МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ РЕПЕТИТОРСЬКОГО ЦЕНТРУ ЗАСОБАМИ REACT ТА MONGODB»**

Виконала: студентка 2 курсу, групи 8.1214-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.О. Васильєва
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент в.о. завідувача кафедри комп'ютерних наук, доцент, к.пед.н. Пшенична О.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Васильєвій Дарії Олексіївні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку для репетиторського центру засобами React та MongoDB

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 03.12.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та вибір засобів реалізації.
2. Моделювання проєкту.
3. Особливості реалізації та результати тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	21.03.2025	
2.	Збір вихідних даних.	18.04.2025	
3.	Обробка методичних та теоретичних	30.05.2025	
	джерел.		
4.	Розробка першого та другого розділу.	08.09.2025	
5.	Розробка третього розділу.	24.11.2025	
6.	Оформлення та нормоконтроль	01.12.2025	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	17.12.2025	

Студент _____ Д.О. Васильєва
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку для репетиторського центру засобами React та MongoDB»: 59 с., 30 рис., 22 джерела, 2 додатки.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, РЕПЕТИТОРСЬКИЙ ЦЕНТР, EXPRESS, FULL-STACK, JAVASCRIPT, MONGODB, NODEJS, REACT.

Об’єкт дослідження – процес розробки платформи для потреб репетиторського центру.

Предмет дослідження – мова програмування JavaScript, фреймворки React та Node.js, база даних MongoDB, налаштування взаємодії між ними.

Мета роботи: змоделювати та розробити додаток із певним набором функцій для репетиторського центру за допомогою фреймворків React для клієнтської частини та Node.js для серверної частини на основі системи керування базами даних MongoDB.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, конструювання та тестування програмного забезпечення.

У кваліфікаційний роботі реалізовано вебзастосунок для репетиторського центру, використано наступні інструменти: мова програмування JavaScript, фреймворки React, Express, Node.js, Tailwind, СКБД MongoDB. У результаті отримано повноцінний функціональний застосунок, який дозволяє керувати освітнім процесом у репетиторському центрі STEAM City у місті Запоріжжя. На перспективу можна додати наступний функціонал: оплату за заняття безпосередньо у застосунку, публікація матеріалів, чат тощо.

SUMMARY

Master's qualifying paper "Development of a Web Application for a Tutoring Center Using React and MongoDB": 59 pages, 30 figures, 22 references, 2 supplements.

DATABASE, EXPRESS, FULL-STACK, JAVASCRIPT, MONGODB, NODEJS, REACT, TUTORING CENTER, WEBAPPLICATION.

The object of the study is the process of developing a platform to meet the needs of a tutoring center.

The subject of the study is the JavaScript programming language, the React and Node.js frameworks, the MongoDB database, and the configuration of interaction between them.

The aim of the study is to model and to develop an application with a specific set of functions for a tutoring center using the React framework for the client side and Node.js for the server side. based on the MongoDB database management system.

The methods of research are methods of collecting and analyzing software requirements, methods of modeling, designing, and testing software.

A web application has been implemented and the tools have been used, such as JavaScript programming language, React, Express, Node.js, Tailwind frameworks, and MongoDB DBMS. As a result, a fully functional application has been obtained, which makes it possible to manage the educational process at the STEAM City tutoring center in Zaporizhzhia. In the future, the following functionality can be added: payment for classes directly in the application, publication of the materials, chat etc.

ЗМІСТ

Завдання на кваліфікаційну роботу ……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області та вибір засобів реалізації 11
1.1 Опис предметної області 11
1.2 Огляд сучасних освітніх рішень у контексті дистанційного навчання ……...12
1.3 Вибір технологій для розробки вебзастосунку 14
1.3.1 Мова програмування JavaScript..... 14
1.3.2 Фреймворк React 16
1.3.3 Node.js та Express 17
1.3.4 Система керування базами даних MongoDB 18
1.3.5 CSS-фреймворк Tailwind..... 19
1.4 Висновки до розділу 1 20
2 Моделювання проєкту 21
2.1 Аналіз вимог 21
2.2 Розробка архітектури сайту 22
2.2.1 Формулювання загальної структури системи..... 22
2.2.2 Моделі взаємодії та сценарії використання 24
2.2.3 Структура бази даних 27
2.3 Висновки до розділу 2 29
3 Реалізація та тестування 30
3.1 Загальна структура проєкту 30
3.2 Розробка фронтенд-частини..... 30
3.3 Розробка бекенд-частини 38
3.4 Тестування функціональності, продуктивності та безпеки..... 41

3.5 Оптимізація та адаптивність застосунку для різних пристроїв 44
3.6 Розгортання в мережі 48
3.7 Висновки до розділу 3 49
Висновки 51
Перелік посилань 53
Додаток А Будова сторінок у клієнтській частині 56
Додаток Б Реалізація серверної частини 59

ВСТУП

У сучасному світі цифрових технологій онлайн-середовище активно проникає в усі сфери життя, зокрема, й у освіту. Протягом останніх п’яти років стрімко розвивається дистанційне навчання, адаптуючись під зовнішні умови: карантинні обмеження, спричинені пандемією COVID-19, і повномасштабне вторгнення не дозволяють у повній мірі повернутися до традиційної освіти в очному форматі. І саме з розвитком дистанційного навчання та індивідуального підходу до здобуття знань зростає попит на ефективні та гнучкі рішення. Так, наприклад, вже декілька років користується попитом Google Classroom, Moodle тощо. Вони роблять комунікацію між учнями та викладачами набагато простіше, дозволяють зберігати навчальні матеріали. Вони чудово підходять для шкіл та університетів. Однак є певні заклади, для яких даного функціоналу недостатньо.

Так, наприклад, є запит від місцевого репетиторського центру STEAM City, який хоче мати свій власний застосунок, де буде загальна інформація про заклад, окремий кабінет кожного викладача та учня, ресурси для навчання тощо. Така проблема виникла внаслідок того, що для ведення розкладу вони використовують Notion, для онлайн занять – Google Meet або Zoom, ресурси для навчання кожен викладач зберігає у себе окремо та індивідуально надсилає учням. Це достатньо незручно, адже потребує постійно мати з собою пристрій, на якому збережені всі матеріали; або тримати все на Google акаунті, що теж є не найбезпечнішим варіантом через постійне підключення свого профілю до різних пристроїв.

Отже, поставлена задача з розробки застосунку поєднує в собі декілька основних функцій: керування заняттями, комунікацію між викладачем і учнем, перелік матеріалів (при чому як завдань, які просто використовуються під час уроку, так і повноцінних збірників чи методичних посібників, які доступні до купівлі). Окрім цього, так як розробка стосується саме конкретного

репетиторського центру, необхідно створити стандартні сторінки, де можна ознайомитися з його діяльністю.

Таким чином, **актуальність** проєкту полягає в безпосередньому запиті замовника та своєчасності викликів сьогодення. Його **метою** є моделювання та розробка додатку для репетиторського центру з реалізацією всіх визначених функцій за допомогою фреймворків React для створення клієнтської частини та Node.js – для серверної, зберігаючи при цьому всі дані в базі даних на основі MongoDB.

Для досягнення даної мети необхідно реалізувати наступні **завдання**:

- проаналізувати предметну область і вже існуючі проєкти в мережі, інструменти та технології, визначені метою;
- спроєктувати схему сутностей та моделей, базу даних, загальну структуру додатку;
- розробити всі необхідні діаграми для ілюстрації процесів, функціоналу (діаграми прецедентів, діяльностей);
- розробити застосунок відповідно до вимог замовника;
- протестувати розроблений продукт;
- представити продукт замовнику, за наявності виправити всі неточності та зауваження.

Об’єктом дослідження є процес розробки платформи для потреб репетиторського центру.

Предмет дослідження – мова програмування JavaScript, фреймворки React та Node.js, база даних MongoDB, налаштування взаємодії між ними.

Під час роботи застосовувалися наступні **методи** дослідження: методи збору та аналізу вимог до програмного забезпечення, методи моделювання, конструювання та тестування програмного забезпечення.

Для досягнення мети було обрано фреймворки React та Node.js на основі JavaScript, оскільки вони є чудовим інструментом для роботи з онлайн додатками. Їхнє використання забезпечує наявність та підключення компонентів, маршрутизацію між сторінками, зв’язок між серверною та

клієнтською частинами. Зберігати дані про учнів, викладачів і заняття було вирішено в базі даних на основі NoSQL системи керування базами даних MongoDB. Нереляційні бази даних дозволяють легко вносити зміни до записів та до самої структури таблиці, що, зі свого боку, дозволяє швидко адаптувати вже розроблений проєкт до змін. Для швидшої та зручнішої стилізації проєкту замість класичного ручного написання було підключено CSS-фреймворк Tailwind.

Основні положення і результати кваліфікаційної роботи доповідались, обговорювалися та знайшли схвалення на Шістнадцятій Всеукраїнській, Двадцять третій регіональній науковій конференції молодих дослідників «Актуальні проблеми математики та інформатики» (Запоріжжя, 24-25 квітня 2025 р.), за результатами якої опубліковано тези «Вебзастосунок для оптимізації роботи репетиторського центру: досвід розробки засобами React та MongoDB».

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань та двох додатків. Загальний обсяг роботи становить 59 сторінок. Додатки викладено на 4 сторінках. У переліку посилань 22 використаних джерела.

У першому розділі наведені основні теоретичні дані, а саме опис предметної області, аналіз існуючих освітніх платформ та визначення інструментарію розробки. У другому розділі описується аналіз вимог, моделювання проєкту, наводяться всі необхідні діаграми. У третьому розділі висвітлено процес розробки та тестування, наведено фрагменти коду та ілюстрації реалізованого проєкту, а також процес розгортання в мережі. У додатках наведено програмну реалізацію.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

У цьому розділі наведено порівняльний аналіз існуючих платформ, їхнього функціоналу, на основі чого змодельований проєкт. Також описано технології, які було обрано для реалізації. Вибір саме таких засобів обумовлений забезпеченням надійної взаємодії і зручністю в розробці та адаптації.

1.1 Опис предметної області

Предметною областю дослідження є діяльність репетиторського центру STEAM City в Запоріжжі, де передбачено організацію та проведення індивідуальних та групових занять для учнів різних вікових категорій і з різних предметів. Основними учасниками є адміністратор, учень (безпосередньо сам здобувач або його батьки) та вчитель. Кожен із учасників взаємодіє із центром у межах певних завдань, із яких складається загальна структура роботи закладу.

Серед процесів репетиторського центру можна виділити наступні основні: формування розкладу, ведення обліку учнів, фіксація відвідування, спілкування між учасниками освітнього процесу. Однак, як зазначалося вище, для фіксації відвідування використовується одна платформа, для фіксації оплат – інша, кожен викладач окремо надає доступ до своїх матеріалів кожному учню індивідуально тощо.

Використання багатьох інструментів призводить до неузгодженості інформації, складності в комунікації та проблем із доступом до навчальних матеріалів.

Крім того, даний репетиторський центр працює в змішаному форматі, тому критичним є питання отримання оперативних даних щодо змін у розкладі чи навчальних матеріалах. Відсутність єдиної платформи значно ускладнює

адміністрування, збільшує навантаження на персонал, знижує ефективність комунікації, і, як наслідок, ефективність роботи загалом.

Отже, предметна область передбачає потребу у створенні застосунку, який дозволить поєднати ключові функції – ведення розкладу, контроль оплат, керування учнями, зберігання навчальних матеріалів. Також у процесі розвитку центру зростає кількість учнів, що робить потребу у конкретному одному сервісі критичною. Реалізація такого рішення забезпечить оптимізацію внутрішніх процесів репетиторського центру та підвищення якості надання освітніх послуг.

1.2 Огляд сучасних освітніх рішень у контексті дистанційного навчання

Підготовчим етапом до реалізації проєкту є ознайомлення з аналогічними готовими рішеннями в Інтернет-мережі. Аналіз останніх наукових публікацій також підтверджує актуальність використання освітніх платформ для організації дистанційного навчання.

У статті «Цифрові освітні платформи як інструмент реалізації неформальної освіти» автори наводять дуже багато теоретичних матеріалів про неформальну освіту та її популярність [17], що є актуальним для обраної теми кваліфікаційної роботи. Також звертається увага на відкриті освітні ресурси, їхні елементи та типи. Наводяться конкретні приклади різних платформ, на яких розміщені курси, причому як іноземні, так і українські. Однак більшість із них уже є повноцінним готовим набором теорії і завдань, і адаптувати їх – окремі витрати часу (якщо це взагалі доступно).

Звернемося до статті Н.І. Корильчук, М.П. Первака та Т.Ю. Чернової [15]. Дослідники описують переваги дистанційного навчання та наводять приклади платформ із різним функціоналом. У статті надано мінімум статистичної інформації та названо лише найпопулярніші платформи.

У своїй статті «Використання освітніх платформ в освітньому середовищі» Інна Яковлєва проводить опитування та аналіз вчителів із різних областей і з різним стажем роботи. Авторка наводить низку статистичних даних, де виокремлює, що найбільше використовується Google Classroom для публікації завдань і Google Meet для онлайн зустрічей. Також користується популярністю платформа Zoom для спілкування та Learning.ua для використання вже готових завдань. Як віртуальна дошка найчастіше використовується Canva та Jamboard, а для тестів – LearningApps [22]. Отже, це підтверджує тезу про використання багатьох ресурсів водночас, що не завжди є зручним.

Ознайомлюючись зі статтею М. Карпушиної і І. Шумило, було виявлено ще ряд додаткових платформ для створення інтерактивних завдань при вивченні іноземних мов [12]. Так, враховуючи, що одним із напрямків репетиторського центру є філологія, то інтеграція даних сервісів також є доцільною.

У матеріалах К.О. Січкаренка описуються освітні онлайн платформи, які автор класифікує за різними ознаками (за вартістю, часом, авторством, кількістю слухачів, процесом навчання). Він перечислює також існуючі ресурси, які вже наповнені контентом, чи які можна наповнювати самостійно. Окрім цього автор порівнює міжнародний ринок послуг з дистанційної освіти та проводить порівняльний аналіз існуючих платформ із повноцінним наповненням курсів [20].

Стаття Д.Л. Широкова ілюструє різновидність освітніх платформ, аналогічно до попереднього класифікуючи їх на вже готові і ті, що необхідно наповнювати самостійно [21].

Любомир Сірський описує зміни в освіті, як впливають Інтернет-ресурси на навчання, про Нову українську школу, і зокрема для чого використовується платформа Kwiga [19]. І дійсно, даний ресурс виглядає досить корисно на відміну від всіх попередніх – він дозволяє вести розклад, причому максимально детально, з веденням обліку оплати, відвідування, статистики тощо. Однак мінус у тому, що вона пропонується на комерційній основі, до

того ж у ній немає повного переліку функціоналу, який планується замовником.

Таким чином, проведений аналіз свідчить про доцільність і необхідність використання платформ для дистанційного навчання, однак на даному етапі не виявлено жодних посилань на конкретно існуючі платформи, які містять весь проаналізований раніше функціонал в одному місці. Сучасні освітні застосунки чудово виконують окремі функції, але не забезпечують комплексного підходу, необхідного для організації повноцінної роботи репетиторського центру. Реалізація запланованого продукту передбачатиме поєднання всіх визначених функцій.

1.3 Вибір технологій для розробки вебзастосунку

Для розробки якісного вебзастосунку необхідно обрати набір технологій, які забезпечать стабільну роботу, зручність розробки та подальше масштабування. Вибір інструментарію визначається як вимогами замовника, так і специфікою предметної області, яка передбачає обробку великої кількості даних, підтримку взаємодії між користувачами, збереження даних про розклад, оплату тощо.

У процесі аналізу визначено ключові критерії вибору: продуктивність, легкість у розгортанні. Особливо важливим є питання інтеграції клієнтської та серверної частини, оскільки застосунок повинен працювати як цілісна система. Також необхідно врахувати можливість подальшого розширення платформи додатковим функціоналом без перепланування всієї системи.

