КВАЛІФІКАЦІЙНА РОБОТА

Група МІПЗс-22
Насадик В.В.

2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Насадик Владислав Вікторович

УДК 004.378

Розробка та дослідження нового програмного забезпечення для вдосконалення та автоматизації процесів освітнього наставництва

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації магістра

Нормоконтроль

_____Стисло О.В.
(підпис, дата, розшифрування підпису)

Студент

_____ Насадик В.В.
(підпис, дата, розшифрування підпису)

Допускається до захисту
Завідувач кафедри

_____к.т.н., доц. Ващишак С.П.
(підпис, дата, розшифрування підпису)

Керівник роботи

к.т.н. доц.
_____ Ващишак С.П.
(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «магістр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

« 19 » лютого 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Насадик Владислав Вікторович

(прізвище, ім'я, по батькові)

1. Тема магістерської роботи

Розробка та дослідження нового програмного забезпечення для вдосконалення та автоматизації процесів освітнього наставництва

керівник роботи:

Ващищак Сергій Петрович, к.т.н., доцент

затверджена наказом вищого навчального закладу від « 26 » червня 2023 року № 32/1 с

Термін подання студентом роботи 16.02.2024

3. Зміст магістерської роботи (перелік питань, які потрібно розробити)

1. Опис та аналітика конкурентів

2. Вибір веб-платформ та використаних технологій

3.Розробка структури проекту, програмного забезпечення та аналіз ефективності веб-програми

4. Дата видачі завдання 29.06.2023

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Опис та аналітика даних (графіків та таблиць) конкурентів зібраних за період дослідження.	12.08.2023	Виконано
2.	Опис використовуваних мов програмування та їх алгоритм роботи.	11.09.2023	Виконано
4.	Опис роботи та відмінність браузерів в яких буде проводитись дослідження.	23.11.2023	Виконано
5.	Розробка структури проекту та розробка програмного забезпечення.	28.12.2023	Виконано

Студент _____ Насадик В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ващишак С.П.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
12	Інтерфейс Zoom	48	Реалізація маршрутів для авт.
22	Синтаксис HTML	52	Middleware для генерації паролю
23	Синтаксис CSS	54	Модель уроку
23	Синтаксис JS	68	Сторінка реєстрації
25	Приклад вибірки з MongoDB	69	Сторінка логіну
31	Сценарій роботи Cross-Site Scripting	70	Сторінка предмету
34	Діаграма компонентів архітектури проекту	74	Модальне вікно з заняттям
38	Діаграма зв'язків таблиць в базі даних MongoDB	77	Інт. звіту про прогрес студента
45	Код для реалізації сервера в файлі index.js	88	Приклад моніт. сервера

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці та дослідженню нового програмного забезпечення для вдосконалення та автоматизаії процесів освітнього наставництва.

В першому розділі проведено аналітичний огляд, опис та аналітику конкурентів та їх стратегій.

В другому розділі розроблено оглянуто та обгрунтовано вибір технологій та веб-платформи, яка буде використуватись для розробки веб-програми.

В третьому розділі проведена розробка структури веб-програми, яка включає в себе архітектуру програмного забезпечення та розроблене програмне забезпечення та проведений аналіз ефективності веб-програми.

КЛЮЧОВІ СЛОВА: HTML, CSS, JAVASCRIPT, REACT, NODE.JS, ВЕБ-ПРОГРАМА.

SUMMARY

The qualification work is devoted to the development and research of new software for the improvement and automation of educational mentoring processes.

In the first section, an analytical review, description and analysis of competitors and their strategies was carried out.

In the second section, the choice of technologies and web platform, which will be used for the development of the web application, is developed in a comprehensive and justified manner.

In the third section, the structure of the web application, which includes the software architecture, is developed.

In the fourth chapter, the software was developed and the analysis of the effectiveness of the web program was carried out.

KEYWORDS: HTML, CSS, JAVASCRIPT, REACT, NODE.JS, WEB PROGRAM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ 8

ВСТУП 9

РОЗДІЛ 1. ОПИС ТА АНАЛІТИКА КОНКУРЕНТІВ 12

1.1. Визначення конкурентів. Перелік основних конкурентів та їх профілі 12

1.2. Аналіз стратегій, які використовуються конкурентами 15

1.3. Виявлення сильних та слабких сторін конкурентів 16

1.4. Аналіз аудиторії та клієнтської бази конкурентів 17

1.5. Аналіз інновацій та нових тенденцій 18

1.6 Постановка задачі 20

Висновок до розділу 1 20

РОЗДІЛ 2. ВИБІР ВЕБ-ПЛАТФОРМ ТА ВИКОРИСТАНИХ ТЕХНОЛОГІЙ 22

2.1 Технологічний ландшафт. Огляд загальних технологій 22

2.2. Обгрунтування вибору конкретних технологій для проекту 26

2.3. Визначення технічних вимог до програмного забезпечення 28

2.4. Огляд заходів забезпечення безпеки використаних технологій 30

Висновок до розділу 2 32

РОЗДІЛ 3. РОЗРОБКА СТРУКТУРИ ПРОЕКТУ, ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ВЕБ-ПРОГРАМИ 33

3.1. Огляд загальної архітектури проекту 33

3.2. Опис розділення проекту на модулі та їх функціональність 34

3.3. Опис структури баз даних та методів зберігання і обробки даних 37

3.4. Інтерфейс користувач. Розгляд інтерфейсу та його функціональність 39

3.5. Опис методів управління розробкою проекту та ресурсами 42

3.6 Розробка модулів серверної частини та їх функціональності 44

3.7 Розробка структури бази даних та методів зберігання даних 53

3.8 Розробка окремих контролерів для управління даними 57
3.9 Розробка інтерфейсу користувача, та його функціональність 67
3.10 Забезпечення безпеки веб-програми та захист від потенційних загроз 80
3.11 Управління проектом та задачами 82
3.12 Опис процесу інтеграції програмного забезпечення 84
3.13 Плани для моніторингу веб-програми після впровадження 86
3.14 Аналіз продуктивності. Оцінка ефективності веб-програми 88
3.15 Аналіз користувацького досвіду (UX) та ітеративне поліпшення інтерфейсу 89
3.16 Розгляд сумісності з різними браузерами та пристроями 94
3.17 Обгрунтування можливих напрямків для подальшого розвитку 97
3.18 Способи отримання фідбеку від користувачів 99
3.19 Виконання вимог законодавства щодо захисту даних 102
3.20 Розробка тестових сценаріїв та їх виконання 103
3.21 Вивчення та інтеграція нових технологій 105
3.22 Розробка резервних планів та стратегій відновлення 105
3.23 Оцінка впливу на навколишнє середовище 106
3.24. Розробка стратегії масштабування та росту 107
Висновок до розділу 3 110
ВИСНОВКИ 111
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 113

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

HTML – HyperText Markup Language; CSS – Cascading Style Sheets;

JS – JavaScript;

NPM – Node Package Manager;

IDE – Integrated development environment; CDN – Content Delivery Networ;

CLI – Command line interface; DOM – Document Object Model;

LTS – Long-Term Support.

