КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Програмна система е-щоденник»

Виконав _____
(Підпис)
Гончаренко Олександр Леонідович
(прізвище, ім'я, по батькові)

Керівник _Домрачев Володимир Миколайович
(прізвище, ім'я, по батькові)

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")

Завідувач кафедри _____ Плескач В.Л. _____
(Підпис) (Прізвище, ініціали) (Дата)

Київ – 2021

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1 ст.
Відомість дипломної роботи	1 ст.
Анотація	2 ст.
Анотація (іноземною мовою-англійською)	2 ст.
Перелік умовних позначень і скорочень	1 ст.
Зміст	1 ст.
Вступ	2 ст.
1. Теоретичні основи побудови програмної системи «е-щоденник»	18 ст.
2. Проектні та технічні рішення. Проектування та створення програмної системи	18 ст.
3. Опис роботи програмної системи	14 ст.
Висновки	1 ст.
Перелік посилань	4 ст.

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.				Відомість дипломної роботи	Лист	Листів
Керівн.						
Н/контр.	Макаренко С.А.					
Зав.каф.	Плескач В.Л.					

РЕФЕРАТ (АНОТАЦІЯ)

Кваліфікаційна робота бакалавра складається з 3 розділів та містить: 65 сторінок, 35 рисунків, 50 літературних джерел. Метою кваліфікаційної роботи є створення програмної системи е-щоденник.

Кваліфікаційна робота виконана студентом – Гончаренко О.Л.

Назва роботи – Програмна система е-щоденник.

Шифр та назва спеціальності – 122 «Комп’ютерні науки».

Освітня програма – Прикладне програмування.

Актуальність кваліфікаційної роботи полягає в зростанні потреби в дистанційному та простому способі вести записи про оцінки учнів, а також в легкому інформуванні учнів про їх оцінки. Окрім виконання мети завданням також є дослідження сучасних технологій створення веб-застосунків.

Об’єкт дослідження – програмна система «Е-щоденник», її функціональні рішення, принципи та підходи до організації основної програмної системи з визначеним функціоналом.

Предмет дослідження – програмно-технічні засоби для створення веб-застосунків, що були використані для створення програмної системи.

Дана кваліфікаційна робота описує створення клієнтської та серверної частини програмної системи, а також різний спектр технологій, котрі було використано для створення прототипу програмної системи.

Було створено базу даних, для створення якої було обрано базу даних MongoDB. Клієнтська частина була виконана з використанням HTML, CSS та JS. Було застосовано JS-бібліотеку React для створення клієнтської частини у вигляді SPA, а також ряд бібліотек з React-екосистеми. Для створення серверної частини у вигляді веб-служби REST Web API було використано технологію NodeJS, фреймворк NestJS та мову програмування JavaScript. Для створення автентифікації та авторизації користувачів було використано технологію JSON Web Token (JWT токен).

Створений прототип програмної системи може бути використано на практиці та є реалізацією нового підходу до вирішення такого виду завдань. Також створений прототип має потенціал до продовження його розробки в подальшому та розширення наявного функціонала електронного щоденника до функціонала системи більш широкого застосування.

Ключові слова: програмна система, веб-застосунок, клієнт, сервер, Web API, NodeJS, NestJS, HTML, CSS, JS, TS, React, C#, JWT.

ANNOTATION

The qualification work of the bachelor consists of three sections and contains 65 pages, 35 drawings, 50 literary sources. The purpose of the qualification work is to create grade book software system.

Qualification work performed by a student – Oleksandr Honcharenko.

Title of the work – Grade book software system.

Code and name of the specialty – 122 "Computer Science".

Educational program – Applied programming.

The relevance of the qualification work is the growing need for a remote and easy way to keep records of student assessments, as well as easy information to students about their assessments. In addition to fulfilling the goal, the task is also to study modern technologies for creating web applications.

The object of research is the software system "Grade Book", its functional solutions, principles, and approaches to the organization of the main software system with certain functionality.

The subject of research – software, and hardware for creating web applications that were used to create a software system.

This qualification work describes the creation of client and server parts of the software system, as well as a variety of technologies that were used to create a prototype software system.

A database was created, for which a MongoDB database was selected. The client part was created using HTML, CSS, and JS. The React JS library was used to create the client part in the form of a SPA, as well as a number of libraries from the React ecosystem. NodeJS technology, the NestJS framework, and the JavaScript programming language were used to create the server part in the form of the REST Web API web service. JSON Web Token technology was used to create user authentication and authorization.

The created prototype of the software system can be used in practice and is the implementation of a new approach to solving this type of problem. In addition, the created prototype has the potential to continue its development in the future and expand the

existing functionality of the electronic grade book to the functionality of the system of wider application.

Keywords: software system, web application, client, server, Web API, NodeJS, NestJS, HTML, CSS, JS, TS, React, C #, JWT.

Перелік умовних позначень і скорочень

англ. – англійська

БД – база даних

СКБД – система керування базами даних

JS – JavaScript

TS – TypeScript

JWT – JSON Web Token

SPA – Single Page Application

SRS – Software Requirements Specification

npm – Node Package Manager

ЗМІСТ

ВСТУП

В наш час все частіше й частіше постає питання переносу оцінок учнів в різних сферах навчання в електронний вигляд для надання учням можливості в дистанційному режимі переглядати свої оцінки. Існує чимало шкільних електронних систем, що надають різний спектр можливостей, в тому числі виставляти оцінки учням, але при цьому більшість таких систем є або приватними, або дозволяють використовувати їх лише школам та іншим подібним установам.

Актуальність теми цієї кваліфікаційної роботи зумовлено тим, що перенесення оцінок в електронний вигляд надає учням зручний спосіб перегляду своїх оцінок. Особливо це актуально для дистанційного навчання.

Метою кваліфікаційної роботи є створення програмної системи, яка б дозволила користувачам в он-лайн режимі переглядати свої оцінки з різних курсів, за різні завдання, а також надавала можливості для створення власних класів в системі і відгравати роль викладача виставляючи оцінки учням даного класу.

Завдання дослідження. Повинні бути вирішені наступні завдання:

- Дослідити технології для створення веб-застосунків, а також визначити що саме і як повинно бути реалізовано для вирішення поставленої задачі;
- Створити базу даних, яка буде зберігати всю інформацію програмної системи, таку як інформація про користувачів, їх оцінки, списки класів, курсів, завдань;
- Спроектувати та створити серверну та клієнтську частини програмної системи.

Об’єктом дослідження є програмна система «Е-щоденник», її функціональні рішення, принципи та підходи до організації основної програмної системи з визначеним функціоналом.

Предметом дослідження є програмно-технічні засоби для створення веб-застосунків, ряд бібліотек, що будуть використані для створення програмної системи «Е-щоденник».

Методи дослідження полягають в системному аналізі та синтезі, теорії систем та застосуванні клієнт−серверної моделі архітектурного рішення досліджуваної системи, методах експериментально-теоретичного рівня для збору даних про створення веб-застосунків та веб-серверів, їх використання на практиці, узагальненні всієї дослідженої інформації.

Практичне значення полягає в демонстрації роботи прототипу програмної системи «Е-щоденник», який може бути використаний у практичній діяльності за своїм прямим призначенням, а також над яким може бути продовжена розробка для розширення функціоналу систему, що дозволить перетворити систему з електронного щоденника в певну систему автоматизованого електронного навчання.

У процесі виконання кваліфікаційної роботи були використані наступні документації використаних технологій: офіційна документація бібліотеки React, офіційна документація NodeJS, офіційна документація NestJS, офіційна документація бази даних MongoDB, документації мов програмування JavaScript та TypeScript, документації по мові розмітки HTML, мові стилів CSS, а також документації до використаних в процесі розробки різних бібліотек різного призначення. Було використано редактор коду Visual Studio Code.

Розділ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ ПРОГРАМНОЇ СИСТЕМИ «Е-ЩОДЕННИК»

1.1 Аналіз актуальності розв'язуваної задачі

В наш час багато аспектів життєдіяльності людини переходять в цифровий формат, реалізуються різні програмні системи для вирішення абсолютно різних завдань, для покращення різних буденних для нас речей. Створення програмної системи «Е-щоденник» в режимі он-лайн дозволить користувачам створювати свої власні навчальні класи, додавати до них учнів після чого виставляти їм оцінки за завдання. Окрім взяття на себе ролі викладача, користувачі зможуть приєднуватися до створених іншими користувачами класів і переглядати в цих класах виставлені їм оцінки. Наявність такої системи електронного щоденника дозволить викладачам та учням покращити процес інформування учнів оцінками, особливо у випадках коли навчання відбувається в дистанційному форматі.

Провівши пошук на просторах інтернету за існуючими системами електронних щоденників було виявлено, що більшість електронних щоденників існують як підсистеми більщ глобальних програмних систем шкільної автоматизації. Більшість таких систем є або внутрішніми (приватними) система певного навчального закладу, або вимагають, щоб користувач, який буде використовувати систему був викладачем в певному навчальному закладі. Також такі системи в основному орієнтуються на шкільні заклади та не є універсальними.

При створенні своєї системи електронного щоденника було прийнято створити систему, яка не була б націлена тільки на шкільні заклади, а була б схожа на систему навчання Google Classroom, де немає прив'язки до типу навчального закладу, де викладачем може бути будь-який користувач.

Отже, в рамках цієї кваліфікаційної роботи буде створено систему, яка не буде залежати від типу навчального закладу, дозволить бути викладачем будь-якому користувачу, а також своїм функціоналом буде нагадувати Google Classroom, але з прицілом саме на оцінки.

Програмна система буде реалізована саме у вигляді веб-застосунку, адже цей вид програмної системи найкраще підходить для реалізації такого роду задачі, оскільки користувачі повинні мати доступ до системи саме за допомогою мережі Інтернет.

1.2 Обґрунтування мети вирішення та постановка задачі

Отже ми визначили, що метою є створення веб-застосунку, який би надавав можливість створювати свої класи та виставляти оцінки учням цього класу, а також в той же час мати можливість приєднатися до інших класів в ролі учня і мати змогу там отримувати та переглядати свої оцінки.

Тепер же більш детально розглянемо конкретні аспекти нашої програмної системи, які повинні бути реалізовані. Зазвичай при проектуванні певної програмної системи створюється спеціальний технічний документ, який називається Software Requirements Specification (SRS) – Специфікація на розробку програмного забезпечення. Саме в цьому документі описуються всі вимоги до програмної системи, весь функціонал, який повинна надавати система, особливості та інтерфейси програмного забезпечення, те, що буде робити програмне забезпечення, та обмеження, за яких воно має працювати. Цей документ призначений для користувачів програмного забезпечення, команд розробників та тестувальників, які будуть реалізовувати і перевіряти коректність роботи системи. Писати даний документ в його повному обсязі немає сенсу в даній кваліфікаційній роботі, але деякі його розділи все ж таки чудово підходять для опису програмної системи, а саме – функціонал програмної системи і класи та характеристики користувачів системи.

1.2.1 Функціонал системи

Реєстрація на сайті

Опис: Користувачі повинні мати змогу реєструватися на веб-сайті для створення свого облікового запису, який дозволить в подальшому користуватися системою.

Вимоги: Інформація, вказана користувачем при реєстрації буде складатися з імені, прізвища та електронної адреси і буде достатньою для того, щоб почати користуватися системою. Реєстрація повинна бути доступна на головній сторінці сайту, а також бути доступною лише якщо користувач не увійшов в систему. Вже авторизований користувач не повинен мати доступ до сторінки реєстрації. Електронний адрес повинен мати структуру, яка відповідає структурі електронних адрес, пароль повинен складатися щонайменше з 6 символів.

Послідовність дій / відповідей:

1. **Дія:** Користувач зайшов на головну сторінку.

Відповідь системи: Надає можливість реєстрації в системі.

2. **Дія:** Користувач обирає реєстрацію.

Відповідь системи: Відкриває форму для реєстрації.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Зареєструватися».

4. **Дія:** Користувач натискає кнопку «Зареєструватися».

Відповідь системи: Реєструє нового користувача в системі.

Авторизація на сайті

Опис: Користувачі повинні мати змогу авторизуватися на веб-сайті для того, щоб почати користуватися можливостями системи використовуючи раніше створений обліковий запис користувача.

Вимоги: Інформація, вказана користувачем при авторизації буде складатися з електронної адреси та пароля. Авторизація, так само як і реєстрація, повинна бути доступна на головній сторінці сайту, а також бути доступною лише якщо користувач не увійшов в систему. Вже авторизований користувач не повинен мати доступ до сторінки авторизації. Вимоги до електронної адреси та паролю співпадають з відповідними вимогами при реєстрації.

Послідовність дій / відповідей:

1. **Дія:** Користувач зайшов на головну сторінку.

Відповідь системи: Надає можливість увійти в систему.

2. **Дія:** Користувач обирає опцію входу у систему.

Відповідь системи: Відкриває форму для входу.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Увійти».

4. **Дія:** Користувач натискає кнопку «Увійти».

Відповідь системи: Авторизує користувача для доступу до системи.

Вихід з сайту

Опис: Користувачі повинні мати змогу вийти з системи.

Вимоги: Кнопка для виходу з сайту повинна бути завжди присутня на всіх сторінках сайту.

Послідовність дій / відповідей:

1. **Дія:** Авторизований користувач натискає кнопку «Вийти».

Відповідь системи: Користувач успішно виходить з системи, сайт переадресовує користувача на головну сторінку.

Перегляд списку класів

Опис: Користувачі повинні мати змогу після авторизації у системі отримати доступ до списку класів.

Вимоги: Список класів повинен знаходитися на окремій сторінці та бути розподіленим на класи, які створив користувач, та класи в яких користувач навчається. Повинні бути присутні кнопки для створення нового класу або приєднання до вже існуючого в ролі учня.

Послідовність дій / відповідей:

1. **Дія:** Користувач увійшов до системи.

Відповідь системи: Переадресовує користувача з головної сторінки на сторінку зі списком класів.

Створення нового класу

Опис: Авторизовані користувачі повинні мати змогу створювати нові класи на сторінці зі списком всіх класів.

Вимоги: Новостворені класи повинні відразу з’являтися у списку всіх класів надаючи змогу перейти на сторінку класу. У формі створення класу повинні бути присутні наступні поля: назва класу, короткий опис класу. Максимальна кількість символів для поля назви класу – 50 символів, для короткого опису класу – 100 символів.

Послідовність дій / відповідей:

1. **Дія:** Користувач увійшов до системи.

Відповідь системи: Відображає список класів разом з кнопкою створення нового класу.

2. **Дія:** Користувач натискає кнопку «Створити клас».

Відповідь системи: Відкриває форму для створення нового класу.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Створити клас».

4. **Дія:** Користувач натискає кнопку «Створити клас».

Відповідь системи: Створює новий клас, в якому користувач стає керівником. Додає новостворений клас до списку класів.

Приєднання до класу

Опис: Авторизовані користувачі повинні мати змогу приєднуватися до класів на сторінці зі списком всіх класів. Приєднання повинно відбуватися за допомогою введення спеціального коду класу, який учні можуть дізнатися у викладача класу.

Вимоги: Форма приєднання повинна містити єдине поле – код класу. Повинне бути присутнє відображення помилок при спробі приєднатися до класу, до якого користувач уже приєднався або в якому є викладачем, а також при спробі приєднатися до класу, якого не існує.

Послідовність дій / відповідей:

1. **Дія:** Користувач увійшов до системи.

Відповідь системи: Відображає список класів разом з кнопкою приєднання до класу.

2. **Дія:** Користувач натискає кнопку «Приєднатися до класу».

Відповідь системи: Відкриває форму для приєднання до класу.

3. **Дія:** Користувач вводить код класу.

Відповідь системи: Стає доступною для натиску кнопка «Приєднатися до класу».

4. **Дія:** Користувач натискає кнопку «Приєднатися до класу».

Відповідь системи: Приєднує користувача до класу в ролі учня, оновлює список класів користувача.

Перегляд сторінки класу

Опис: Авторизовані користувачі повинні мати змогу обрати один з класів зі списку класів для того, щоб переглянути інформацію про цей клас, а також отримати доступ до певного ряду дій пов’язаних з обраним класом та роллю користувача в ньому. На сторінці класу повинна бути представлена коротка інформація про клас, а також список курсів, що відносяться до даного класу.

Вимоги: Кожен клас на сторінці зі списком класів повинен представляти собою кнопку, при натиску на яку відкриватиметься сторінка з цим класом.

Послідовність дій / відповідей:

1. **Дія:** Користувач переглядає список класів.

Відповідь системи: Відображає список класів у вигляді кнопок.

2. **Дія:** Користувач натискає на один з класів.

Відповідь системи: Відкриває сторінку обраного класу.

Перегляд сторінки зі списком користувачів класу

Опис: Користувачі класу повинні мати змогу переглянути список користувачів класу, який поділяється на викладачів та учнів.

Вимоги: Відображення кількості користувачів. Керівник класу повинен мати змогу видаляти користувачів.

Послідовність дій / відповідей:

1. **Дія:** Користувач переглядає сторінку класу.

Відповідь системи: Відображає кнопку для перегляду списку користувачів класу.

2. **Дія:** Користувач натискає на кнопку «Користувачі».

Відповідь системи: Відкриває сторінку зі списком користувачів класу.

Налаштування класу

Опис: На сторінці класу повинна бути присутня кнопка для налаштування класу. Повинна надаватися можливість перейменувати клас та змінити його короткий опис, переглянути код класу для учнів та код класу для викладачів. Повинна бути можливість видалити клас.

Вимоги: Доступ до налаштувань класу повинен мати лише керівник класу. У формі налаштування класу повинні бути присутні наступні поля: назва класу, короткий опис класу. Максимальна кількість символів для поля назви класу – 50 символів, для короткого опису класу – 100 символів. Повинна бути присутня кнопка для видалення класу.

Послідовність дій / відповідей:

1. **Дія:** Користувач перейшов на сторінку класу та є керівником цього класу.

Відповідь системи: Відображає кнопку для відкриття налаштувань класу.

2. **Дія:** Користувач натискає кнопку «Налаштування класу».

Відповідь системи: Відкриває форму для редагування класу.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Зберегти зміни».

4. **Дія:** Користувач натискає кнопку «Зберегти зміни».

Відповідь системи: Сторінка класу оновлюється з новою інформацією про клас.

Створення нового курсу в класі

Опис: Користувач, який є викладачем класу, повинен мати змогу створювати нові курси в цьому класі. Курси можуть бути приватними, тобто доступні лише до обраних викладачем учнів класу. Учні можуть мати оцінку за курс, а тому при створені курсу викладач повинен обрати спосіб вирахування автоматичної оцінки

за курс. Згодом, на сторінці оцінок за курс, викладач курсу зможе переглянути автоматично вирахувані оцінки учнів за цей курс і виставити підсумкові оцінки, що мають більший пріоритет аніж автоматичні.