1.3.1 Мова програмування JavaScript

JavaScript – одна з найпопулярніших мов програмування на сьогодні. Початок її створення припадає на 1995 рік, розробником є Brendan Eich. З того

часу вона змінила декілька назв, але основний принцип був у потребі додавання інтерактивності до статичних сторінок. Якщо спочатку це було можливо тільки на клієнтській частині, то з часом це можна реалізувати і на сервері [5].

На сучасному ринку веброзробки JavaScript використовується для великої кількості проєктів, від невеликих допоміжних додатків до великих відомих платформ. За даними рейтингів мов програмування JavaScript стабільно входить до трійки найпопулярніших мов у світі. За статистикою лютого 2025 року ця мова займала одне з провідних місць серед розробників, а в лютому 2024 року – очолювала список [18]. Така тенденція пояснюється певними факторами, які свідчать про гнучкість, універсальність і широке застосування JavaScript.

По-перше, навіть розробляючи верстку на мові розмітки HTML, можна додати певну динаміку за допомогою скриптів – програм написаних на JavaScript. Дана мова програмування дуже легко інтегрується і з HTML, і з CSS [11].

По-друге, для запуску коду не потрібно використовувати додатковий компілятор, все виконується безпосередньо в браузері. Всі сучасні середовища підтримують JavaScript.

По-третє, вона достатньо легка для навчання та зручна у використанні, оскільки містить логічний і гнучкий синтаксис.

До того ж, зараз існує велика кількість фреймворків на основі JavaScrypt. Кожен із них має свій певний функціонал, який окремо встановлюється та використовується. Так, зокрема, Node.js, express та React використовуються в роботі для забезпечення роботи серверної та клієнтської частини відповідно. Роботу з MongoDB забезпечує фреймворк mongoose тощо.

Окрім фреймворків останнім часом з’являються також нові мови, які конвертуються в JavaScript. До таких належать TypeScript, Kotlin, CoffeeScript, Flow тощо.

Таким чином, вибір JavaScript та фреймворків на її основі є обґрунтованим та актуальним, а її поширеність і підтримка в галузі гарантують

стабільність і перспективність. Тому розробка даного проєкту з використанням зазначених бібліотек є сучасним підходом.

1.3.2 Фреймворк React

Структура сучасного сайту, як правило, включає повторювані елементи, які присутні на більшості або навіть на всіх сторінках. До таких елементів належать, як приклад, заголовок, який найчастіше містить меню навігації, підвал з інформацією про рік, контактні дані, посилання на соціальні мережі або автора, а також інші допоміжні блоки – бокове меню, спливаючі вікна тощо.

У випадку класичної верстки за допомогою HTML подібні елементи необхідно копіювати вручну на кожну сторінку сайту, що створює проблеми і незручності. Так, наприклад, найменша зміна в одному елементі вимагає редагування кожної сторінки окремо.

Натомість фреймворк React дозволяє ефективно вирішити дану проблему завдяки компонентому підходу. Компоненти – це незалежні, повторно використовувані частини інтерфейсу, які функціонують окремо один від одного [14]. Таким чином можна створити компонент і використовувати його на всіх потрібних сторінках, вказавши у відповідному місці.

Ще однією перевагою є уніфікація коду. Тобто якщо з’являється необхідність змінити певний компонент, достатньо відредагувати лише конкретний файл із кодом, і зміни автоматично будуть внесені всюди, де він підключений.

Також React дозволяє передавати дані в компоненти через властивості. Таким чином у залежності від певних умов один і той самий компонент може виглядати по-іншому. Так, наприклад, може змінюватися стиль конкретного компонента через підключення його на декількох сторінках із різними властивостями.

Як бачимо, даний фреймворк має достатню кількість переваг, і цілком справедливо, що він залишається одним із найпопулярнішим у використанні протягом довгого часу. Як зазначає Андрій Козюля у своїй статті: «React – це бібліотека для створення інтерфейсів від Meta» [13]. Він також пише, що цей фреймворк чудово використовувати при роботі з великими інтерактивними проєктами, тож для розробки додатку для репетиторського центру він буде вдалим варіантом.

1.3.3 Node.js та Express

Node.js – це середовище виконання JavaScript поза межами браузера. Використання цього фреймворку дозволяє достатньо зручно розробити динамічні вебсторінки. Сценарії для створення таких сторінок виконують запити на стороні сервера, тому це значно пришвидшує роботу додатка та зменшує навантаження [16].

Серед переваг цього фреймворку виділяють легкість у вивченні, оскільки він використовує JavaScript, масштабування для обробки кількох підключень водночас, швидкість завантаження. Також можна виокремити менеджер пакетів npm, який використовується для швидкого підключення готових модулей для розв’язання різних задач.

Node.js використовується у розробці додатків багатьох відомих компаній, таких як Netflix, PayPal, LinkedIn тощо. Застосування ними даної технології підтверджує його переваги та свідчить про його стабільність, масштабованість і практичність у реальних проєктах.

Для організації серверної частини також використовують Express.js. Це досить потужний інструмент, який містить велику кількість корисних функцій для розробки додатків і спрощує роботу з HTTP-запитами. Він заснований на вищезгаданому Node.js, і, відповідно, теж є достатньо нескладний у освоєнні [2].

Однією з ключових функцій Express є маршрутизація – можливість визначати різні шляхи для клієнтських запитів. Це забезпечує обмін даними між клієнтською частиною, компонентами застосунку та базою даних.

До того ж, цей інструмент і дозволяє під’єднатися до обраної системи керування базами даних за допомогою відповідної бібліотеки. Додатково він має інші бібліотеки, які дозволяють налаштувати безпеку застосунку, забезпечити роботу з конфігураційними файлами тощо.

Таким чином, поєднання Node.js та Express створює надійну основу для серверної частини вебзастосунку, дозволяючи забезпечити швидку обробку запитів, гнучкість у розробці та можливість розширення функціоналу в майбутньому.

1.3.4 Система керування базами даних MongoDB

Максим Вальченко, описуючи MongoDB, формулює свою думку наступним чином: «MongoDB – це найбільш популярна документоорієнтована СКБД із сімейства NoSQL» [10].

Дана СКБД активно використовується у сучасних вебзастосунках завдяки своїй гнучкості, масштабованості та високій продуктивності при роботі з великими обсягами даних.

Однією з переваг є дуже легка інтеграція з JavaScript. Так, наприклад, після встановлення фреймворку mongoose можна під’єднувати базу даних, редагувати її, дивитися записи напряму в додатку.

Також можна виокремити гнучкість у структурі даних. На відміну від реляційних баз даних, MongoDB дозволяє змінювати модель і зв’язки протягом роботи. Це забезпечує високий рівень адаптивності у випадку змін вимог до проєкту.

Отже, обрана СКБД є сучасним та ефективним рішенням для зберігання інформації у вебзастосунках. У межах даного проєкту MongoDB використовується для ведення записів про викладачів, учнів, заняття тощо.

1.3.5 CSS-фреймворк Tailwind

Звичайно, жоден сучасний вебзастосунок не може обійтися без стилізації для забезпечення зручного користувацького інтерфейсу. Для цього слугують каскадні таблиці стилів – CSS, які виділяють як окрему мову. За допомогою CSS-файлів описують зовнішній вигляд сторінки: починаючи від шрифту сторінки, закінчуючи анімаціями. Стилі дозволяють легко змінити відображення одних і тих самих елементів, якщо їх застосувати до конкретних тегів або вказати класи. Також пишучи CSS можна зробити сторінки доступними та зручними для різних пристроїв – себто адаптивними [4].

Під час роботи з маленькими проєктами можна без проблем написати власні стилі для всього застосунку, окремо для сторінок, компонентів тощо. Однак зі збільшенням коду може зрости потреба в ручному написанні стилів для усіх елементів та класів. І для того, щоб зробити це швидше та ефективніше, можна використовувати CSS-фреймворки. Так, наприклад, існує фреймворк Tailwind, і він є дуже зручним та інтуїтивно зрозумілим у використанні. Як зазначено у статті sparklingman: «…він надає набір маленьких, цільових утилітних класів, які ви додаєте безпосередньо в HTML» [8]. І в цьому його перевага серед інших CSS-фреймворків, оскільки деякі з них (наприклад, Bootstrap) надають готові компоненти. Додатково, звичайні CSS-файли впливають на весь додаток або на весь компонент, у якому його підключено, і інколи це може заважати, тому доводиться переписувати стиль. Натомість, класи Tailwind призначаються конкретному елементу, що дозволяє легше зробити їх унікальними за потреби.

Загалом до переваг Tailwind можна віднести швидкість розробки, адже не потрібно перемикатися між файлами, вигадувати назву класу для конкретних елементів та писати стилі вручну. До того ж за допомогою вбудованих префіксів адаптивні стилі пишуться значно швидше. Фреймворк має детальну документацію з прикладами та наведенням коду в стандартному CSS-файлі [6].

1.4 Висновки до розділу 1

У результаті проведеного аналізу предметної області було визначено ключові процеси, які потребують автоматизації в роботі репетиторського центру. Проведене дослідження сучасних освітніх платформ засвідчило, що існуючі рішення не забезпечують реалізації всіх необхідних функцій в єдиному середовищі. Це підтвердило актуальність створення окремого вебзастосунку, орієнтованого на конкретні запити замовника.

Окрему увагу було приділено аналізу існуючих технологій для реалізації такого проєкту. На основі визначених критеріїв обрано конкретні інструменти: мова програмування JavaScript, а також фреймворки React для розробки якісної клієнтської частини, Node.js і Express для побудови серверної частини та забезпечення зв’язку між клієнтом і сервером, СКБД MongoDB для обробки та збереження всіх необхідних даних. Для пришвидшення і зручності розробки замість класичних файлів стилів було обрано CSS-фреймворк Tailwind.

Вибір зазначених технологій обумовлений їхніми очевидними перевагами та зручністю роботи з ними як окремо, так і разом. Це створює міцну основу для подальшого етапу, яким є визначення основних вимог та проєктування застосунку.

2 МОДЕЛЮВАННЯ ПРОЄКТУ

У цьому розділі наведено процес проєктування основних компонентів вебплатформи, яка розробляється відповідно до потреб репетиторського центру. Розглянуто вимоги до системи, побудовано її архітектуру, описано ключові сценарії взаємодії користувачів та сформовано структуру бази даних. Моделювання дозволяє узгодити функціональні можливості платформи, виявити залежності між її елементами та забезпечити цілісність під час подальшої реалізації.

2.1 Аналіз вимог

Аналіз вимог є ключовим етапом розробки, оскільки визначає перелік можливостей, які має забезпечувати майбутня платформа. На основі предметної області та очікувань замовника, які були зазначені в попередньому розділі, визначено цільові групи користувачів, основні функції, процеси, сценарії взаємодії та характеристики, яких необхідно дотримуватися під час реалізації.

До користувачів системи відносяться адміністратор (власник центру і, якщо передбачено, окремо адміністратор), викладач, учень (або батьки учня, на даному етапі вони мають такі ж налаштування доступу) та гість.

Вимоги можна розділити на функціональні та нефункціональні. *Функціональні вимоги*:

– ведення розкладу занять;
– створення та редагування навчальних матеріалів;
– створення завдань (або прикріплення зовнішніх);
– облік відвідуваності;
– облік оплат;
– особистий кабінет для кожної ролі (окрім гостя);

– повідомлення та сповіщення;

– можливість адміністрування користувачів.

Перелік *нефункціональних* вимог менший: зручний інтерфейс, адаптивність, можливість масштабування.

Також досить важливими є технічні вимоги до середовища функціонування застосунку. Тобто він повинен працювати у сучасних браузерах та підтримувати роботу на мобільних пристроях.

Суттєвим є питання безпеки даних, оскільки в акаунтах заплановано розміщувати певну персональну інформацію. Для цього потрібно забезпечити хешування паролів і токенізацію.

2.2 Розробка архітектури сайту

Архітектуру застосунку найзручніше побудувати на принципах модульності, себто розділити функціонал за логічними компонентами та забезпечити незалежність окремих частин застосунку. Платформа орієнтується на вебтехнології та передбачає клієнт-серверний підхід: клієнтська частина відповідає за інтерфейс і взаємодію з користувачем, а серверна – за обробку даних і керування базою даних. Такий підхід дозволяє масштабувати проєкт, розширювати функціонал і забезпечувати безпечну роботу всіх ролей у системі.

2.2.1 Формулювання загальної структури системи

Загальна структура вебплатформи ґрунтується на принципах модульності та чіткого розподілу відповідальності між компонентами. Можна умовно розділити систему на логічні частини, кожна з яких виконує ряд конкретних завдань, але взаємодіє з іншими через інтерфейс та API. До основних частин належать клієнтська частина, серверна частина та частина

керування даними (тобто база даних).

Клієнтська частина відповідає за відображення інтерфейсу та взаємодію користувачів із функціоналом платформи. Вона також забезпечує авторизацію, перегляд розкладу, керування навчальними матеріалами, перегляд оплат та інші можливості, доступні відповідно до ролі. Завданням фронтенд-частини є надання зручного та інтуїтивного інтерфейсу, що адаптується під різні пристрої.

Серверна частина здійснює обробку запитів, реалізує бізнес-логіку та координує доступ до даних. Вона також виконує перевірку прав доступу, автентифікацію та авторизацію користувачів, обробку даних про заняття, оплату, домашні завдання, створення та редагування навчальних матеріалів.

Взаємодія між клієнтською та серверною частинами реалізується наступним чином: сервер надає набір кінцевих точок, до яких звертається клієнт для отримання даних. Обмін інформації відбувається у форматі json, і потім він оброблюється для виводу.

Так як серед вимог є забезпечення безпеки даних, то варто також додати перевірку прав доступу в залежності від ролі користувача, щоб не допустити ситуацій, коли, наприклад, учень перейшов за посиланням, доступним адміністратору, таким чином отримавши доступ до всіх даних. Для цього кожен запит повинен перевірятися на наявність згенерованого токена, і відповідно до цього дати або відхилити доступ.

База даних зберігає інформацію про користувачів, розклад занять, оплату, навчальні матеріали та всі інші сутності, які визначені в п. 2.2.3. Так як обраний інструмент для керування базами даних є нереляційним, то структуру документів можна досить гнучко модифікувати відповідно до вимог проєкту при змінах. Це спрощує розширення функціоналу, додавання нових полів та адаптацію моделі даних під нові можливості платформи.

Узгоджена робота і взаємодія цих трьох частин забезпечує масштабованість і стабільність застосунку. Така архітектура дозволяє ефективно додавати нові модулі, розширювати функціонал та підтримувати різні ролі користувачів.

2.2.2 Моделі взаємодії та сценарії використання

Для опису поведінки системи та особливостей взаємодії між різними ролями було побудовано діаграму прецедентів. Вона відображає основні сценарії використання та дозволяє визначити, які функції доступні кожному користувачеві. На рисунку 2.1 подано узагальнену діаграму, що демонструє ключові можливості платформи.