API - Application Programming Interface

SPA - Single Page Application

JSX - JavaScript XML

CRUD - Create, Read, Update, Delete

SSR - Server-Side Rendering

JWT - JSON Web

REST - Representational State Transfer

ORM - Object-Relational Mapping

UI - User Interface

UX - User Experience

CORS - Cross-Origin Resource Sharing

API endpoint - точка доступу до API

ВСТУП

Актуальність теми. Розробка та дослідження нового програмного забезпечення для вдосконалення та автоматизації процесів освітнього наставництва є надзвичайно актуальною темою в сучасному світі. Наразі, у зв'язку з швидкими змінами та викликами, які постала в освіті та навчанні, ефективні інструменти та програмне забезпечення, що допомагають вчителям та студентам, набули особливого значення.

З підвищенням популярності дистанційного навчання та онлайн-курсів, які стали необхідними через події останніх років, з'явилася потреба у розвитку інноваційних інструментів для навчання. Програмне забезпечення для освіти повинно бути зручним, ефективним та відповідати сучасним вимогам користувачів.

Цей проект може вирішити ці виклики та надати нові можливості для покращення освітнього процесу. Дослідження та розробка програмного забезпечення для освітнього наставництва може допомогти підвищити якість навчання та сприяти зручності вчителів та студентів. Така робота є актуальною та важливою для покращення якості освіти та навчання.

Мета і завдання дослідження. Метою дослідження є розробка та дослідження нового програмного забезпечення для вдосконалення та автоматизації процесів освітнього наставництва. Дане дослідження спрямоване на досягнення наступних завдань:

- Аналіз потреб освітнього наставництва. Дослідження конкретних потреб та вимог освітнього наставництва, які можуть бути вдосконалені та автоматизовані за допомогою програмного забезпечення.
- Вибір технологічних рішень. Обгрунтування вибору певних технологій та інструментів для розробки програмного забезпечення, які найкраще відповідають потребам освітнього наставництва.

- Розробка програмного забезпечення. Створення програмного забезпечення, яке дозволить вдосконалити та автоматизувати конкретні процеси освітнього наставництва, зокрема взаємодію вчителів та студентів, оцінювання успішності тощо.
- Аналіз ефективності програмного забезпечення. Оцінка ефективності та корисності розробленого програмного забезпечення для освітнього наставництва, враховуючи його вплив на якість навчання та робочий процес вчителів.
- Рекомендації та подальший розвиток. Надання рекомендацій для поліпшення та розвитку програмного забезпечення, а також визначення можливих напрямків для подальшого вдосконалення проекту та збільшення його користі.

Об’єктом дослідження є процеси та практики, пов'язані з освітнім наставництвом, а також розробка програмного забезпечення для їх поліпшення та автоматизації.

Предметом дослідження є нове програмне забезпечення, яке розробляється для вдосконалення та автоматизації процесів освітнього наставництва.

Методи дослідження базуються на теорії інформації, включають аналіз літературних джерел, експерименти та тестування для отримання інформації.

Наукова новизна одержаних результатів. В роботі вперше проведена розробка та дослідження програмного забезпечення, яке вдосконалює та автоматизує процеси освітнього наставництва, і визначення оптимальних технологічних рішень для підвищення якості навчання та робочого процесу вчителів і студентів.

Практичне значення одержаних результатів. Полягає в можливості впровадження розробленого програмного забезпечення в освітнє середовище для поліпшення навчання та спілкування між вчителями та студентами. Це допоможе зробити освіту більш доступною та ефективною для всіх учасників.

Апробація результатів дослідження. Матеріали кваліфікаційної роботи були представлені на І Всеукраїнській науково-практичній інтернет-конференції «ІТ ЕКОСИСТЕМА: Цифровізація бізнес-процесів в умовах війни», яка відбулась 23-24 листопада 2023 року в університеті Короля Данила.

Структура. Загальний обсяг сторінок основної частини є 115 сторінок. Список використаних джерел містить 41 позицію.

РОЗДІЛ 1. ОПИС ТА АНАЛІТИКА КОНКУРЕНТІВ

1.1 Визначення конкурентів. Перелік основних конкурентів та їх профілі

У контексті розробки та дослідження нового програмного забезпечення для вдосконалення та автоматизації процесів освітнього наставництва, важливо провести аналіз конкурентного середовища для кращого розуміння потенційної конкуренції та визначення унікальних переваг нашого продукту.

Одним з основних конкурентів є популярна платформа для віддаленого навчання та репетиторства, така як Zoom (рис. 1.1). Zoom володіє широким функціоналом для віддалених занять, включаючи відеоконференції та можливості спільної роботи над документами. Проте, наш продукт має на меті не лише забезпечити спілкування, але і повністю охопити всі аспекти організації та ведення занять.

Рисунок 1.1 – Приклад інтерфейсу Zoom

Іншим конкурентом може бути платформа для електронного навчання, така як Moodle (рис. 1.2). Moodle забезпечує можливості створення курсів та взаємодії з учнями. У нашому випадку, ми прагнемо зосередитися на індивідуальному підході до взаємодії між наставником і учнем, надаючи інструменти для персоналізації та ведення статистики.

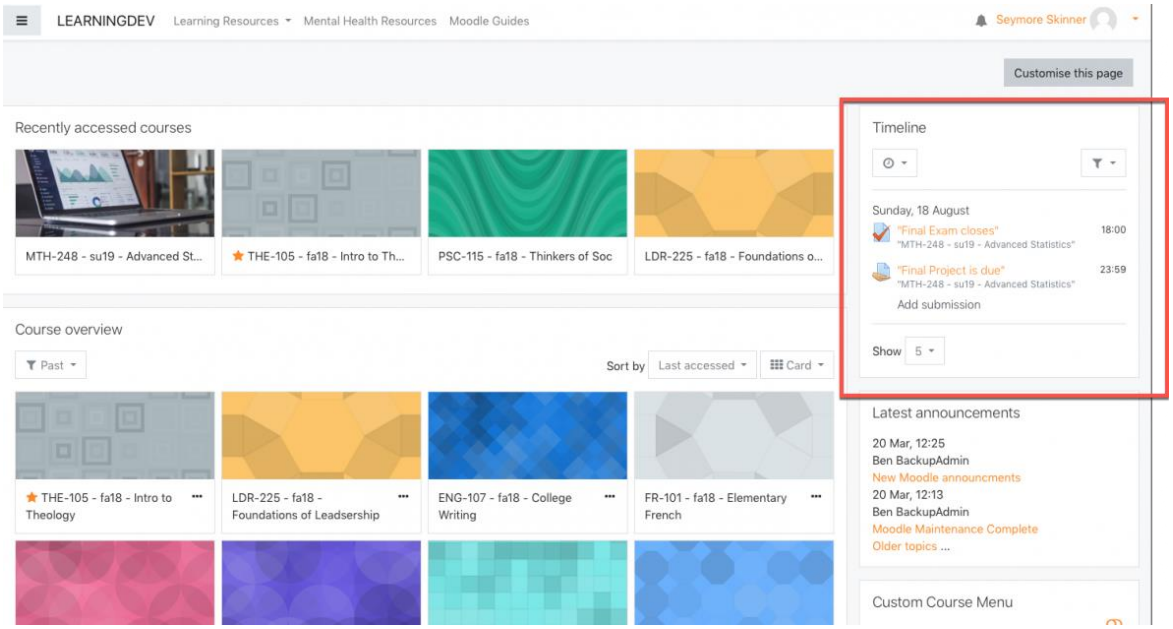


Рисунок 1.2 – Приклад інтерфейсу Moodle

Також слід врахувати конкуренцію від інших платформ репетиторства, таких як Tutor.com чи Chegg Tutors. Ці платформи спеціалізуються на забезпеченні послуг репетиторства, і наше програмне забезпечення повинно виокремитися як унікальний та більш інтегрований інструмент для наставників.