Вимоги: Новостворені курси повинні відразу з’являтися у списку всіх курсів класу, надаючи змогу перейти на сторінку курсу. У формі створення нового курсу повинні бути присутні наступні поля: назва курсу, викладач цього курсу, спосіб вирахування оцінки, приватний курс, студенти приватного курсу. Максимальна кількість символів для поля назви курсу – 50 символів. Поле «приватний курс» повинно бути у вигляді перемикача, включивши який повинне відобразитися поле «студенти приватного курсу» у вигляді таблиці зі студентами класу. Поле вибору викладача курсу повинно мати вигляд випадного списку, який надає можливість обрати користувач серед викладачів класу.

Послідовність дій / відповідей:

1. **Дія:** Користувач перейшов на сторінку класу.

Відповідь системи: Відображає список курсів разом з кнопкою створення нового курсу на сторінці класу.

2. **Дія:** Користувач натискає кнопку «Новий курс».

Відповідь системи: Відкриває форму для створення нового курсу.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Створити курс».

4. **Дія:** Користувач натискає кнопку «Створити курс».

Відповідь системи: Створює новий курс в класі. Додає новостворений курс до списку курсів класу.

Перегляд сторінки курсу

Опис: Авторизовані користувачі повинні мати змогу обрати один з курсів зі списку курсів обраного класу, щоб переглянути інформацію про цей курс, а також отримати доступ до певного ряду дій пов’язаних з обраним курсом та роллю користувача в обраних класі та курсі. На сторінці курсу повинна бути представлена

коротка інформація про курс, а також список завдань, що відносяться до даного курсу, та за які будуть виставлятися оцінки учням.

Вимоги: Кожен курс зі списку курсів повинен представляти собою кнопку, при натиску на яку відкриватиметься сторінка з обраним курсом. Інформація про курс повинна містити наступну інформацію: назва курсу, дата створення, викладач курсу, спосіб вирахування оцінки за курс. Якщо користувач, що відкрив сторінку курсу є викладачем цього курсу, то для нього також повинні відображатися кнопки для відкриття налаштувань курсу та для відкриття сторінки з оцінками учнів цього курсу. Якщо ж користувач є учнем цього курсу, то для нього повинні відображатися додаткові дані, такі як автоматично вирахувана оцінка за цей курс та підсумкова оцінка за курс виставлена викладачем цього курсу.

Послідовність дій / відповідей:

1. **Дія:** Користувач переглядає список курсів на сторінці класу.

Відповідь системи: Відображає список курсів у вигляді кнопок.

2. **Дія:** Користувач натискає на один з курсів.

Відповідь системи: Відкриває сторінку обраного курсу.

Налаштування курсу

Опис: На сторінці курсу повинна бути присутня кнопка для налаштування курсу. Повинна надаватися можливість перейменувати курс, змінити викладача цього курсу, змінити спосіб вирахування автоматичної оцінки курсу, змінити статус курсу з приватного на не приватний і навпаки, додати або видалити учнів у випадку, коли курс приватний. Повинна бути можливість видалити курс.

Вимоги: Доступ до налаштувань курсу повинен мати лише керівник класу. У формі налаштування курсу повинні бути присутні наступні поля: назва курсу, викладач цього курсу, спосіб вирахування оцінки, приватний курс, список студентів приватного курсу. Вимоги до полів форми редагування курсу повинні співпадати з відповідними вимогами до полів форми створення курсу. Повинна бути присутня кнопка для видалення курсу.

Послідовність дій / відповідей:

1. **Дія:** Користувач знаходиться на сторінці курсу та є керівником класу, до якого відноситься обраний курс.

Відповідь системи: Відображає кнопку для відкриття налаштувань курсу на панелі з інформацією про курс.

2. **Дія:** Користувач натискає кнопку «Налаштування курсу».

Відповідь системи: Відкриває форму для редагування курсу.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Зберегти зміни».

4. **Дія:** Користувач натискає кнопку «Зберегти зміни».

Відповідь системи: Сторінка класу та сторінка курсу оновлюються у відповідності з новою інформацією про курс.

Оцінки курсу

Опис: На сторінці курсу повинна бути присутня кнопка для перегляду оцінок учнів курсу. На цій сторінці викладач зможе переглянути список учнів, які мають доступ до курсу, автоматично вирахувані оцінки цих учнів на основі їх оцінок за всі завдання курсу, а також матиме змогу виставити підсумкові оцінки учнів за курс. Автоматично вирахувані оцінки не є підсумковими, а лише допомагають викладачам виставляти оцінки за курс, пропонуючи оцінки, які вираховуються у відповідності з обраними викладачем курсу налаштуваннями вирахування оцінки.

Вимоги: Доступ до сторінки з оцінками курсу повинен мати лише викладач цього курсу. Виставляти оцінки також повинен мати змогу лише викладач курсу. Список учнів повинен мати вигляд таблиці з трьома стовпчиками: студент, вирахувана оцінка, підсумкова оцінка. Останній стовпчик повинен мати опцію редагування для надання викладачам курсу відповідного функціоналу зміни оцінки учня.

Послідовність дій / відповідей:

1. **Дія:** Користувач знаходиться на сторінці курсу та є викладачем цього курсу.

Відповідь системи: Відображає кнопку для відкриття сторінки з оцінками курсу на панелі з інформацією про курс.

2. **Дія:** Користувач натискає кнопку «Оцінки курсу».

Відповідь системи: Відкриває сторінку з оцінками учнів за курс.

3. **Дія:** Користувач змінює оцінку певного учня.

Відповідь системи: Система автоматично зберігає в базі даних нову оцінку за курс для обраного учня.

Перегляд списку завдань курсу

Опис: Користувачі при відкриті сторінки курсу повинні мати змогу переглянути загальний список завдань курсу. Цей список повинен поділятися на завдання на рівні курсу та на групи завдань, також на рівні курсу. Саме оцінки за ці завдання та групи повинні використовуватися при вирахувані оцінок за курс. Групи завдань можуть поділятися на завдання групи та на підгрупи завдань. Підгрупи завдань можуть поділятися на завдання підгрупи та на модулі завдань. Модулі завдань, в свою чергу, можуть містити лише завдання. Оцінки за групу, підгрупу та модуль вираховуються на основі сутностей, які знаходяться всередині них. При цьому, наприклад, оцінки за завдання в групі мають рівнозначну силу з оцінками за підгрупи, тобто, при вирахувані оцінки за групу, підгрупи повинні розглядатися як звичайні завдання.

Вимоги: Користувач повинен бачити список завдань, де кожне завдання відображається з вказанням назви цього завдання, дати виконання завдання, а також кнопкою для перегляду оцінок за це завдання, якщо користувач викладач курсу, або з вказанням оцінки за завдання, якщо користувач є учнем курсу. Якщо завдання приватне, то біля назви цього завдання повинна бути присутня відповідна відмітка, яка дасть змогу користувачам зрозуміти, що це завдання є приватним. Для груп, підгруп та модулів відображення повинно бути однаковим та мати вигляд розкривного блоку, в якому міститиметься список завдань та підгруп або модулів обраної сутності. Цей розкривний блок повинен мати заголовок, який міститиме тип сутності, назву сутності, кнопки для відкриття оцінок за цю сутність та

відкриття налаштувань сутності, якщо користувач є викладачем цього курсу, або відображення оцінки за сутність разом з відображенням автоматично вирахуваної оцінки за цю сутність, якщо користувач є учнем курсу. Також в заголовку розкривного блоку сутності повинні бути присутні кнопки для створення завдань всередині обраної сутності, а також для створення нової сутності всередині (для модулів повинна бути відсутня, оскільки модулі будуть містити лише завдання). Зверху над списком завдань та груп курсу повинні бути присутні кнопки для створення завдань та груп на рівні курсу.

Послідовність дій / відповідей:

1. **Дія:** Користувач перейшов на сторінку курсу.

Відповідь системи: Відображає список завдань та груп обраного курсу.

2. **Дія:** Користувач розкриває одну з груп, підгруп, модулів.

Відповідь системи: Розкриває групу, підгрупу або модуль і відображає список завдань та підгруп або модулів вибраної сутності.

Створення нового завдання

Опис: Користувач, який є викладачем обраного ним курсу, повинен мати змогу створювати нові завдання в цьому курсі. Завдання можуть бути приватними, тобто доступними лише для обраних викладачем учнів курсу. Учні можуть мати оцінки за завдання. Завдання можна створювати як на рівні курсу так і всередині групи, підгрупи або модуля.

Вимоги: Новостворені завдання повинні відразу з’являтися у списку всіх завдань курсу, надаючи змогу викладачам виставляти оцінки за ці завдання, а учням переглядати оцінки. У формі створення нового завдання повинні бути присутні наступні поля: назва завдання, дата завдання, приватне завдання, студенти приватного завдання. Максимальна кількість символів для поля назви завдання – 50 символів. Поле «приватне завдання» повинно бути у вигляді перемикача, включивши який повинне відобразитися поле «студенти приватного завдання» у вигляді таблиці зі студентами класу.

Послідовність дій / відповідей:

1. **Дія:** Користувач перейшов на сторінку курсу та є викладачем цього курсу.

Відповідь системи: Відображає список завдань та груп обраного курсу разом з кнопками для створення нових завдань.

2. **Дія:** Користувач натискає кнопку «Створити завдання» на рівні курсу або всередині групи, підгрупи або модуля.

Відповідь системи: Відкриває форму для створення нового завдання.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Створити завдання».

4. **Дія:** Користувач натискає кнопку «Створити завдання».

Відповідь системи: Створює нове завдання або на рівні курсу або всередині групи, підгрупи, модуля. Відбувається оновлення списку завдань.

Редагування завдання

Опис: На сторінці курсу у списку завдань для кожного завдання повинна бути присутня кнопка для редагування завдання. Повинна надаватися можливість перейменувати завдання, змінити дату проведення завдання, змінити статус завдання з приватного на не приватне і навпаки, додати або видалити учнів у випадку, коли завдання є приватним. Повинна бути можливість видалити завдання.

Вимоги: Доступ до редагування завдань повинен мати лише викладач курсу, в якому знаходяться ці завдання. У формі редагування завдання повинні бути присутні наступні поля: назва завдання, дата завдання, приватне завдання, список студентів приватного завдання. Вимоги до полів форми редагування завдання повинні співпадати з відповідними вимогами до полів форми створення завдання. Повинна бути присутня кнопка для видалення завдання.

Послідовність дій / відповідей:

1. **Дія:** Користувач знаходиться на сторінці курсу, є викладачем цього курсу та переглядає список завдань.

Відповідь системи: Відображає кнопки для відкриття редагування завдання біля кожного з завдань курсу та завдань всередині групи, підгрупи або модуля.

2. **Дія:** Користувач натискає кнопку «Редагувати завдання».

Відповідь системи: Відкриває форму для редагування завдання.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Зберегти зміни».

4. **Дія:** Користувач натискає кнопку «Зберегти зміни».

Відповідь системи: Сторінка курсу зі списком завдань оновлюється і завдання, що редагувалося, відображається з новою інформацією.

Оцінки за завдання

Опис: На сторінці курсу у списку завдань для кожного завдання повинна бути присутня кнопка для перегляду оцінок учнів курсу за відповідне завдання. На цій сторінці викладач зможе переглянути список учнів, які мають доступ до завдання, а також матиме змогу виставити оцінки учням за завдання. Якщо завдання є приватним, то учнями завдання можуть бути тільки ті учні, які були вказані як учні цього приватного завдання при створені або редагуванні завдання.

Вимоги: Доступ до сторінки з оцінками за завдання повинен мати лише викладач курсу, до якого відноситься це завдання. Виставляти оцінки також повинен мати змогу лише викладач курсу. Список учнів повинен мати вигляд таблиці з двома стовпчиками: студент, оцінка. Останній стовпчик повинен мати опцію редагування для надання викладачам курсу відповідного функціоналу зміни оцінки учня за завдання. Для приватних завдань таблиця повинна містити лише учнів, які були додані до завдання, тобто приватні завдання, які не були назначені певному учню, не повинні в тому числі відображатися для учня у списку завдань та не враховуватися, відповідно, при вирахувані автоматичних оцінок.

Послідовність дій / відповідей:

1. **Дія:** Користувач знаходиться на сторінці курсу, є викладачем цього курсу та переглядає список завдань курсу.

Відповідь системи: Відображає кнопку для відкриття сторінки з оцінками за завдання для кожного з завдань у списку.

2. **Дія:** Користувач натискає кнопку «Оцінки» одного з завдань, що знаходиться біля відповідного завдання.

Відповідь системи: Відкриває сторінку з оцінками учнів за завдання.

3. **Дія:** Користувач змінює оцінку певного учня.

Відповідь системи: Система автоматично зберігає в базі даних нову оцінку за завдання для обраного учня.

Створення нових груп, підгруп та модулів

Опис: Користувач, який є викладачем обраного ним курсу, повинен мати змогу створювати нові групи, підгрупи та модулі в цьому курсі.

Вимоги: Новостворені групи, підгрупи та модулі повинні відразу з’являтися у списку всіх завдань курсу, надаючи змогу викладачам виставляти оцінки за ці групи, підгрупи та модулі, а учням переглядати автоматичні та підсумкові оцінки за них. У формах створення нових груп, підгруп та модулів повинні бути присутні наступні поля: назва групи, підгрупи або модуля, спосіб вирахування оцінки групи, підгрупи або модуля. Максимальна кількість символів для поля назви – 50 символів. Поле способу вирахування оцінки групи, підгрупи або модуля повинно бути у вигляді випадного списку, та містити дві можливі опції: середнє арифметичне всіх оцінок, сума всіх оцінок.

Послідовність дій / відповідей:

1. **Дія:** Користувач перейшов на сторінку курсу та є викладачем цього курсу.

Відповідь системи: Відображає список завдань та груп обраного курсу разом з кнопками для створення нових груп, підгруп та модулів.

2. **Дія:** Користувач натискає кнопку для створення групи на рівні курсу, підгрупи всередині групи або модуля всередині підгрупи.

Відповідь системи: Відкриває форму для створення нової групи, підгрупи або модуля.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка створення групи, підгрупи або модуля.

4. **Дія:** Користувач натискає кнопку створення групи, підгрупи або модуля.

Відповідь системи: Створює нову групу, підгрупу або модуль. Відбувається оновлення списку завдань та груп на сторінці курсу.

Редагування груп, підгруп та модулів

Опис: На сторінці курсу у списку завдань для кожної групи, підгрупи та модуля повинна бути присутня кнопка для редагування відповідної сутності. Повинна надаватися можливість перейменувати групу, підгрупу та модуль, змінити спосіб вирахування автоматичної оцінки. Повинна бути можливість видалити групи, підгрупи та модулі.

Вимоги: Доступ до редагування груп, підгруп та модулів повинен мати лише викладач курсу, в якому знаходяться ці сутності. У формах редагування повинні бути присутні наступні поля: назва групи, підгрупи або модуля, спосіб вирахування оцінки групи, підгрупи або модуля. Вимоги до полів форм редагування повинні співпадати з відповідними вимогами до полів форм створення груп, підгруп та модулів. Повинна бути присутня кнопка для видалення груп, підгруп та модулів.

Послідовність дій / відповідей:

1. **Дія:** Користувач знаходиться на сторінці курсу, є викладачем цього курсу та переглядає список завдань та груп.

Відповідь системи: Відображає кнопки для відкриття редагування групи, підгрупи або модуля біля кожної з сутностей.

2. **Дія:** Користувач натискає відповідну кнопку для редагування однієї з сутностей.

Відповідь системи: Відкриває форму для редагування вибраної сутності.

3. **Дія:** Користувач заповнює всі поля форми.

Відповідь системи: Стає доступною для натиску кнопка «Зберегти зміни».

4. **Дія:** Користувач натискає кнопку «Зберегти зміни».

Відповідь системи: Сторінка курсу зі списком завдань та груп оновлюється і група, підгрупа чи модуль, що редагувалися, відображається з новою інформацією.

Оцінки за групи, підгрупи та модулі

Опис: На сторінці курсу у списку завдань та груп для кожної групи, підгрупи та модуля повинна бути присутня кнопка для перегляду оцінок учнів курсу за відповідну сутність. На цих сторінках викладач зможе переглянути список учнів, а також матиме змогу виставити оцінки учням за обрану групу, підгрупу або модуль.

Вимоги: Доступ до сторінок з оцінками повинен мати лише викладач курсу, до якого відносяться сутності для яких переглядаються оцінки. Виставляти оцінки також повинен мати змогу лише викладач курсу. Список учнів повинен мати вигляд таблиці з трьома стовпчиками: студент, вирахувана оцінка, підсумкова оцінка. Останній стовпчик повинен мати опцію редагування для надання викладачам курсу відповідного функціоналу зі зміни оцінок учнів за групу, підгрупу чи модуль.

Послідовність дій / відповідей:

1. **Дія:** Користувач знаходиться на сторінці курсу, є викладачем цього курсу та переглядає список завдань та груп курсу.

Відповідь системи: Відображає кнопки для відкриття сторінок з оцінками за групу, підгрупу або модуль для кожної з цих сутностей у списку завдань та груп.

2. **Дія:** Користувач натискає кнопку «Оцінки» однієї з сутностей, що знаходиться біля відповідної сутності.

Відповідь системи: Відкриває сторінку з оцінками учнів для обраної сутності.

3. **Дія:** Користувач змінює оцінку певного учня.

Відповідь системи: Система автоматично зберігає в базі даних нову оцінку учня за групу, підгрупу або модуль.

1.2.2 Класи та характеристики користувачів системи

Загалом створювана система буде мати в системі лише один тип користувачів з технічної точки зору. Тобто, всі користувачі в базі даних будуть зберігатися разом і в них не буде ніякого поля ролі, яке б визначало їх роль у системі. Таке рішення було прийнято у зв’язку з тим, що в системі будь-який користувач може як створювати власні класи і бути викладачем, так і приєднуватися до класів

створених іншими користувачами і відігравати роль учня. Тобто, виходить так, що роль користувача визначається не для всієї системи в цілому, а тільки для конкретного класу в системі. Отже, в рамках певного класу можна розрізняти наступні класи користувачів:

- **Керівник клас**у – користувач, що створив клас. Може створювати курси в класі та редагувати їх, має доступ до коду класу, що надає можливість запрошувати учнів до класу. Також може додавати до класу інших користувачів в ролі викладачів і назначати їх викладачами курсів класу;
- **Викладач курсу** – користувач, який був назначений викладачем певного курсу. Не може редагувати курс в якому викладає, але при цьому має виключне право на створення завдань, груп, підгруп та модулів в курсі, який викладає, а також тільки викладач курсу може ставити оцінки учням відповідного курсу. Викладачем курсу може бути керівник класу, якщо він назначить себе викладачем курсу;
- **Учень** – користувач, що приєднався до класу, до якого був запрошений за допомогою спеціального коду класу. Може переглядати свої оцінки за курс, групи, підгрупи, модулі та завдання. Не може бачити оцінки інших учнів та не може бачити приватні завдання, на які не був назначений.