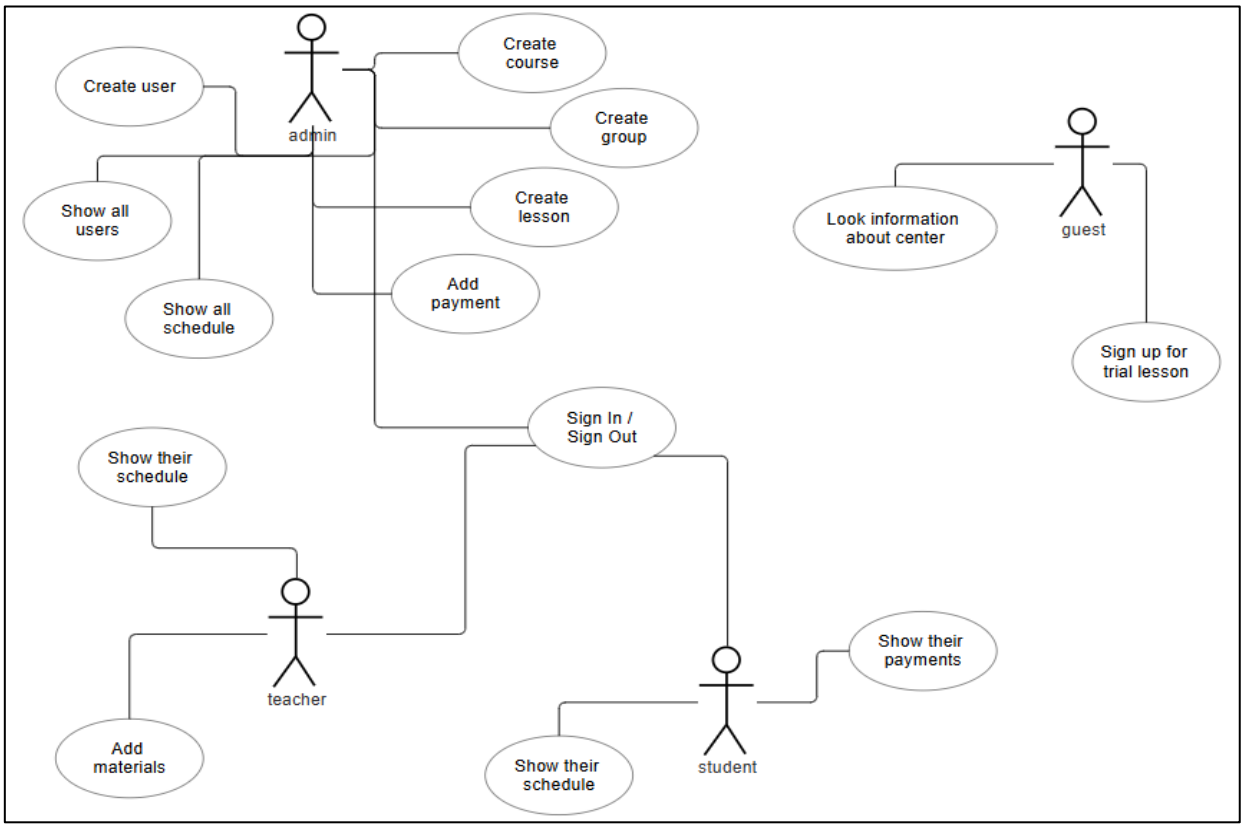


Рисунок 2.1 – Діаграма прецедентів

Отже, у нас є чотири ролі: «Адміністратор», «Вчитель», «Учень» і «Гість». Адміністратор має найбільше повноважень: він реєструє нових користувачів і може переглядати всіх існуючих, створює курси та групи і, відповідно, може їх переглядати, додає внесену учнем оплату, додає уроки і може переглядати розклад. Вчитель бачить свій розклад і може додавати матеріали до існуючих занять. Учень може переглядати свій розклад і свої оплати занять. Всі ці три ролі мають спільну можливість зайти та вийти з акаунту для доступу до облікового запису. У гостя такої можливості немає,

йому доступні лише більш статичні функції застосунку: перегляд інформації та реєстрація на курс за зовнішньою формою.

Для більш повноцінного розуміння внутрішніх процесів для кожної ролі (окрім гостя – у нього занадто мало можливостей) було розроблено діаграму активностей (рис. 2.2 – 2.4).

На рисунку 2.2 зображено орієнтовний цикл активності учня. Як попередньо вказано на діаграмі прецедентів, він може зайти в акаунт, проглянути весь контент, який йому доступний за налаштуваннями доступу, і вийти з акаунту.

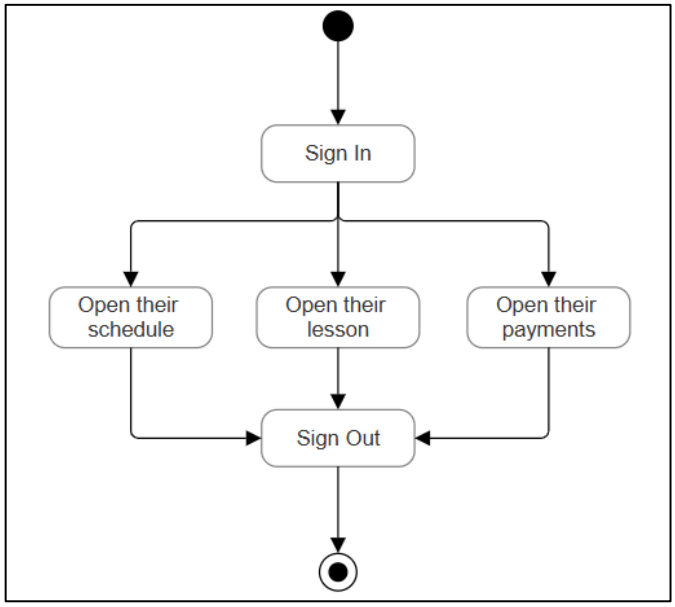


Рисунок 2.2 – Діаграма активностей для користувача «Учень»

В цілому, дещо схожим до попереднього є цикл активності вчителя (див. рис. 2.3). Він може зайти, переглянути доступний контент, а також додати матеріали до конкретного уроку. Аналогічно, цикл закінчується, коли вчитель виходить з акаунту.

Логічно, що діаграма активностей для адміністратора значно розширена через велику кількість доступних йому функцій (див. рис. 2.4). Заплановано, що в нього буде умовна панель керування, де буде чотири модулі, у кожному з яких є певні функції. Після будь-якої дії адміністратор може вийти зі свого акаунту.

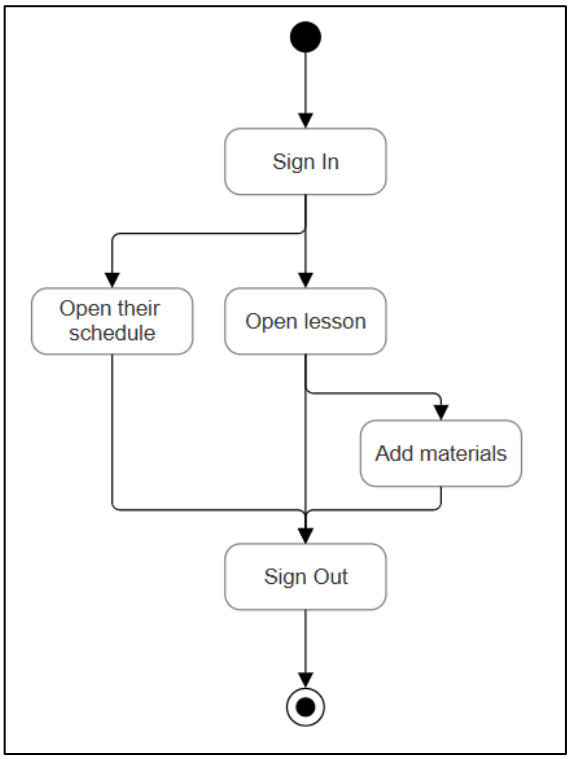


Рисунок 2.3 – Діаграма активностей для користувача «Вчитель»

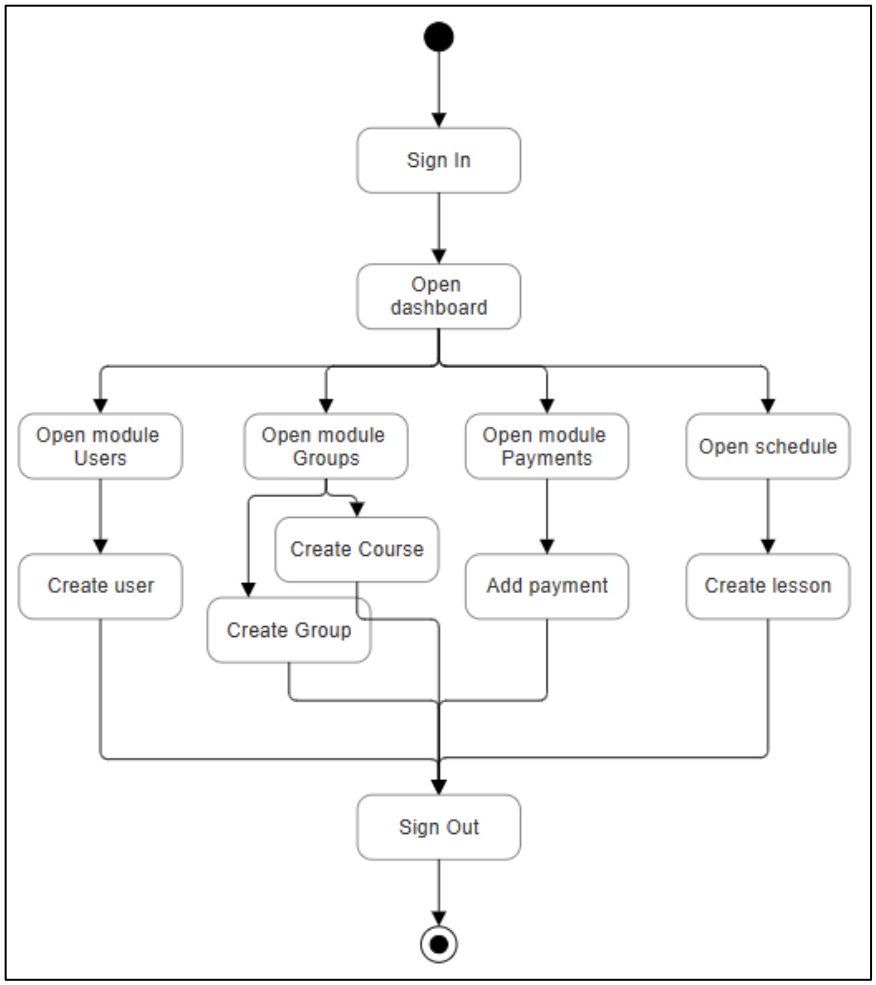


Рисунок 2.4 – Діаграма активностей для користувача «Адміністратор»

2.2.3 Структура бази даних

Розуміння функцій кожного користувача – це значний крок, але для кращого розуміння зв’язків і взаємодій між ними необхідно розробити ER-діаграму.

Не дивлячись на те, що у нас лише три користувача, які взаємодіють із динамічним контентом, сутностей буде дещо більше. І адміністратора, і вчителя, і учня можна віднести до однієї сутності «Користувач» і взаємодіяти з нею, залежно від визначеної ролі. Отже, полями цієї сутності буде ім’я, електронна пошта та пароль для входу, роль, телефон і, бажано, логічна змінна про статус (активність) користувача, оскільки адміністратор матиме право додавати або блокувати користувача.

Кожен вчитель викладає конкретний «Курс», який буде окремою сутністю. Зв’язок між ними забезпечує поле з ідентифікатором вчителя. Серед додаткових полів є назва, опис (за необхідності) та вартість за годину.

Кожне заняття повинно бути оплачене, тому варто створити сутність «Оплата». Для розуміння за кого заплатили і за який курс, необхідно додати ідентифікатор студента та курсу, на який він записаний. Також є поле з сумою оплати (так як передбачена можливість платити частинами), дата оплати та спосіб, щоб відслідковувати платежі за квитанціями.

На один і той самий курс можуть записуватися різні учні, причому як на індивідуальні заняття, так і на групові. Раціональним буде створити одну сутність «Група» та додавати до неї необхідних учнів, навіть якщо він тільки один. Отже, необхідно поле з масивом ідентифікаторів студентів, поле з ідентифікатором вчителя, який викладає цей курс (один і той самий курс можуть викладати різні вчителі), і також ідентифікатор безпосередньо самого курсу. Для розрізнення груп – назва групи та формат (онлайн або офлайн).

У кожної групи відбуваються заняття, для яких теж необхідно створити сутність. Зв’язок забезпечується через ідентифікатор групи. Обов’язково потрібно вказувати дату та час заняття, його визначену тривалість (так як це

впливає на оплату), відвідуваність, бажано вказати тему та окремо виділити формат – наприклад, курс може бути офлайн, але учень може попросити про додаткове онлайн-заняття тощо.

Остання сутність – «Матеріали». В ідеалі, щоб вчитель до кожного уроку підкріпляв певні дані, отже, щоб забезпечити зв’язок, потрібно додати ідентифікатор вчителя. Так як матеріали стосуються конкретного уроку, то варто додати його ідентифікатор теж. Можна уточнити тип матеріалу, оскільки це може бути файл, текст чи посилання на зовнішній контент. Якщо в матеріалах є домашнє завдання, доречно додати дату дедлайну. Також можна вказати назву, якщо вона відрізняється від назви уроку.

Всі сутності обов’язково мають унікальний ідентифікатор. На основі описаних даних побудовано діаграму, яка представлена на рисунку 2.5.

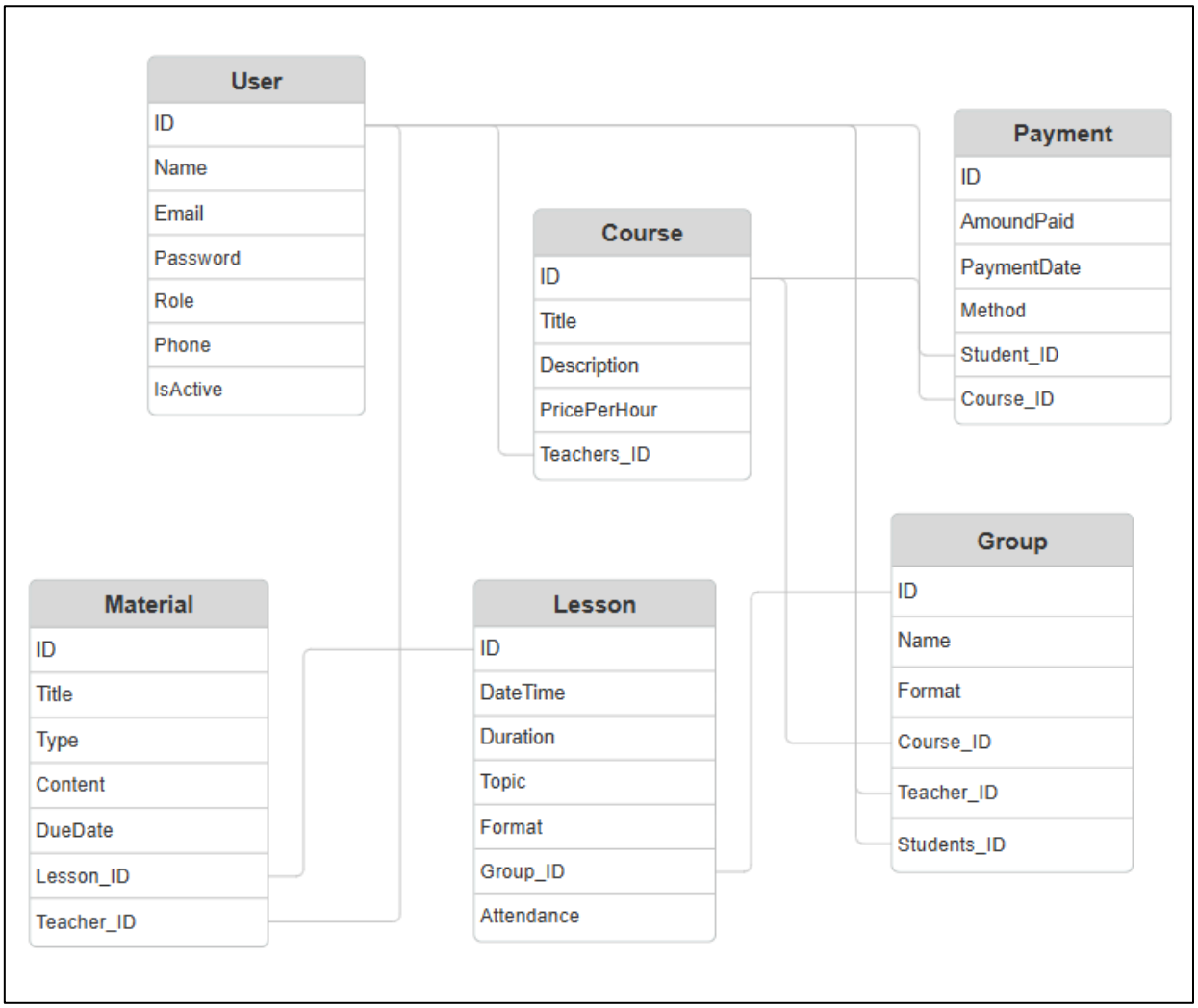


Рисунок 2.5 – ER-діаграма

2.3 Висновки до розділу 2

У цьому розділі було проведено комплексне моделювання вебзастосунку для автоматизації роботи репетиторського центру. На етапі аналізу вимог було деталізовано функціональні вимоги, а також сформовано нефункціональні та технічні вимоги.

