, покращити якість написаного коду, прискорити швидкість його написання, зберегти його на віддаленому сховищі для запобігання його втрати або навіть налаштування автоматизованого процесу тестування та розгортання.

Git – це безкоштовна розподілена найрозповсюдженіша система контролю версій із відкритим вихідним кодом, призначена для швидкої та ефективної обробки будь-яких проектів, від малих до дуже великих [13]. Є однією з найважливіших складових у розробці будь-якого програмного продукту. Базовим поняттям у системі контролю версій Git є репозиторій. Це універсальний контейнер для збереження файлів вихідного коду який також тримає у собі всю історію змін коду, який коли-небудь був збережений у ньому. При зміні будь-якого з файлів всередині репозиторію він переходить в Unstaged changes. Це зміни, які не відстежуються Git. Він підтримує проміжну область (також відому як staging area) для відстеження змін, які входять до наступного комміту (Рисунок 3.6). Поетапне внесення змін помістить файли в проміжну область, а наступний комміт перенесе всі елементи з неї до вашого репозиторія. Ще однією важливою складовою Git є комміти. Вони є фіксацією зміни файлу або групи файлів, котрі знаходилися в проміжній області, всередині репозиторія. Саме комміти утворюють історію змін при розробці програмного продукту. Також важливим поняттям є так звані гілки, котрі дозволяють створювати нові галуження коммітів, на основі інших. Зазвичай вони використовується для розділу роботи декількох людей над одним проектом, що дозволяє їм працювати над розробкою різної функціональності програмного продукту одночасно.

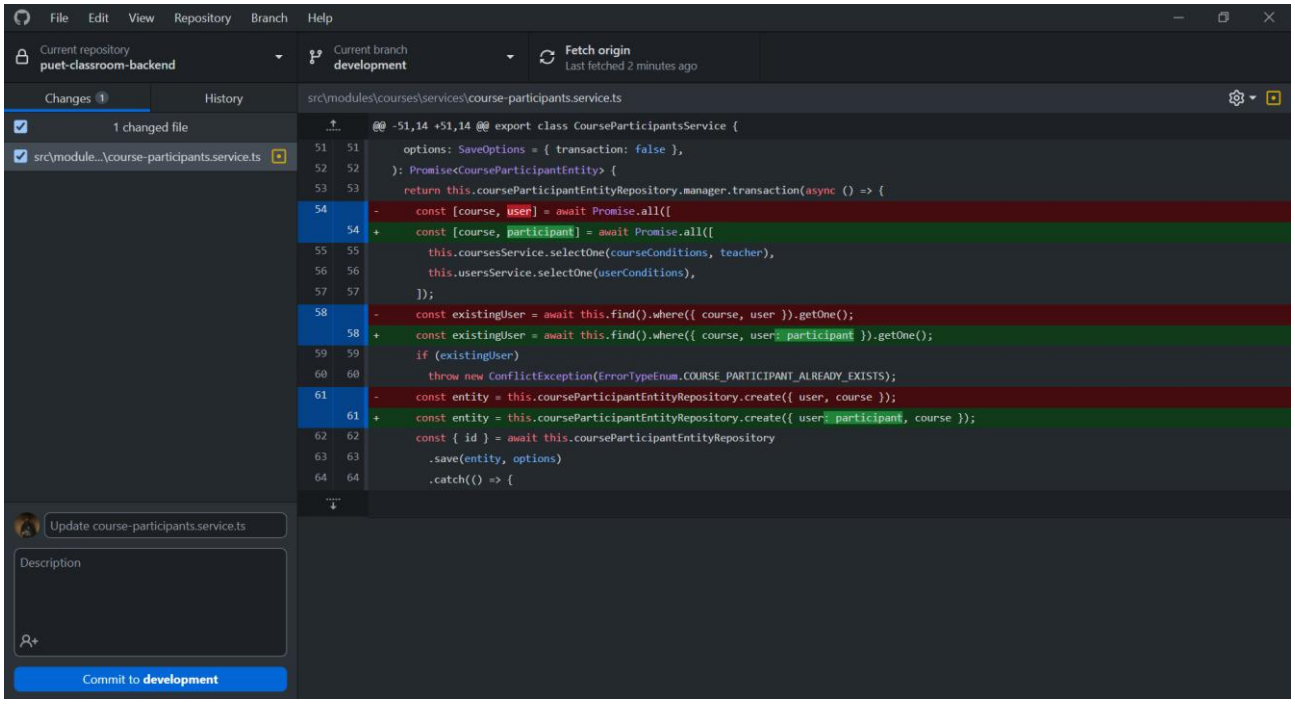

Рисунок 3.6 – Візуалізація відстеження змін у коді за допомогою програмного засобу GitHub Desktop

Prettier – це опціонований форматувальник коду, котрий забезпечує узгоджений стиль написання вихідного коду, на основі своїх правил з деякими можливостями кастомізації. Він підтримує JavaScript, JSX, Angular, Vue, Flow, TypeScript, CSS, Less, SCSS, HTML, Ember/Handlebars, JSON, GraphQL, Markdown, YAML (Рисунок 3.7). Також він інтегрується з більшістю середовищ для розробки. Його можливості кастомізації включають в себе налаштування максимальної довжини рядку, використання табуляції або пробілів, використання крапки з комою та багато іншого. Його використання дозволяє притримуватись одного стилю написання коду, коли мова йде про проекти над якими працюють команди розробників [14].

```typescript
/**
 * [description]
 * @param conditions
 * @param entityLike
 * @param owner
 * @param options
 */
public async createOne(
  conditions: Partial<CourseTopicEntity>,
  entityLike: Partial<CourseActivityEntity>,
  owner?: Partial<UserEntity>,
  options: SaveOptions = { transaction: false },
): Promise<CourseActivityEntity> {
  return this.courseActivityEntityRepository.manager.transaction(async () => {
    const topic = await this.courseTopicsService.selectOne(conditions, owner);
    const entity = this.courseActivityEntityRepository.create({
      ...entityLike,
      topic,
      file: { owner },
    });
    const { id } = await this.courseActivityEntityRepository.save(entity, options).catch(() => {
      throw new ConflictException(ErrorTypeEnum.COURSE_ACTIVITY_ALREADY_EXISTS);
    });

    return this.selectOne({ id }, {}, { loadEagerRelations: true });
  });
}
```

Рисунок 3.7 – Приклад форматування програмного коду мови TypeScript за допомогою Prettier

ESLint – проект із відкритим вихідним кодом, який допомагає знаходити та виправляти проблеми з кодом JavaScript або TypeScript, котрий інтегрується з більшістю середовищ для розробки. Він представляє із себе набір правил, котрі можна налаштовувати і також створювати свої власні (Рисунок 3.8). ESLint охоплює аналізування як якості, так і стилю кодування. ESLint підтримує поточні стандарти ECMAScript та експериментальний синтаксис для майбутніх стандартів. Зазвичай розробники використовують заготовлені набори правил, котрі використовують при розробці свого програмного забезпечення популярні світові IT компанії. Цей проект дозволяє виявляти потенційні проблеми з вихідним кодом та швидко їх вирішити, надавши розробнику можливі варіанти полагодження [15].