В одному й тому ж класі користувач не може бути одночасно і учнем і викладачем. Незареєстровані користувачі не можуть отримати доступ до функціоналу системи поки не пройдуть процедуру реєстрації.

1.3 Висновки до розділу

Отже, з даного розділу було отримано розгляд питання актуальності теми кваліфікаційної роботи, її мета, а також ми отримали спланований функціонал програмної системи, який буде реалізовуватися і становить собою новий підхід до вирішення поставленої задачі.

Розділ 2. ПРОЕКТНІ ТА ТЕХНІЧНІ РІШЕННЯ. ПРОЕКТУВАННЯ ТА СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ

В попередньому розділі було визначено обсяг функціональних можливостей програмної системи, який потрібно реалізувати. В цьому розділі буде описано процес створення програмної системи «Е-щоденник», реалізацію описаного в попередньому розділі функціоналу за допомогою технологій, мов програмування, фреймворків та бібліотек, що також будуть розглянуті в рамках даного розділу.

Для реалізації програмної системи обрано клієнт-серверну архітектуру, яка найкраще підходить для створення програмних систем у вигляді веб-застосунків, у вигляді, в якому буде реалізовуватися програмна система електронного щоденника.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти [5]:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів [5].

Клієнт, або клієнтську частину дуже часто ще називають – Frontend, а сервер, або серверну частину – Backend. Створювана система буде складатися якраз з цих двох частин, а також з бази даних, яка буде розташовуватися в хмарних сервісах.

Frontend – це клієнтська частина веб-застосунку, тобто та частина, з якою безпосередньо взаємодіють користувачі системи, має вигляд певного графічного

інтерфейсу. Для створення клієнтських частин програмних систем майже завжди використовують мову розмітки HTML, мову стилів сторінок CSS та мову програмування JavaScript. Разом вони створюють досить гнучкий та повний набір для створення веб-застосунків. Але в наш час на так часто веб-застосунки створюються лише за допомогою вище вказаних мов. Все частіше й частіше використовуються різні набори фреймворків та бібліотек, які допомагають тим чи іншим чином покращити процес створення веб-застосунків. Пізніше ми розглянемо які існують технології для розробки клієнтських частин на основі тих, які були обрані для розробки даної програмної системи.

Backend – це серверна частина веб-застосунку, тобто та частина програмної системи яка недоступна для користувачів веб-застосунку, оскільки не має ніякого графічного інтерфейсу і знаходиться на певному фізичному сервері, який може бути розташований далеко від користувачів, які через клієнтську частину взаємодіють з ним. Серверна частина приймає запити від клієнтських частин та певним чином оброблює інформацію, що надійшла, працює з наявною базою даних (надалі – БД) та повертає клієнтам інформацію, яку було опрацьовано та витягнуто з БД. Серверна частина містить в собі БД, де зберігається вся інформація програмної системи, а також програмне рішення для моделювання та контролю даних з бази даних з метою надання доступу до них клієнтським частинам. При цьому не завжди серверна частина напряму містить в собі БД. Інколи БД може бути розташовано на іншому фізичному сервері або в хмарних сервісах. Програмуватися серверна частина може різними мовами програмування, а дані, що використовуються програмною системою можуть зберігатися в різних БД. Зазвичай для адміністрування БД використовуються так званні системи керування базами даних (надалі – СКБД). У випадку використання БД в хмарних сервісах СКБД може бути представлена у вигляді веб-застосунку, що надає доступ до відповідного хмарового сервісу збереження та обробки даних в БД.

Отже створювану програмну систему буде розділено на три умовні частини: клієнтську частину, серверну частину та хмарову БД. Обрано саме хмарову БД, оскільки у випадку обраної БД хмарові сервіси роблять роботу з БД більш легкою

та зручною. Детальніше про обрану БД буде розказано в одному з наступних підрозділів. Також окремо буде розглянуто обрані технології для клієнтської та серверної частин. Нижче наведено схематичне представлення обраної архітектури програмної системи.

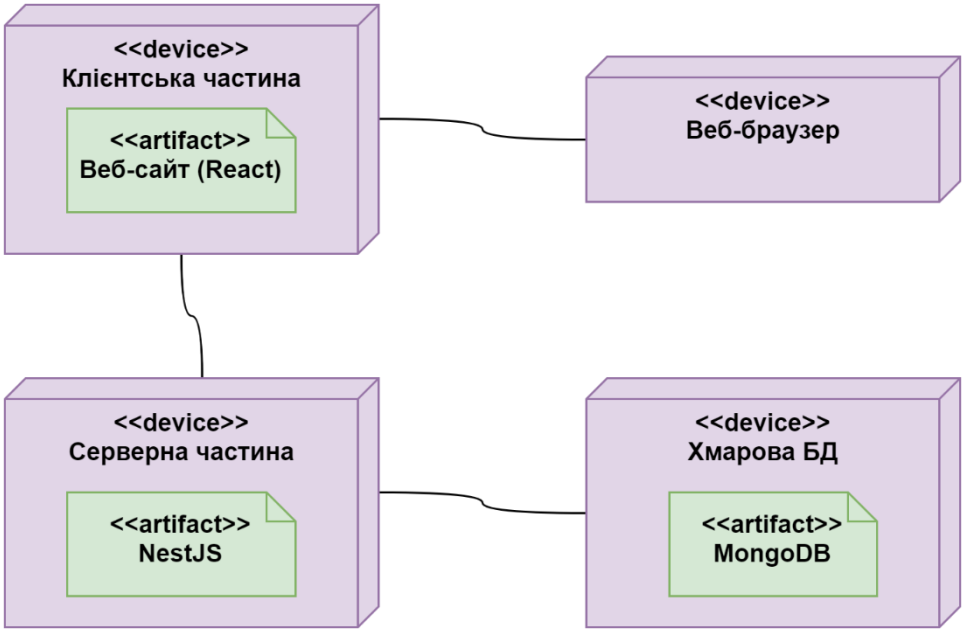

Рисунок 2.1 – Схематичне представлення архітектури програмної системи

2.1 Створення бази даних

База даних (англ. database) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами. В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім саме даних, містить їх опис та може містити засоби для їх обробки [6].

Система керування базами даних (СКБД, англ. Database Management System, DBMS) – це система, заснована на програмних та технічних засобах, яка забезпечує визначення, створення, маніпулювання, контроль, керування та використання баз

даних. Застосунки для роботи з базою даних можуть бути частиною СКБД або автономними. Найпопулярнішими СКБД є MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase, Interbase, Firebird та IBM DB2. СКБД дозволяють ефективно працювати з базами даних, обсяг яких робить неможливим їх ручне опрацювання [6].

Для створення БД було прийняте рішення обрати документ-орієнтовану NoSQL БД під назвою MongoDB. Дана БД не тільки може бути розгорнута на власній машині чи сервері, а також може бути розміщена в хмаровому сервісі. Для цих цілей існує спеціальний офіційний хмаровий сервіс – MongoDB Atlas.

Зазвичай при розробці програмних систем використовують так звані реляційні БД, в яких дані представлені у вигляді таблиць з записами. В таких таблицях стовпчики відіграють роль властивостей певної сутності, а рядки – сама сутність зі значеннями властивостей під кожний стовпчик. При цьому схеми БД розробляються на рівні самої БД незалежно від програми, яка буде в подальшому взаємодіяти з цією БД. Для взаємодії з реляційними БД використовується спеціальна мова структурованих запитів SQL.

Поглиблюватися в реляційні БД ми не будемо, оскільки, як було сказано вище, було обрано БД MongoDB, яка є протилежністю реляційним БД. MongoDB реалізує зовсім новий підхід до розробки БД без використання таблиць, які притаманні реляційним БД. Також MongoDB не використовує мову структурованих запитів SQL.

MongoDB пропонує розробникам зберігати дані в БД у вигляді так званих документів. За своєю структурою ці документи дуже сильно нагадують поширений в сфері веб-розробки формат даних JSON (JavaScript Object Notation), але насправді в MongoDB створений власний формат для збереження документів – BSON (Binary JSON). Тобто, MongoDB зберігає документи, в той час як реляційні бази зберігають рядки в таблицях. Замість таблиць же в MongoDB присутні так звані колекції. Як можна зрозуміти з назви, колекції створені для того, щоб об’єднувати документи за певним принципом. Наприклад, колекція документів, які представляють собою користувачів системи або колекція документів, які представляють собою завдання

або групу завдань. При цьому документи в MongoDB можуть мати поля які будуть не простими числами чи строками, а повноцінними документами. Тобто, документи можуть містити в собі інші документи. Також варто відзначити, що хоч колекції в MongoDB і виконують роль об’єднувача документів, але при цьому документи в колекції не обов’язково повинні мати однакову структуру.

Варто зазначити, що хоч колекції і можуть містити документи з різною структурою, в рамках розробки даної програмної системи всі колекції, які були створені, містять документи з однаковою структурою. Це робить роботу з БД та системою більш зручною та передбачуваною. При цьому структури всіх колекцій БД були створені на програмному рівні, а не на рівні БД. Детальніше цей процес буде описаний при описі серверної частини.

Нижче відображено загальний вигляд СКБД, яку пропонує хмаровий сервіс MongoDB Atlas. Саме цей хмаровий сервіс було використано для створення БД для програмної системи, та саме до БД в цьому сервісі підключається серверна частина розробленої системи.

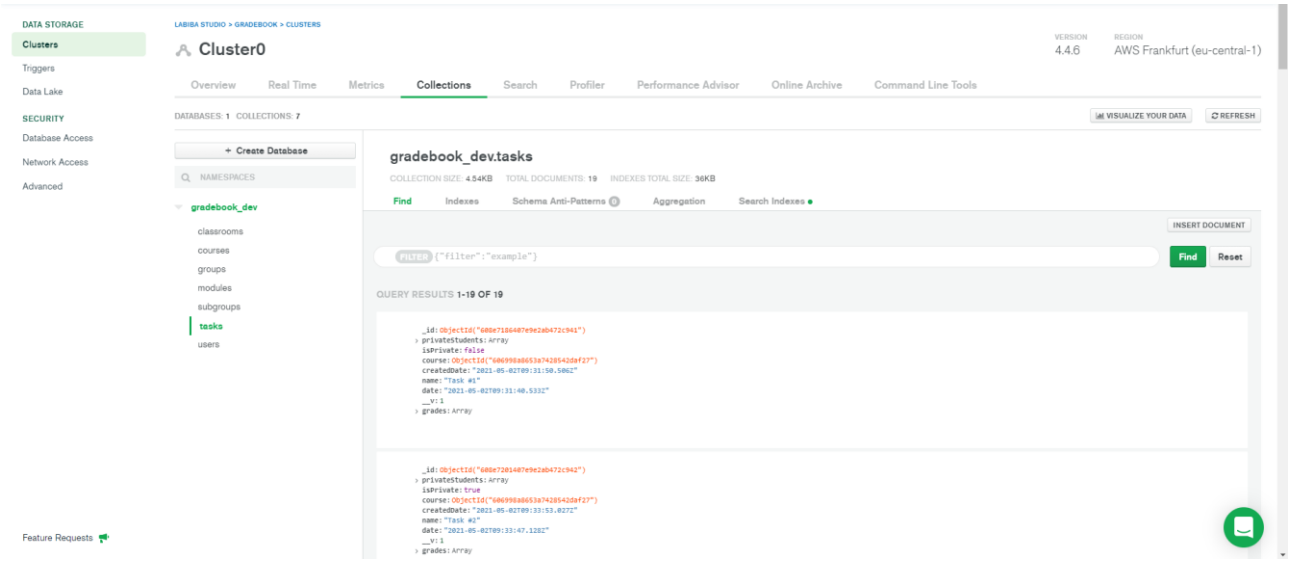

Рисунок 2.2 – Загальний вигляд СКБД MongoDB Atlas

Коротко розглянемо створені колекції:

- classrooms – містить класи, створені користувачами;
- courses – містить інформацію про курси класів;

- groups – містить інформацію про групи курсів;
- subgroups – містить інформацію про підгрупи груп;
- modules – містить інформацію про модулі підгруп;
- tasks – містить інформацію про всі завдання в системі;
- users – містить інформацію про всіх користувачів системи.

Для оцінок не створювалося колекції. Оцінки зберігаються в якості вбудованих документів всередині таких документів як завдання, група, підгрупа, модуль та курс.

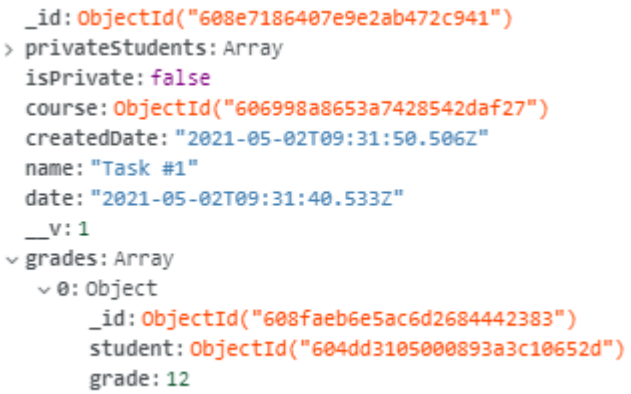

Рисунок 2.3 – Документ завдання з вбудованим масивом документів оцінок

2.2 Створення серверної частини.

Отже ми створили БД, яка готова для використання нашим сервером. Як вже було сказано, структура БД буде створюватися на рівні серверного застосунку, а не самої БД. Це дуже зручний функціонал, який надає використання обраної БД, адже таким чином вся робота по створенні БД полягає в тому, щоб створити сам екземпляр БД, та підключити до неї будь-який застосунок, який буде вирішувати як і яким чином працювати з БД, які колекції та документи зберігати в ній.

Отже, створення серверної частини умовно можна розподілити на такі етапи:

1. Створення серверного застосунку за допомогою обраних засобів та технологій;
2. Налаштування з’єднання з БД та створення моделей сутностей, що будуть використовуватися в програмній системі;

3. Створення сервісів для обробки будь-яких дій пов’язаних з цими сутностями;
4. Створення контролерів для кожної сутності, які будуть використовувати створені сервіси і надавати зовнішнім застосункам, таким як клієнтським частинам, зручний спосіб доступу до роботи з функціоналом серверної частини.

Будь-який процес розробки програмної системи починається з пошуку та вибору технологій, бібліотек та фреймворків за допомогою яких буде створюватися відповідна програмна система. В даному розділі буде описано обрані технології для серверної частини. Короткий перелік обраного стеку технологій можна представити у вигляді наступного списку:

- NestJS – загальний фреймворк, для створення серверних застосунків;
- Mongoose – бібліотека для роботи з БД MongoDB;
- ExpressJS – бібліотека для створення сервера;
- PassportJS – бібліотека для реалізації автентифікації та авторизації в серверних застосунках;
- NanoId – бібліотека для генерації унікальних ідентифікаторів;
- Bcrypt – бібілотека для хешування даних;
- NodeJS – середовище виконання мови програмування JavaScript;
- TypeScript – мова програмування.

Програмний застосунок повністю створюється за допомогою платформи NodeJS, яка дозволяє запускати JavaScript код не в браузері, якому зазвичай притаманна дана мова програмування, а на стороні сервера. У зв’язці з NodeJS використовується так названий Node Package Manager, який використовується для обробки, управління, збереження та установки різного призначення NodeJS пакетів та бібліотек.

npm (Node Package Manager) – це менеджер пакетів для мови програмування JavaScript. Для середовища виконання Node.js це менеджер пакетів за замовчуванням. Включає в себе клієнт командного рядка, який також називається

npm, а також онлайн-базу даних публічних та приватних пакетів, яка називається реєстром npm. Реєстр доступний через клієнт, а доступні пакети можна переглядати та шукати через веб-сайт npm [7].

npm може управляти пакетами, які є локальними залежностями певного проекту, а також глобально інстальованими інструментами JavaScript. При використанні npm як менеджера залежності для локального проекту, можна встановити одною командою всі залежності проекту через файл package.json. У файлі package.json кожна залежність може визначати діапазон дійсних версій, використовуючи схему семантичної версії, що дозволяє розробникам автоматично оновлювати свої пакети, одночасно уникаючи небажаних змін [7].

Так як і серверний і клієнтський застосунки використовують NodeJS, то обидва також використовують npm для управління своїми залежностями та саме використовуючи npm було інстальовано всі пакети та бібліотеки в серверному застосунку, що були перелічені вище.

Отже, було обрано середовище для розробки серверної та клієнтської частини. Використання однієї і тієї ж мови програмування у всіх частинах програмної системи робить процес розробки більш зручнішим для розробників, а також дозволяє вивчивши лише одну мову програмування використовувати її для написання різних видів застосунків. Саме таку мову програмування представляє з себе JavaScript.

JavaScript (JS) – динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки [12].

JavaScript, наразі, є однією з найпопулярніших мов програмування в інтернеті. В перші роки існування, більшість професійних програмістів скептично ставилися до мови, цільова аудиторія якої складалася з програмістів-аматорів. Поява AJAX змінила ситуацію та звернула увагу професійної спільноти до мови, а її подальші модифікації за стандартами ES6+ внесли багато корисних можливостей, яких не

вистачало для ефективного програмування. В результаті, були розроблені та покращені багато практик використання JavaScript (зокрема, тестування та налагодження), створені бібліотеки та фреймворки, поширилося використання JavaScript поза браузером [12].

Оскільки JavaScript є інтерпретованою мовою програмування, без строгої типізації, і може виконуватися в різних середовищах, кожне зі своїми власними особливостями сумісності, програміст має бути уважним, і повинен перевіряти, що його код виконується як очікується в широкому переліку можливих конфігурацій. Типізація вважається одною з ключових проблем JavaScript, тому восени 2012 року, компанія Microsoft презентувала мову програмування TypeScript, що компілюється в JavaScript та містить декілька важливих для програмістів доповнень, що полегшують розробку [12].