Розробку обраної клієнт-серверної архітектури доречно розбити на окремі модулі, щоб забезпечити масштабування проєкту в майбутньому. Побудовані діаграми прецедентів та активностей демонструють логіку процесів і сценарії функцій, які доступні кожній ролі.

Спроєктована ER-діаграма враховує специфіку предметної області, забезпечуючи ефективну взаємодію між визначеними сутностями даних. Така структура даних дозволить мінімізувати надлишковість інформації та забезпечити цілісність зв’язків.

Таким чином, результати здійсненої роботи є надійним фундаментом для подальшої розробки.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

У попередніх розділах було детально проаналізовано предметну область, інструментарій, визначено вимоги, основні процеси, функції та сутності. Цей розділ присвячено безпосередньо процесу реалізації клієнтської та серверної частини, а також налаштування взаємодії між ними. Важливим є також забезпечення адаптивності застосунку для використання на різних пристроях. Окрему увагу приділено тестуванню розробленого продукту.

3.1 Загальна структура проєкту

Проєкт розроблено в одному репозиторії steamcity [3]. У кореневій теці чітко розділяється frontend і backend – вміст клієнтської і серверної частини відповідно. Також там знаходяться необхідні конфігураційні файли, наприклад, package.json, у якому є скрипт на запуск одночасно обох частин проєкту і вказано відповідні встановлені для цього пакети.

3.2 Розробка фронтенд-частини

Клієнтську частину реалізовано за допомогою фреймворку React. Він дозволяє зручно розробити динамічний інтерфейс, підключати компоненти, щоб уникнути дублювання коду, та гнучко додавати інші функції за потреби.

Структура фронтенд-частини знаходиться в теці проєкту frontend (див. рис. 3.1). Там же розташовані основні файли конфігурації для клієнта: package.json, у якому описані версії встановлених пакетів, tailwind.config.js для використання фреймворку стилізації та опису стилів, і postcss.config.js для коректної роботи попереднього файлу. Подібно до будь-якої іншої стандартної

будови React-проєкту, тут знаходяться теки public із публічними файлами та src із основним вмістом клієнтської частини.

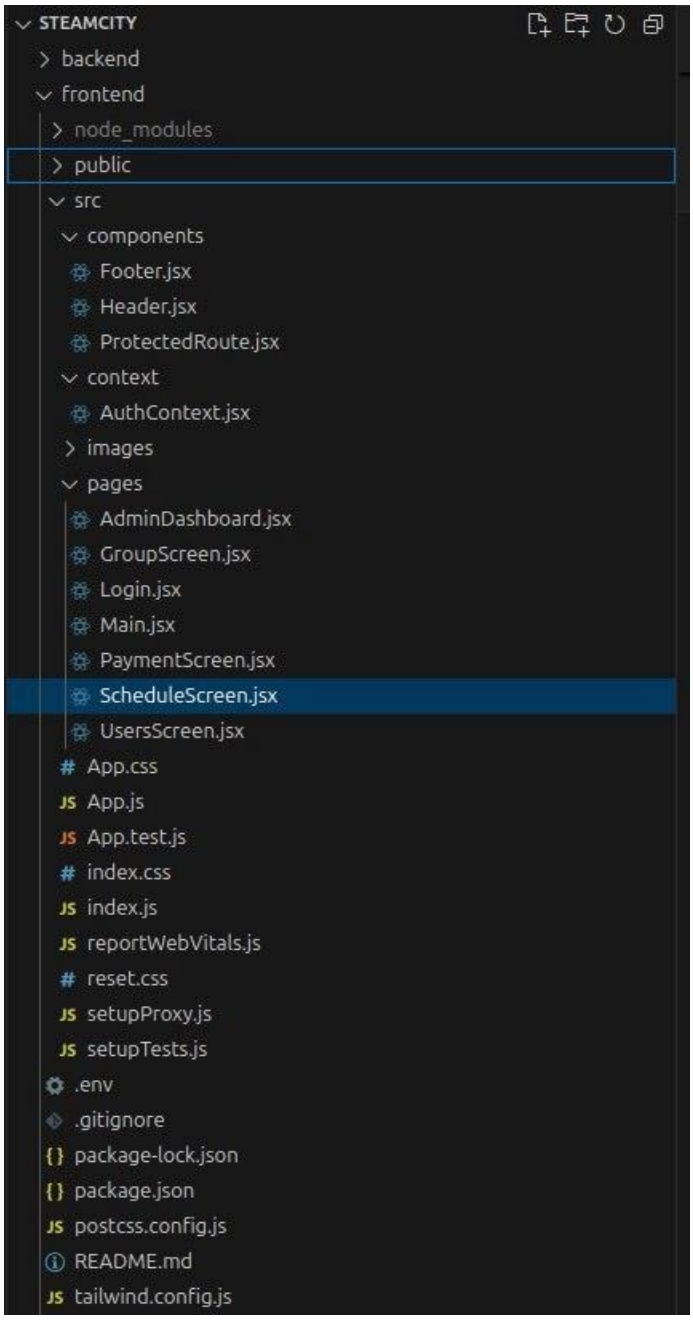


Рисунок 3.1 – Структура клієнтської частини

Перейдемо детальніше до розгляду теки src. У її корені знаходяться основні файли, які забезпечують роботу додатку – index.js, коректну маршрутизацію, підключення компонентів і контексту – App.js. Також там знаходиться файл із глобальними стилями index.css. Для зручності всі інші файли поділено на відповідні теки, розглянемо їх по черзі.

Отже, тека components містить компоненти додатку. Поділ окремих фрагментів коду на ізольовані компоненти дозволяє уникнути повторення коду на різних сторінках. Достатньо підключити їх у потрібне місце; у проєкті вони підключаються один раз у файлі App.js, щоб відображатися на будь-якій іншій сторінці. Сюди входять header, footer, а також компонент забезпечення захищеного з'єднання. Даний компонент є критично важливим, оскільки навіть переходячи по різним можливим варіаціям посилань застосунку, користувач повинен отримувати доступ виключно до дозволених йому даних.

Наступною текою є context лише з одним файлом – контекст авторизації (див. рис. 3.2). Він є надзвичайно суттєвим у будь-яких застосунках, які потребують входу в акаунт. Так як проєкт розбитий на компоненти, то саме для забезпечення доступу до даних користувача з будь-якого компоненту існує цей контекст. Він зберігає дані при вході, очищує при виході, і дозволяє залишатися в акаунті після оновлення сторінки.

Обов'язковим є створення теки pages, оскільки для адміністратора описаний досить великий функціонал, і робити йому це все на одній сторінці буде досить незручно. Тому для забезпечення кращої та більш продуктивної роботи виділено окремо сторінки з консоллю адміністратора, з керуванням користувачами, курсами, оплатами та розкладом. Остання сторінка доступна всім користувачам, які здійснили вхід у свій акаунт, але її контент буде відрізнятися в залежності від ролі. Так, наприклад, у той час, як адміністратор бачить абсолютно всі заняття і може їх додавати, вчитель і учень бачать виключно свої заняття.

Також є сторінка для входу в акаунт. Сторінка створення акаунту не запланована замовником, оскільки самостійна реєстрація не матиме сенсу до тих пір, доки адміністратор не підключить нового користувача до конкретного курсу. Це також дозволить уникнути великій кількості незрозумілих пустих акаунтів.

Наостанок, окремо виділяються статичні сторінки: головна, про нас, відгуки, контакти.

```
// початковий стан перевіряє наявність токена
const initialState = {
   user: localStorage.getItem('userInfo')
      ? JSON.parse(localStorage.getItem('userInfo'))
      : null,
   loading: false,
   error: null,
};

const authReducer = (state, action) => {
   switch (action.type) {
      case 'LOGIN_SUCCESS':
         localStorage.setItem('userInfo', JSON.stringify(action.payload)); // збереження сесії в
браузері
         return {
            ...state,
            user: action.payload,
            loading: false,
            error: null,
         };
      case 'LOGOUT':
         localStorage.removeItem('userInfo'); // очищення даних при вході
         return {
            ...state,
            user: null,
            loading: false,
            error: null,
         };
      default:
         return state;
   }
};

export const AuthProvider = ({ children }) => {
   const [state, dispatch] = useReducer(authReducer, initialState);

   return (
      <AuthContext.Provider value={{ state, dispatch }}>
         {children}
      </AuthContext.Provider>
   );
};
```

Рисунок 3.2 – Фрагмент коду контексту авторизації

При запуску проєкту відразу відкривається головна сторінка (див. рис. 3.3). Кнопки у верхньому меню посилаються на однойменні сторінки. «Кабінет» із роллю користувача посилається на його особистий кабінет або зникає, якщо вхід не здійснено. «Вихід» змінюється на «Авторизацію», якщо

користувач не увійшов в акаунт. Кнопки з іконками соціальних мереж посилаються на відповідні сторінки в мережах. Кнопка «Дізнатись більше» в середній секції сторінки перенаправляє на сторінку «Про нас».

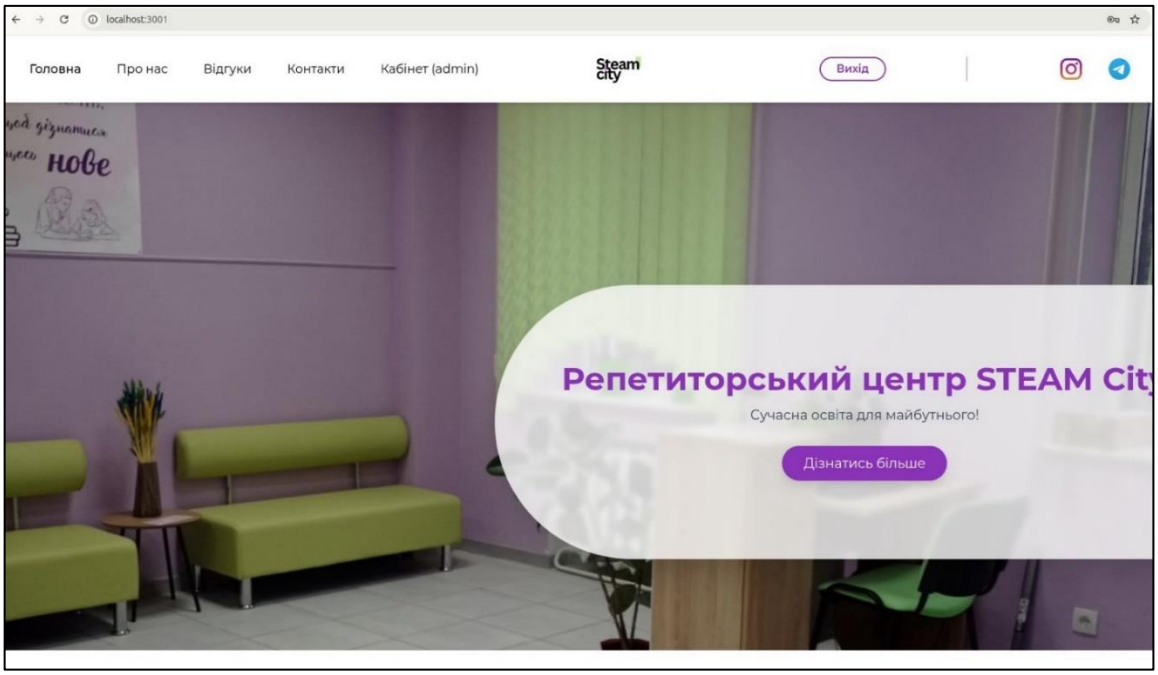


Рисунок 3.3 – Головна сторінка застосунку

Розглянемо детальніше саме на динамічні функції. Перейдемо у кабінет адміністратора, де відкривається його панель керування (див. рис. 3.4).

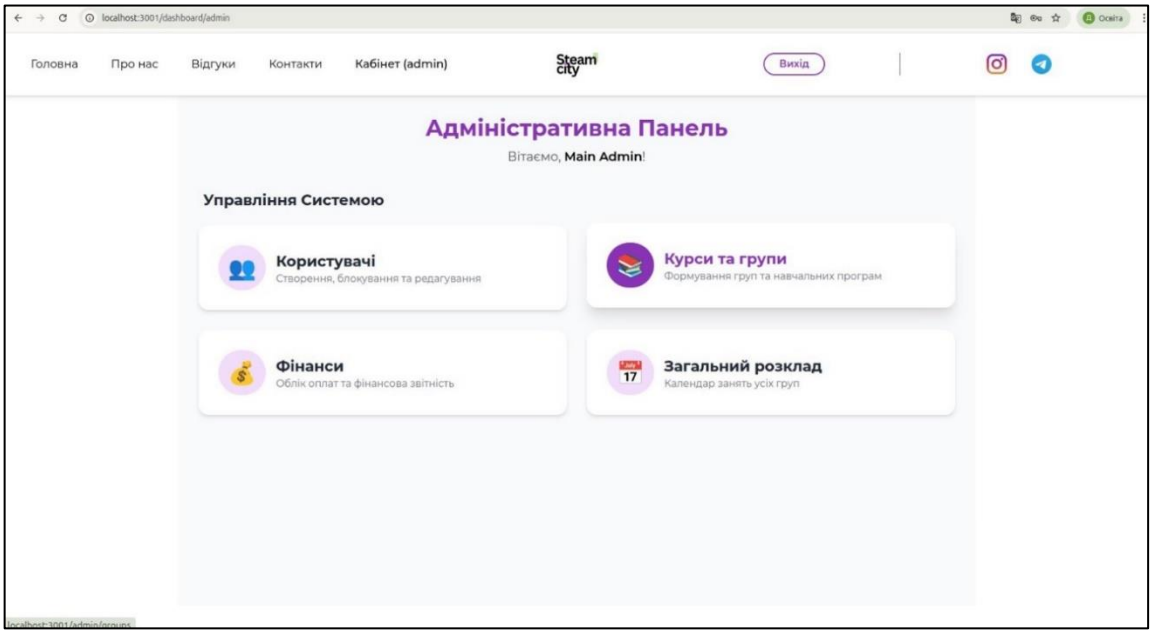


Рисунок 3.4 – Панель керування адміністратора

Із повним кодом цієї сторінки можна ознайомитися в додатку А.1. Всі інші сторінки реалізовані схожим чином. Отже, на панелі адміністратора бачимо 4 кнопки, які є посиланнями і переходять на відповідні сторінки. Переглянемо детальніше кожну з них.

При переході на сторінку керування користувачами бачимо форму для створення нового користувача та перелік усіх користувачів системи (див. рис. 3.5). Адміністратор вказує ім’я, електронну пошту, роль та пароль. Також він має право заблокувати акаунт студента чи викладача, якщо вони не приймають наразі участі в освітньому процесі.