Рисунок 3.8 – Візуалізація роботи правила ESLint у програмному коді

Swagger – це платформа програмного забезпечення з відкритим вихідним кодом, яка підтримується великою екосистемою інструментів, яка допомагає автоматично створювати документацію для створеного RESTful API. Також за допомогою цієї платформи можна з легкістю тестувати це ж саме API [16].

Docker – це платформа з відкритим кодом, яка дозволяє розробникам створювати, розгортати, запускати, оновлювати та керувати контейнерами – стандартизованими виконуваними компонентами, які поєднують вихідний код програми з бібліотеками операційної системи (ОС) і залежностями, необхідними для запуску цього коду в будь-якому середовищі [17]. Контейнери ізольовані один від одного та містять власне програмне забезпечення, бібліотеки та конфігураційні файли (Рисунок 3.9). Вони можуть спілкуватися один з одним через чітко визначені канали. Контейнери можуть бути створені на основі образів Docker, котрі містять виконуваний вихідний код програми, а також усі інструменти, бібліотеки та залежності, необхідні для запуску коду програми. Ці образи зазвичай беруться з так званого Docker Hub – місця в якому знаходяться близько 100 000 різних образів для майже будь-якого популярного програмного забезпечення будь то СУБД або файлове сховище. Налаштування процесу розгортання за допомогою Docker описується у

спеціальному файлі Dockerfile, котрий тримає у собі список кроків, котрі необхідно виконати щоб запустити контейнер.

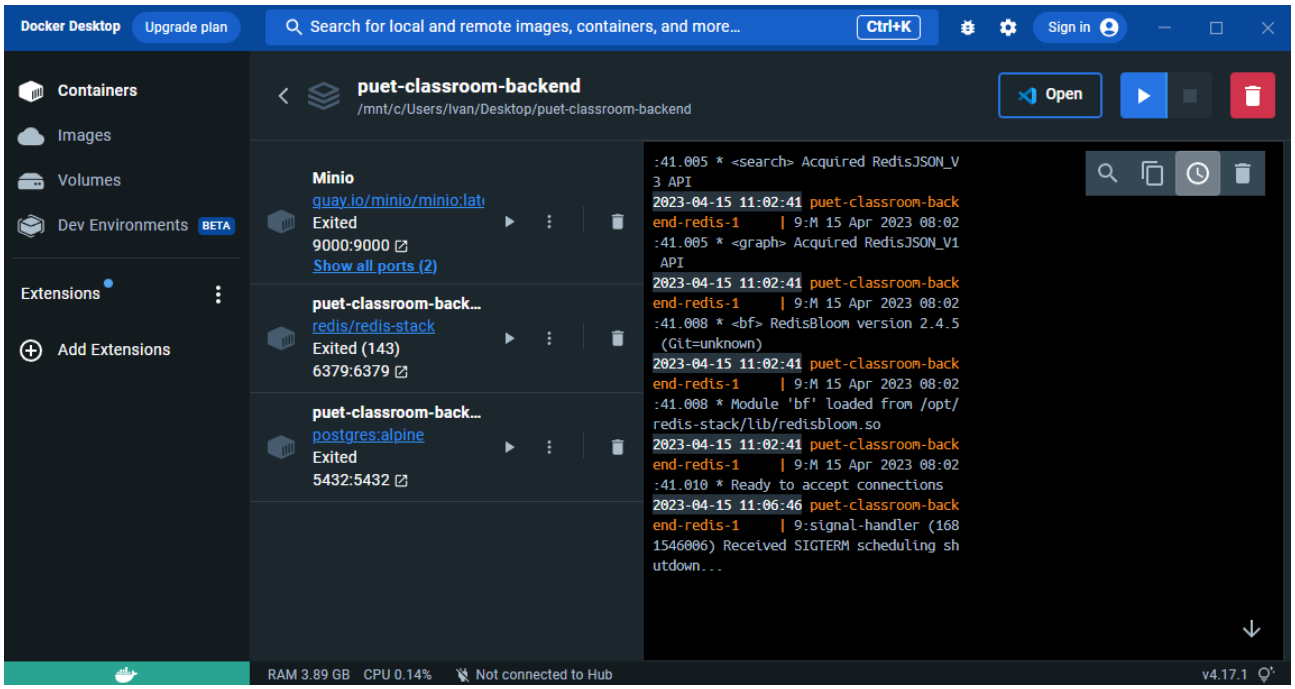

Рисунок 3.9 – Візуалізація запуску контейнерів Docker за допомогою програмного засобу Docker Desktop

MinIO – файлове сховище високого навантаження з відкритим вихідним кодом API котрого сумісне з Amazon S3 і котрий ліцензується під GNU AGPL v3. На відміну від багатьох інших сховищ, він також надає для користувача зручну панель керування усім контентом, правами доступу та налаштуваннями (Рисунок 3.10). Дана технологія використана лише для процесу розробки платформи дистанційного навчання [18].