TypeScript (TS) надає розробникам певний ряд покращень над JS, що дозволяє підвищити швидкість розробки, читабельність коду, його рефакторинг, надає кращі можливості для повторного використання коду, а також для знаходження потенційних помилок в коді ще на етапі розробки. Ключовою відмінністю TS від JS є те, що TS дозволяє чітко визначати типи даних для змінних, в той час як JS має динамічну типізацію, що дуже часто може призвести до помилок, коли розробник помилково задав змінній значення одного типу, а далі в коді працює з нею як зі змінною іншого типу даних. Оскільки TS дуже сильно покращує процес розробки при створенні будь-яких застосунків, що базуються на JS, було прийнято рішення писати обидві частини створюваної програмної частини саме на цій мові програмування.

Для створення серверної частини було обрано фреймворк NestJS, який базується на NodeJS та бібліотеці ExpressJS. NestJS дозволяє створювати ефективні, легко масштабовані серверні застосунки на мові програмування JS або TS завдяки своїй детально та якісно продуманій архітектурі. Під капотом NestJS використовує ExpressJS, який дозволяє легко створювати серверні застосунки в середовищі NodeJS. Хоч ExpressJS і надає можливості для створення якісних та швидких веб-серверів, він ніяк не визначає архітектуру, якої повинні дотримуватися розробники

при створенні серверних застосунків. З одного боку це надає розробникам свободу при розробці, а з іншого, якщо при початку створення застосунку розробник обере неправильну архітектуру, то в подальшому при виявленні таких проблем їх буде важко виправити, оскільки доведеться переписувати всю структуру застосунку. NestJS створює певний ряд абстракцій над ExpressJS тим самим задаючи розробникам уже якісно продуману архітектуру, яка чудово підходить для створення серверних застосунків.

В NestJS існує цілий ряд абстракцій, які надають зручний спосіб для організації коду. При розробці серверної частини було використано наступні основні абстракції NestJS: модулі, контролери та сервіси.

Таким чином, для всіх сутностей системи, які було перераховано в підрозділі про БД, було створено окремий модуль. Модуль представляє собою ізольований об’єкт, який може містити контролери та сервіси. При цьому логіка в цих контролерах та сервісах зазвичай повинна співпадати з темою модуля. Тобто, якщо модуль створений для користувачів, то він повинен містити тільки контролери та сервіси, які виконують обробку інформації пов’язаної з користувачами. Але при цьому модулі можуть експортувати певні сервіси, а інші модулі можуть імпортувати ці модулі при цьому отримуючи доступ до сервісів, що експортує модуль, що було імпортовано.

Контролери представляють собою класи, що описують так звані кінцеві точки веб-сервера (endpoints). Саме на них будуть приходити HTTP-запити від клієнтів, коли клієнту потрібно буде отримати або зберегти певну інформацію у веб-сервері.

Сервіси схожі з контролерами, але вони не можуть бути використанні клієнтами напряму і використовуються контролерами для обробки даних. Саме сервіси виконують всі операції пов’язані з взаємодією веб-сервера з БД.

Таким чином, контролери відіграють роль HTTP рівня для обробки вхідних запитів до сервера, а сервіси реалізують будь-яку логіку самого веб-застосунку.

При розробці серверної частини в кожному модулі для кожної сутності було розроблено відповідні контролери та сервіси. Наведемо приклад коду одного з контролерів:

```
@Controller('courses')
@UseGuards(JwtAuthGuard)
export class CoursesController {
  constructor(private readonly coursesService: CoursesService) {}

  @Post()
  create(@Request() req, @Body() createCourseDto: CreateCourseDto) {
    return this.coursesService.create(createCourseDto, req.user);
  }

  @Put(':id')
  update(
    @Param('id') id: string,
    @Request() req,
    @Body() createCourseDto: CreateCourseDto,
  ) {
    return this.coursesService.update(id, createCourseDto, req.user);
  }

  @Post(':id/delete')
  delete(
    @Param('id') id: string,
    @Request() req,
    @Body() deleteCourseDto: DeleteCourseDto,
  ) {
    return this.coursesService.deleteCourse(id, deleteCourseDto, req.user);
  }

  @Get(':classroomUrlCode/c/:courseUrlCode')
  getInfoForCoursePage(
    @Request() req,
    @Param('courseUrlCode') courseUrlCode: string,
    @Param('classroomUrlCode') classroomUrlCode: string,
  ) {
    return this.coursesService.getInfoForCoursePage(
      req.user,
      courseUrlCode,
      classroomUrlCode,
    );
  }

  @Patch('grade')
  updateGrade(
    @Request() req,
    @Body() updateCourseGradeDto: UpdateCourseGradeDto,
  ) {
    return this.coursesService.updateGrade(updateCourseGradeDto, req.user);
  }
}
```

Рисунок 2.4 – Контролер для роботи з курсами

В NestJS використовуються TypeScript декоратори для того, щоб надати класам, функціям або змінним певних метаданих, які потім будуть використані фреймворком для того, щоб правильно перетворити все в відповідну реалізацію на ExpressJS. У наведеному вище прикладі були використанні наступні декоратори:

- @Controller – визначає клас контролером;
- @UseGuards – визначає, що для доступу до кінцевих точок даного контролера повинен бути використаний спеціальний захисник, який визначає чи авторизований користувач, від імені якого робиться запит. Якщо користувач не авторизований доступ буде заборонено і клієнту буде повернена відповідна помилка;
- @Post, @Get, @Patch, @Put – визначають відповідні HTTP методи, які повинні бути використані для виконання запиту до відповідної кінцевої точки. Можна сказати, що саме ці декоратори перетворюють методи класу в кінцеві точки, які можуть бути доступні для виконання запитів до них;
- @Request – визначає спеціальний об’єкт запиту з бібліотеки ExpressJS;
- @Body – визначає об’єкт тіла запиту;
- @Param – визначає параметри запиту, які отримуються з адреси запиту.

Як було сказано вище для роботи з БД MongoDB використовується бібліотека mongoose. Ця бібліотека надає зручний спосіб описувати моделі MongoDB документів та колекцій, які будуть містити ці документи. За допомогою mongoose було створено та описано структуру всіх сутностей в програмній системі. Таким чином, mongoose було використано для створення структури всієї БД, оскільки це в нашому випадку виконується саме на рівні веб-застосунку.

При використанні mongoose в NestJS використовується спеціальний пакет @nestjs/mongoose, який надає певний ряд абстракцій над бібліотекою, а також надає розробникам такий же ряд декораторів за допомогою яких і виконується опис структури документів. Крім цього варто зазначити, що для NodeJS середовища існує спеціальний офіційний пакет для взаємодії з MongoDB, але бібліотека

mongoose надає більш зручніший спосіб для взаємодії з MongoDB, а тому використовується набагато частіше при розробці веб-застосунків.

Наведемо приклад коду, для створення моделі, що описує структуру однієї з сутностей:

```
7   export type ClassroomDocument = Classroom & Document;
8
    You, 2 months ago | 1 author (You)
9   @Schema()
10  export class Classroom {
11    @Prop({ required: true, trim: true })
12    name: string;
13
14    @Prop({ trim: true })
15    description: string;
16
17    @Prop({ required: true })
18    createdDate: string;
19
20    @Prop({ required: true, unique: true })
21    code: string;
22
23    @Prop({ required: true, unique: true })
24    urlCode: string;
25
26    @Prop({ required: true, type: mongoose.Schema.Types.ObjectId, ref: 'User' })
27    classhead: User | mongoose.Types.ObjectId;
28
29    @Prop({ type: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }] })
30    teachers: User[] | mongoose.Types.ObjectId[];
31
32    @Prop({ type: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }] })
33    students: User[] | mongoose.Types.ObjectId[];
34
35    @Prop({ type: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Course' }] })
36    courses: Course[] | mongoose.Types.ObjectId[];
37  }
38
39  export const ClassroomSchema = SchemaFactory.createForClass(Classroom);
40
```

Рисунок 2.5 – Код опису структури моделі класу

Наведемо короткий опис використаних декораторів:

- @Schema – визначає клас, який повинен бути схемою для всіх документів у відповідній колекції документів в БД;
- @Prop – помічає поле класу як поле документу в колекції БД.

2.3 Створення клієнтської частини.

Для взаємодії користувачів зі створеним серверним застосунком було створено клієнтську частину у вигляді веб-сайту. В цьому розділі розглянемо технології, які було при цьому використано та наведемо приклади їх використання в клієнтській частині програмної системи. Перш за все варто зазначити такі технології як мова розмітки HTML, мова стилів CSS та мова програмування JavaScript, які є стандартом для розробки веб-сайтів. Але у нашому випадку замість JavaScript було використано TypeScript, про який було розказано в попередньому розділі.

Мова гіпертекстової розмітки (HyperText Markup Language – HTML) використовується для створення і візуального відображення веб-сторінок. Основу кожної веб-сторінки в мережі Інтернет становить HTML. Під терміном «гіпертекст» мається на увазі текст, сформований за допомогою мови розмітки і який містить гіперпосилання, якими зв'язані веб-сторінки одна з одною, роблячи Всесвітню павутину тим, чим вона є сьогодні. HTML підтримує як візуальні зображення, так і інший медіа контент. HTML є мовою, що описує структуру і семантику вмісту веб-документу. Контент веб-сторінки розмічений за допомогою тегів, що представляють HTML-елементи. Прикладами таких елементів є <head>, <title>, <body>, <article>, <section>, <p>, , , <img>, <picture>, і т.д. Ці елементи формують будівельні блоки для будь-якого веб-сайту [17].

CSS (Cascading Style Sheets) це декларативна мова програмування, яка контролює зовнішній вигляд веб-сторінок у браузері. Браузер застосовує декларації стилів CSS до обраних елементів для їх коректного відображення. Декларація стиля складається з властивостей та їх значень, які обумовлюють вигляд веб-сторінки. Правило CSS це набір властивостей, асоційованих із селектором. CSS використовується авторами веб−сторінок для вказування кольорів, шрифтів, розташування блоків та інших аспектів представлення документа. Основною метою CSS є розділення вмісту (написаного на HTML) та подання документа (написаного на CSS). Цей поділ документа збільшує доступність документа, надає велику

гнучкість та можливість виконувати управління його показу, а також зменшити складність та повторюваність в структурі самого документа [16].

Використовуючи HTML, CSS та JS можна створювати будь-які динамічні веб-сайти. Але в сучасній веб-розробці на сьогодні частіше використовуються спеціальні фреймворки та бібліотеки, які покращують та полегшують веб-розробку, а також надають специфічні можливості, яких було б важко досягти використовуючи лише HTML, CSS та JS. Саме тому для створення програмної системи «Е-щоденник» також було використано JS-бібліотеку React, яка дозволяє створювати Single Page Application (SPA) – односторінкові веб-застосунки.

React – відкрита JS-бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових веб-застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників. React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки інтерфейс користувача у застосунках. Це відповідає частині View у шаблоні MVC, і може бути використане у поєднанні з іншими JavaScript бібліотеками [21].

Будівельними блоками в React є спеціальні компоненти, які можуть бути або класами, або функціями. В розробці було використано функціональні компоненти, а тому саме на їх прикладі буде виконано опис технологій.

Стандартний функціональний React компонент приймає як параметри певні властивості, які він отримує від батьківського компонента, тобто компонента, який рендерить його. Після чого певним чином оброблює отримані дані та повертає спеціальні React елементи, які схожі на HTML за розміткою. Також компоненти можуть повертати в розмітці крім стандартних HTML тегів (точніше їх аналогів у вигляді React елементів) ще й інші React компоненти.

Також функціональні компоненти можуть не тільки просто відображати отримані від іншого компонента дані, а ще й мати певний стан, який може змінюватися з часом. При зміні стану компонента відбувається вирахування змін в

розмітці, яку повертає компонент після чого відбувається оновлення компонента в інтерфейсі користувача. Якщо стан компонента передається в інші компоненти у вигляді властивостей, то відображення цих компонентів також буде оновлено.

Для відслідковування змін в графічному інтерфейсі та зменшенні кількості оновлень використовується концепція віртуального DOM (Document Object Model). При зміні в компонентах, React створює новий віртуальний DOM (нову структуру сторінки), порівнює його з попереднім, після чого оновлює лише ті частини справжнього DOM, які справді змінилися.

Нижче наведено приклад компонента, який використовується для відображення спеціальної відмітки біля приватних завдань та курсів:

```
import { Tag, Tooltip } from 'antd';
import React from 'react';

const getAlign = (align: string = 'center') => {
  if (align === 'center') return 'self-center';
  else if (align === 'start') return 'self-start';
  else if (align === 'end') return 'self-end';
}

...
interface PTagProps {
  text?: string;
  children?: any;
  align?: 'center' | 'start' | 'end';
  tooltip?: string;
}

export default function PTag({ text, children, tooltip, align = 'center' } : PTagProps) {
  const tag = (
    <Tag
      color="cyan"
      className={`bg-transparent text-green-500 border-green-500 ${getAlign(align)} text-sm`}
    >
      {text || children}
    </Tag>
  );

  return tooltip ? (
    <Tooltip title={tooltip}>
      {tag}
    </Tooltip>
  ) : tag;
}
```

Рисунок 2.6 – Приклад функціонального React-компонента

Також при розробці клієнтської частини було використано бібліотеку компонентів Ant Design, бібліотеку styled-components, а також CSS-фреймворк TailwindCSS.

Використання бібліотеки компонентів можна побачити на рисунку наведеному вище, де використовуються компоненти Tag та Tooltip з цієї бібліотеки. Також можна побачити використання TailwindCSS. Компоненту Tag присвоюються CSS класи саме з цього фреймворка, де кожен клас відповідає за певну CSS властивість.

Що ж стосується бібліотеки styled-components, то вона дозволяє створювати спеціальні компоненти обгортки поверх інших React-компонентів або елементів переписуючи їх стилізацію або додаючи нову. Це дуже зручний функціонал, особливо при використанні сторонніх бібліотек компонентів, адже в такому випадку ми не маємо доступу до коду таких компонентів, але використання бібліотеки styled-components надає можливість змінити стилізацію таких компонентів. Нижче наведено приклад використання цієї бібліотеки:

```
14  const StyledClassroomForm = styled(CreateClassroomForm)`
15    max-width: 500px;
16    width: 100%;
17    margin: 0 auto;
18    padding-top: 5px;
19    padding-bottom: 5px;
20
21    & .ant-input {
22      background-color: rgba(31, 41, 55, var(--tw-bg-opacity));
23      color: whitesmoke;
24      border: none;
25    }
26
27    & .ant-input-textarea-show-count::after {
28      color: whitesmoke;
29    }
30  `;
```

Рисунок 2.7 – Приклад використання styled-components

2.4 Висновки до розділу

Отже, серверна частина програмної системи «Е-щоденник» представляє собою веб-сервер з архітектурою REST і була написана на мові програмування TypeScript з використанням фреймворка NestJS. При цьому було створено певну кількість класів моделей, які представляють собою відповідні їм колекції в БД MongoDB. Було створено класи контролерів API з методами, які приймають на вхід запити, виконують обробку інформації з БД на основі вказаних в запитах даних та повертають відповідні дані клієнту. Було створено клієнтську частину за допомогою бібліотеки React, а також додаткових бібліотек для створення інтерфейсу користувача на веб-сайті.

Використання описаних технологій дозволило створити прототип програмної системи «Е-щоденник», роботу якої описано в наступному розділі цієї кваліфікаційної роботи.

Розділ 3. ОПИС РОБОТИ ПРОГРАМНОЇ СИСТЕМИ

Створена програмна система дозволяє користувачам створювати власні класи, запрошувати до класу інших користувачів в якості учнів або викладачів, виставляти оцінки учням за завдання та групи завдань, а також автоматично вираховує оцінки учнів за групи завдань на основі оцінок завдань в цих групах. Також автоматична оцінка вираховується і для всього курсу. Нижче буде наведено приклади користування програмною системою за різними сценаріями.

При переході на веб-сайт неавторизованому користувачу відразу відобразиться кнопки для входу або реєстрації.

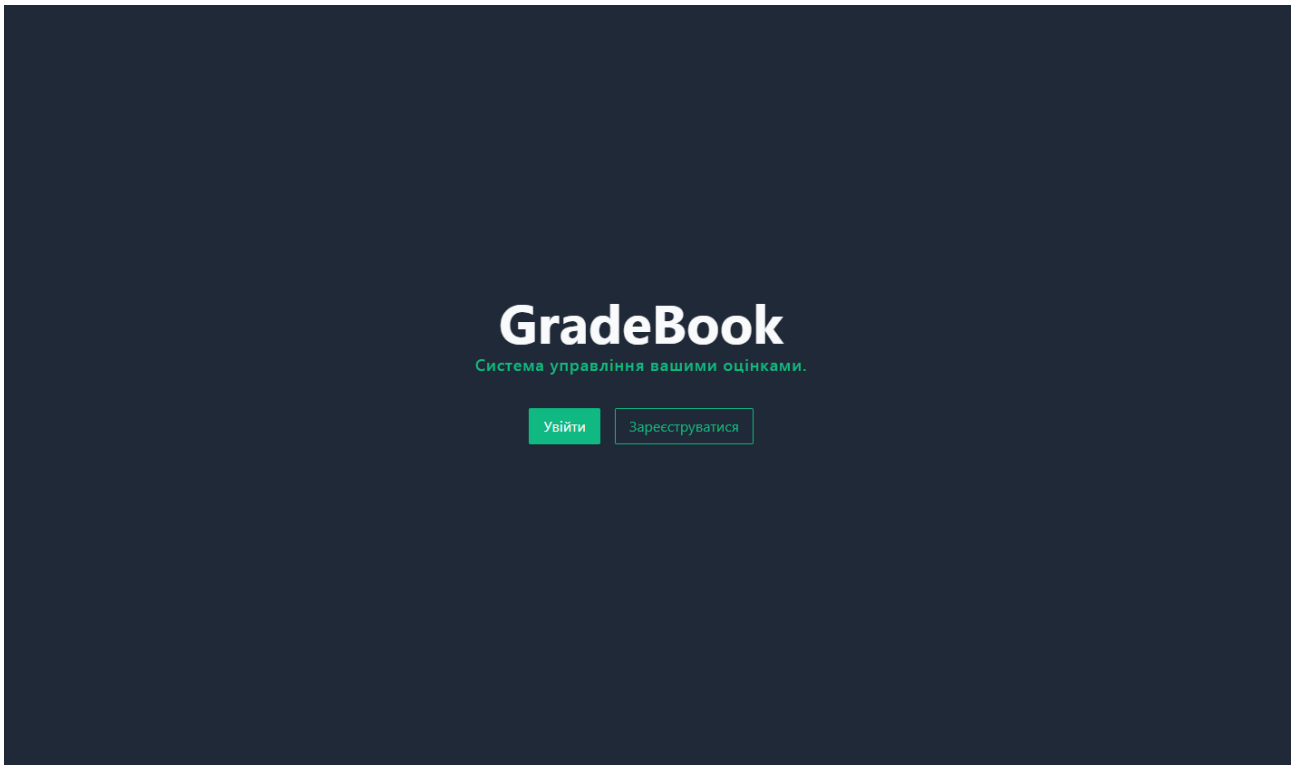

Рисунок 3.1 – Головна сторінка

Натиснувши кнопку «Увійти» відкриється модальне вікно з формою для входу до системи. У цій формі користувачу буде запропоновано увести такі дані як електронна пошта та пароль, які дозволять йому увійти в систему за умови, що користувач з вказаною електронною поштою існує і пароль введено правильний.