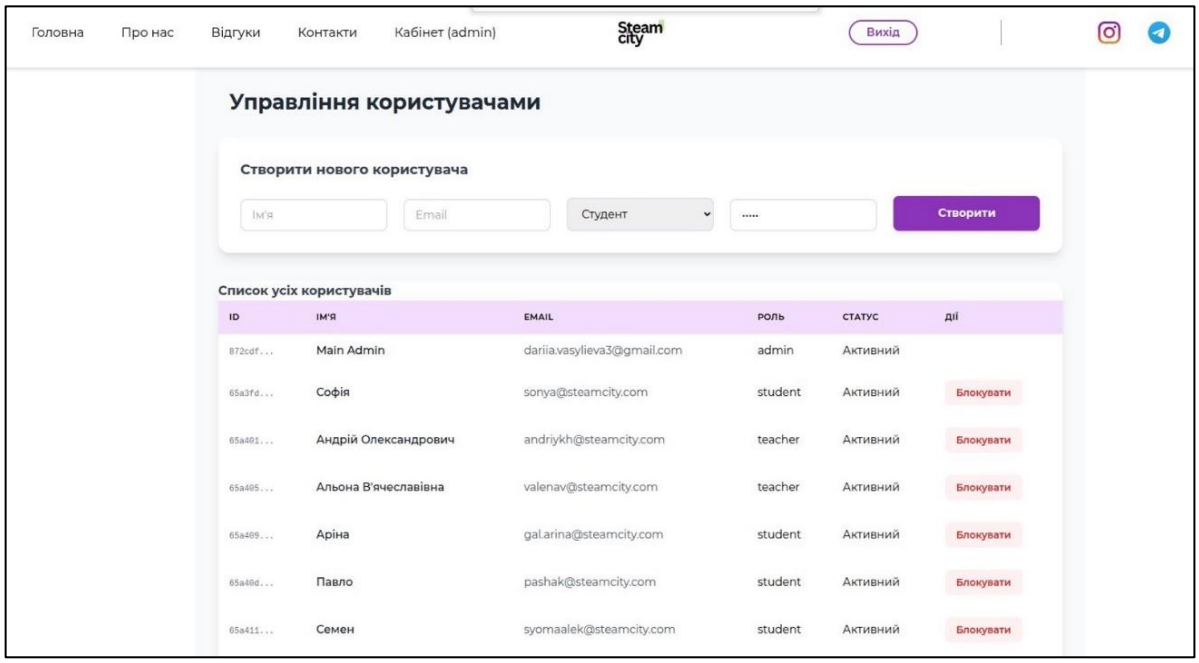


Рисунок 3.5 – Панель керування користувачами

Адміністратор має право створити курс та групу на відповідній сторінці (див. рис. 3.6). У лівій частині сторінки є невелика форма для створення курсу: достатньо ввести назву та ціну за заняття. Натомість у правій частині – створення групи. Обов’язково необхідно вказати назву групи, обрати конкретний курс, оскільки програми навчання можуть відрізнятися від одного до іншого, викладача, студентів. Так як це створення групи, то можна обрати як одного студента, так і декілька. У нижній частині відображається перелік усіх існуючих груп.

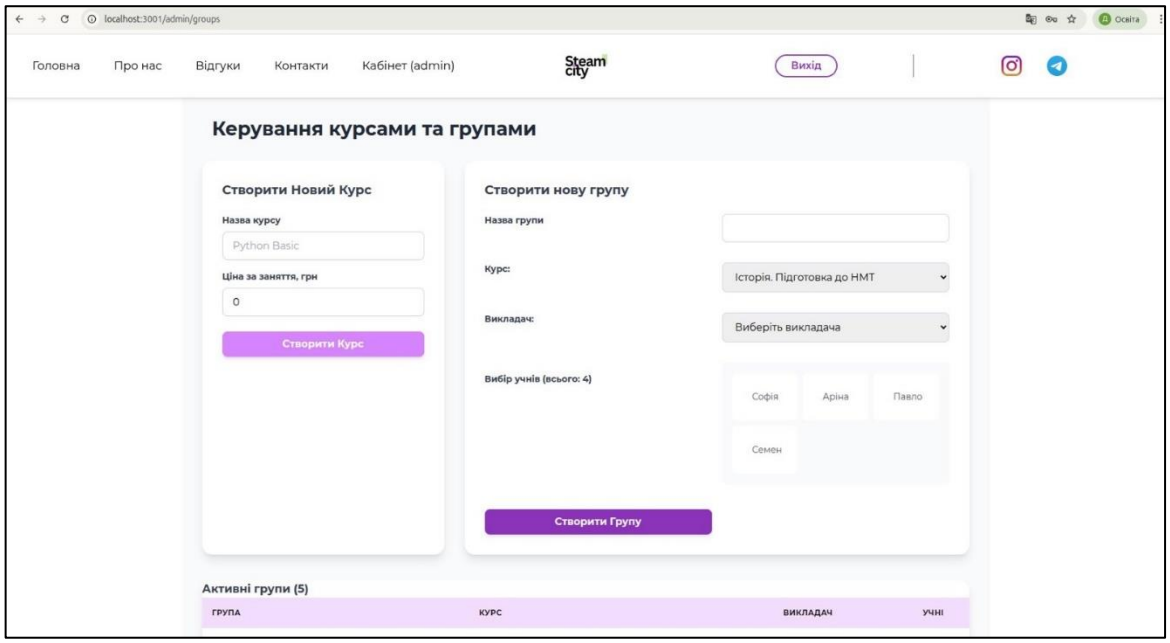


Рисунок 3.6 – Панель керування курсами та групами

Окремою сторінкою є керування оплатами (див. рис. 3.7). Для зручності сторінку подібно до попередньої поділено на дві частини. У лівій знаходиться форма внесення оплати, за якого учня, курс, яку суму внесли і яким способом. У правій – загальні дані по курсу конкретно по обраній дитині: скільки занять пройдено, сума до оплати та скільки сплачено. Також додано перелік пройдених занять, щоб переглянути та перевірити коректність суми до сплати.

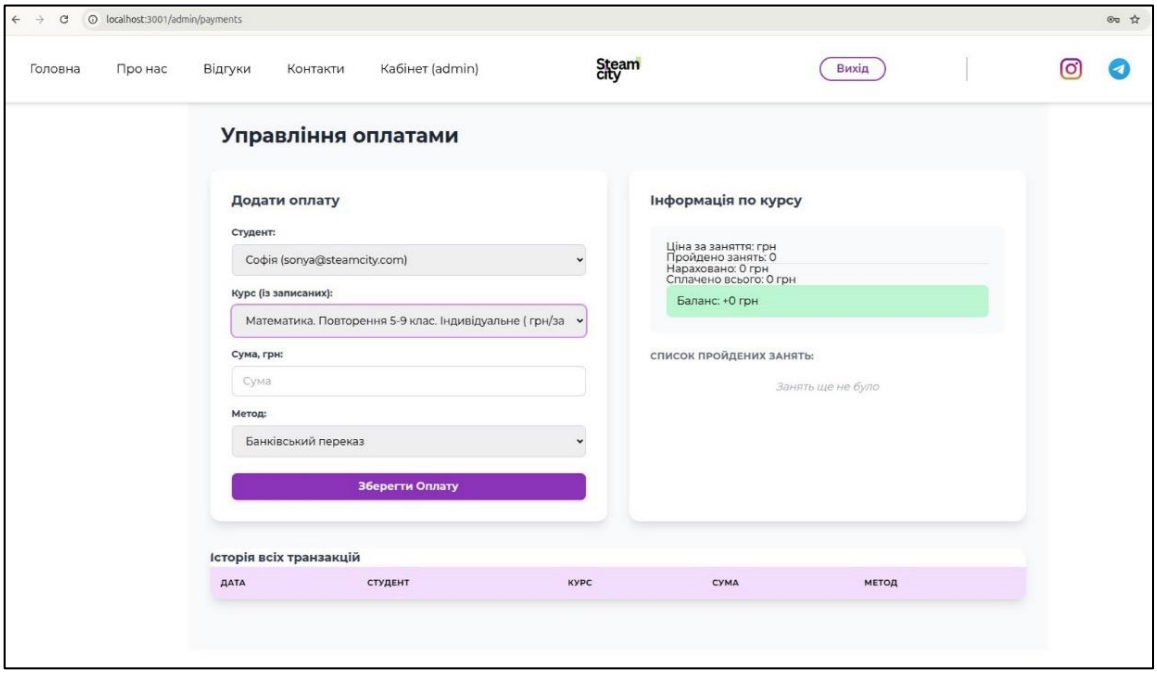


Рисунок 3.7 – Панель керування оплатами

І остання, але не менш важлива частина керування адміністратора – розклад (див. рис. 3.8). На початку є форма створення заняття, вибір групи, дата та час, тривалість заняття (з деяких курсів заняття можуть тривати годину, півтори або дві) та формат. Після форми відображається перелік усіх доданих занять із детальною інформацією.

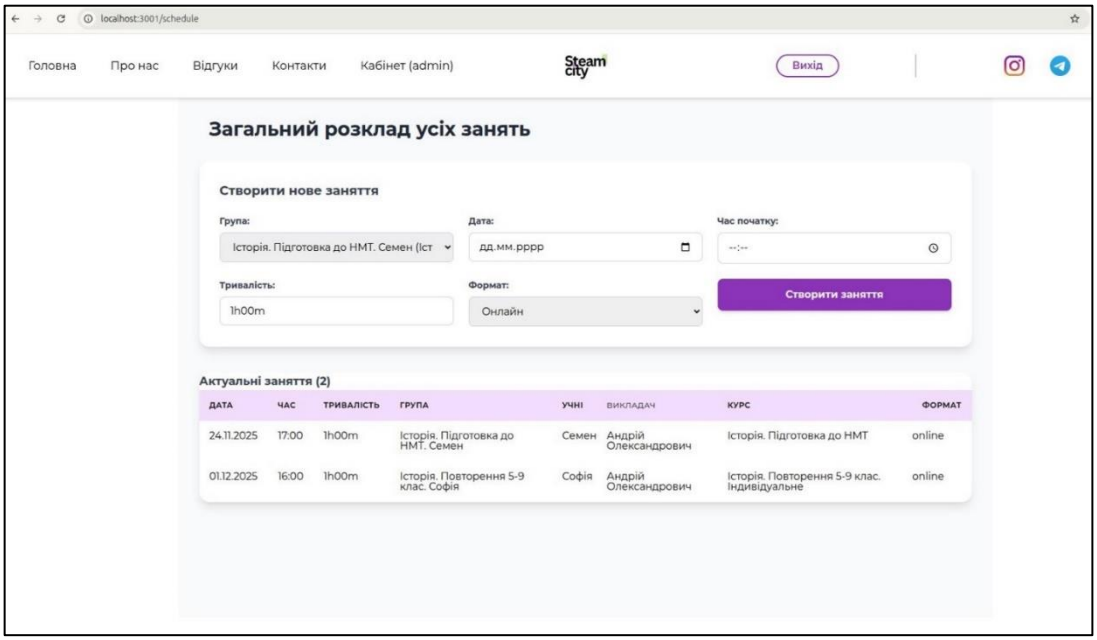


Рисунок 3.8 – Панель керування заняттями

Спробуємо перемкнутися на користувача з іншою роллю. Введемо дані для входу одного з вчителів та у їхньому кабінеті отримаємо лише розклад цієї людини (див. рис. 3.9).

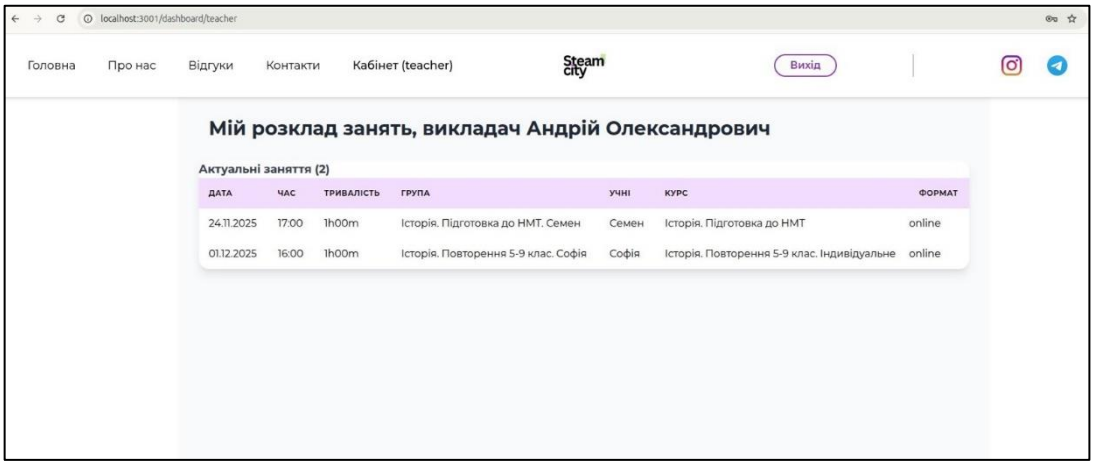


Рисунок 3.9 – Сторінка розкладу вчителя

Аналогічно, для студента відображатимуться тільки його заняття (див. рис. 3.10).

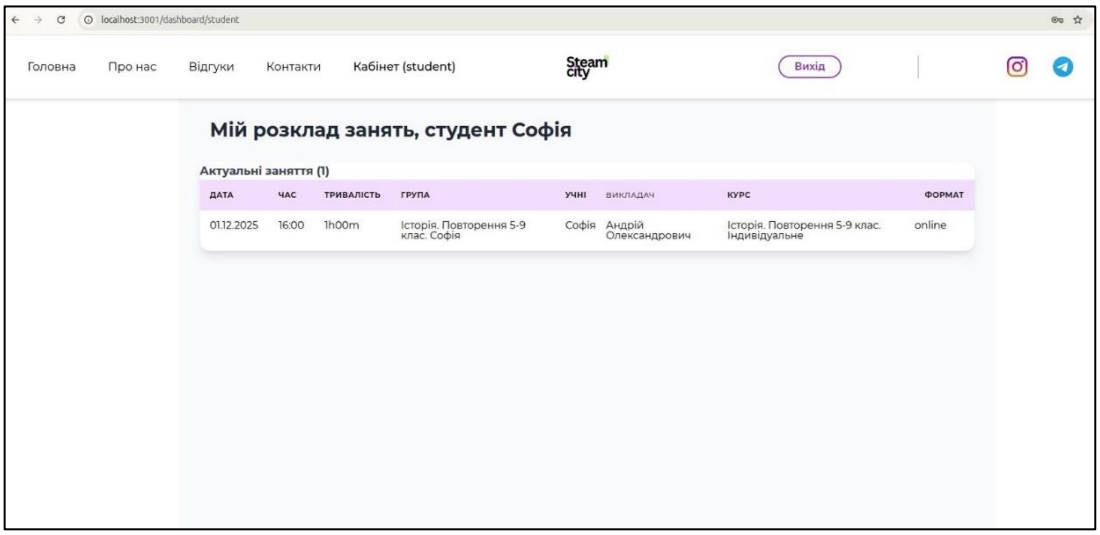


Рисунок 3.10 – Сторінка розкладу студента

Таким чином, реалізовано користувацький інтерфейс для зручного користування системою.

3.3 Розробка бекенд-частини

Серверна частина – «мозок» усього проєкту, так як там містяться описи сутностей, шляхи тощо. Вона побудована на основі Node.js та Express. Погляньмо детальніше на структуру теки backend (див. рис. 3.11).

У корені цієї теки лежать конфігураційні файли package.json, як і в попередньому пункті – про підключення необхідних пакетів, а також .env – файл, у якому визначений порт серверу та посилання на підключення до бази даних. Ці дані винесені в окремий файл для посилення безпеки застосунку. У цій же теці знаходиться і основний файл серверу – server.js. У ньому відбуваються всі підключення маршрутів, зовнішніх функцій і безпосередній запуск сервера. Код даного файлу наведено в додатку Б.1. Окрім вказаних файлів також у корені знаходиться перелік інших тек.