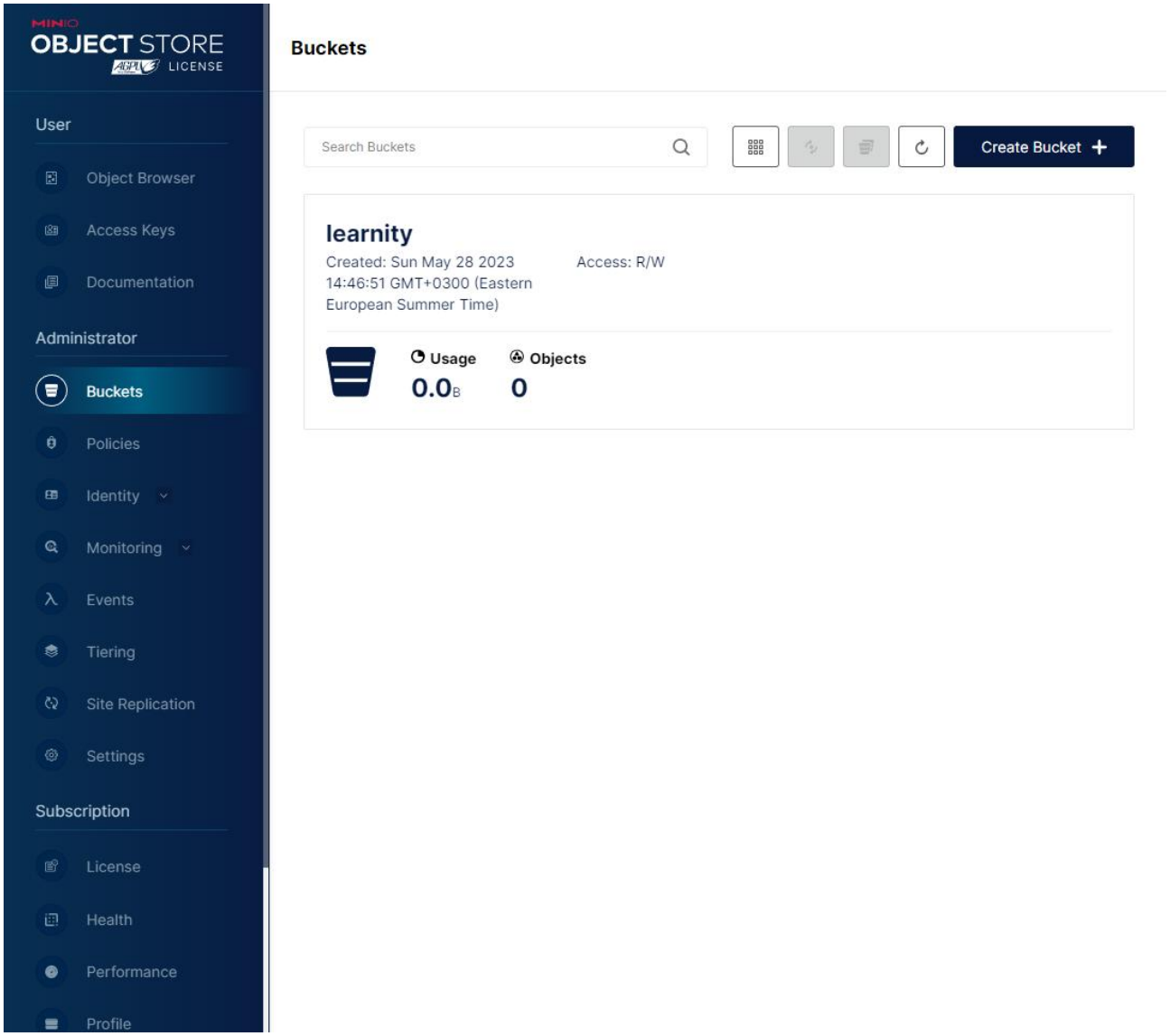

Рисунок 3.10 – Головний екран панелі адміністрування MinIO

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис основних модулів системи

Серверна частина складається з наступних модулів:

- бази даних;
- конфігурацій (взаємодія зі змінними середовища, глобальний);
- взаємодії із файловим сховищем (глобальний);
- файлів;
- авторизації;
- користувачів;
- кешу;
- курсів (курси, учасники курсу, матеріали курсу, виконані практичні завдання).

Деякі з цих модулів є глобальними. Якщо модуль є глобальним, це означає, що даний модуль можуть використовувати всі інші модулі без потреби мати пряму залежність з ним. Це також дозволяє уникнути так званих circular dependencies, котрі є звичайною проблемою фреймворку NestJS, коли один модуль залежить від іншого, а останній від першого. Це не дає системі правильно ініціалізувати програму та видає помилку. Далі наведена діаграма модулів та їхніх залежностей один з одним (Рисунок 4.1).