У світі ринку освітніх технологій є й інші значущі конкуренти. Наприклад, платформа Google Classroom (рис. 1.3) вже завоювала визнання серед освітян за можливість швидкого і простого створення віртуальних класів, обмін документами та зручне ведення спільних завдань. Google Classroom зараз є гігантом на ринку перемогти її в конкуренції буде дуже складно, адже люди довіряють платформі Google.

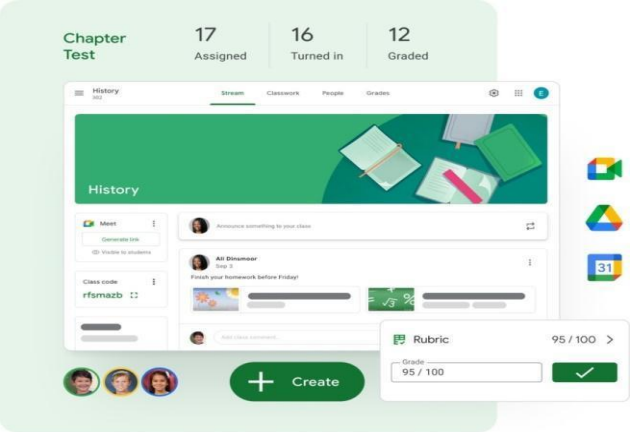


Рисунок 1.3 – Приклад інтерфейсу Google Classroom

Ще одним конкурентом може бути платформа Microsoft Teams (рис. 1.4), яка також пропонує інструменти для віддалених занять та співпраці. Microsoft Teams вражає розширеним набором інтегрованих програм та можливістю проведення онлайн-зустрічей. Наше програмне забезпечення буде ставити на завдання підвищення ефективності освітнього процесу через інтеграцію всіх аспектів репетиторської роботи, щоб сприяти зростанню якості освіти.

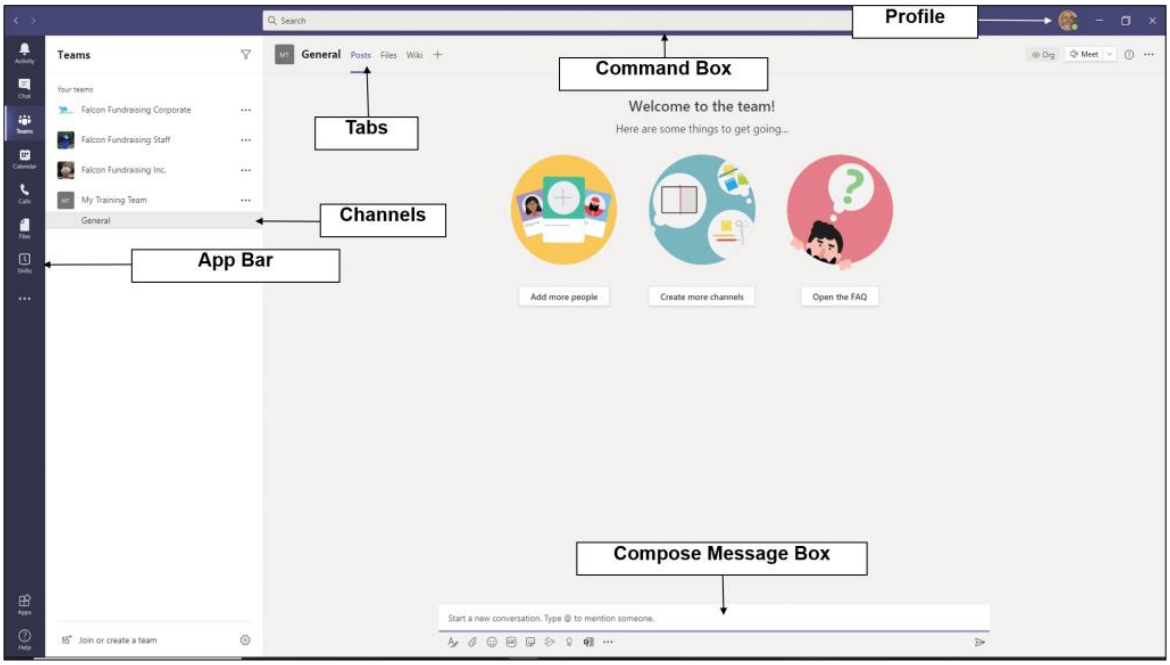


Рисунок 1.4 – Приклад інтерфейсу Microsoft Teams

Ми спрямовуємося на покращення індивідуального навчання, надаючи наставникам інструменти для персоналізації та взаємодії з учнями на більш глибокому рівні.

Важливо враховувати різноманітність конкурентів і їхні підходи до вирішення схожих завдань. Це надасть можливість визначити прогалини в ринковій пропозиції та підготувати ефективний стратегічний план для подальшого розвитку нашого програмного забезпечення.

1.2 Аналіз стратегій, які використовуються конкурентами

Основні стратегії конкурентів наведені в таблиці 1.1.

Таблиця 1.1

Аналіз основних стратегій конкурентів

Конкурент	**Основна стратегія**
Zoom	Платформа фокусується на відеоконференціях та спільній роботі над документами. Забезпечує широкі можливості віддаленого спілкування та співпраці, але більше зорієнтована на широкий спектр використання.
Moodle	Зосереджена на створенні та управлінні електронними курсами. Надає функціонал для взаємодії з учнями, але менше акцентується на індивідуальному репетиторстві та персоналізації навчального процесу.
Google Classroom	Надає швидке та просте створення віртуальних класів, обмін документами та роботу з завданнями в колективі. Фокусується на ефективності в організації групового навчання.
Microsoft Teams	Інтегрована платформа для спільної роботи, яка пропонує широкий спектр інструментів для організації онлайн-зустрічей та співпраці в команді. Спрямована на корпоративне та освітнє використання.

Аналіз стратегій ваших конкурентів дозволить виявити важливі аспекти, на які вони наголошують у своєму підході до освітніх технологій.

Кожен конкурент визначає свої основні стратегії для вирішення конкретних завдань у дистанційній освіті та репетиторських послугах.

Один із головних гравців, Zoom, прагне забезпечити ефективні відеоконференції та співпрацю над документами. Це робить платформу ідеальною для різноманітних додатків, але може бути непридатною для потреб навчання та персоналізації навчальних курсів.

Moodle спеціалізується на створенні та управлінні електронними курсами, але менше фокусується на індивідуальному навчанні.

Це робить платформу ефективною для організації масштабних процесів навчання, також вона широко використовується в університетах але може бути непридатною для особистої взаємодії між викладачами та студентами.

Google Classroom пропонує корисні інструменти для віртуального командного навчання з акцентом на ефективності організації групових уроків. Однак він не підходить для керівництва в класі та персоналізації.

Microsoft Teams поєднує різноманітні інструменти для командної співпраці та організації онлайн зустрічей. Він зосереджений на бізнесі та освіті.