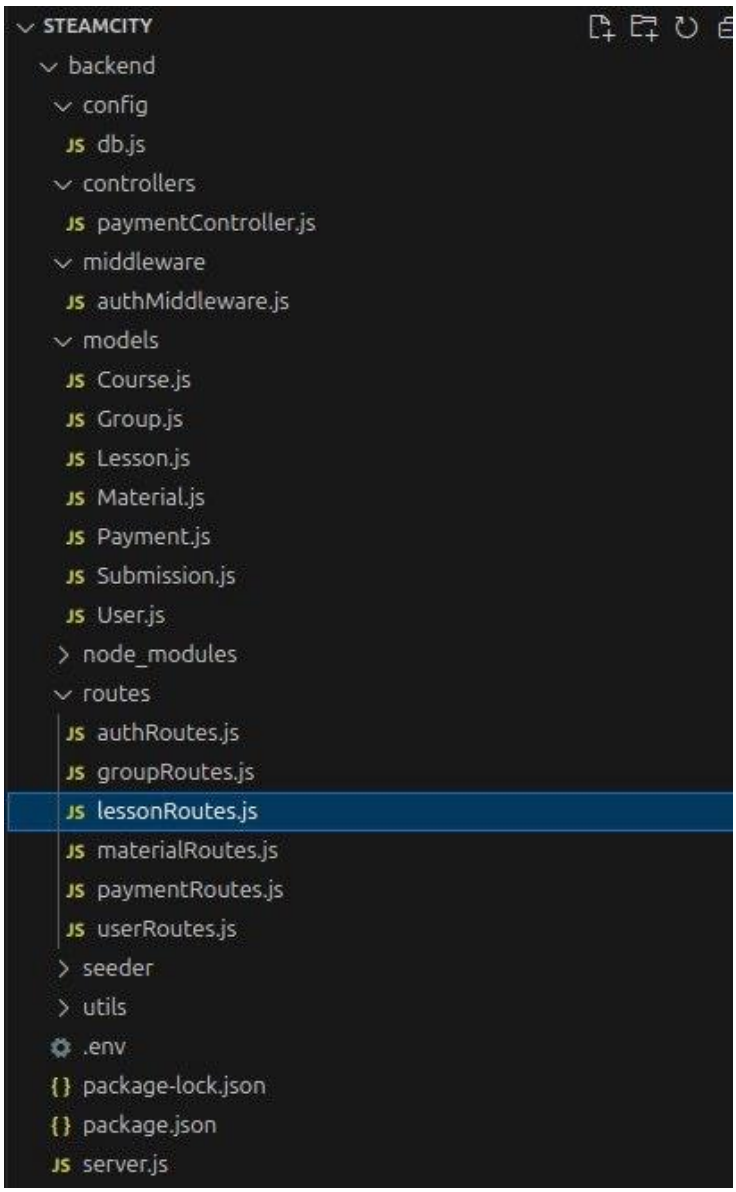


Рисунок 3.11 – Структура серверної частини

По-перше, там є тека config із файлом db.js. У ньому міститься функція встановлення зв’язку із базою даних, яка підключається у загальний файл серверу. Цей процес винесено окремо, щоб не нагромаджувати серверний файл.

По-друге, є тека controllers із файлом paymentController.js. Контролер містить більшу кількість операцій та певну логіку обробки даних: перевірка вхідних параметрів, звернення до бази даних, математичні обчислення та формування відповіді. Поки що для цього проєкту достатньо одного контролеру оплат, оскільки цей процес складніший за всі інші, а вони, у свою чергу, описані у файлах маршрутизації. Однак у майбутньому, для

покращення та масштабування застосунку можна окремо винести контролери груп, користувачів тощо.

Наступним важливим елементом у теці middleware є файл authMiddleware.js. Він відповідає за автентифікацію та авторизацію користувачів. Це вкрай важливий функціонал, і його також винесено окремо, щоб уникнути дублювання, оскільки перевірка поточного користувача та його прав доступу потребують майже всі маршрути.

Тека models містить опис усіх сутностей у форматі mongoose схем, які потім перетворюються на моделі для коректної роботи. У кожному файлі визначається структура кожного документа, його поля і формат. Для прикладу наведемо фрагмент коду із моделлю «Користувач» (див. рис. 3.12). Отже, спочатку створюється схема, конкретно в цій будь-який користувач повинен обов’язково мати ім’я, електронну пошту, пароль і роль. Для кожного поля визначений його формат. У кінці зі схеми утворюють модель, із якою програма надалі працює. За таким принципом реалізовано всі інші моделі.

```
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
    },
    email: {
        type: String,
        required: true,
        unique: true,
    },
    password: {
        type: String,
        required: true,
    },
    role: {
        type: String,
        enum: ['admin', 'teacher', 'student'],
        default: 'student',
        required: true,
    },
    phone: {
        type: String,
    },
    isActive: {
        type: Boolean,
        default: true,
    },
}, {
    timestamps: true,
});

const User = mongoose.model('User', userSchema);
```

Рисунок 3.12 – Фрагмент коду модель «Користувач»

Також суттєвим є маршрутизація в застосунку, яка реалізована у теці routes. Вона наповнена файлами маршрутизації, які об’єднують усі кінцеві точки, пов’язуючи http-методи з відповідними функціями. Так, наприклад, адміністратор може отримати весь перелік користувачів за допомогою GET запиту (див. рис. 3.13). Ця функція доступна лише адміністратору, вона асинхронно завантажує всі дані про користувачів (окрім паролю) і передає їх клієнту, який вже відображає цю інформацію на сторінці.

```
router.get('/', protect, isAdmin, async (req, res) => {
    const filter = req.query.role ? { role: req.query.role } : {};

    try {
        const users = await User.find(filter).select('-password');
        res.json(users);
    } catch (error) {
        console.error('Помилка отримання списку користувачів:', error);
        res.status(500).json({ message: 'Помилка сервера при отриманні користувачів' });
    }
});
```

Рисунок 3.13 – Фрагмент коду із маршрутизацією користувачів

Тека seeder містить файл seeder.js, який використовується для швидкого наповнення бази даних тестовою інформацією або для заміни певної інформації програмним шляхом.

Так як питання безпеки в застосунку є одним із критичних, то необхідно десь генерувати токен для входу. Це відбувається у файлі generateToken.js у теці utils.

3.4 Тестування функціональності, продуктивності та безпеки

Для забезпечення якості продукту було проведено ряд ручних тестів, спрямованих на перевірку коректності роботи всього функціоналу.

Одним із найважливіших пунктів є безпека даних, отже, треба перевірити, чи не пускає студента чи викладача на сторінки адміністратора, якщо ті просто введуть посилання. Заходимо в акаунт студента, вказуємо

адресу, наприклад, із переліком усіх користувачів. У результаті отримуємо відразу помилку (див. рис. 3.14).

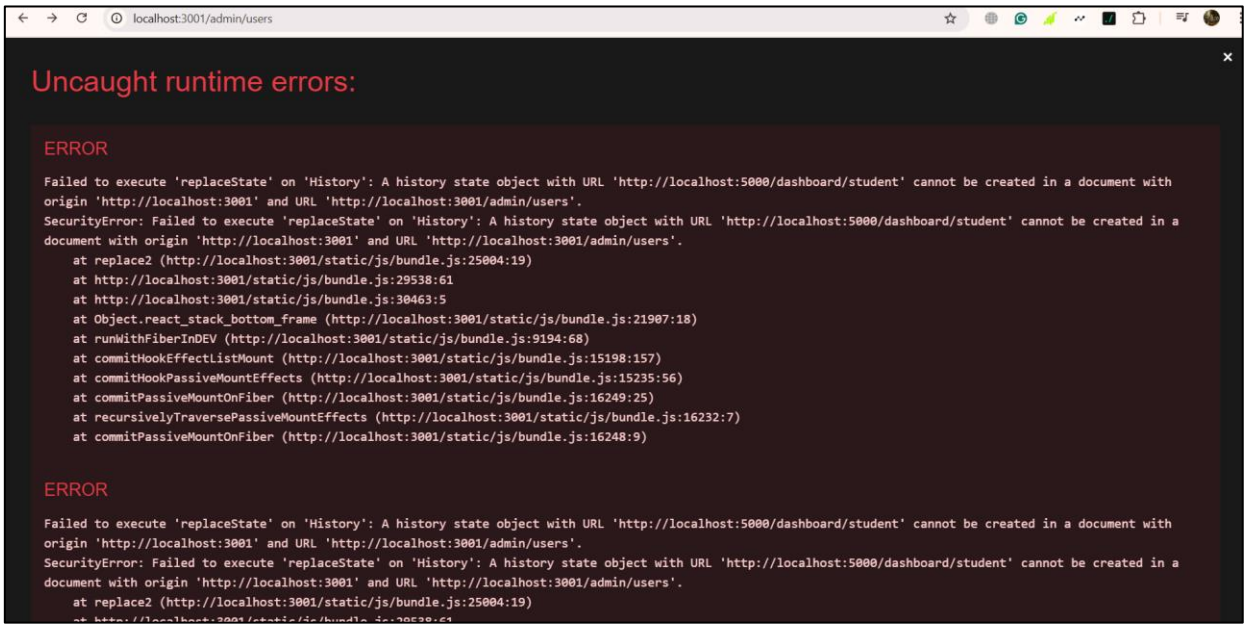


Рисунок 3.14 – Помилка доступу студента до переліку користувачів

Аналогічно перевіримо права доступу вчителя. Здійснюємо вхід у акаунт та вказуємо адресу всіх оплат. Отримуємо результат із тією ж помилкою, але із вказанням сторінки, на яку ми хотіли перейти та ролі поточного користувача (див. рис. 3.15).

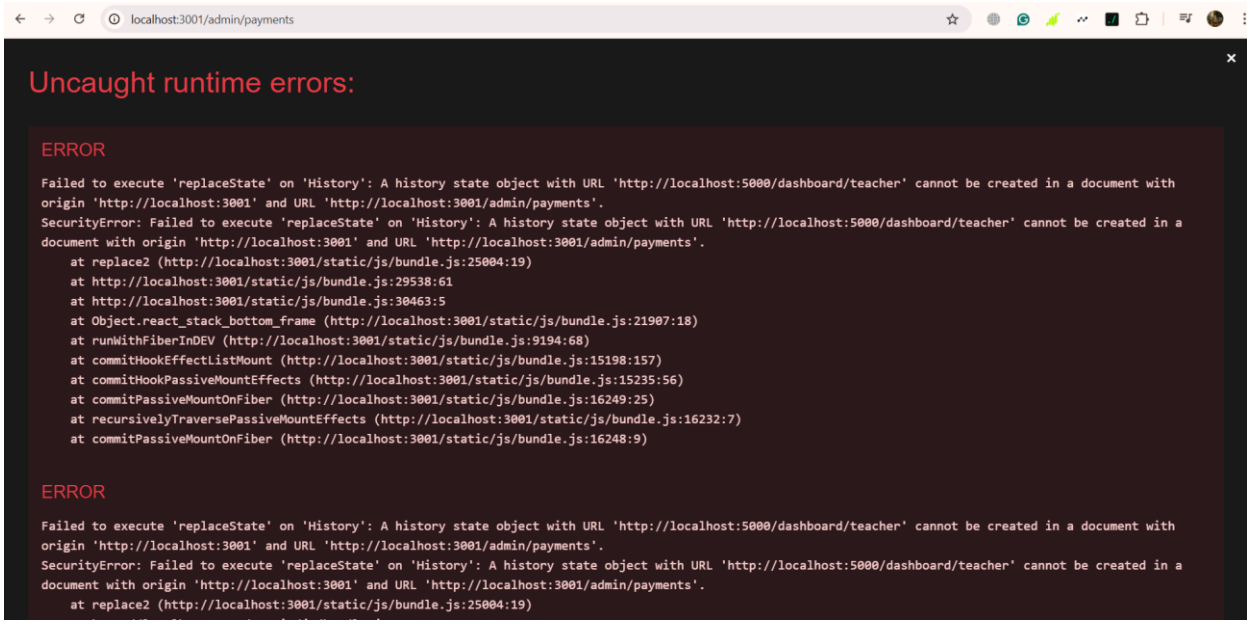


Рисунок 3.15 – Помилка доступу вчителя до сторінки з оплатами

Цей функціонал надійно реалізовано в самому файлі App.js у кореневій теці клієнтської частини. Кожен шлях, до якого має доступ тільки конкретна роль, є захищеним компонентом ProtectedRoute (див. рис. 3.16). Реалізація даного компоненту наведена в додатку А.2.

```
<BrowserRouter>
      <Header />
      <Routes>
        <Route path='/' element={<Main />} />
        <Route path='/authorize' element={<Login />} />

        <Route path='/dashboard/admin' element={<ProtectedRoute
allowedRoles={['admin']}><AdminDashboard /></ProtectedRoute>} />
        <Route path='/admin/users' element={<ProtectedRoute allowedRoles={['admin']}><UsersScreen
/></ProtectedRoute>} />
        <Route path='/admin/groups' element={<ProtectedRoute allowedRoles={['admin']}><GroupScreen
/></ProtectedRoute>} />
        <Route path='/admin/payments' element={<ProtectedRoute allowedRoles={['admin']}><PaymentScreen
/></ProtectedRoute>} />

        <Route path='/schedule' element={<ProtectedRoute allowedRoles={['admin', 'teacher',
'student']}><ScheduleScreen /></ProtectedRoute>} />

        <Route path='/dashboard/teacher' element={<ProtectedRoute
allowedRoles={['teacher']}><ScheduleScreen /></ProtectedRoute>} />
        <Route path='/dashboard/student' element={<ProtectedRoute
allowedRoles={['student']}><ScheduleScreen /></ProtectedRoute>} />
      </Routes>
    </BrowserRouter>
```

Рисунок 3.16 – Реалізація захисту шляхів у файлі App.js

Також було проведено ручне додавання користувачів, курсів, груп, оплат і уроків. Після кожної операції додавання на екран виводиться відповідне повідомлення (див. рис. 3.17 та 3.18).

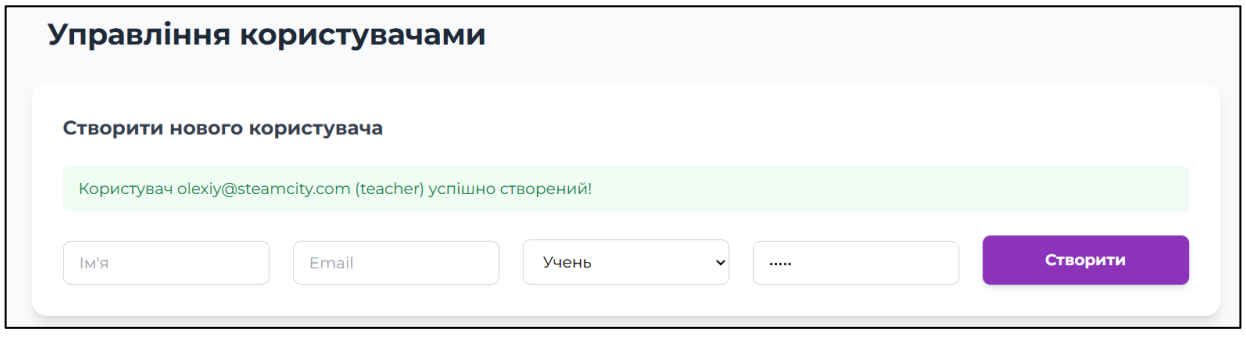


Рисунок 3.17 – Повідомлення про створення нового користувача

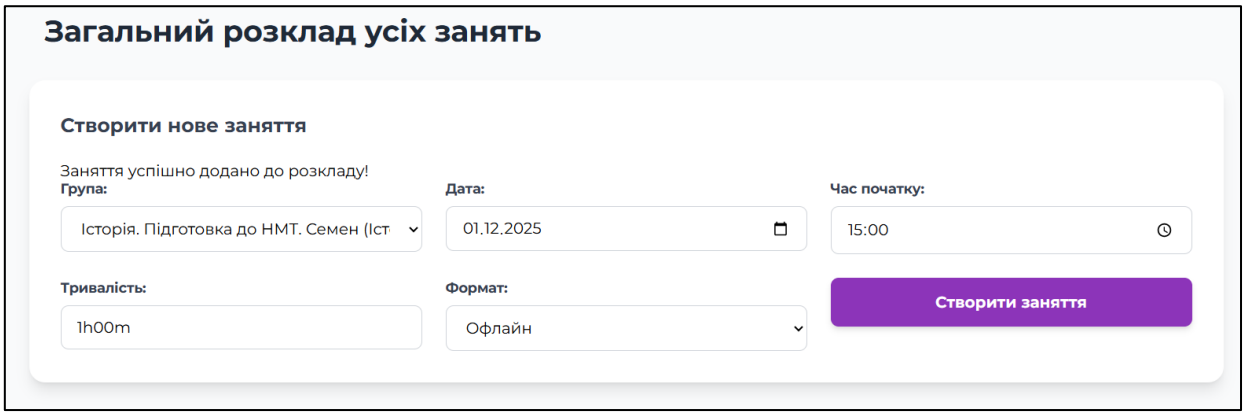


Рисунок 3.18 – Повідомлення про створення нового заняття

3.5 Оптимізація та адаптивність застосунку для різних пристроїв

Сучасні додатки повинні мати приємний та інтуїтивно зрозумілий користувацький інтерфейс, причому як на комп'ютерах, так і на мобільних пристроях. Це дозволить зручно переглядати свій розклад зі смартфону, а адміністраторам – швидко додавати зміни за потреби.