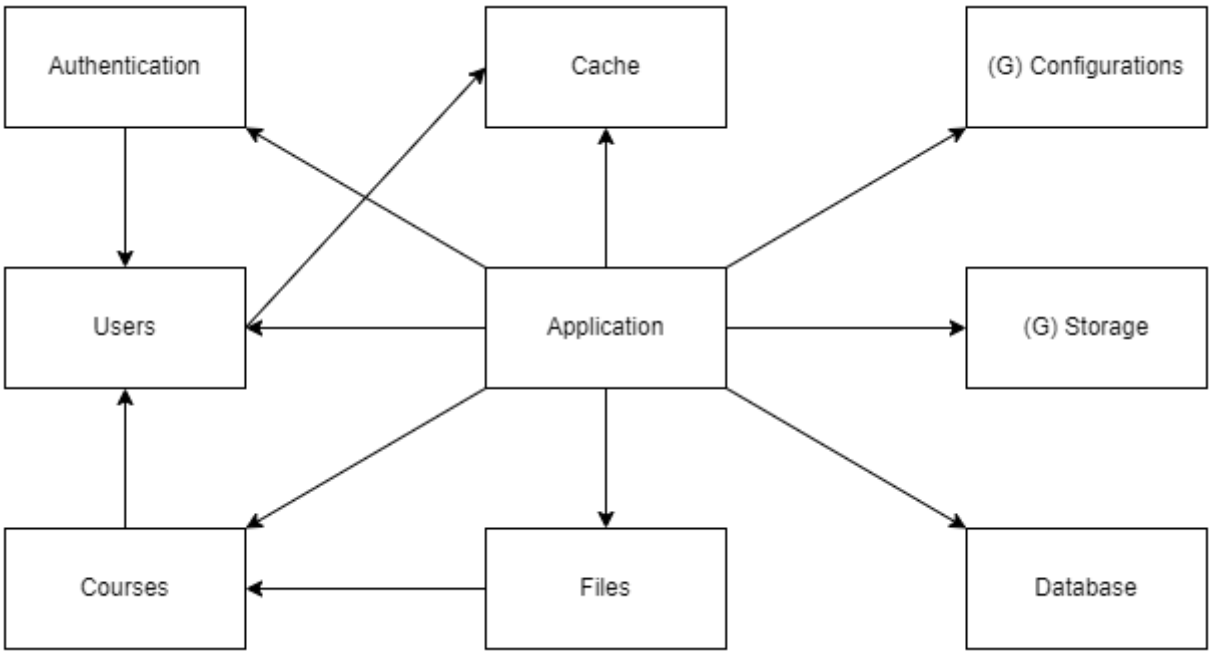

Рисунок 4.1 – Діаграма системних модулів

Кожен модуль відповідає за взаємодію із конкретною складовою системи та має у своїй структурі контролери та сервіси. Варто відзначити, що всі модулі, контролери та сервіси мають вигляд класів. Контролери відповідають за маршрутизацію вхідних запитів з API, обробку котрих потім беруть на себе сервіси модуля. Основні модулі сутностей мають набір ендпоінтів, котрі складають із себе RESTful API, за допомогою яких і йде взаємодія із ними. Кожен модуль має при собі такий набір ендпоінтів для взаємодії із сутностями:

- створення сутності
- отримання списку сутностей з можливостями сортування, вибору конкретних полів для виведення, завантаження eager relations сутності та взяття списку по сторінкам
- отримання сутності за її ідентифікатором у базі даних із тими самими можливостями, що і отримання списку сутностей
- редагування сутності (деякі поля неможливо редагувати після створення та навпаки)
- видалення сутності.

Для зручного відображення повного списку ендпоінтів кожного модулю та взаємодії з ними без використання клієнтської частини системи використовується Swagger, котрий надає зручний веб-інтерфейс з усіма необхідними для цього можливостями. Далі неведений інтерфейс Swagger, котрий був створений на основі серверного API (Рисунок 4.2).

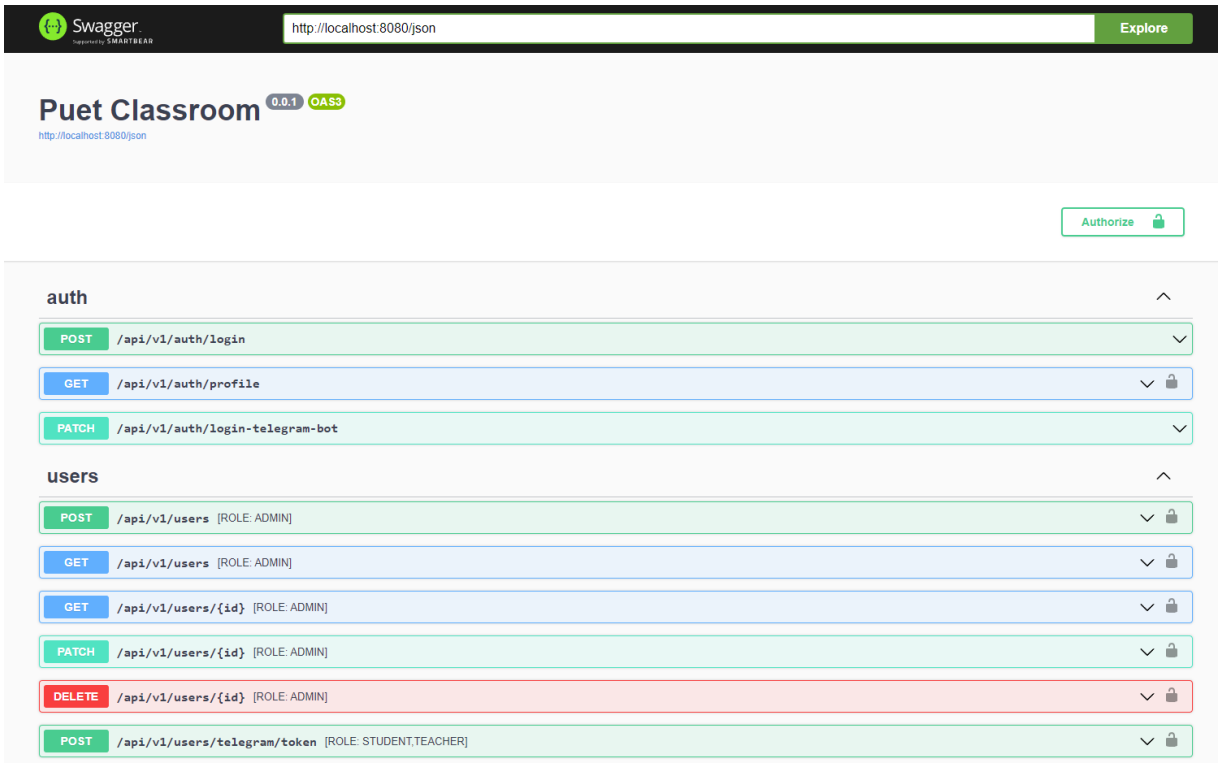

Рисунок 4.2 – Інтерфейс Swagger

Всі модулі так чи інакше пов’язані між собою і використовують можливості одне одного. Саме так сервіс може взаємодіяти з іншим сервісом для виконання необхідних завдань, при чому не залежно від того чи є ці сервіси складовою одного модулю. Проте тут треба бути обережним для того, щоб не виникла так звана circular dependencies, тобто коли модуль використовує інший модуль, а цей модуль використовує перший. При такій залежності неможливо створити об’єкти сервісів для подальшої dependency injection.

Важливими для роботи серверу є його налаштування – ті складові, які контролюють окремі його частини або зберігають критичну для його роботи інформацію. Така інформація надійно захищена та збережена у недоступному

для сторонніх осіб місці. Крім того, деякі налаштування та інформація можуть відрізнятися в залежності від того в якому режимі працює сервер (розробки або розгорнутому для користувачів). Для цього сервер використовує змінні середовища, котрі дозволять йому зберігати налаштування та критичну інформацію в безпечному місці з можливістю її розподілу.