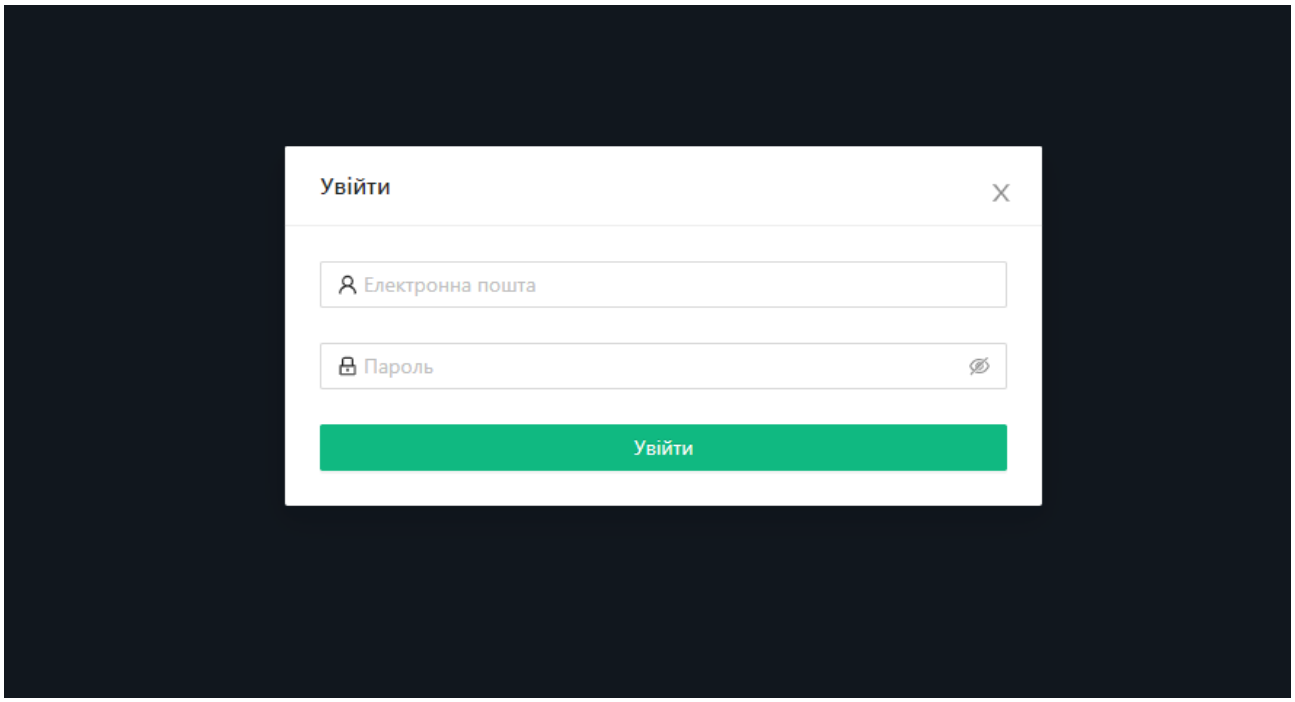

Рисунок 3.2 – Форма входу

Натиснувши кнопку «Зареєструватися» відкриється модальне вікно з формою для реєстрації в системі. У цій формі користувачу буде запропоновано увести такі дані як електронна пошта, ім’я, прізвище та пароль, які дозволять йому зареєструватися в системі.

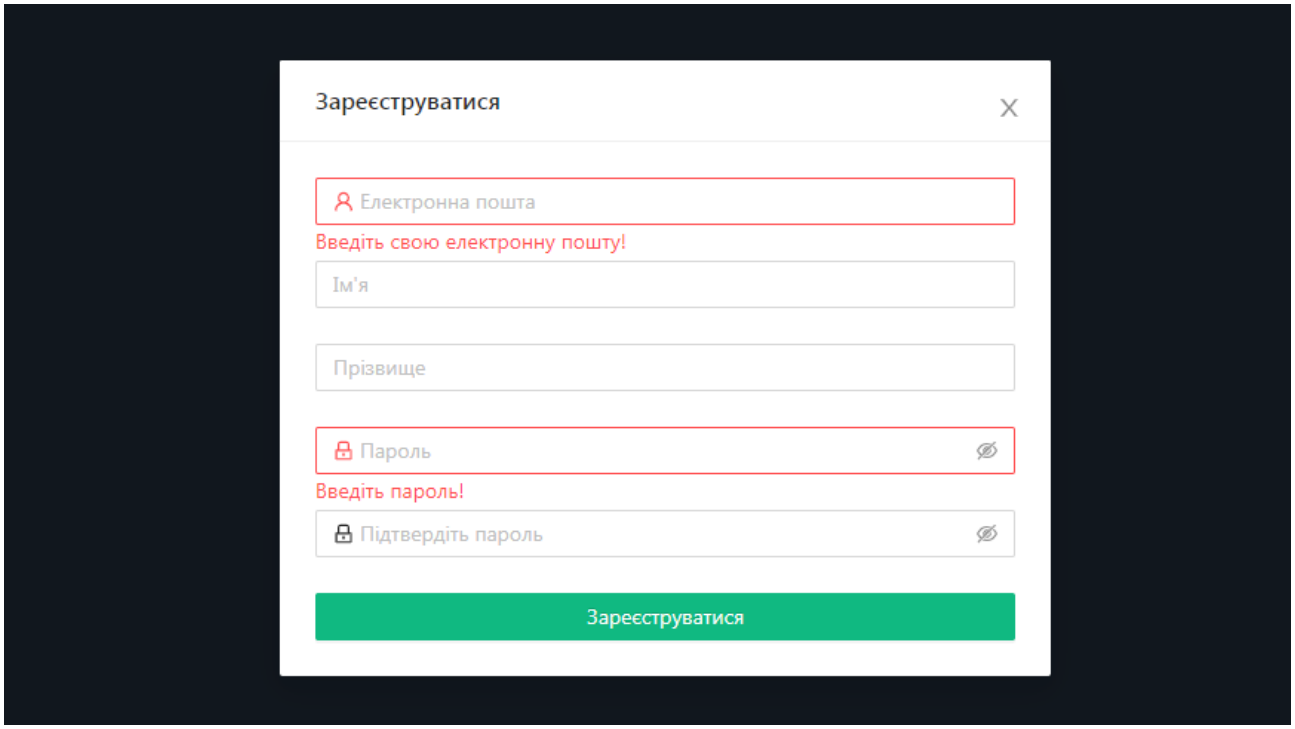

Рисунок 3.3 – Форма реєстрації

Після входу або реєстрації користувач попадає на основну сторінку сайту, яка містить список класів.

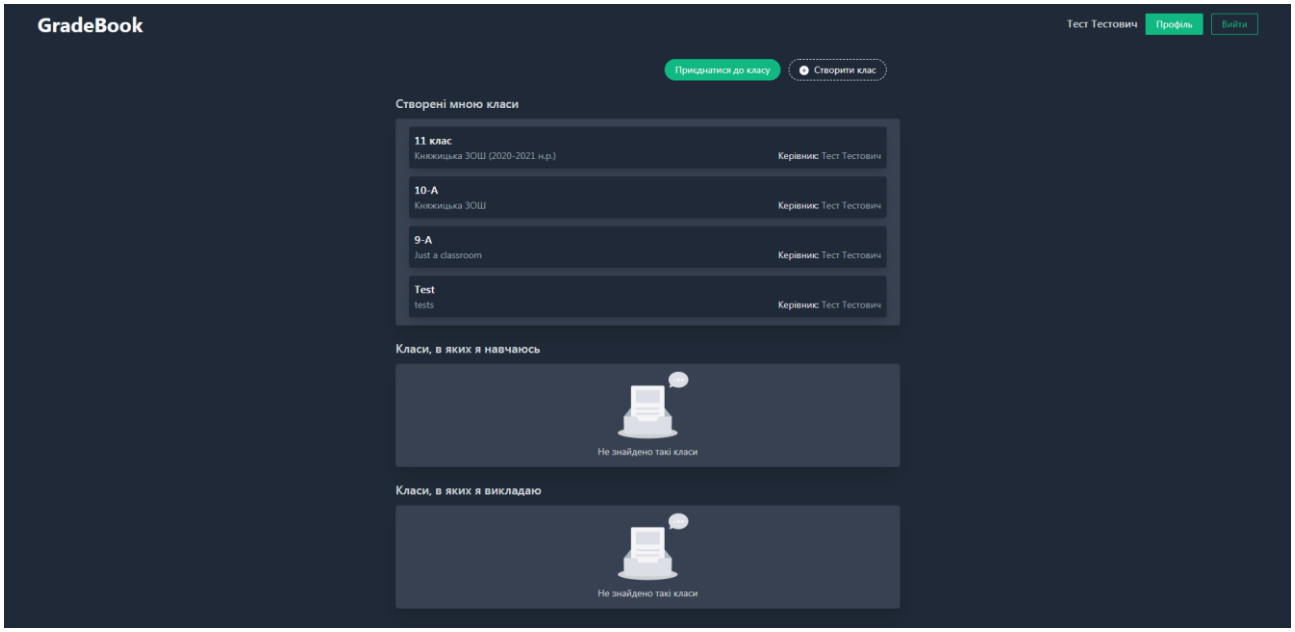

Рисунок 3.4 – Сторінка зі списком класів

Зверху списку присутні кнопки для створення нового класу або приєднання до класу створеного іншим користувачем.

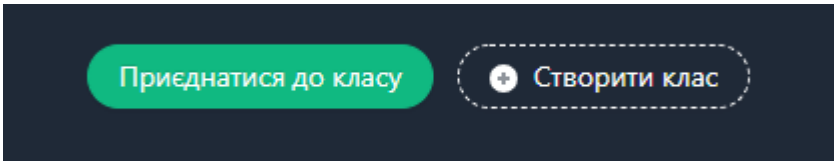

Рисунок 3.5 – Опції зі створення або приєднання до класу

Форма приєднання до класу має наступний вигляд та вимагає введення коду класу:

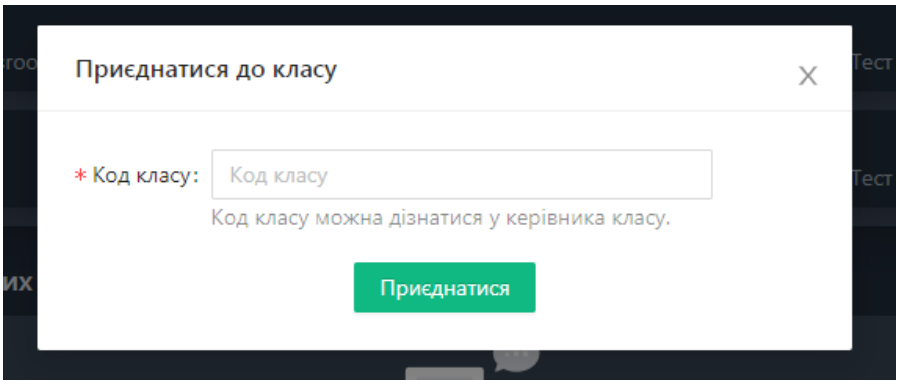

Рисунок 3.6 – Форма приєднання до класу

Форма створення нового класу має наступний вигляд та вимагає введення назви класу та короткого опису:

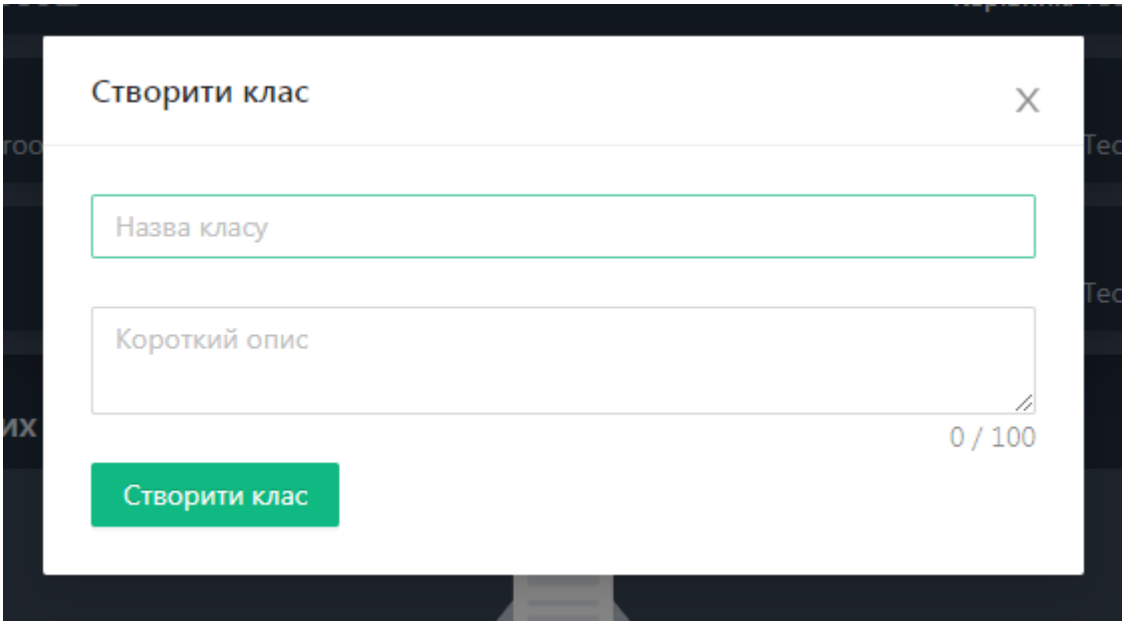

Рисунок 3.7 – Форма створення нового класу

Список класів складається з трьох блоків: створені користувачем класи; класи в яких користувач навчається; класи в яких користувач викладає.

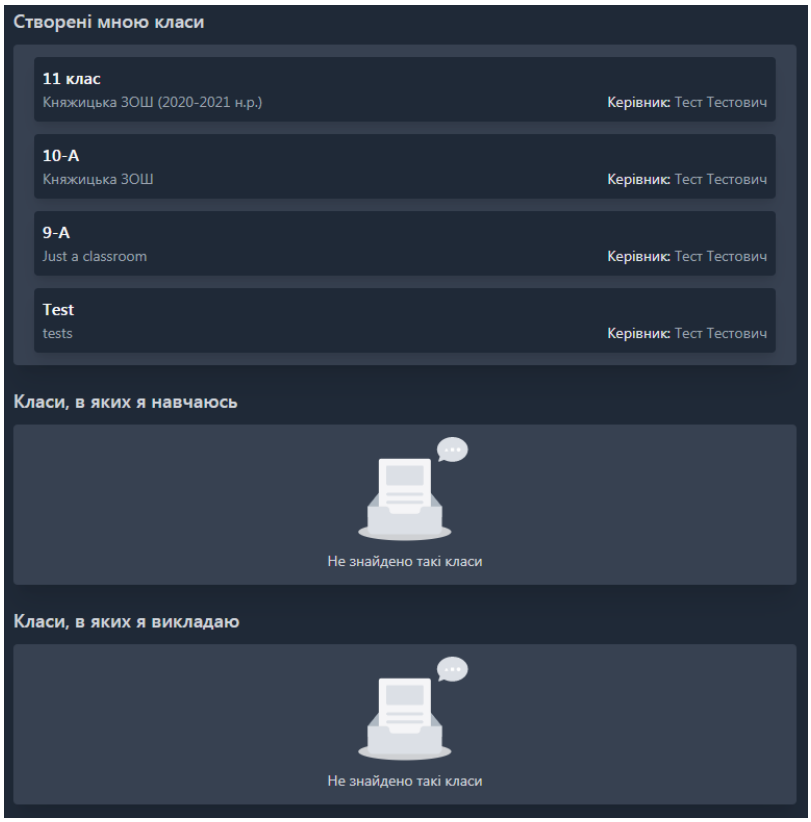

Рисунок 3.8 – Список класів

Розглянемо сценарій при якому користувач є керівником класу та викладачем в курсах цього класу. Вибравши клас, користувач попадає на сторінку класу.

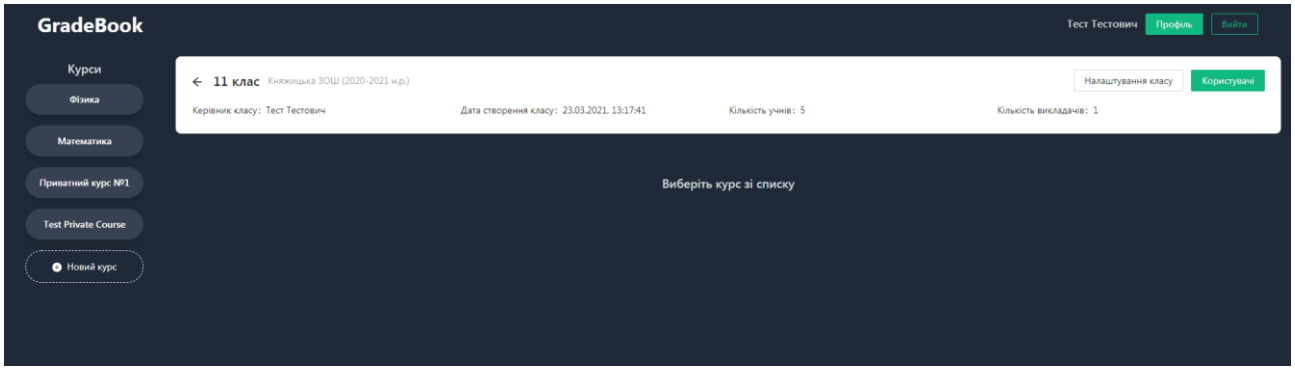

Рисунок 3.9 – Сторінка класу

На верхній панелі з короткою інформацією про клас присутні також кнопки для переходу на сторінки налаштування класу та списку користувачів класу.