По аналогії із загальним стилем, для адаптивності також використовується фреймворк Tailwind CSS.

Використання адаптивних сіток і флексів дозволяє зручно розробити інтерфейс для різних розширень екрану. Так, наприклад, якщо порівняти вигляд адміністративної панелі на комп'ютері і смартфоні, можна побачити, що в першому випадку елементи керування розташовані в дві колонки, а в другому – в одну (див. рис. 3.4 та 3.19).

Завдяки гнучкій системі класів Tailwind, всі інтерактивні елементи, тобто кнопки, поля введення, автоматично адаптують свій розмір для зручного натискання. Таким чином, застосунок забезпечує однаковий рівень функціональності, незалежно від пристрою користувача.

Аналогічним чином реалізовані всі сторінки: панель керування користувачами (див. рис. 3.5 та 3.20), панель керування курсами (див. рис. 3.6 та 3.21), панель керування оплатами (див. рис. 3.7 і 3.22), сторінка з розкладом і додавання нових уроків (див. рис. 3.8 і 3.23).

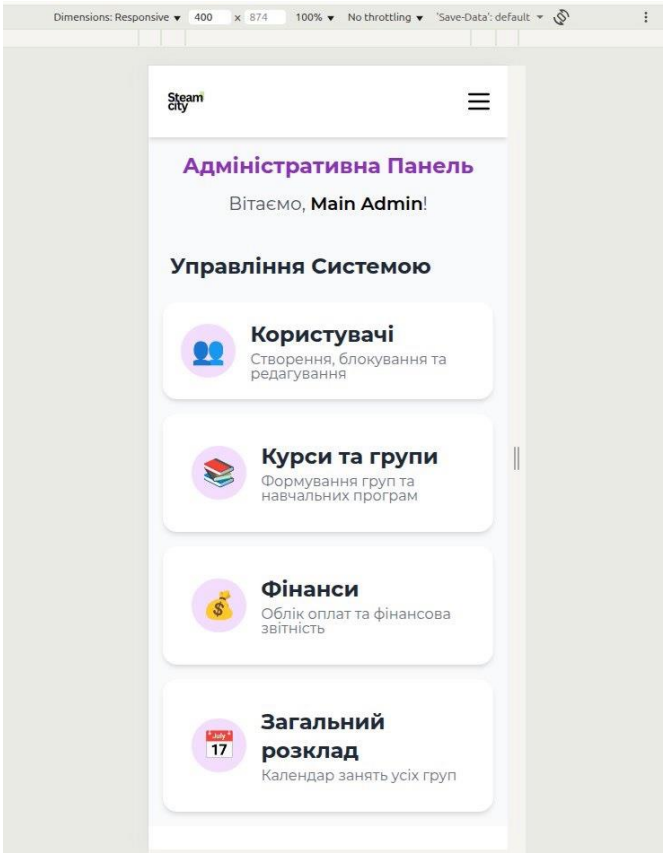


Рисунок 3.19 – Вигляд адміністративної панелі на мобільному пристрої

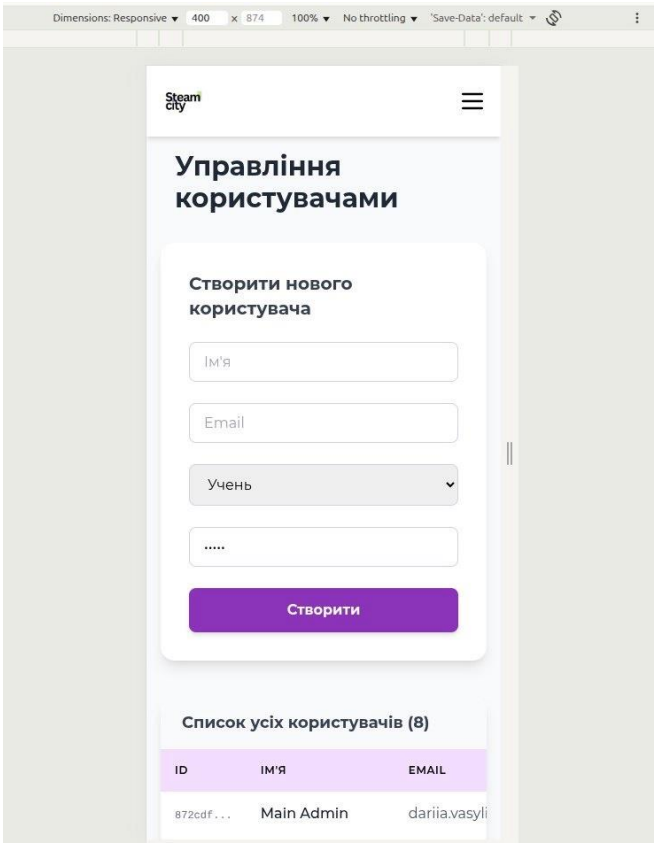


Рисунок 3.20 – Вигляд панелі керування користувачами на мобільному пристрої

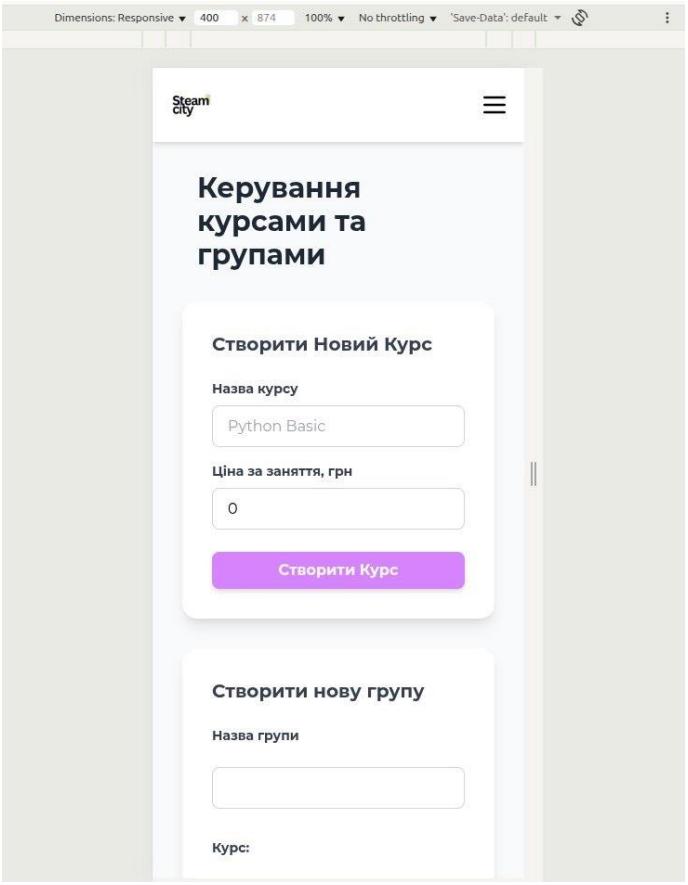


Рисунок 3.21 – Вигляд панелі керування курсами на мобільному пристрої

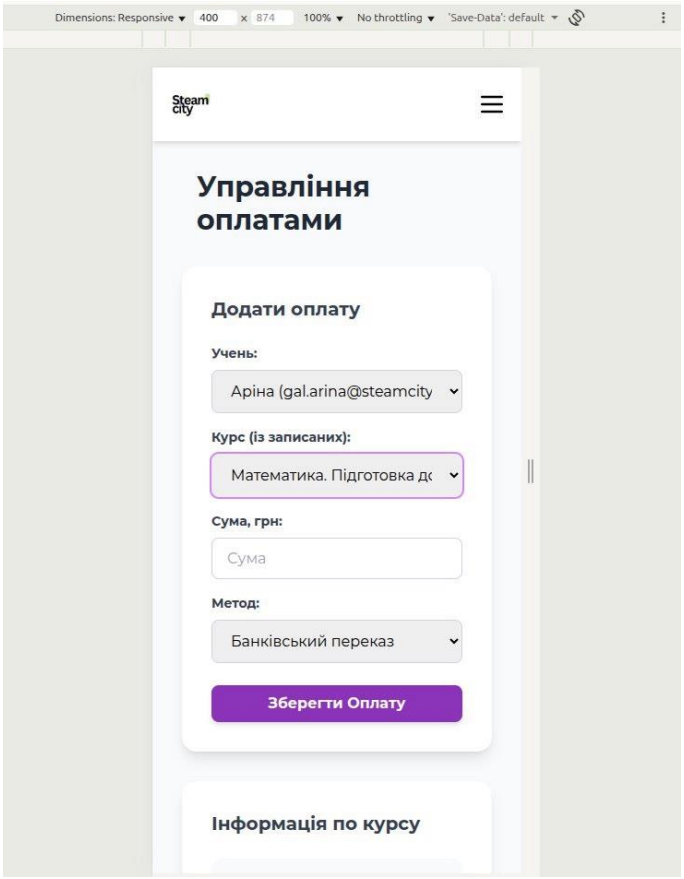


Рисунок 3.22 – Вигляд панелі керування оплатами на мобільному пристрої

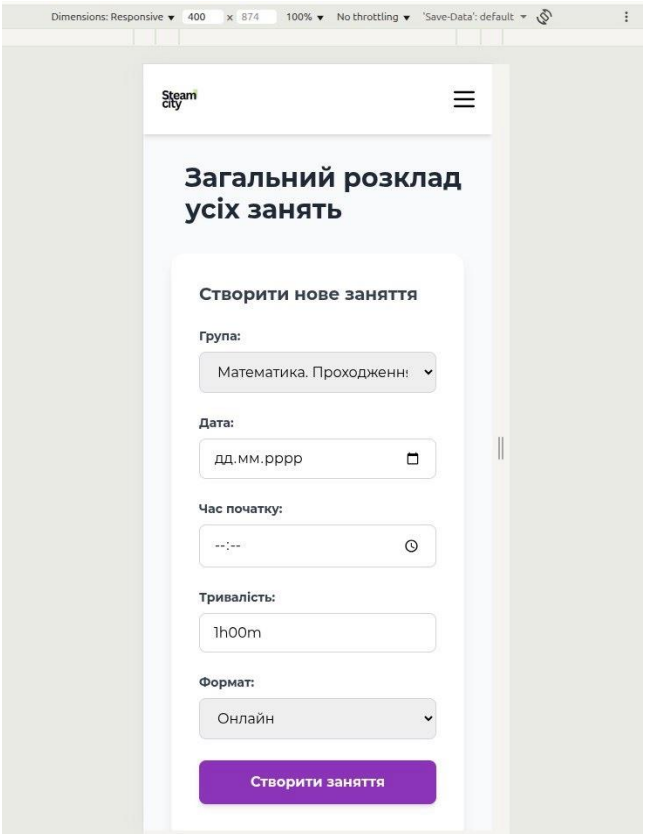


Рисунок 3.23 – Вигляд сторінки з розкладом на мобільному пристрої

Як уже було зазначено, адаптивність сторінки досягнуто за допомогою сіток і флексбоксів. У Tailwind є зручні префікси sm, md та lg для отримання властивостей різного виду – small, medium та large відповідно. Префікс відповідає мінімальній ширині екрану, для вказаних – 640, 768 та 1024 пікселів відповідно. Наприклад, реалізація сторінки з адміністративною панеллю представлено на рисунку 3.24.

```
<div className="min-h-[85vh] bg-gray-50 p-4 md:p-8 max-w-7xl mx-auto">
  <div className="mb-10 text-center">
    <h2 className="text-2xl md:text-4xl font-bold text-primary mb-3">
      Адміністративна Панель
    </h2>
    <p className="text-xl text-gray-600">
      Вітаємо, <span className="font-semibold text-black">{state.user.name}</span>!
    </p>
  </div>

  {/*вміст*/}
</div>
```

Рисунок 3.24 – Реалізація основного контейнеру та заголовку на сторінці адміністративної панелі

Розробка адаптивності вимагає підключення коректних класів із необхідним префіксом. Написання стилів вручну зайняло би дещо більше часу та вимагало би підключення CSS-файлів, а так всі властивості видно відразу в коді.

Отже, як видно, на даній сторінці для основного контейнеру за замовчуванням встановлено внутрішні відступи класом p-4, а при мінімальній ширині екрану 768 пікселів вони будуть збільшені до класу p-8. Для заголовку це працює аналогічно: за замовчуванням розмір тексу становить 24 пікселі (клас text-2xl), а при вказаній мінімальній ширині він зміниться до 36 пікселів (клас text-4xl) тощо.

Загалом реалізація адаптивності по всьому коду виглядає подібним чином, тобто це вказання конкретних властивостей із різними префіксами, тому детальніше можна поглянути на реалізацію в додатках. Такий підхід забезпечує цілісність дизайну та зручність підтримки проєкту.

3.6 Розгортання в мережі

Так як цей продукт запланований для роботи репетиторського центру, цілком виправданою є необхідність у його розгортанні. Варіантом реалізації цього завдання є використання інструментів Render для серверної частини та Vercel для клієнтської.

Render – це платформа, яка дозволяє розгорнути вебсервіси, статичні сайти, Docker контейнери та інше. Проєкт завантажується через посилання на github. Усі оновлення підтягуються автоматично [1].

Vercel – це платформа, яка розгортає проєкт у хмарі. Аналогічно проєкт завантажується через github. При зміні коду необхідно запустити процес розгортання знову [9].

При запуску на одному комп’ютері, сервер і клієнт будуть запускатися на локальних портах 5000 та 3001 відповідно, а для розгортання необхідні актуальні посилання. Для налаштування цього необхідно додати змінну

середовища під назвою REACT_APP_API_URL, яка буде посилатися на наш сервер. За допомогою вказаних інструментів розгортаємо проєкт і отримуємо результат у мережі [7]. Для коректного налаштування зв'язку між серверною та клієнтською частинами, спочатку завантажуємо сервер, вказуємо всі необхідні змінні середовища, такі як токен безпеки та посилання на базу даних. Розгортаємо клієнтську частину та додаємо лише одну змінну середовища – посилання на вже запущений сервер. Таким чином, отримуємо робочий застосунок, розгорнутий у мережі (див. рис. 3.25).