4.2. Робота з даними

Однією з головних задач сервісу є робота з даними у СУБД. Ця робота налаштована за допомогою ORM, котра бере на себе роль створення необхідних запитів до бази даних. Цією ORM стала TypeORM, з котрою дуже зручно працювати у комбінації з обраним фреймворком NestJS. Для того щоб ORM розуміла з чим їй доведеться працювати, були створені класи, котрі будуть описувати сутності. Кожній сутності була відведена окрема таблиця у базі даних. Сутність має вигляд класу з набором полів, котрі будуть зберігатися в таблицях, та набором декораторів для кожного з полів, які надає сама ORM. За допомогою декораторів конкретні поля були описані параметрами, котрими так само можна було б описати поле у таблиці бази даних (первинний ключ, тип даних поля, чи є воно nullable або унікальним та інше). Після надання ORM класів-сутностей, вона надає об’єкти класів через які і можна працювати із даними. Класи цих об’єктів є реалізаціями паттерну «Репозиторій» котрий дозволяє нам працювати за допомогою простих абстракцій над СУБД. Також ORM має важливу особливість – вона бере на себе роль захисту від SQL ін’єкцій, котрі можуть негативно вплинути на цілісність даних у базі даних та роботу всієї системи в цілому.

Особливу увагу було приділено базі даних, у котрій зберігаються усі дані про користувачів, курси, файли та інше. Її структура знаходиться у третій нормальній формі, що вказує на те, що схема даних створена максимально оптимізовано і не має надлишкових даних. У базі даних будуть наявні наступні таблиці:

- користувачі;
- курси;
- учасники курсів;
- теми курсів;
- матеріали курсів;
- здані практичні роботи курсів;
- файли;
- міграції;
- кеш результатів запитів;
- метадані ORM.

Всі таблиці так чи інакше пов’язані між собою вторинними ключами. Таблиці міграцій, кешу результатів запитів та метадані ORM необхідні для коректної та швидкої роботи ORM, через яку проходять усі дії із базою даних. Усі таблиці містять у собі колонки для необхідних для збереження полів, котрі були описані у постановці задачі. Далі наведена візуальна схема даних (Рисунок 4.3).

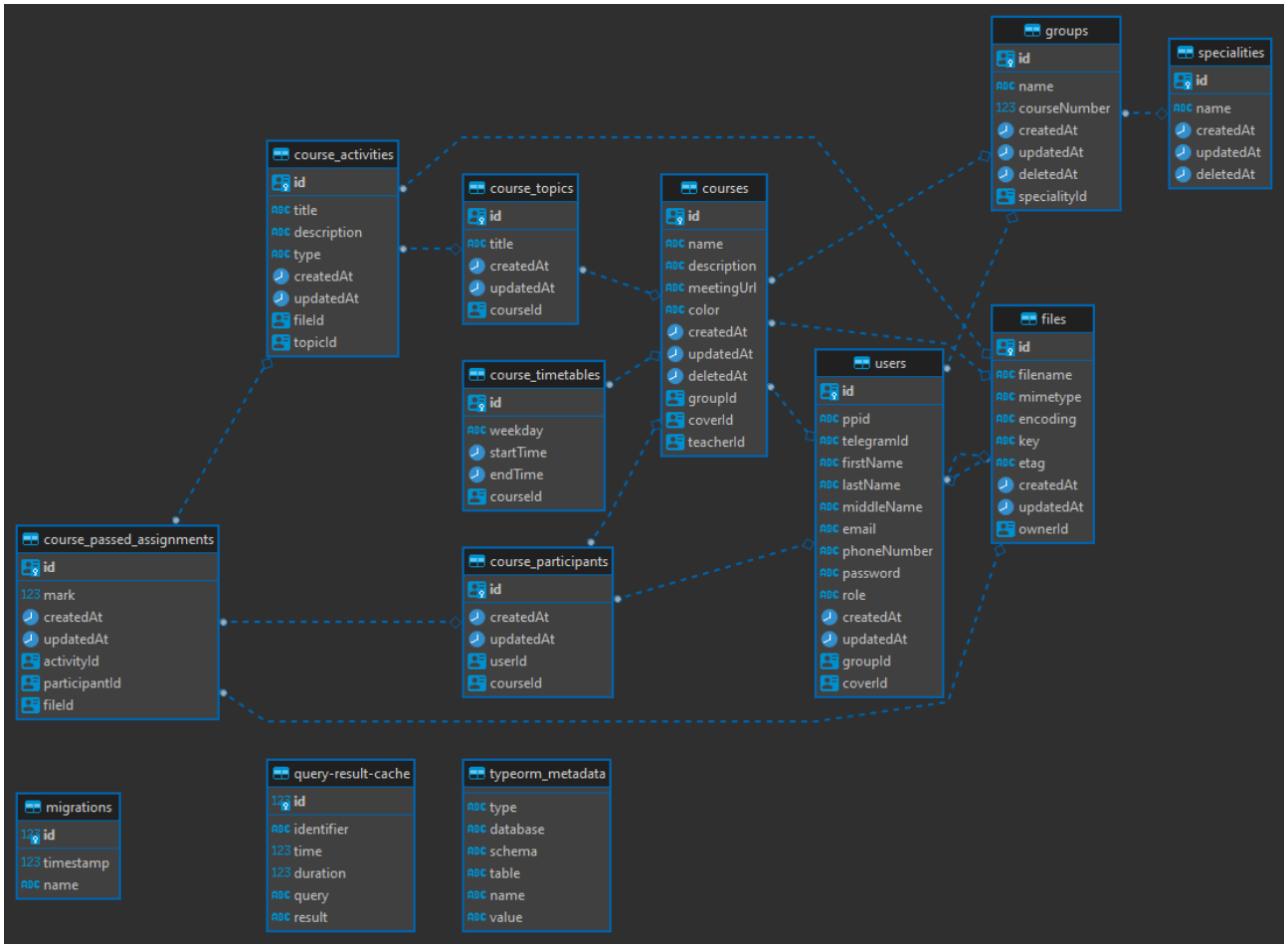

Рисунок 4.3 – Схема даних системи

4.3. Опис роботи серверного API

Невід’ємною складовою якісного та стійкого API є валідація параметрів вхідних запитів. Саме для цього були створені так звані DTO (data transfer object), які мають вигляд звичайного класу з набором полів, котрі є вхідними параметрами конкретного запиту, та набором валідаторів у вигляді декораторів. Якщо валідація була пройдена успішно, об’єкт даного класу передається далі до контролеру для подальшої роботи. Якщо дані не пройшли валідацію, сервер повертає помилку із статусом 400 та текст з поясненням чому дані не пройшли валідацію. Опис валідації вхідних параметрів реалізований за допомогою декораторів сторонньої бібліотеки class-validator, котра також допомагає генерувати повідомлення про помилки, якщо деякі з параметрів запиту не пройшли валідацію.