Наша стратегія полягає в тому, щоб розробити програмне забезпечення, яке поєднує в собі сильні сторони наших конкурентів і усуває їхні слабкі сторони, забезпечуючи інтуїтивно зрозумілий інтерфейс, високі налаштування та ефективну взаємодію між викладачами та студентами.

1.3 Виявлення сильних та слабких сторін конкурентів

1. Zoom**:**

1.1 Сильні сторони:

- висока якість відеоконференцій та зручна спільна робота;
- широкий функціонал для віддаленого спілкування.

1.2 Слабкі сторони:

- обмежені можливості для організації навчального процесу;
- може бути менше підходить для індивідуального репетиторства.

2. Moodle:

2.1 Сильні сторони:

- висока функціональність для створення електронних курсів;
- підтримка великої кількості користувачів та груп.

2.2 Слабкі сторони:

- обмежені можливості для індивідуального репетиторства;
- може бути менше зручним для використання вчителями.

3. Google Classroom:

3.1 Сильні сторони:

- легкість використання та швидкість створення віртуальних класів;
- зручна організація групових завдань.

3.2 Слабкі сторони:

- обмежені функціональність для індивідуальних навчальних потреб;
- може виявитися менш ефективним для особистого репетиторства.

4. Microsoft Teams:

4.1 Сильні сторони:

- інтегровані інструменти для спільної роботи;
- підтримка великої кількості користувачів.

4.2 Слабкі сторони:

- може бути менше спрямованим на освітній сектор;
- можливі обмежені можливості для індивідуального навчання.

Аналіз сильних та слабких сторін конкурентів допоможе нам не лише виявити їхні недоліки, але і спрямує розробку програмного забезпечення на виправлення цих вад та надання переваг у сфері індивідуального репетиторства та ефективного взаємодії між наставниками та учнями.

1.4 Аналіз аудиторії та клієнтської бази конкурентів

Zoom привертає різноманітну аудиторію, включаючи корпорації, освітні

установи та індивідуальних користувачів. Його клієнтська база розподілена між великими компаніями, які використовують платформу для віддалених нарад, та індивідуальними користувачами, які обирають її для особистого використання та навчання.

Moodle основно використовується у сфері освіти, зокрема в університетах, школах та інших навчальних закладах. Його клієнтська база складається з освітніх установ, які використовують платформу для електронного навчання та організації курсів для студентів.

Аудиторія Google Classroom складається з учителів, учнів та шкіл, які використовують платформу для проведення онлайн-уроків та навчання в групах. Його клієнтська база орієнтована на навчальні установи та індивідуальних вчителів.

Microsoft Teams привертає корпоративних клієнтів, освітні заклади та підприємства. Клієнтська база Microsoft Teams складається з великих корпорацій, які використовують платформу для спільної роботи та комунікації, а також освітніх установ, що організовують онлайн-навчання та взаємодію зі студентами.

Аналіз цільової аудиторії та клієнтської бази конкурентів надає уявлення про сегменти ринку, які вони успішно обслуговують, і сприяє визначенню наших конкурентних переваг у залученні та задоволенні потреб аудиторії.

1.5 Аналіз інновацій та нових тенденцій

В сучасному світі розробка програмного забезпечення для навчання та репетиторства відзначається активним впровадженням інновацій та врахуванням нових тенденцій. Для досягнення успіху в цьому сегменті ринку необхідно уважно слідкувати за останніми технологічними розвитками та враховувати змінні потреби користувачів.

Інтеграція штучного інтелекту (ШІ) та машинного навчання (МН):

однією з ключових тенденцій є використання ШІ та МН для персоналізації навчання. Системи можуть аналізувати студентські дані та надавати індивідуальні рекомендації, щоб оптимізувати процес навчання.

Розширена реальність (AR) та віртуальна реальність (VR): використання AR та VR дозволяє створювати іммерсивні навчальні середовища, що поліпшують розуміння матеріалу та забезпечують взаємодію на більш глибокому рівні.

Безпека та конфіденційність даних: у світлі зростання кількості онлайн-навчання, безпека та конфіденційність даних стають пріоритетом. Використання захищених протоколів та шифрування гарантує захист інформації користувачів.

Мобільні застосунки та доступність: застосунки для мобільних пристроїв стають все більш популярними, адже вони забезпечують гнучкість та доступність навчання з будь-якого місця та в будь-який час.

Співпраця та соціальні взаємодії: підтримка функцій співпраці та соціальної взаємодії, таких як форуми та чати, сприяє створенню сприятливого середовища для обміну ідеями та колективного навчання.

Вивчення інновацій та тенденцій у сфері програмного забезпечення для навчання надає нам важливий інструмент для формування нашого продукту відповідно до найсучасніших вимог користувачів. На основі проведеного аналізу ми визначили ключові напрямки розвитку, які включають персоналізацію та адаптивність, використання віртуальної та змішаної реальності, аналітику великих даних, мобільність та доступність, а також акцент на онлайн-спільнотах та співпраці.

Цей інструмент визначає нашу стратегію активного впровадження цих інновацій у розробку програмного продукту. Наша мета - забезпечити користувачам не лише сучасні, але й ефективні інструменти для навчання, враховуючи найновіші досягнення технологій та відповідаючи вимогам сучасного освітнього середовища.

1.6 Постановка задачі

Дослідження свідчить, що сучасні технології можуть впливати на покращення освітнього процесу, зокрема за допомогою розробки нового програмного забезпечення.

Освітнє наставництво вимагає систематизації та автоматизації, тому метою цього дослідження є створення та аналіз нового програмного забезпечення для вдосконалення процесів освітнього наставництва.

Завдання кваліфікаційної роботи:

- розробити нове програмне забезпечення для освітнього наставництва з урахуванням потреб користувачів та сучасних тенденцій у галузі;
- порівняти розроблене програмне забезпечення з існуючими конкурентами на ринку освітніх технологій;
- визначити стратегії, які використовують конкуренти в галузі освітнього наставництва та встановити їх недоліки;
- виявити сильні та слабкі сторони конкурентів у сфері освітніх технологій;
- проаналізувати аудиторію та клієнтську базу конкурентів для визначення потенційної аудиторії нового програмного забезпечення.

Висновок до розділу 1

У розділі наведено обґрунтування та контекст для проведення дослідження. Аналіз сучасних технологій підтвердив можливість їхнього позитивного впливу на освітній процес.

Зазначено необхідність створення нового програмного забезпечення для оптимізації освітнього наставництва.

В розділі проведений аналіз причин, які роблять цю кваліфікаційну роботу актуальною та унікальною в онлайн-середовищі. Детально розглянуто

причину, через яку дана робота набуває важливості та особливості в інтернеті. При цьому були розкриті технології, які планується використовувати при розробці проекту.

Сформульовано завдання, яке передбачає вирішення в рамках цієї кваліфікаційної роботи. Такий аналіз створює обґрунтування для дослідження та розробки, підкреслюючи унікальний характер та важливість цього проекту в онлайн-середовищі.