На сторінці налаштування класу можна змінити інформацію про клас, переглянути код класу, який потрібно буде відправити учням, форма для додавання викладачів до класу, а також кнопка для видалення класу.

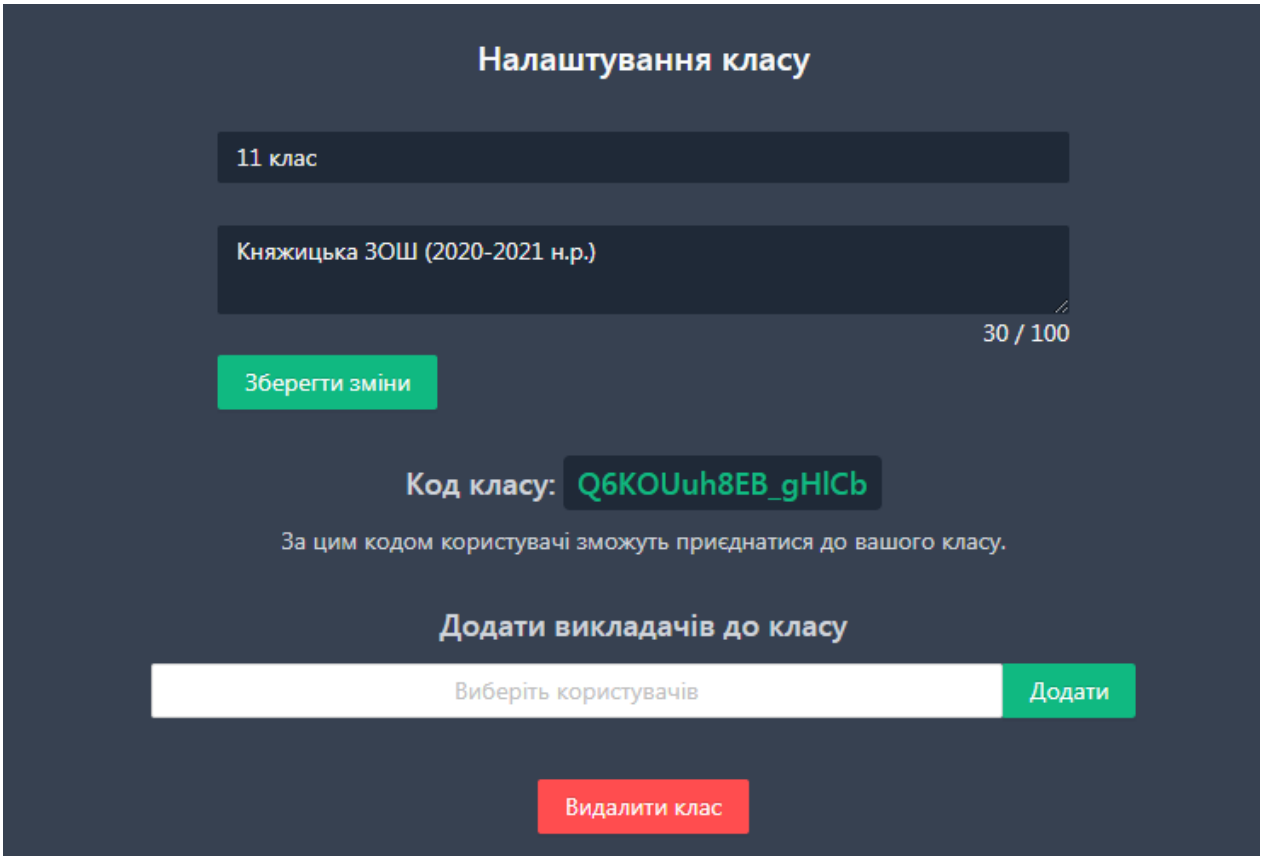

Рисунок 3.10 – Налаштування класу

На сторінці зі списком користувачів класу можна переглянути список викладачів та учнів. Керівник класу може видаляти учнів або викладачів.

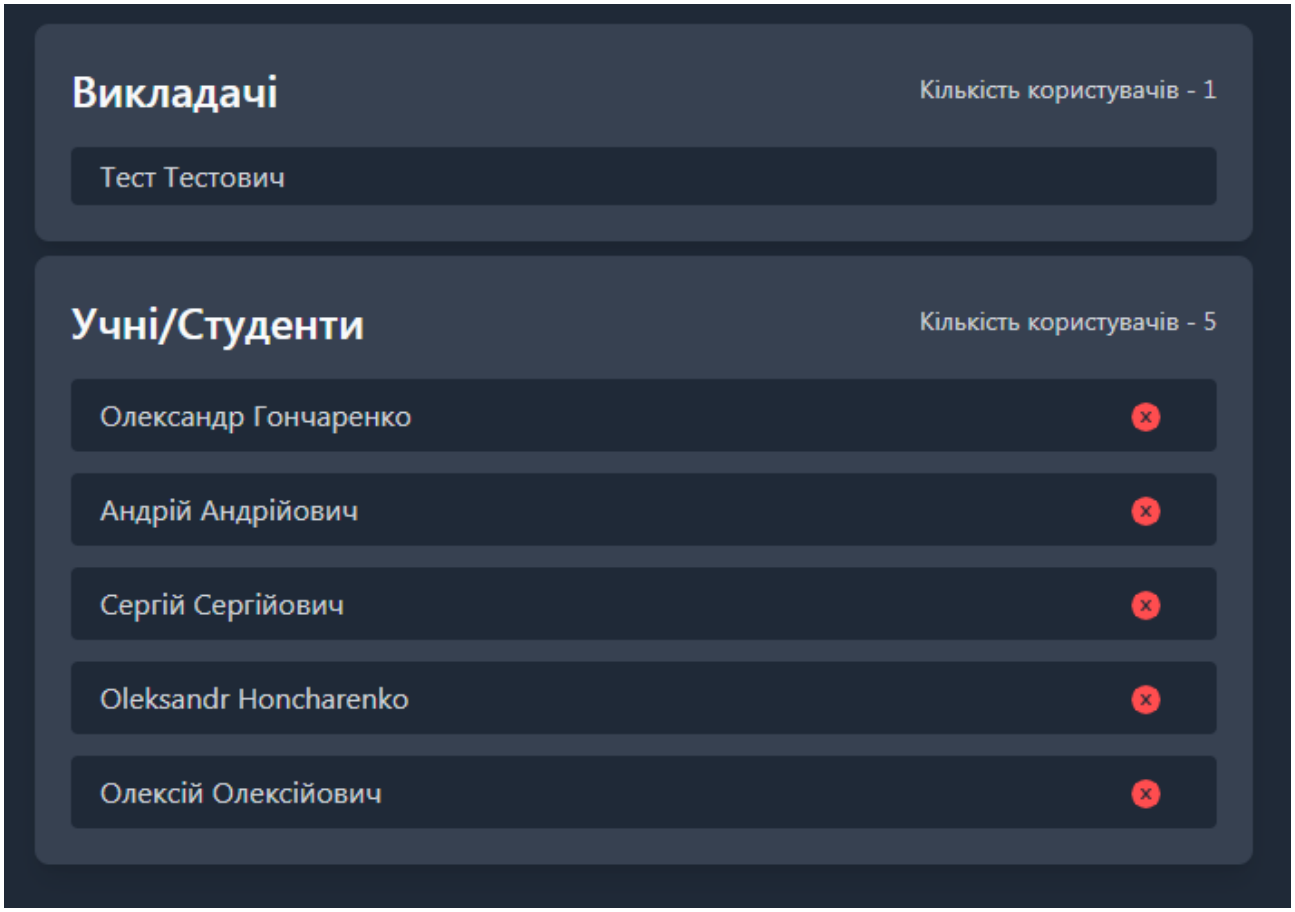

Рисунок 3.11 – Список користувачів класу

В лівому боці сторінки класу присутній список курсів цього класу.

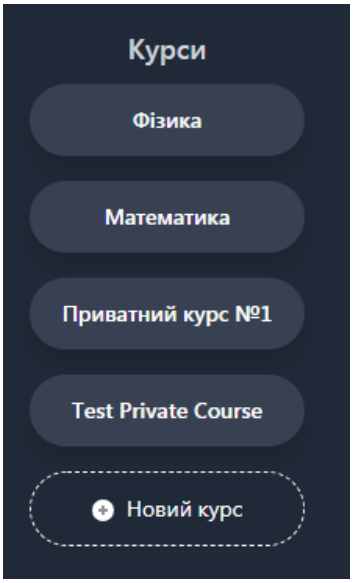

Рисунок 3.12 – Список курсів класу

Вибравши один з курсів відкриється інформація про курс та список завдань курсу.

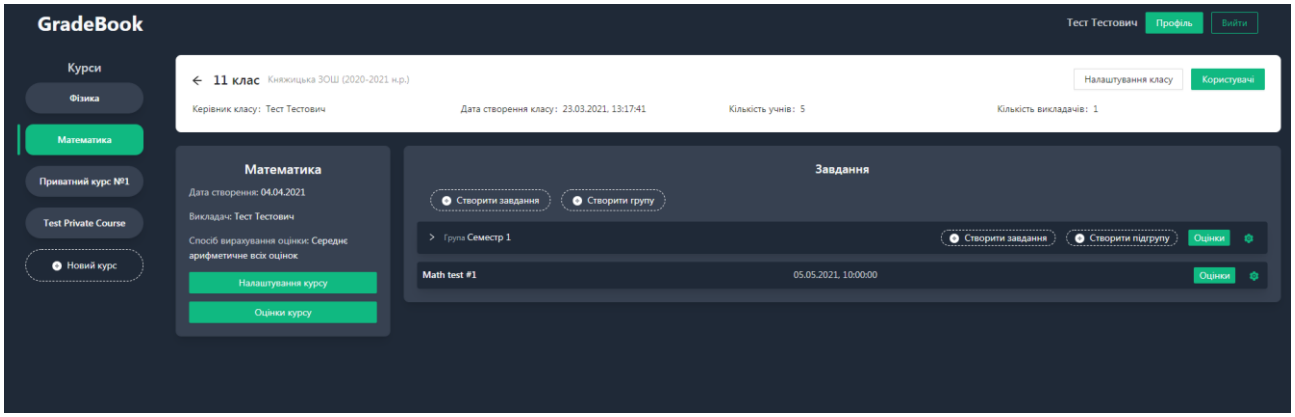

Рисунок 3.13 – Сторінка класу з обраним курсом

Форма створення нового курсу має наступний вигляд:

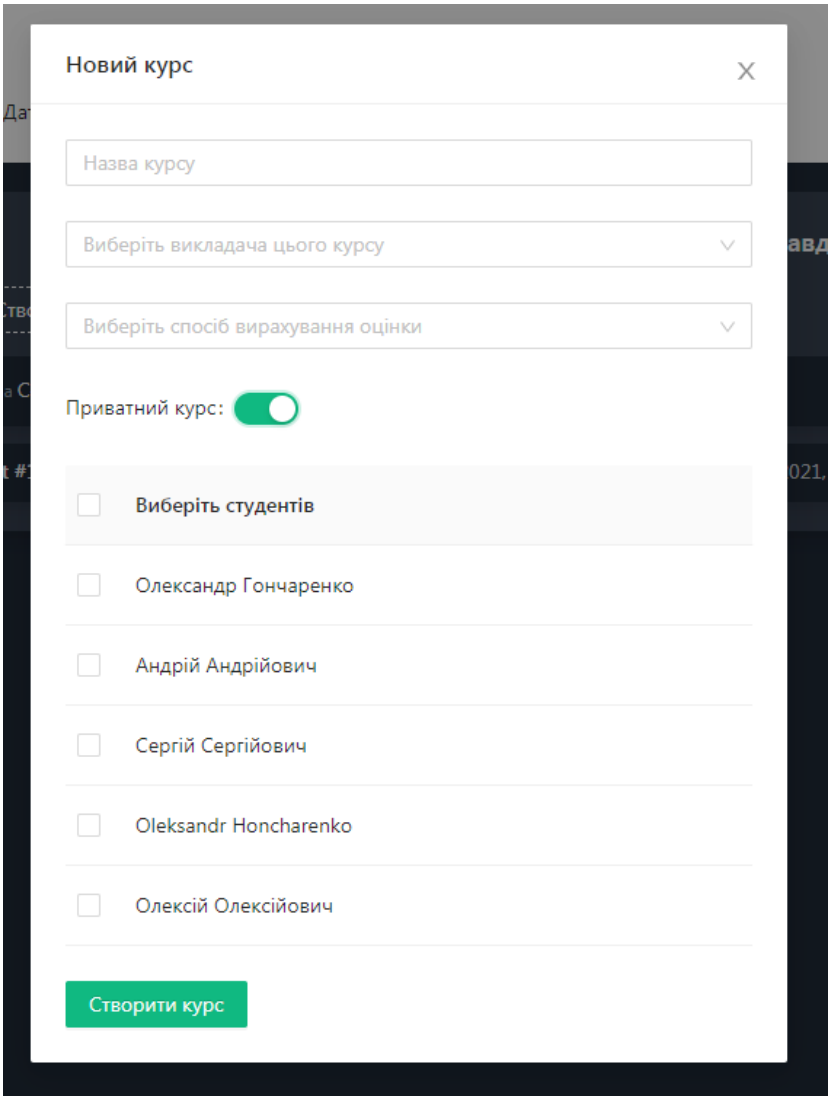

Рисунок 3.14 – Форма створення курсу

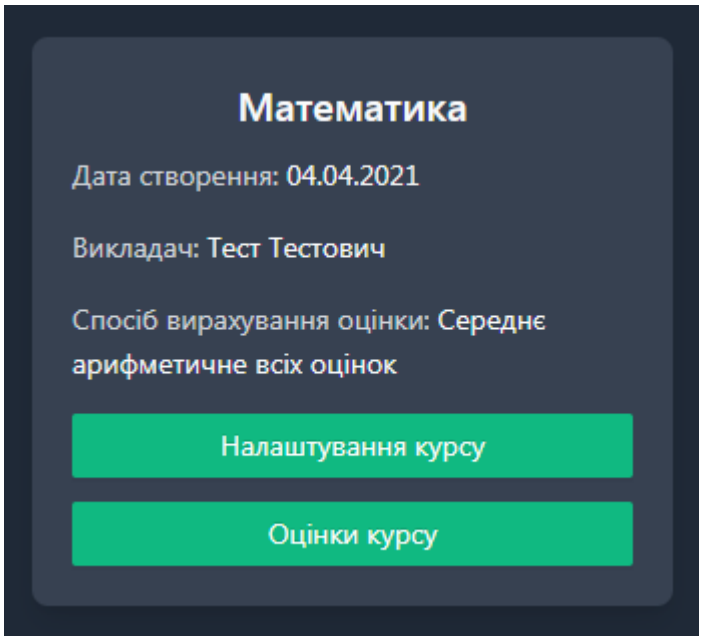

Рисунок 3.15 – Інформація про обраний курс

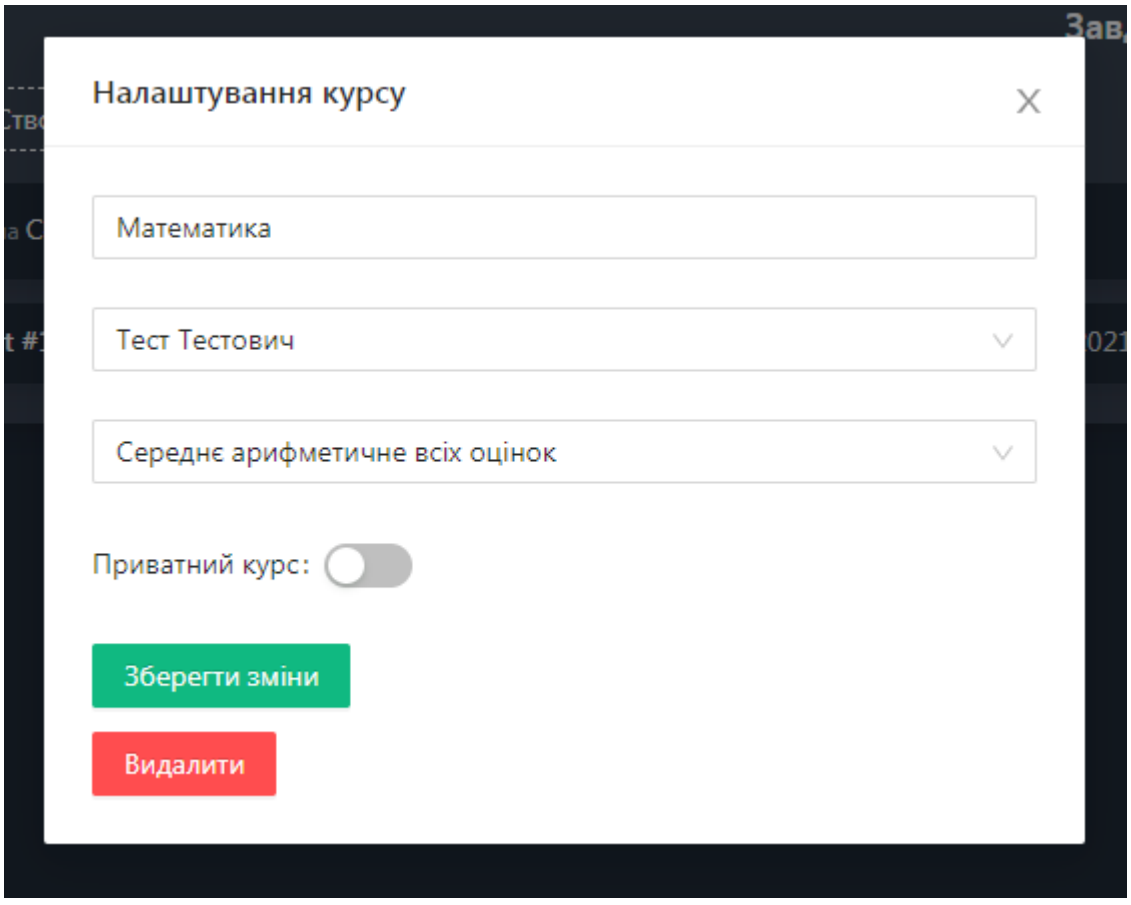

Рисунок 3.16 – Форма налаштування курсу

Список завдань складається з завдань на рівні курсу, а також груп завдань, які можуть містити як завдання так і підгрупи завдань.