4.4. Опис роботи з файлами системи

Важливою складовою системи дистанційного навчання є зберігання файлів з можливістю надання контрольованого доступу до них. Це необхідний аспект системи, котрий не дає доступу до перегляду файлів тим користувачам, котрі не мають доступу до них. Хоча і немає нічого страшного в тому, щоб видавати на показ фото профілю користувача, але зовсім інше діло, коли йде річ про обмеження доступу до файлів практичних робіт, які студенти надсилають викладачам на оцінювання.

Середовищем для збереження файлів було обране Amazon S3 (Simple Storage Service), котре є одним із найпопулярніших та простих способів зберігати файли для роботи різного роду сервісів на сьогодні. Його концепт є доволі простим – файли зберігаються у вигляді так званих об’єктів, котрий складається з 3 частин: його контент (тобто файл), його унікальний ідентифікатор (unique object identifier) та метадані, котра зберігається як пари ключ-значення і містять таку інформацію, як ім’я, розмір, дата, атрибути безпеки, тип вмісту об’єкту. Самі ж об’єкти зберігаються у так званих бакетах, котрі є базовими логічними контейнерами, де зберігаються файли. Бакет не має обмежень по кількості збережених об’єктів, але сам об’єкт не може бути більшим ніж 5 ТБ за розміром. Далі наведена схема роботи з сервісом Amazon S3 (Рисунок 4.4).

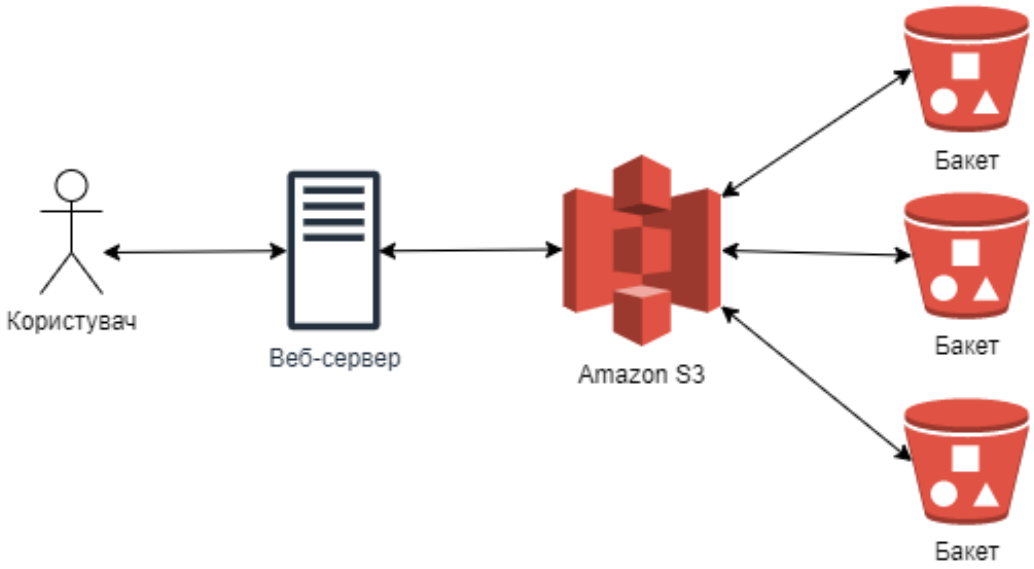

Рисунок 4.4 – Схема роботи системи з Amazon S3

Використовуючи дану технологію, була реалізована система для збереження та контрольованого доступу до файлів всієї системи. Доступ до файлів надається у вигляді підписаних посилань. За допомогою них можна надавати доступ до файлу на конкретний період часу у вигляді URL. Часом доступу до файлу був обраний проміжок в 1 годину.

ВИСНОВКИ

Використання сучасних технологій у проведенні навчального процесу в освітніх закладах будь-якого типу стало базовою складовою цієї системи, що змінило її назавжди. В ході виконання завдання мною були вдосконалені навички по розробці серверних додатків за допомогою сучасних методів та інструментів проектування та розробки. В рамках кваліфікаційної роботи поставлене завдання по розробці серверної частини для системи дистанційного навчання було виконано в повному обсязі.

Основні виконані завдання:

- сформульовані вимоги до функціональності та безпеки для розробленої системи;
- проведено огляд схожих програмних продуктів;
- описані обрані проектні рішення, підходи, технології та інструменти розробки;
- розроблено усі необхідні складові системи для її коректного функціонування з урахуванням усіх поставлених вимог;
- проведене тестування усіх модулів та доступних можливостей системи.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. About Classroom – Classroom Help [Електронний ресурс]. – Режим доступу: URL: https://support.google.com/edu/classroom/answer/6020279 – Назва з екрану.
2. Moodle App Overview [Електронний ресурс]. – Режим доступу: URL: https://moodledev.io/general/app/overview – Назва з екрану.
3. What is Canvas? [Електронний ресурс]. – Режим доступу: URL: https://community.canvaslms.com/t5/Canvas-Basics-Guide/What-is-Canvas/ta-p/45 – Назва з екрану.
4. Dan Vanderkam. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript – O'Reilly Media; 1st edition (November 12, 2019) – 264 с.
5. TypeScript Documentation [Електронний ресурс]. – Режим доступу: URL: https://www.typescriptlang.org/docs – Назва з екрану.
6. Node.js Documentation [Електронний ресурс]. – Режим доступу: URL: https://nodejs.org/uk/docs/ – Назва з екрану.
7. Node.js Architecture and 12 Best Practices for Node.js Development [Електронний ресурс]. – Режим доступу: URL: https://scoutapm.com/blog/nodejs-architecture-and-12-best-practices-for-nodejs-development – Назва з екрану.
8. Mario Casciaro, Luciano Mammino. Node.js Design Patterns – Packt Publishing; 2nd edition (July 18, 2016) – 777c.
9. PostgreSQL: Documentation [Електронний ресурс]. – Режим доступу: URL: https://www.postgresql.org/docs – Назва з екрану.
10. NestJS Documentation [Електронний ресурс]. – Режим доступу: URL: https://docs.nestjs.com – Назва з екрану.
11. TypeORM - Amazing ORM for TypeScript and JavaScript (ES7, ES6, ES5). Supports MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle,

WebSQL databases. Works in NodeJS, Browser, Ionic, Cordova and Electron platforms. [Електронний ресурс]. – Режим доступу: URL: https://typeorm.io – Назва з екрану.