РОЗДІЛ 2. ВИБІР ВЕБ-ПЛАТФОРМ ТА ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1 Технологічний ландшафт. Огляд загальних технологій

Щоб розпочати розробку веб-додатків, я використовую HTML для створення базової структури сторінок і їх розмітки (рис. 2.1). Це допомагає забезпечити логічну організацію вмісту та базову архітектуру програми.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <main class="container">
    <section class="features">
      <div class="feature-item">
        <img src="#" width="300" height="146" alt="Laptop">
        <h2 class="feature-name">Web sites</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. In facilis amet. </p>
      </div>
      <div class="feature-item">
        <img src="#" width="300" height="146" alt="Phone">
        <b class="feature-name">Apps</b>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
      </div>
      <div class="feature-item">
        <img src="#" width="300" height="146" alt="Presentation">
        <b class="feature-name">Presentations</b>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Doloremque asperiores at quos quasi nobis,
        beatae vel iure quae.</p>
      </div>
    </section>
  </main>
</body>
</html>
```

Рисунок 2.1 – Базова структура HTML

У стилях і дизайні CSS використовується для керування зовнішнім виглядом елементів (рис. 2.2). Використання SCSS, препроцесора для CSS, робить стиль чистішим і простіше підтримувати (рис. 2.3). Використання вкладених змінних і селекторів у SCSS полегшує розробку та ефективну підтримку стилів.

```
body {
    width: 100%;
    height: 100%;
    font-family: 'Open Sans', sans-serif;
    font-weight: 300;
    color: #666;
    background-color: #ddd;
    font-size: 16px;
    line-height: 1.6em;
}

html {
    width: 100%;
    height: 100%;
}

label,
```

Рисунок 2.2 – базова структура CSS

SCSS

```
section {
    height: 100px;
    width: 100px;

    .class-one {
        height: 50px;
        width: 50px;

        .button {
            color: #074e68;
        }
    }
}
```

CSS

```
section {
    height: 100px;
    width: 100px;
}

section .class-one {
    height: 50px;
    width: 50px;
}

section .class-one .butto
    color: #074e68;
}
```

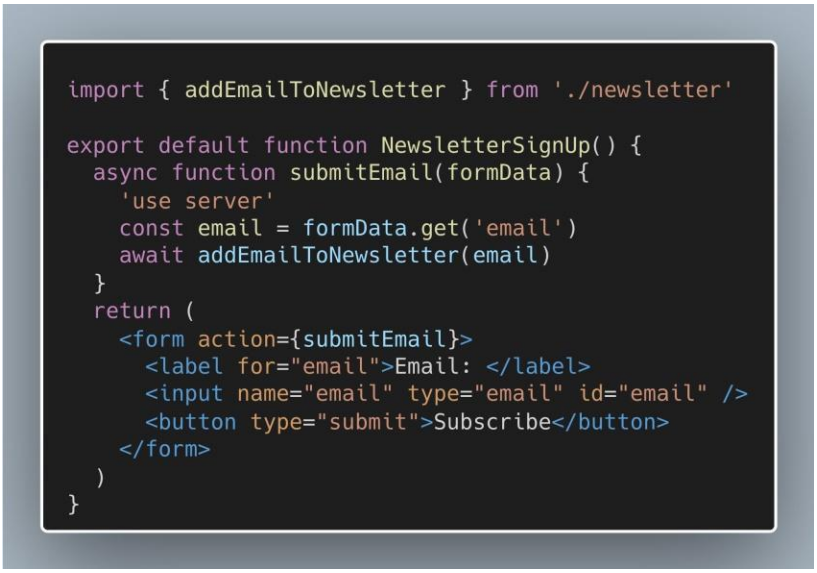
Рисунок 2.3 – Порівняння CSS і SCSS

JavaScript відіграє ключову роль у створенні інтерактивності веб-сторінок. Використання JavaScript дозволяє взаємодіяти з користувачем без необхідності перезавантажувати сторінку (рис. 2.4), створюючи більш зручний і динамічний інтерфейс.

```
const LOCALE = globalThis.navigator.language

const div = document.body.appendChild(document.createElement('div'))
const list = div.appendChild(document.createElement('ol'))

const dayNames = new Map()

for (let i = 0; i < 7; ++i) {
    const d = Temporal.PlainDate.from({
        year: Temporal.Now.plainDateISO().year,
        month: 1,
        day: i + 1,
    })

    dayNames.set(d.dayOfWeek, d.toLocaleString(LOCALE, { weekday: 'long' }))
}

for (const num of [ ...dayNames.keys()].sort((a, b) => a - b)) {
    list.appendChild(Object.assign(
        document.createElement('li'),
        { textContent: dayNames.get(num) },
    ))
}
```

Рисунок 2.4 – Приклад інструкцій JS

Однією з ключових технологій у моєму стеку є React. Я використовую цю бібліотеку для створення інтерфейсів користувача та розділення програми на компоненти (рис. 2.5). Це спрощує розробку коду, тестування та підтримку та дозволяє швидше вводити нові функції. Я використовую React Router Dom для забезпечення навігації. Ця бібліотека полегшує реалізацію маршрутизації та навігації між різними частинами програми (рис. 2.6), забезпечуючи при цьому швидкість і ефективність взаємодії.



```
import { addEmailToNewsletter } from './newsletter'

export default function NewsletterSignUp() {
  async function submitEmail(formData) {
    'use server'
    const email = formData.get('email')
    await addEmailToNewsletter(email)
  }
  return (
    <form action={submitEmail}>
      <label for="email">Email: </label>
      <input name="email" type="email" id="email" />
      <button type="submit">Subscribe</button>
    </form>
  )
}
```

Рисунок 2.5 – Компонент React