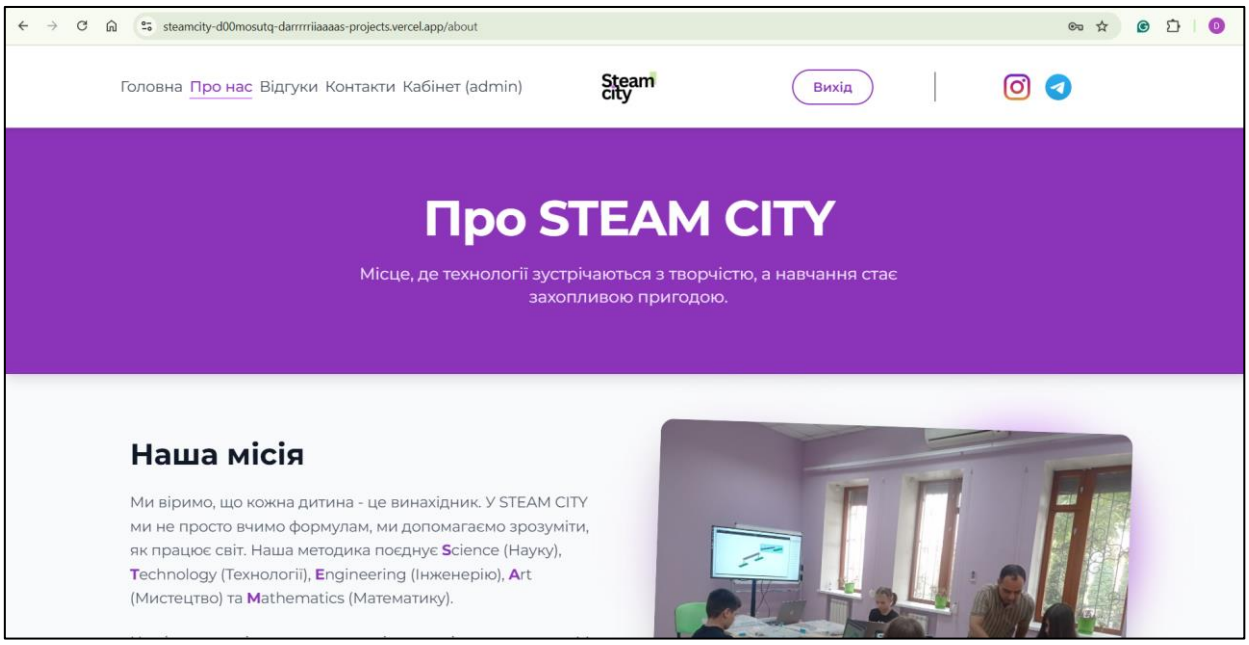


Рисунок 3.25 – Розгорнутий застосунок

3.7 Висновки до розділу 3

У даному розділі було детально описано процес програмної реалізації продукту. Розглянуто окремо розробку клієнтської та серверної частини, а також взаємодію між ними за допомогою обраних інструментів. Із використанням Tailwind CSS досягнуто сучасного стилю вебзастосунку, а також його адаптивності. Було реалізовано перевірку доступу за ролями, щоб забезпечити захист даних.

У результаті проведеного тестування доведено, що реалізовані функції є робочими, дані надійно захищено.

Для забезпечення роботи із продуктом, його було розгорнуто на платформах Render (серверна частина) та Vercel (клієнтська частина).

ВИСНОВКИ

У результаті проведеної роботи було досягнуто поставленої мети та виконано всі завдання, ключовим результатом чого стало створення повністю функціонального вебзастосунку для автоматизації процесів репетиторського центру.

По-перше, було проведено аналіз предметної області, а також пошук і порівняння існуючих рішень. У результаті оптимального рішення для вирішення поставленої проблеми не було знайдено, що і обумовило актуальність даної роботи.

По-друге, було досліджено і проаналізовано стек сучасних технологій. Обрано було мову програмування JavaScript, та фреймворки на її основі: React для розробки клієнтської частини, оскільки він забезпечує зручну розбивку на компоненти, навігацію, керування станами тощо, Node.js та Express для розробки серверної частини, налагодження зв’язку з базою даних, керування маршрутами, безпеки даних. Також було використано фреймворк Tailwind CSS для стилізації і адаптивності відповідно, оскільки він є сучасним і зручним рішенням. Для збереження інформації використовувалася система керування базами даних MongoDB, так як у разі радикальних змін моделей даних вона дуже легко адаптується.

По-третє, для якісної розробки було змодельовано ряд діаграм. Для розуміння взаємодії між користувачами та процесів було розроблено діаграми прецедентів та активностей для кожної ролі. Перед тим як приступати до роботи з базою даних було розроблено ER-діаграму для розгляду всіх зв’язків між сутностями.

Усі попередні етапи підготовки, моделювання та проєктування дозволили успішно реалізувати функціональний застосунок. Він є фулстак проєктом, себто має і клієнтську, і серверну частину, які реалізовані обраними інструментами. Клієнтська частина поділяється на статичні та динамічні сторінки. Статичні сторінки містять загальну інформацію про репетиторський

центр, а динамічні, у залежності від ролі користувача, забезпечують перегляд конкретної інформації. Наприклад, для адміністратора доступні всі дані по центру – перелік учнів, курсів, груп, оплат, загальний розклад. Вчитель і учень бачить кожен свій розклад, і не має доступу до чужого контенту.

Тестування перевірки доступу до чужих даних мала місце, що проілюстровано в п. 3.4. Якщо вчитель або учень спробує перейти за посиланням для адміністратора, клієнтська частина виведе помилку на екран. Також було проведено тестування функціоналу адміністратора, а саме додавання користувачів, курсів, груп, оплат і занять. Усі дані відображаються коректно.

Слід відзначити, що застосунок є адаптивним для різних пристроїв, що реалізовано вже зазначеним фреймворком Tailwind.

Продукт успішно розгорнуто за допомогою платформ Render та Vercel, тож замовник вже може почати працювати з ним безпосередньо з будь-якого комп’ютера.

Перспективним напрямом розвитку проєкту може стати розширення функціоналу для вчителів: наприклад, додати їм доступ на створення своїх занять або зміна дати та часу існуючих, реалізувати функціонал із додаванням матеріалів до занять. Для учнів чудовим буде перегляд своїх оплат, а також можливість оплачувати прямо у вебзастосунку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Cloud Application Platform | Render. *Render*. URL: https://render.com/ (дата звернення: 08.11.2025).

2. Express js: встановлення, налаштування та особливості. *FoxmindEd*. URL: https://foxminded.ua/express-js/ (дата звернення: 27.05.2025).

3. GitHub - darrrrriiaaaa/steamcity. *GitHub*. URL: https://github.com/darrrrriiaaaa/steamcity (дата звернення: 11.11.2025).

4. HTML і CSS: що це, кому та для чого потрібно. *goit.global*. URL: https://goit.global/ua/articles/html-i-css-shcho-tse-komu-ta-dlia-choho-potribno/ (дата звернення: 02.06.2025).

5. JavaScript Історія. Уроки для початківців. W3Schools українською. *W3Schools українською. Безплатні уроки онлайн для початківців, школярів та студентів*. URL: https://w3schoolsua.github.io/js/js_history.html#gsc.tab=0 (дата звернення: 27.05.2025).

6. Responsive design - Core concepts. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. URL: https://tailwindcss.com/docs/responsive-design (дата звернення: 06.11.2025).

7. STEAM City. *STEAM City*. URL: https://steamcity.vercel.app/ (дата звернення: 08.11.2025).

8. TailwindCSS – основи та переваги | sparklingman. *sparklingman*. URL: https://sparklingman.com/blog/stilizuj-z-tailwind (дата звернення: 02.06.2025).

9. Vercel: Build and deploy the best web experiences with the AI Cloud – Vercel. *Vercel*. URL: https://vercel.com/ (дата звернення: 08.11.2025).

10. Вальченко М. MongoDB: що це за СУБД, переваги та недоліки. *goit.global*. URL: https://goit.global/ua/articles/mongodb-shcho-tse-take-ta-yak-prazuie/ (дата звернення: 08.08.2025).

11. Вступ до JavaScript. *Сучасний підручник з JavaScript*. URL: https://uk.javascript.info/intro (дата звернення: 27.05.2025).

12. Карпушина М., Шумило І. Використання освітніх інтернет-платформ у навчанні іноземної мови. *Львівська політехніка*. URL: https://ena.lpnu.ua:8443/server/api/core/bitstreams/d27b4789-1142-43f4-9faf-6922c672b230/content (дата звернення: 14.04.2025).

13. Козюля А. Огляд найпопулярніших JavaScript-фреймворків: прості пояснення та практичні поради. *Dan.it Education*. URL: https://dan-it.com.ua/uk/blog/obzor-samyh-populyarnyh-javascript-frejmvorkov-prostye-obyasneniya-i-prakticheskie-sovety/ (дата звернення: 21.05.2025).

14. Компоненти і пропси – React. *React – JavaScript-бібліотека для створення користувацьких інтерфейсів*. URL: https://uk.legacy.reactjs.org/docs/components-and-props.html (дата звернення: 26.05.2025).

15. Корильчук Н. І., Первак М. П., Чернова Т. Ю. Аналіз дистанційних платформ для навчання і саморозвитку здобувачів вищої освіти в контексті воєнних реалій. *Академічні візії*. URL: https://academy-vision.org/index.php/av/article/download/116/106/107 (дата звернення: 11.04.2025).

16. Легше, швидше, масштабніше. Чому топові компанії використовують Node.js. *Projector – Creative & Tech Online Institute*. URL: https://prjctr.com/mag/whynode (дата звернення: 27.05.2025).

17. Острога М., Шамоня В., Шершень О. Цифрові освітні платформи як інструмент реалізації неформальної освіти. *Освітаю Інноватика. Практика*. URL: https://oip-journal.org/index.php/oip/article/view/60/46 (дата звернення: 11.04.2025).

18. Рейтинг мов програмування 2025. *DOU*. URL: https://dou.ua/lenta/articles/language-rating-2025/ (дата звернення: 26.05.2025).

19. Сірський Л. Вплив платформ для онлайн-курсів на навчання в школах. *Освіта.UA*. URL: https://osvita.ua/news/89291/ (дата звернення: 11.04.2025).

20. Січкаренко К. О. *Журнал «Ефективна економіка» - наукове фахове видання з питань економіки*. URL:

http://www.economy.nayka.com.ua/pdf/12_2018/117.pdf (дата звернення: 11.04.2025).

21. Широков Д. Л. Платформатизація освтінього процесу під час дистанційного та змішаного навчання в школі. *Research Gate*. URL: https://www.researchgate.net/publication/352430583_Platformatizacia_osvtinogo_procesu_pid_cas_distancijnogo_ta_zmisanogo_navcanna_v_skoli (дата звернення: 11.04.2025).

22. Яковлєва І. Використання освітніх платформ в освітньому середовищі | Український Педагогічний журнал. *Український Педагогічний журнал*. URL: https://uej.undip.org.ua/index.php/journal/article/view/619/592 (дата звернення: 11.04.2025).

ДОДАТОК А

Будова сторінок у клієнтській частині

А.1 Панель керування адміністратора – AdminDashboard.jsx

```
import React, { useEffect } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const AdminDashboard = () => {
  const { state } = useAuth();
  const navigate = useNavigate();

  console.log("Поточний користувач:", state.user);

  useEffect(() => {
    if (!state.user) {
      navigate('/authorize');
    }
    else if (state.user.role !== 'admin') {
      alert('Доступ заборонено! Ви не адміністратор.');
      navigate(`/dashboard/${state.user.role}`);
    }
  }, [state.user, navigate]);

  if (!state.user || state.user.role !== 'admin') {
    return (
      <div className="min-h-[80vh] flex flex-col items-center justify-center bg-gray-50 px-4">
        <div className="bg-white p-8 rounded-2xl shadow-lg text-center max-w-md border border-gray-100">
          <h2 className="text-2xl font-bold text-gray-800 mb-2">Доступ заборонено</h2>
          <p className="text-gray-600 mb-6">
            Ця сторінка доступна лише адміністраторам. <br/>
            Ваша роль: <span className="font-bold text-primary">{state.user ? state.user.role : 'Гість'}</span>
          </p>
          <button
            onClick={() => navigate('/')}
            className="bg-primary text-white px-6 py-2 rounded-full font-semibold hover:bg-opacity-90
transition shadow-md"
          >
            Повернутися на головну
          </button>
        </div>
      </div>
    );
  };

  return (
    <div className="min-h-[85vh] bg-gray-50 p-4 md:p-8 max-w-7xl mx-auto">
      <div className="mb-10 text-center">
```

```
        <h2 className="text-2xl md:text-4xl font-bold text-primary mb-3">
          Адміністративна Панель
        </h2>
        <p className="text-xl text-gray-600">
          Вітаємо, <span className="font-semibold text-black">{state.user.name}</span>!
        </p>
      </div>

      <section>
        <h3 className="text-2xl font-bold text-gray-800 mb-6 pl-2 border-l-4 border-accent">Управління Системою</h3>
        <div className="grid grid-cols-1 md:grid-cols-2 gap-4 md:gap-8">
          <Link to="/admin/users" className="group bg-white p-5 md:p-8 rounded-2xl shadow-md hover:shadow-xl transition-all duration-300 transform hover:-translate-y-1 border border-gray-100">
            <div className="flex items-center space-x-4">
              <div className="text-3xl md:text-4xl bg-secondary p-3 md:p-4 rounded-full group-hover:bg-primary group-hover:text-white transition-colors">
                👥
              </div>
              <div>
                <h4 className="text-2xl font-bold text-gray-800 group-hover:text-primary transition-colors">Користувачі</h4>
                <p className="text-gray-500 mt-1">Створення, блокування та редагування</p>
              </div>
            </div>
          </Link>
          <Link to="/admin/groups" className="group bg-white p-8 rounded-2xl shadow-md hover:shadow-xl transition-all duration-300 transform hover:-translate-y-1 border border-gray-100">
            <div className="flex items-center space-x-4">
              <div className="text-3xl md:text-4xl bg-secondary p-3 md:p-4 rounded-full group-hover:bg-primary group-hover:text-white transition-colors">
                📚
              </div>
              <div>
                <h4 className="text-2xl font-bold text-gray-800 group-hover:text-primary transition-colors">Курси та групи</h4>
                <p className="text-gray-500 mt-1">Формування груп та навчальних програм</p>
              </div>
            </div>
          </Link>
          <Link to="/admin/payments" className="group bg-white p-8 rounded-2xl shadow-md hover:shadow-xl transition-all duration-300 transform hover:-translate-y-1 border border-gray-100">
            <div className="flex items-center space-x-4">
              <div className="text-3xl md:text-4xl bg-secondary p-3 md:p-4 rounded-full group-hover:bg-primary group-hover:text-white transition-colors">
                💰
              </div>
              <div>
                <h4 className="text-2xl font-bold text-gray-800 group-hover:text-primary transition-colors">Фінанси</h4>
                <p className="text-gray-500 mt-1">Облік оплат та фінансова звітність</p>
              </div>
            </div>
          </Link>
```

```
            <Link to="/schedule" className="group bg-white p-8 rounded-2xl shadow-md hover:shadow-xl transition-all duration-300 transform hover:-translate-y-1 border border-gray-100">
              <div className="flex items-center space-x-4">
                <div className="text-3xl md:text-4xl bg-secondary p-3 md:p-4 rounded-full group-hover:bg-primary group-hover:text-white transition-colors">
                  
                </div>
                <div>
                  <h4 className="text-2xl font-bold text-gray-800 group-hover:text-primary transition-colors">Загальний розклад</h4>
                  <p className="text-gray-500 mt-1">Календар занять усіх груп</p>
                </div>
              </div>
            </Link>
          </div>
        </section>
      </div>
  );
};

export default AdminDashboard;
```

А.2 Забезпечення захисту за допомогою компонента ProtectedRoute.jsx

```
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const ProtectedRoute = ({ allowedRoles = [], children }) => {
  const { state } = useAuth();
  const user = state.user;

  if (!user) {
    return <Navigate to="/authorize" replace />;
  }

  if (allowedRoles.length > 0 && !allowedRoles.includes(user.role)) {
    return <Navigate to={`http://localhost:5000/dashboard/${user.role}`} replace />;
  }

  return children;
};

export default ProtectedRoute;
```

ДОДАТОК Б

Реалізація серверної частини

Б.1 server.js

```
import express from 'express';
import dotenv from 'dotenv';
import cors from 'cors';

import connectDB from './config/db.js';

import userRoutes from './routes/userRoutes.js';
import authRoutes from './routes/authRoutes.js';
import groupRoutes from './routes/groupRoutes.js';
import lessonRoutes from './routes/lessonRoutes.js';
import materialRoutes from './routes/materialRoutes.js';
import paymentRoutes from './routes/paymentRoutes.js';

dotenv.config();

connectDB();

const app = express();

app.use(cors());

app.use(express.json());

app.get('/', (req, res) => {
   res.send('API works');
});

app.use('/api/users', userRoutes);
app.use('/api/auth', authRoutes);
app.use('/api/groups', groupRoutes);
app.use('/api/schedule', lessonRoutes);
app.use('/api/materials', materialRoutes);
app.use('/api/payments', paymentRoutes);

app.use((err, req, res, next) => {
   const statusCode = res.statusCode === 200 ? 500 : res.statusCode;
   res.statusCode(statusCode);
   res.json({message: err.message});
});

const PORT = process.env.PORT || 5000;

app.listen(
   PORT,
   () => console.log(`Server running on port ${PORT}`)
);
```