12. What is Amazon S3? - Amazon Simple Storage Service [Електронний ресурс]. – Режим доступу: URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html – Назва з екрану.
13. Git [Електронний ресурс]. – Режим доступу: URL: https://git-scm.com – Назва з екрану.
14. What is Prettier? Prettier [Електронний ресурс]. – Режим доступу: URL: https://prettier.io/docs/en/index.html – Назва з екрану.
15. Documentation - ESLint - Pluggable JavaScript Linter [Електронний ресурс]. – Режим доступу: URL: https://eslint.org/docs/latest – Назва з екрану.
16. API Documentation & Design Tools for Teams | Swagger [Електронний ресурс]. – Режим доступу: URL: https://swagger.io – Назва з екрану.
17. Docker Documentation | Docker Documentation [Електронний ресурс]. – Режим доступу: URL: https://docs.docker.com – Назва з екрану.
18. MinIO | High Performance, Kubernetes Native Object Storage [Електронний ресурс]. – Режим доступу: URL: https://min.io – Назва з екрану.

ДОДАТОК А. ВИХІДНІ КОДИ

СЕРВІС КУРСІВ

```
import { ConflictException, Injectable, NotFoundException } from
'@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { instanceToPlain } from 'class-transformer';
import {
  FindConditions,
  FindManyOptions,
  FindOneOptions,
  FindOptionsUtils,
  RemoveOptions,
  Repository,
  SaveOptions,
  SelectQueryBuilder,
} from 'typeorm';

import { ErrorTypeEnum } from 'src/common/enums';
import { PaginationCoursesDto } from 'src/modules/courses/dto';
import { CourseEntity } from 'src/modules/courses/entities';
import { GroupsService } from 'src/modules/groups';
import { GroupEntity } from 'src/modules/groups/entities';
import { UserEntity, UserRoleEnum } from 'src/modules/users/entities';

@Injectable()
export class CoursesService {
  constructor(
```

```
    @InjectRepository(CourseEntity)
    private readonly courseEntityRepository: Repository<CourseEntity>,
    private readonly groupsService: GroupsService,
  ) {}
  public async createOne(
    entityLike: Partial<CourseEntity>,
    groupConditions?: Partial<GroupEntity>,
    options: SaveOptions = { transaction: false },
  ): Promise<CourseEntity> {
    return this.courseEntityRepository.manager.transaction(async () => {
      const group = await this.groupsService.selectOne(groupConditions);
      const entity = this.courseEntityRepository.create({
        ...entityLike,
        ...(group && { group }),
        cover: { owner: entityLike.teacher },
      });
      const { id } = await this.courseEntityRepository.save(entity, options).catch(() =>
{
        throw new
ConflictException(ErrorTypeEnum.COURSE_ALREADY_EXISTS);
      });

      return this.selectOne({ id }, {}, { loadEagerRelations: true });
    });
  }
  public find(
    optionsOrConditions?: FindManyOptions<CourseEntity>,
  ): SelectQueryBuilder<CourseEntity> {
    const metadata = this.courseEntityRepository.metadata;
```

```
    const qb = this.courseEntityRepository.createQueryBuilder(
      FindOptionsUtils.extractFindManyOptionsAlias(optionsOrConditions) ||
metadata.name,
    );

    if (
      !FindOptionsUtils.isFindManyOptions(optionsOrConditions) ||
      optionsOrConditions.loadEagerRelations !== false
    ) {
      FindOptionsUtils.joinEagerRelations(qb, qb.alias, metadata);

      /**
       * Place for common relation
       * @example qb.leftJoinAndSelect('CourseEntity.relation_field',
'CourseEntity_relation_field')
       */
    }

    return FindOptionsUtils.applyFindManyOptionsOrConditionsToQueryBuilder(qb,
optionsOrConditions);
  }
  public async selectMany(
    user: Partial<UserEntity>,
    options: FindManyOptions<CourseEntity> = { loadEagerRelations: false },
  ): Promise<PaginationCoursesDto> {
    const qb = this.find(instanceToPlain(options));
    if (options.where) qb.where(options.where);
    this.applyUserAccess(qb, user);
    return qb
```

```
      .getManyAndCount()
      .then((data) => new PaginationCoursesDto(data))
      .catch(() => {
        throw new NotFoundException(ErrorTypeEnum.COURSES_NOT_FOUND);
      });
  }
  public async selectOne(
    conditions: FindConditions<CourseEntity>,
    user?: Partial<UserEntity>,
    options: FindOneOptions<CourseEntity> = { loadEagerRelations: true },
  ): Promise<CourseEntity> {
    const qb = this.find(instanceToPlain(options)).where(conditions);
    if (user) this.applyUserAccess(qb, user);
    return qb.getOneOrFail().catch(() => {
      throw new NotFoundException(ErrorTypeEnum.COURSE_NOT_FOUND);
    });
  }
  public async updateOne(
    conditions: Partial<CourseEntity>,
    entityLike: Partial<CourseEntity>,
    user?: Partial<UserEntity>,
    options: SaveOptions = { transaction: false },
  ): Promise<CourseEntity> {
    return this.courseEntityRepository.manager.transaction(async () => {
      const mergeIntoEntity = await this.selectOne(conditions, user);
      const entity = this.courseEntityRepository.merge(mergeIntoEntity, entityLike);
      const { id } = await this.courseEntityRepository.save(entity, options).catch(() =>
{
```

```
        throw new
ConflictException(ErrorTypeEnum.COURSE_ALREADY_EXISTS);
      });
      return this.selectOne({ id });
    });
  }
  public async deleteOne(
    conditions: FindConditions<CourseEntity>,
    user?: Partial<UserEntity>,
    options: RemoveOptions = { transaction: false },
  ): Promise<CourseEntity> {
    return this.courseEntityRepository.manager.transaction(async () => {
      const entity = await this.selectOne(conditions, user);
      return this.courseEntityRepository.remove(entity, options).catch(() => {
        throw new NotFoundException(ErrorTypeEnum.COURSE_NOT_FOUND);
      });
    });
  }
private applyUserAccess(qb: SelectQueryBuilder<CourseEntity>, user:
Partial<UserEntity>) {
    if (user.role === UserRoleEnum.TEACHER) qb.andWhere({ teacher: user });
    if (user.role === UserRoleEnum.STUDENT) {
      qb.innerJoin(
        'CourseEntity.participants',
        'CourseEntity_participants',
        `CourseEntity_participants.userId = :userId`,
        { userId: user.id },
      );
    }
```

```
  }
}
```