```
import React from 'react';
import {
  BrowserRouter,
  Routes,
  Route,
} from 'react-router-dom';
import Page1 from './pages/page1.js';
import Page2 from './pages/page2.js';
import Page3 from './pages/page3.js';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route index element={<Page1 />}/>
        <Route path="page2" element={<Page2 />}/>
        <Route path="page3/:id" element={<Page3 />}/>
      </Routes>
    </BrowserRouter>
  );
}
```

Рисунок 2.6 – Маршрутизація за допомогою React Router Dom

На стороні сервера я використовую Node.js, платформу для розробки серверних програм (рис. 2.7) за допомогою JavaScript. Express.js допомагає створити потужний і ефективний сервер, обробляючи запити HTTP та спілкуючись із базою даних.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Рисунок 2.7 – Приклад створення сервера на Node.js

MongoDB діє як база даних у моєму стеку. Ця документно-орієнтована база даних використовує документи, подібні до JSON (рис. 2.8), для зберігання даних. Його гнучкість і простота роботи з неструктурованими даними роблять його чудовим вибором для проектів зі змінною структурою даних.

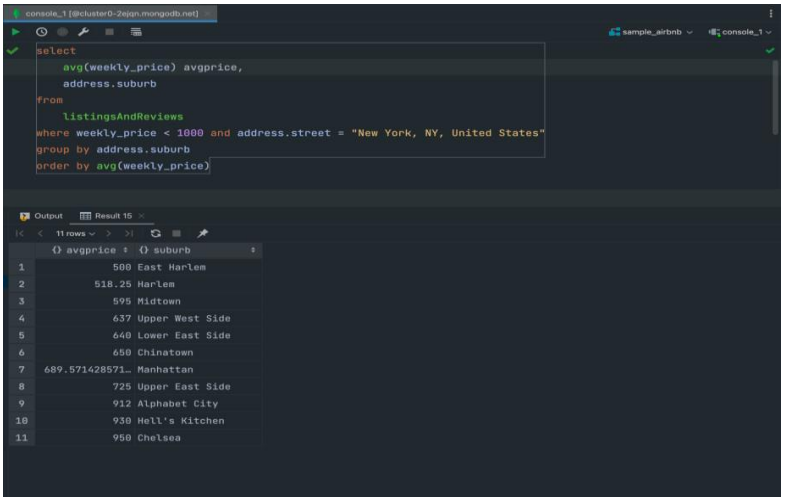


Рисунок 2.8 – Приклад вибірки даних з бази даних MongoDB

Загалом використання цього технологічного ландшафту дозволяє мені розробляти потужні та сучасні веб-програми, зберігаючи високу якість і зручність використання.

2.2 Обгрунтування вибору конкретних технологій для проекту

1. React:

1.1. Основні переваги:

- компонентна архітектура - React дозволяє створювати великі програми, розбиваючи їх на невеликі багаторазові компоненти. Це спрощує розробку коду, тестування та обслуговування;
- віртуальний DOM - ефективне використання віртуального DOM допомагає зменшити кількість маніпуляцій із реальним DOM, що призводить до покращення продуктивності програми;
- розширюваність - велика спільнота та наявність численних сторонніх бібліотек і плагінів роблять React потужним інструментом для проектів будь-якого розміру.

1.2. Порівняння з конкурентами:

- Vue.js - Має простий синтаксис і хорошу документацію, але менш екосистемний, ніж React;
- Angular - Надає повну структуру, але може бути складною для невеликих проектів. React пропонує більше гнучкості.

2. Node.js:

2.1. Основні переваги:

- асинхронний і керований подіями - Node.js використовує асинхронний підхід, що дозволяє йому ефективно обробляти багато одночасних запитів;
- спільна мова (JavaScript) - використання JavaScript як на стороні клієнта, так і на стороні сервера дозволяє ділитися кодом і бібліотеками,

спрощуючи розробку;

- велика кількість модулів - node.js пропонує великий вибір модулів у системі Node Package Manager (NPM), що дозволяє легко інтегрувати додаткові функції.

2.2. Порівняння з конкурентами:

- Django (Python) - Django підходить для швидкої розробки, але Node.js краще для програмування в режимі реального часу та асинхронного програмування;
- Ruby on Rails (Ruby) - Rails є потужним, але Node.js може бути більш ефективним для великої кількості одночасних підключень.

3. MongoDB:

3.1. Основні переваги:

- гнучка схема даних - MongoDB дозволяє зберігати дані як JSON-подібні документи, що забезпечує велику гнучкість і зручність при роботі з неструктурованими даними;
- горизонтальне масштабування - легко масштабувати горизонтально, що робить його чудовим вибором для проектів із різними або великими обсягами даних;
- широкий діапазон операційних систем - MongoDB може працювати на різних платформах і тому є універсальним.

3.2. Порівняння з конкурентами:

- MySQL і PostgreSQL - корисні для реляційних даних, але менш гнучкі при зміні схеми;
- CouchDB - також використовує JSON-подібні документи, але менш популярний і поширений, ніж MongoDB.

Обрані технології (React, Node.js, MongoDB) виправдані своїми основними перевагами, такими як ефективність, гнучкість і розширюваність.

Порівняння з конкурентами свідчить про те, що ці технології відповідають конкретним вимогам проекту.

2.3 Визначення технічних вимог до програмного забезпечення

Технічні вимоги для React:

- використання останньої стабільної версії React для забезпечення найновіших функцій та оптимізацій;
- система компонентів повинна бути організована і структурована відповідно до найкращих практик React;
- дотримання стандартів коду: Використання eslint та prettier для забезпечення дотримання стандартів коду та однорідності.

Технічні вимоги для Node.js та Express.js:

- використання актуальної версії Node.js для забезпечення стабільності та безпеки;
- визначення middleware для обробки різних етапів запиту та відповіді;
- впровадження заходів безпеки, таких як захист від атак Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF).

Технічні вимоги для MongoDB:

- використання останньої стабільної версії MongoDB для отримання оновлень безпеки та нових можливостей;
- розроблення гнучкої схеми даних, щоб враховувати змінність у вигляді та структурі даних;
- щоденне автоматизоване резервне копіювання бази даних для запобігання втраті даних.

Вимоги до Інфраструктури:

- використання хмарних платформ (наприклад, AWS, Google Cloud або Azure) для гнучкості та масштабованості;
- налаштування автоматичного масштабування для ефективного використання ресурсів;
- використання SSL-сертифікатів для захисту передачі даних між

клієнтом і сервером;

- налаштування Firewall для блокування непотрібного мережевого трафіку;
- використання інструментів, таких як Prometheus або New Relic, для моніторингу стану додатка та інфраструктури;
- налаштування системи логування (наприклад, ELK стек) для відстеження та аналізу подій у системі.