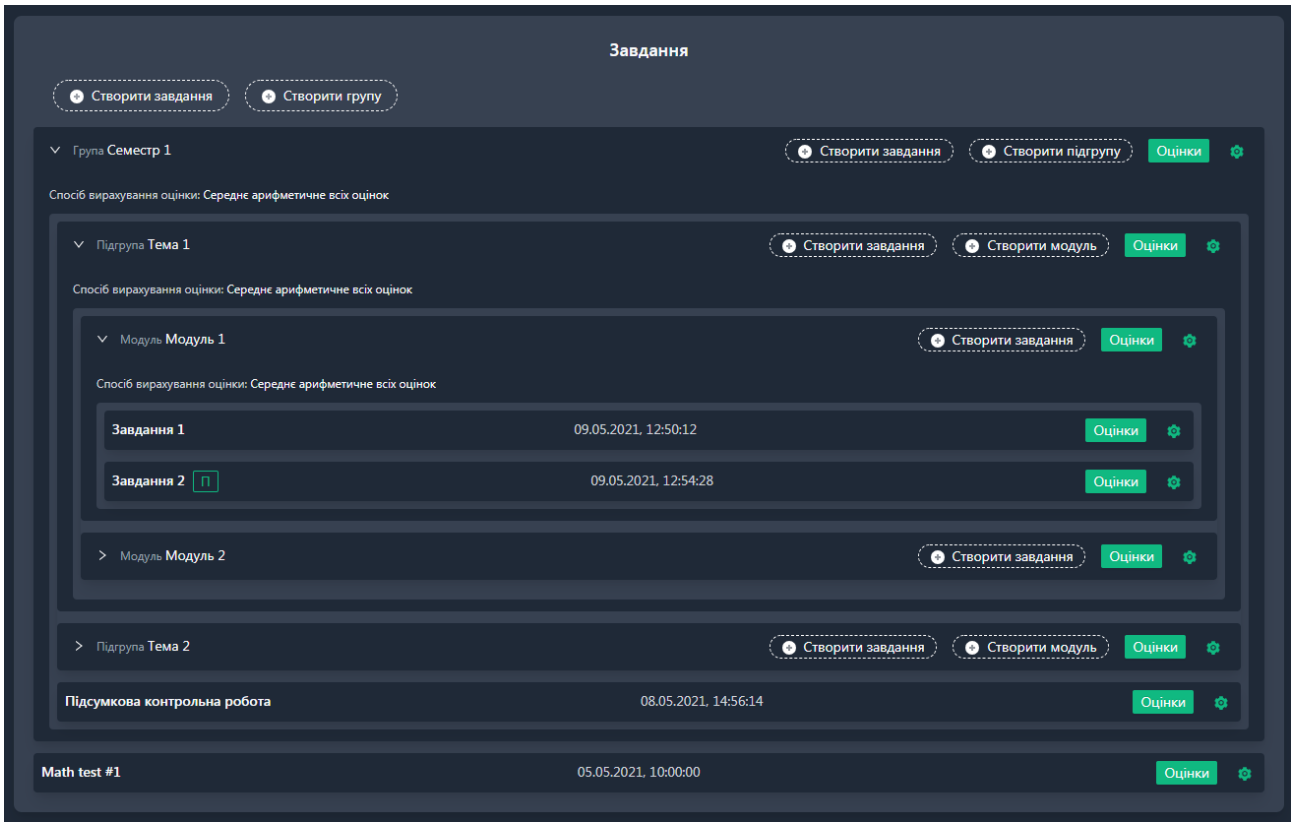

Рисунок 3.17 – Список завдань класу

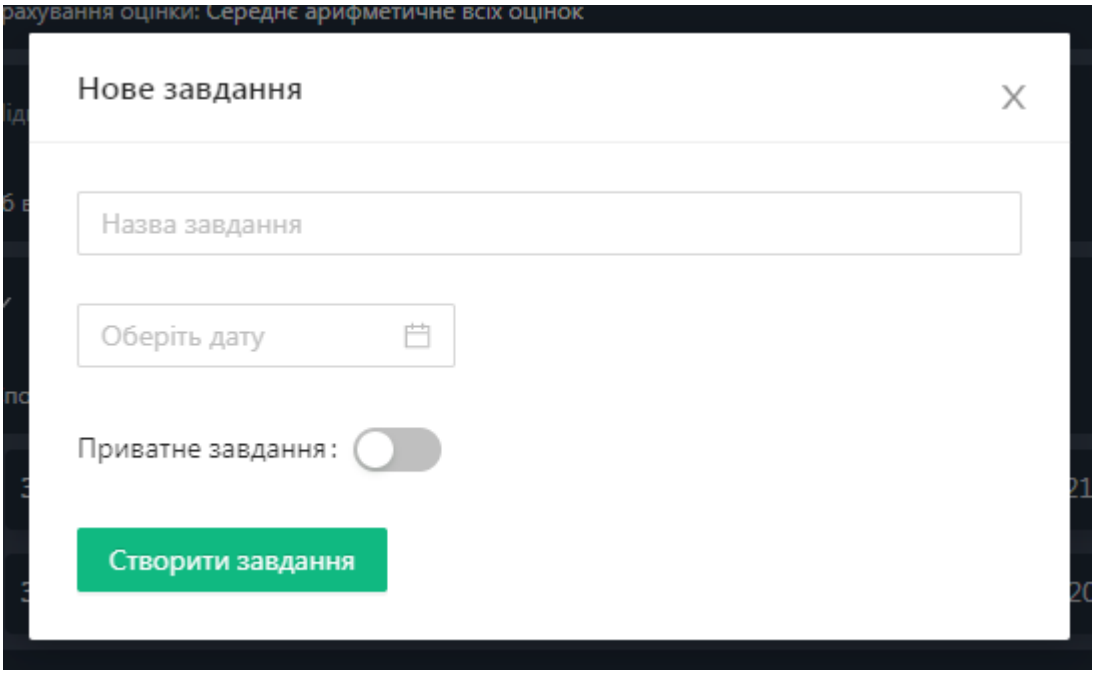

Рисунок 3.18 – Форма створення нового завдання

Завдання можна створювати як на рівні курсу, так і в будь-яких групах. Форми створення груп, підгруп та модулів схожі та вимагають одні й ті ж дані, а тому буде наведено лише приклади для груп.

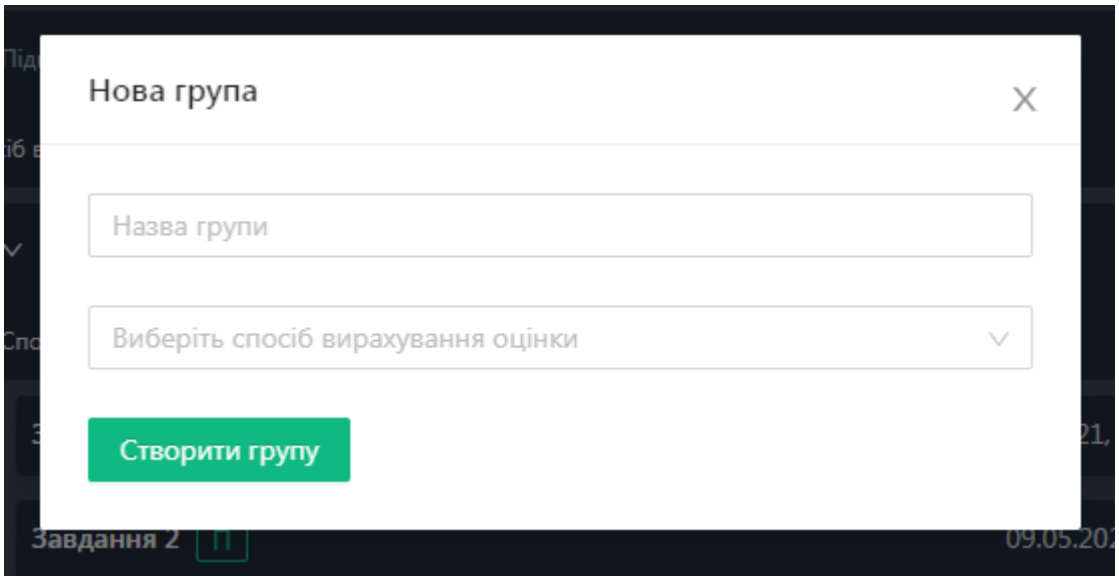

Рисунок 3.19 – Форма створення нової групи завдань

Для завдань та груп присутні також модальні вікна з формами для редагування інформації про них. Важливо відмітити, що для завдань є можливість змінювати тип завдання з приватного на публічне і навпаки.

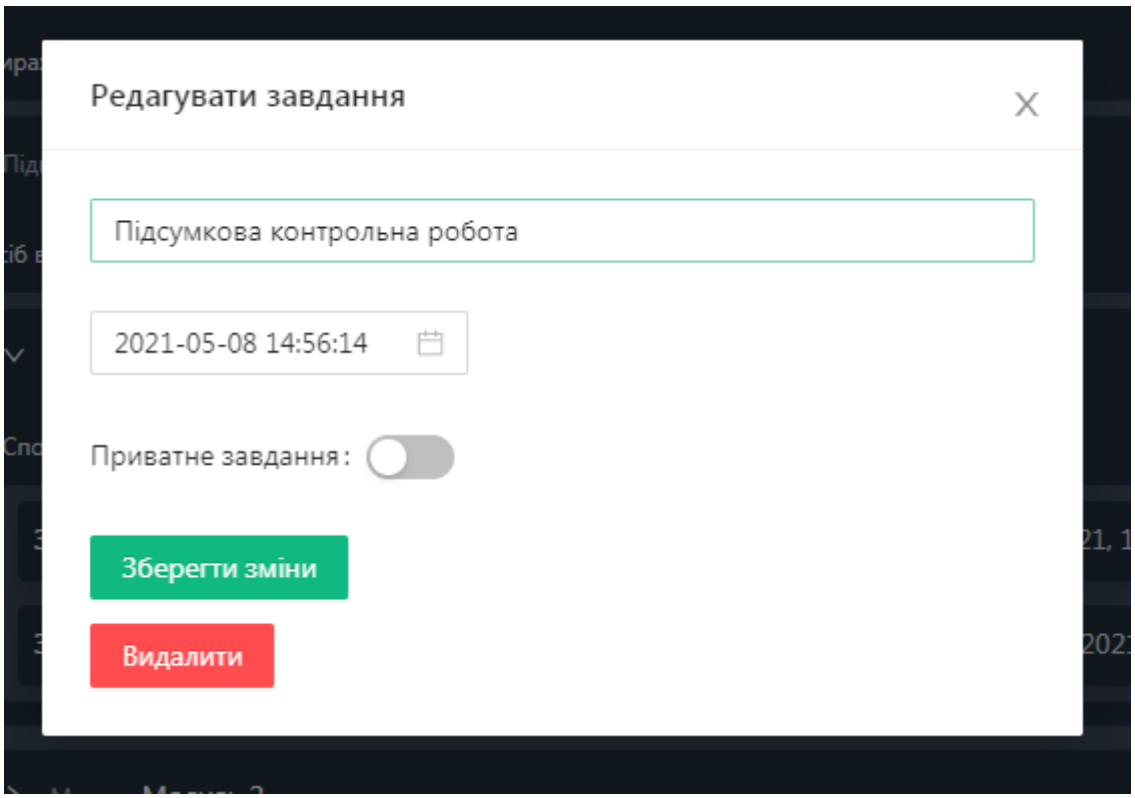

Рисунок 3.20 – Форма редагування завдання

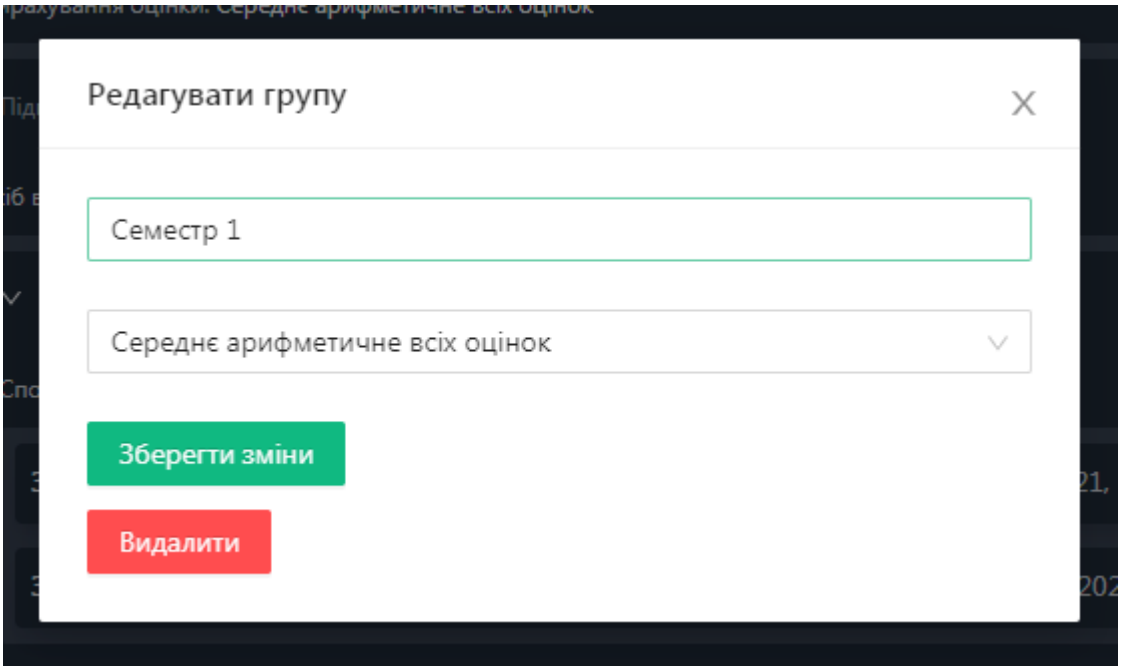

Рисунок 3.21 – Форма редагування групи

Для завдань, груп, підгруп, модулів та курсів присутні модальні вікна з оцінками учнів. Оцінки за завдання мають наступний вигляд:

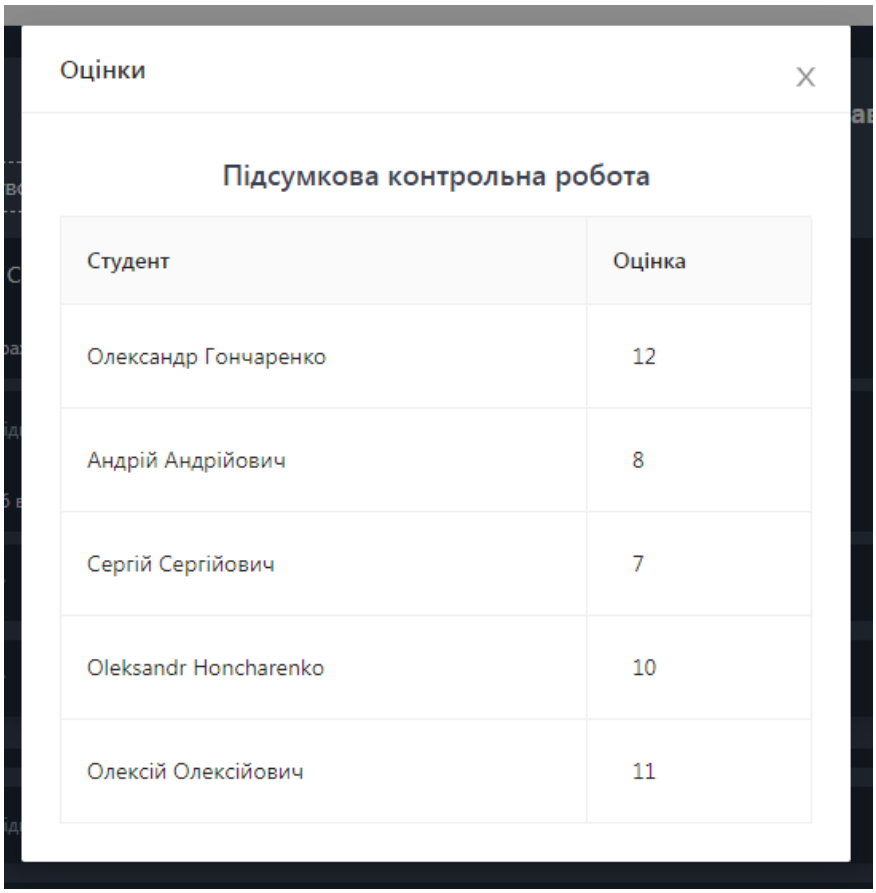

Рисунок 3.22 – Оцінки за завдання

Оцінки за всі інші сутності мають наступний вигляд з додатковим стовпчиком з автоматично вирахуваною оцінкою за цю сутність. При необхідності викладач може заповнити другий стовпчик змінивши оцінку користувача. Другий стовпчик має більший пріоритет в системі при вирахуванні автоматичної оцінки.

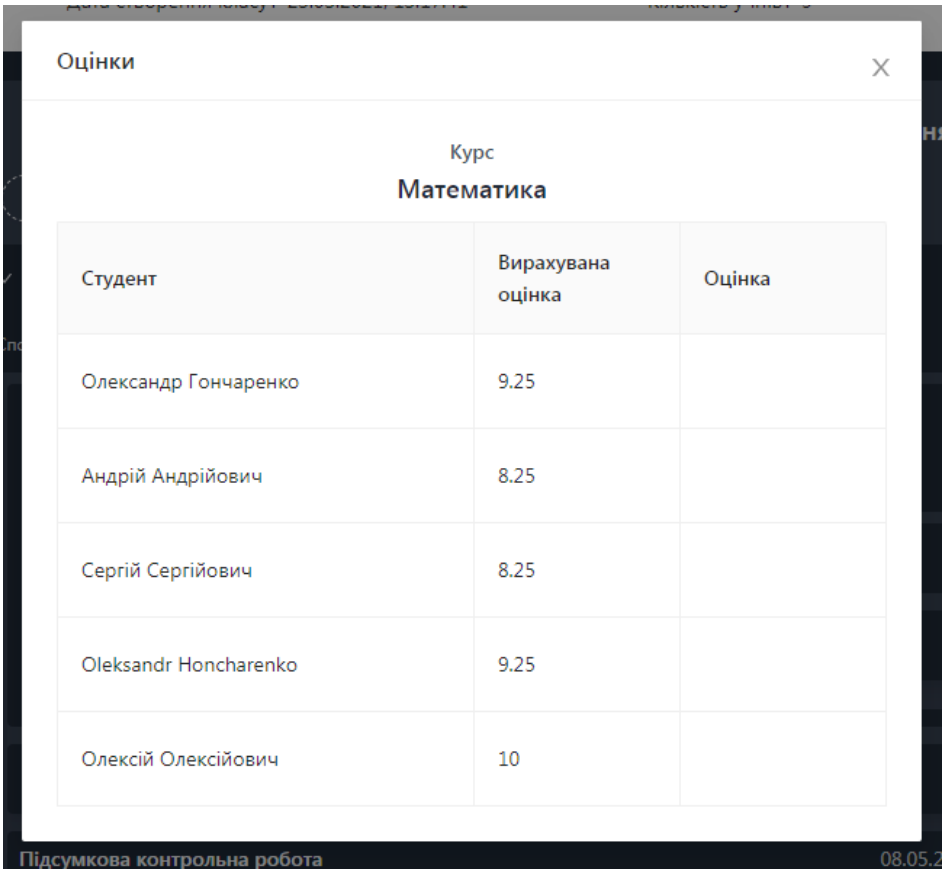

Рисунок 3.23 – Оцінки за курс

Тепер зайдемо в систему як інший користувач, який є учнем в одному класі та викладачем (не керівником) в іншому.