СЕРВІС УЧАСНИКІВ КУРСУ

```
import { ConflictException, Injectable, NotFoundException } from
'@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { instanceToPlain } from 'class-transformer';
import {
  FindConditions,
  FindManyOptions,
  FindOneOptions,
  FindOptionsUtils,
  RemoveOptions,
  Repository,
  SaveOptions,
  SelectQueryBuilder,
} from 'typeorm';

import { ErrorTypeEnum } from 'src/common/enums';
import { PaginationCourseParticipantsDto } from 'src/modules/courses/dto';
import { CourseEntity, CourseParticipantEntity } from 'src/modules/courses/entities';
import { CoursesService } from 'src/modules/courses/services/courses.service';
import { UsersService } from 'src/modules/users';
import { UserEntity, UserRoleEnum } from 'src/modules/users/entities';

@Injectable()
export class CourseParticipantsService {
  constructor(
    @InjectRepository(CourseParticipantEntity)
```

```
    private readonly courseParticipantEntityRepository:
Repository<CourseParticipantEntity>,
    private readonly coursesService: CoursesService,
    private readonly usersService: UsersService,
  ) {}
  public async createOne(
    courseConditions: Partial<CourseEntity>,
    userConditions: Partial<UserEntity>,
    teacher?: Partial<UserEntity>,
    options: SaveOptions = { transaction: false },
  ): Promise<CourseParticipantEntity> {
    return this.courseParticipantEntityRepository.manager.transaction(async () => {
      const [course, user] = await Promise.all([
        this.coursesService.selectOne(courseConditions, teacher),
        this.usersService.selectOne(userConditions),
      ]);
      const existingUser = await this.find().where({ course, user }).getOne();
      if (existingUser)
        throw new
ConflictException(ErrorTypeEnum.COURSE_PARTICIPANT_ALREADY_EXIST
S);
      const entity = this.courseParticipantEntityRepository.create({ user, course });
      const { id } = await this.courseParticipantEntityRepository
        .save(entity, options)
        .catch(() => {
          throw new
ConflictException(ErrorTypeEnum.COURSE_PARTICIPANT_ALREADY_EXIST
S);
        });
```

```
      return this.selectOne({ id }, {}, { loadEagerRelations: true });
    });
  }
  public find(
    optionsOrConditions?: FindManyOptions<CourseParticipantEntity>,
  ): SelectQueryBuilder<CourseParticipantEntity> {
    const metadata = this.courseParticipantEntityRepository.metadata;
    const qb = this.courseParticipantEntityRepository.createQueryBuilder(
      FindOptionsUtils.extractFindManyOptionsAlias(optionsOrConditions) ||
metadata.name,
    );
    if (
      !FindOptionsUtils.isFindManyOptions(optionsOrConditions) ||
      optionsOrConditions.loadEagerRelations !== false
    ) {
      FindOptionsUtils.joinEagerRelations(qb, qb.alias, metadata);

      /**
       * Place for common relation
       * @example qb.leftJoinAndSelect('CourseParticipantEntity.relation_field',
'CourseParticipantEntity_relation_field')
       */
    }
    return FindOptionsUtils.applyFindManyOptionsOrConditionsToQueryBuilder(qb,
optionsOrConditions);
  }
  public async selectMany(
    conditions: Partial<CourseEntity>,
```

```
    user?: Partial<UserEntity>,
    options: FindManyOptions<CourseParticipantEntity> = { loadEagerRelations:
false },
  ): Promise<PaginationCourseParticipantsDto> {
    const course = await this.coursesService.selectOne(conditions, user);
    const qb = this.find(instanceToPlain(options));
    if (options.where) qb.where(options.where);
    qb.andWhere({ course });
    return qb
      .getManyAndCount()
      .then((data) => new PaginationCourseParticipantsDto(data))
      .catch(() => {
        throw new
NotFoundException(ErrorTypeEnum.COURSE_PARTICIPANTS_NOT_FOUND);
      });
  }
  public async selectOne(
    conditions: FindConditions<CourseParticipantEntity>,
    user?: Partial<UserEntity>,
    options: FindOneOptions<CourseParticipantEntity> = { loadEagerRelations: true
},
  ): Promise<CourseParticipantEntity> {
    const qb = this.find({ ...instanceToPlain(options), loadEagerRelations: true
}).where(
      conditions,
    );
    if (user?.role === UserRoleEnum.TEACHER)
      qb.andWhere('CourseParticipantEntity_course.teacherId = :teacherId', {
teacherId: user.id });
```

```
    if (user?.role === UserRoleEnum.STUDENT) qb.andWhere({ user });
    return qb.getOneOrFail().catch(() => {
      throw new
NotFoundException(ErrorTypeEnum.COURSE_PARTICIPANT_NOT_FOUND);
    });
  }
  public async deleteOne(
    conditions: FindConditions<CourseParticipantEntity>,
    user?: Partial<CourseParticipantEntity>,
    options: RemoveOptions = { transaction: false },
  ): Promise<CourseParticipantEntity> {
    return this.courseParticipantEntityRepository.manager.transaction(async () => {
      const entity = await this.selectOne(conditions, user, { loadEagerRelations: true });
      return this.courseParticipantEntityRepository.remove(entity, options).catch(() =>
{
        throw new
NotFoundException(ErrorTypeEnum.COURSE_PARTICIPANT_NOT_FOUND);
      });
    });
  }
}
```