Визначені технічні вимоги до програмного забезпечення та інфраструктури спрямовані на забезпечення стабільності, ефективності та безпеки веб-програми. Використання останніх версій React, Node.js, та MongoDB дозволяє скористатися оновленнями безпеки та новими можливостями. Система компонентів React та middleware для Node.js та Express.js організовані відповідно до найкращих практик, що сприяє читабельності та підтримці кодової однорідності.

Застосування заходів безпеки, таких як захист від атак XSS та CSRF, свідчить про серйозний підхід до забезпечення безпеки веб-програми. Щоденне автоматизоване резервне копіювання бази даних MongoDB спрямоване на уникнення можливих втрат даних.

Щодо інфраструктури, використання хмарних платформ, налаштування автоматичного масштабування та застосування SSL-сертифікатів гарантують гнучкість, ефективність та безпеку передачі даних. Firewall та система логування спрямовані на контроль мережевого трафіку та відстеження та аналіз подій у системі.

Ці технічні вимоги становлять необхідну основу для забезпечення надійності та ефективності веб-програми. Використання eslint та prettier для дотримання стандартів коду та єдність у розробці. Активне використання middleware для обробки етапів запиту та відповіді, а також заходи безпеки, свідчать про високий рівень обережності щодо можливих загроз безпеці.

Щоденне автоматизоване резервне копіювання бази даних MongoDB є

критичним елементом стратегії відновлення, дозволяючи уникнути можливих втрат даних та швидко відновити їх у випадку непередбачуваних подій.

Відданість забезпеченню безпеки також виявляється в застосуванні заходів захисту на рівні інфраструктури, таких як використання хмарних платформ, встановлення SSL-сертифікатів та використання Firewall. Моніторинг стану додатка та інфраструктури здійснюється через інструменти, такі як Prometheus або New Relic, для забезпечення негайного виявлення та вирішення проблем.

У цілому, використання сучасних технологій та строге дотримання технічних вимог є ключовими чинниками успішної інтеграції та функціонування веб-програми в динамічному та конкурентному середовищі.

2.4 Огляд заходів забезпечення безпеки використаних технологій

У рамках проекту безпека стала важливим аспектом, і кожна використовувана технологія має свої унікальні заходи для забезпечення стабільності та захисту. Це означає, що я вживаю різні заходи безпеки для кожної частини додатку.

Почнемо з React. Хоча ця бібліотека вже включає вбудовані заходи безпеки, я активно використовую додаткові заходи для максимального запобігання можливим загрозам. Для виявлення та вирішення можливих уразливостей я використовую бібліотеку OWASP React Security. Це дозволяє мені реагувати та виправляти потенційні проблеми безпеки в реальному часі. Окрім цього, я приділяю особливу увагу валідації введених даних, щоб запобігти атакам Cross-Site Scripting (рис. **2**.9), що можуть виникнути через неправильну обробку користувацьких даних. Такий підхід сприяє забезпеченню високого рівня безпеки програмного забезпечення і захисту від потенційних кібератак.

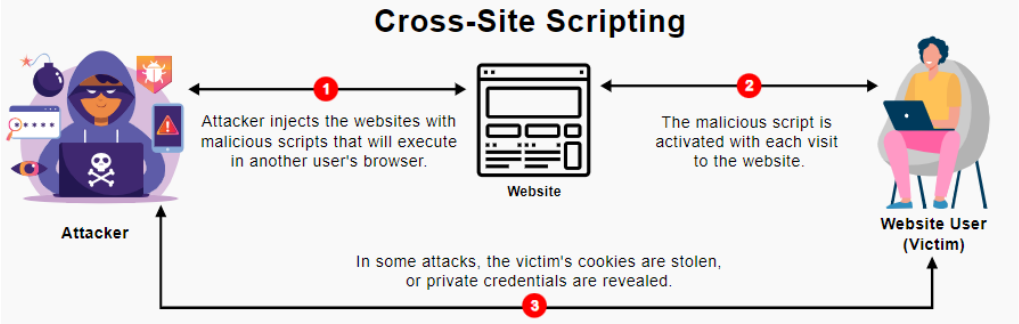


Рисунок 2.9 – Сценарій роботи Cross-Site Scripting

Переходячи до серверної частини, яка базується на Node.js та Express.js, я розумію важливість безпеки для стабільності та надійності. Тут я використовую різноманітні middleware безпеки, включаючи Helmet (рис. 2.10), який автоматизовано встановлює заголовки безпеки та надає додатковий захист від різних атак. Особливу увагу я приділяю валідації введених даних на сервері, щоб мінімізувати ризики атак, пов'язаних з введенням даних, і забезпечити цілісність серверного середовища.

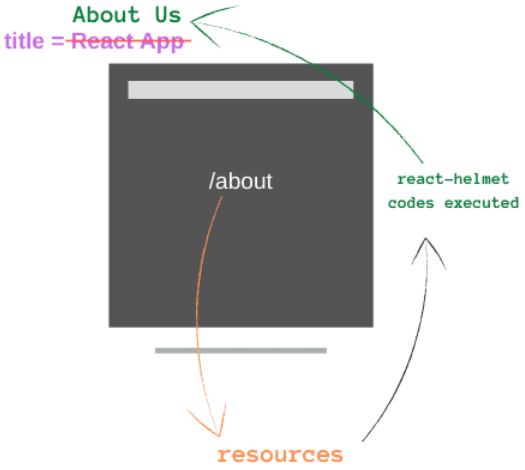


Рисунок 2.10 – Сценарій роботи Cross-Site Scripting

Щодо бази даних MongoDB, я працюю над системою аутентифікації та авторизації. Це включає в себе створення обмежень доступу до бази даних,

забезпечення авторизованого доступу лише для визначених користувачів, виключаючи несанкціонований доступ. Додатково, я використовую шифрування для обраних даних, що дозволяє зберігати конфіденційні дані в зашифрованому вигляді, запобігаючи можливості незаконного доступу.

Загальна ідея полягає в тому, щоб системно захищати кожен аспект проекту. Я розумію, що безпека - це процес, а не лише результат, і я продовжую вдосконалювати заходи безпеки для адаптації до зростаючих загроз. Це включає в себе як внутрішні заходи, так і заходи на рівні комунікацій та зберігання даних. Завдяки цим заходам я не лише реагую на потенційні загрози, а й активно запобігаю їм, що є ключовим для забезпечення довіри та надійності проекту.

Висновок до розділу 2

У розділі 2 був проведений глибокий аналіз веб-платформ та використаних технологій. Початково вивчений технологічний ландшафт, надавши загальний огляд важливих технологій, що визначають сучасне веб-розроблення. Далі, обґрунтовано вибір конкретних технологій, які будуть використовуватись у рамках розробки проекту, враховуючи їхню ефективність та сучасні стандарти.

Визначені технічні вимоги до програмного забезпечення, які включають не лише функціональні аспекти, але й вимоги та масштабованості. Особлива увага приділена огляду заходів забезпечення безпеки використовуваних технологій, щоб гарантувати захищеність користувачів та даних.

Цей розділ становить важливу основу для подальшого розвитку проекту, визначаючи технічні та технологічні аспекти.

РОЗДІЛ 3. РОЗРОБКА СТРУКТУРИ ПРОЕКТУ, ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ВЕБ-ПРОГРАМИ

3.1 Огляд загальної архітектури проекту

Діаграма компонентів ілюструє модульну структуру проекту, де висвітлені ключові компоненти та їх взаємодії (рис. 3.1). Система розділена на фронтенд, бекенд, базу даних та різноманітні сервіси:

- веб-інтерфейс представляє веб-інтерфейс для взаємодії користувачів з системою;
- мобільний інтерфейс визначає мобільний інтерфейс додатка для покращеної доступності.