Рисунок 3.24 – Список класів для іншого користувача

Зайдемо в клас, де користувач навчається.

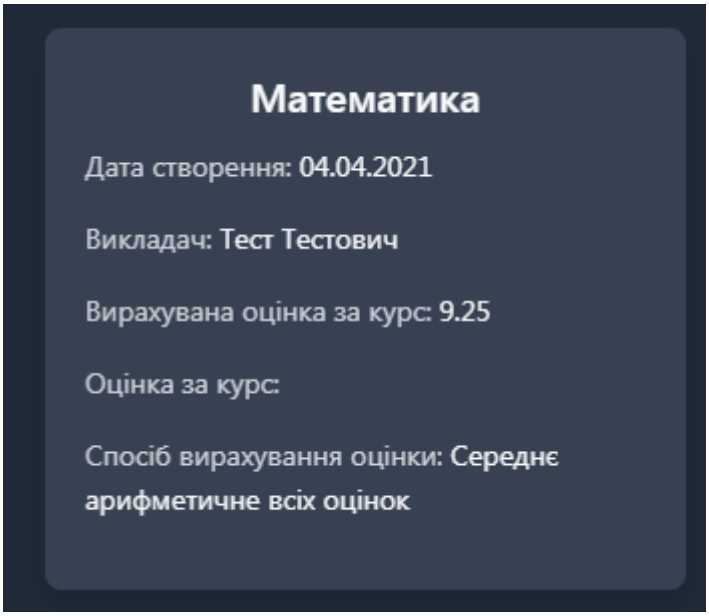

Рисунок 3.25 – Інформація про курс для учнів

Нижче наведено вигляд сторінки завдань для учнів. Біля кожного завдання наведено оцінку учня, біля груп наведено вирахувану та фінальну оцінки. При наведенні курсора на оцінки з’являється підказка.

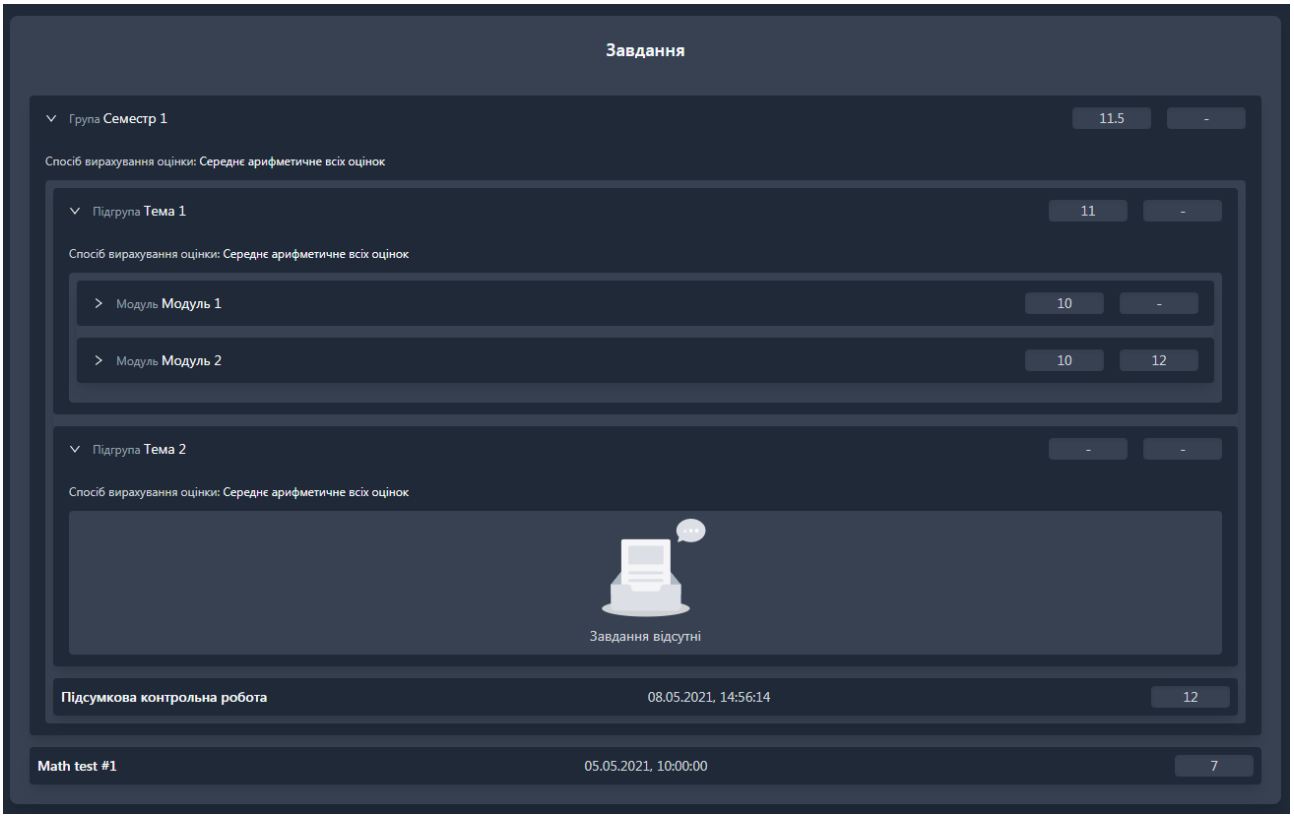

Рисунок 3.26 – Список завдань для учнів

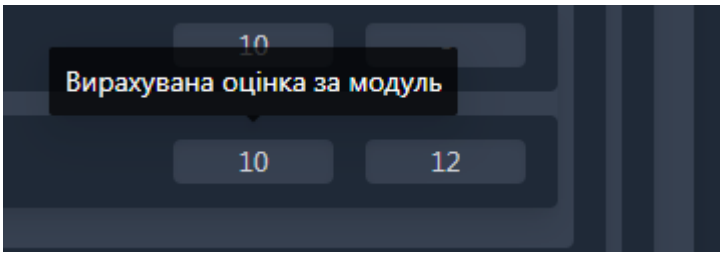

Рисунок 3.27 – Підказка про тип оцінки

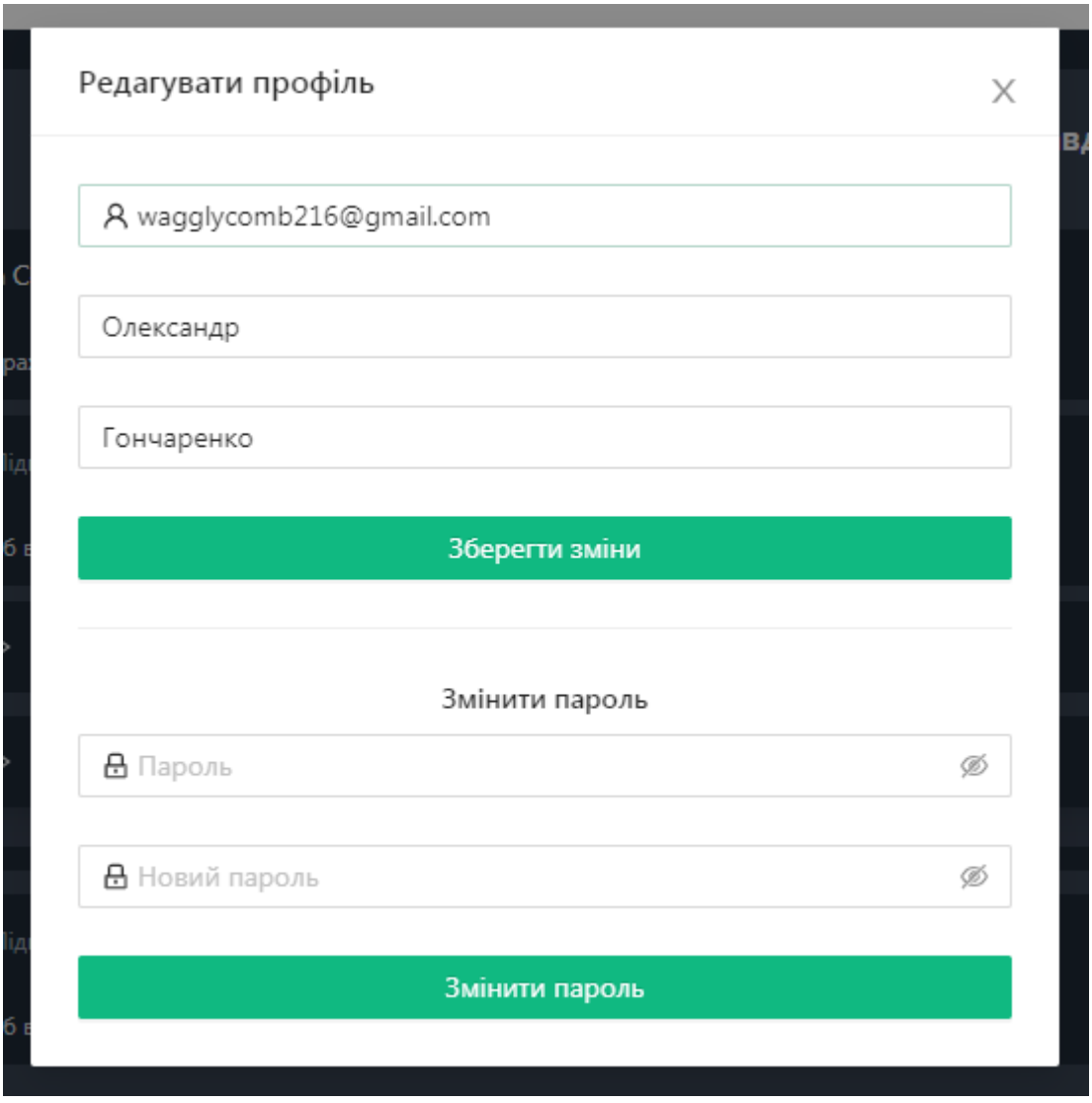

Рисунок 3.28 – Зміна інформації про користувача

ВИСНОВКИ

За результатами виконання кваліфікаційної роботи було створено робочий прототип програмної системи для управління оцінками.

Було проведено дослідження технологій, програмно-технологічних рішень та підходів для розроблення сучасних веб-застосунків та їх аналіз, визначено які саме технології та як потрібно застосувати та реалізувати для виконання поставленої мети створення програмної системи. Було спроектовано та створено клієнтську та серверну частини системи за допомогою вивчених фреймворків, бібліотек та технологій для створення відповідних застосунків. Отримані знання в подальшому можуть бути застосовані при розробці будь-яких програмних систем з клієнт-серверною архітектурою.

Дана система зберігає в базі даних MongoDB всю інформацію про користувачів системи, створені ними класи, курси цих класів, завдання та групи завдань всередині курсів, а також всі оцінки учнів пов’язані з ними. Базу даних було створено та розміщено в хмаровому сервісі. Серверна частина реалізована за допомогою фреймворка NestJS та мови програмування TypeScript. Клієнтська частина реалізована з використанням JS-бібліотеки React, додаткових бібліотек та мови програмування TypeScript.

Було отримано навички створення веб-сервісів, вивчено базові принципи створення API контролерів та моделей даних з баз даних за допомогою обраних технологій та бібліотек. Було отримано та покращено досвід роботи з JS-бібліотекою React та покращено знання про HTML, CSS та JS, вивчено нові бібліотеки з екосистеми React.

Створена система реалізує новий підхід до створення електронних щоденників. При цьому система залишає за собою можливості до розширення її функціональних можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Современный учебник JavaScript [Електронний ресурс]. – Режим доступу : https://learn.javascript.ru/ (дата звернення: 20.03.2021);
2. REST [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/REST (дата звернення: 09.04.2021);
3. Carl Rippon. ASP.NET Core 3 and React: Hands-On full stack web development using ASP.NET Core, React, and TypeScript 3 : навч. посіб. Birmingham B3 2PB, UK, 2019. 768 c.;
4. MySQL и MongoDB — когда и что лучше использовать [Електронний ресурс]. – Режим доступу : https://habr.com/ru/post/322532/ (дата звернення: 02.03.2021);
5. Client–server model [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Client%E2%80%93server_model (дата звернення: 04.03.2021);
6. Database [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Database (дата звернення: 08.05.2021);
7. npm [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/Npm (дата звернення: 18.05.2021);
8. TypeScript [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/TypeScript (дата звернення: 10.03.2021);
9. Онлайн-руководство по MongoDB [Електронний ресурс]. – Режим доступу : https://metanit.com/nosql/mongodb/ (дата звернення: 02.03.2021);
10. CSS-Tricks [Електронний ресурс]. – Режим доступу : https://css-tricks.com/ (дата звернення: 07.04.2021);
11. What is NoSQL? [Електронний ресурс]. – Режим доступу : https://www.mongodb.com/nosql-explained (дата звернення: 05.03.2021);
12. JavaScript [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/JavaScript (дата звернення: 28.04.2021);

13. JSON Web Token [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/JSON_Web_Token (дата звернення: 18.03.2021);
14. Conventional Commits [Електронний ресурс]. – Режим доступу : https://www.conventionalcommits.org/en/v1.0.0/ (дата звернення: 18.05.2021);
15. Introduction to JSON Web Tokens [Електронний ресурс]. – Режим доступу : https://jwt.io/introduction/ (дата звернення: 18.03.2021);
16. MDN CSS: Cascading Style Sheets [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 06.04.2021);
17. MDN HTML: Hypertext Markup Language [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 06.04.2021);
18. MDN HTTP [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 12.04.2021);
19. MDN JavaScript [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 06.04.2021);
20. React [Електронний ресурс]. – Режим доступу : https://reactjs.org/ (дата звернення: 03.04.2021);
21. React [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/React (дата звернення: 03.05.2021);
22. React-Router [Електронний ресурс]. – Режим доступу : https://reacttraining.com/react-router/ (дата звернення: 17.03.2021);
23. The world's largest web developer site [Електронний ресурс]. – Режим доступу : https://www.w3schools.com/ (дата звернення: 06.03.2021);
24. MongoDB [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/MongoDB (дата звернення: 04.05.2021);

25. The MongoDB 4.4 Manual [Електронний ресурс]. – Режим доступу : https://docs.mongodb.com/manual/ (дата звернення: 06.03.2021);
26. MongoDB офіційний сайт [Електронний ресурс]. – Режим доступу : https://www.mongodb.com/ (дата звернення: 06.03.2021);
27. Software framework [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Software_framework (дата звернення: 18.05.2021);
28. Simple, unobtrusive authentication for Node.js [Електронний ресурс]. – Режим доступу : http://www.passportjs.org/ (дата звернення: 05.03.2021);
29. Node.js [Електронний ресурс]. – Режим доступу : https://nodejs.org/en/ (дата звернення: 02.03.2021);
30. Typed JavaScript at Any Scale [Електронний ресурс]. – Режим доступу : https://www.typescriptlang.org/ (дата звернення: 04.03.2021);
31. TypeScript [Електронний ресурс]. – Режим доступу : https://github.com/microsoft/TypeScript/#readme (дата звернення: 04.03.2021);
32. npm [Електронний ресурс]. – Режим доступу : https://www.npmjs.com/ (дата звернення: 02.03.2021);
33. Что такое npm [Електронний ресурс]. – Режим доступу : https://proglib.io/p/chto-takoe-npm-gayd-po-node-package-manager-dlya-nachinayushchih-2020-07-21 (дата звернення: 02.03.2021);
34. NestJS-тот самый, настоящий бэкенд на nodejs [Електронний ресурс]. – Режим доступу : https://habr.com/ru/post/439434/ (дата звернення: 22.02.2021);
35. NestJS [Електронний ресурс]. – Режим доступу : https://github.com/nestjs/nest (дата звернення: 22.02.2021);
36. NestJS documentation [Електронний ресурс]. – Режим доступу : https://docs.nestjs.com/ (дата звернення: 22.02.2021);
37. Endpoint [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Endpoint (дата звернення: 20.02.2021);

38. Service-oriented architecture [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Service-oriented_architecture (дата звернення: 19.02.2021);
39. Document Object Model [Електронний ресурс]. – Режим доступу : https://en.wikipedia.org/wiki/Document_Object_Model (дата звернення: 20.05.2021);
40. Tailwind CSS [Електронний ресурс]. – Режим доступу : https://tailwindcss.com/ (дата звернення: 10.03.2021);
41. CSS Utility Classes and "Separation of Concerns" [Електронний ресурс]. – Режим доступу : https://adamwathan.me/css-utility-classes-and-separation-of-concerns/ (дата звернення: 10.03.2021);
42. Mongoose [Електронний ресурс]. – Режим доступу : https://mongoosejs.com/ (дата звернення: 05.03.2021);
43. styled-components [Електронний ресурс]. – Режим доступу : https://styled-components.com/docs (дата звернення: 11.04.2021);
44. Ant Design [Електронний ресурс]. – Режим доступу : https://ant.design/components/overview/ (дата звернення: 12.03.2021);
45. Create React App [Електронний ресурс]. – Режим доступу : https://create-react-app.dev/ (дата звернення: 06.03.2021);
46. Axios [Електронний ресурс]. – Режим доступу : https://github.com/axios/axios (дата звернення: 09.03.2021);
47. Redux Toolkit [Електронний ресурс]. – Режим доступу : https://redux-toolkit.js.org/ (дата звернення: 15.03.2021);
48. Redux [Електронний ресурс]. – Режим доступу : https://redux.js.org/ (дата звернення: 15.03.2021);
49. React Redux [Електронний ресурс]. – Режим доступу : https://react-redux.js.org/ (дата звернення: 15.03.2021);
50. GitLab CI/CD [Електронний ресурс]. – Режим доступу : https://docs.gitlab.com/ee/ci/README.html (дата звернення: 12.05.2021);