- система автентифікації керує процесом автентифікації користувачів, забезпечуючи безпечний доступ до системи;
- панель управління забезпечує централізовану панель керування для користувачів для моніторингу та управління діяльністю;
- сервер діє як центральний обчислювальний блок, обробляючи запити та керуючи взаємодією між компонентами;
- бізнес-логіка охоплює основні функціональності та правила, що регулюють операції системи;
- сервіс автентифікації спеціалізований сервіс, відповідальний за обробку автентифікації користувачів;
- сервіс обробки відповідає за обробку та управління даними в системі;
- MongoDB використовується як система управління базами даних для зберігання та отримання інформації;
- роутер забезпечує маршрутизацію запитів та перехоплення URL;
- API сервіс надає API для взаємодії з іншими системами;

- сервіс сповіщень відповідає за надсилання сповіщень користувачам.

Огляд загальної архітектури проекту (рис. 3.1) визначає основні компоненти та їх взаємодії. Система має чітко визначені фронтенд та бекенд, які взаємодіють через сервер. База даних MongoDB використовується для зберігання інформації. Сервіси, такі як роутер, API сервіс і сервіс сповіщень, доповнюють функціональність системи та покращують її ефективність.

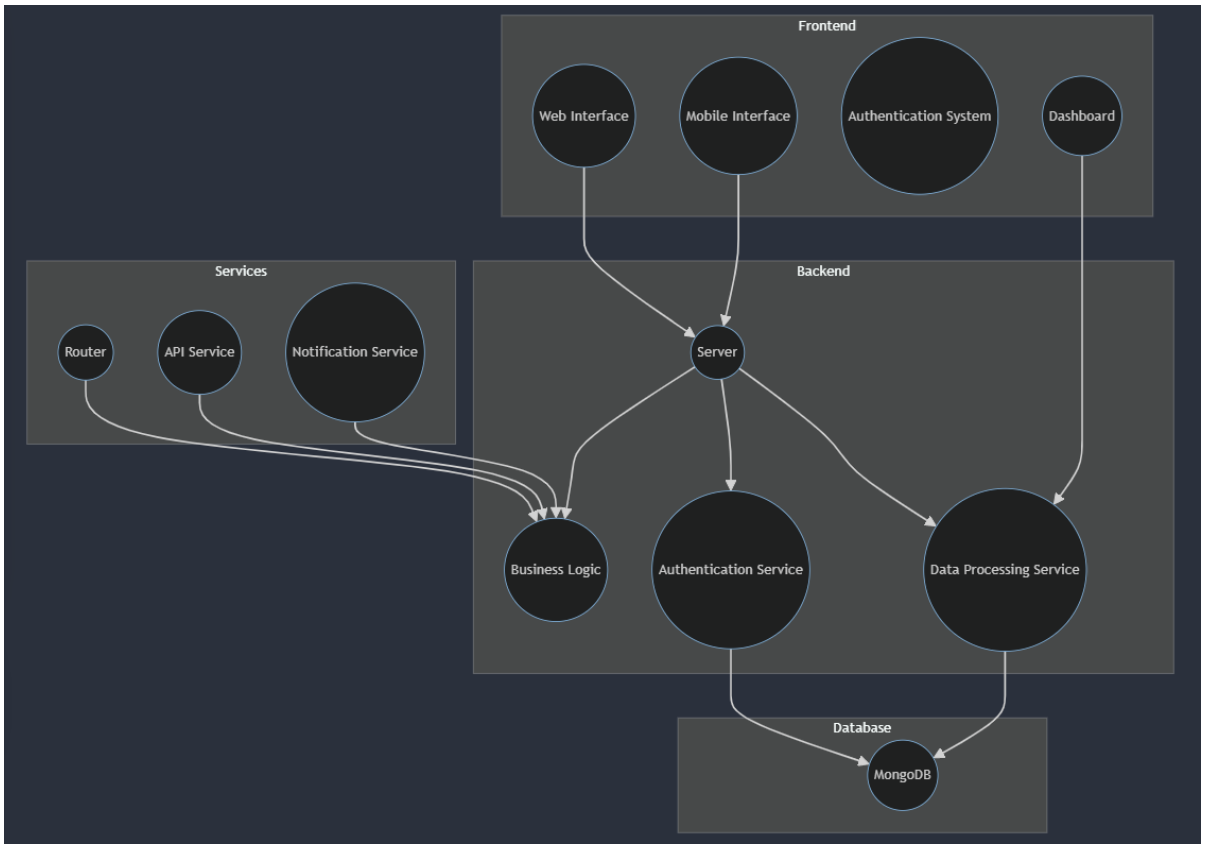


Рисунок 3.1 – Діаграма компонентів архітектури проекту

3.2 Опис розділення проекту на модулі та їх функціональність

Розділ 1. Модуль користувацького інтерфейсу (UI).

Відповідальність: забезпечення зручного та інтуїтивно зрозумілого взаємодії з системою.

1.1. Компоненти:

- веб-інтерфейс - користувацький інтерфейс для веб-браузерів;
- мобільний інтерфейс - інтерфейс, оптимізований для мобільних пристроїв;
- система автентифікації - управління входом, виходом та контролем доступу;
- панель управління - централізована панель для користувачів.

Розділ 2. Модуль бекенду.

Відповідальність: управління логікою на сервері та обробкою даних.

2.1. Компоненти:

- сервер - основний сервер, що обробляє запити від фронтенду;
- бізнес-логіка - реалізація основної логіки та функціональностей;
- сервіс автентифікації - обробка аутентифікації користувачів та контроль доступу. ;
- сервіс обробки даних - управління обробкою та збереженням даних.

Розділ 3. Модуль бази даних.

Відповідальність: обробка зберігання та отримання даних.

3.1. Компоненти:

- MongoDB - NoSQL база даних для ефективного зберігання та отримання даних.

Розділ 4. Модуль сервісів.

Відповідальність: забезпечення додаткових функцій та зовнішніх взаємодій.

4.1. Компоненти:

- роутер - управління маршрутизацією запитів у системі;
- API сервіс - надання API для спілкування між фронтендом та бекендом;
- сервіс сповіщень - обробка доставки сповіщень користувачам.

Розділ 5. Модуль безпеки.

Відповідальність: забезпечення безпеки та цілісності системи.

5.1. Компоненти:

- безпека веб-інтерфейсу - реалізація заходів безпеки для веб-інтерфейсу;
- шифрування даних - забезпечення шифрування чутливих даних;
- контроль доступу - управління доступом до ресурсів системи.

Розділ 6. Модуль тестування та забезпечення якості.

Відповідальність: забезпечення надійності та коректності системи.

6.1. Компоненти:

- юніт-тестування - тести для окремих компонентів на коректність;
- інтеграційне тестування - забезпечення взаємодії між модулями;
- забезпечення якості - контроль загальної якості та продуктивності системи.

Розділ 7. Модуль інновацій та иайбутнього розвитку.

Відповідальність: фокус на дослідженні та впровадженні нових технологій.

7.1. Компоненти:

- дослідження інновацій - вивчення нових технологій та тенденцій;
- розробка функціоналу - впровадження нового функціоналу на основі досліджень;
- планування масштабованості - готовність системи до майбутнього росту та розширення.

Модульна структура розподілу відповідальностей є критичним аспектом успішної розробки проекту, спрямованого на покращення та автоматизацію процесів освітнього наставництва. Цей підхід сприяє високому рівню систематизації та ефективному управлінню, роблячи проект доступним для обслуговування та легким для розширення. Кожен модуль володіє чітко визначеними функціональностями, що сприяє не лише покращенню організації, а й розвитку системи у гнучкий та майбутньоорієнтований спосіб.

Така архітектура дозволяє легко впроваджувати нові функції та технології, забезпечуючи високу надійність та швидкість реакції на зміни у вимогах користувачів та ринку. Загалом, ця модульна структура є основою для стабільної та інноваційної розробки програмного продукту, спрямованого на підвищення якості освітнього наставництва.

3.3 Опис структури баз даних та методів зберігання і обробки даних

В базі даних для системи репетиторства реалізовано структуру для зберігання і обробки різноманітної інформації (рис. 3.2), пов'язаної з учнями, викладачами, уроками та іншими аспектами навчального процесу:

1. Студенти та викладачі - для представлення даних про студентів та викладачів використовуються відповідні схеми, які включають особисті дані (ім'я, електронна пошта, телефон), дані про предмети (для викладачів) та інші атрибути.

2. Предмети та уроки – предмети відображаються в окремій схемі, де для кожного предмету зберігаються назва та індекс. Уроки пов'язані зі студентами та викладачами, містять дані про дату, тему, оплату, тривалість, матеріали для уроку та інші деталі.

3. Дані та логіка уроків – для зберігання текстової інформації про уроки, яка може змінюватися з часом, використовується структура LessonData, яка включає дані про дату, значення та флаг змін. Це дозволяє зберігати історію змін для подальшого аналізу та відновлення даних.

4. Матеріали та файли - для зберігання різноманітних матеріалів, таких як текстові описи, посилання та файли, використовуються схеми Text, Links та Files. Кожна має атрибути, що вказують на деталі матеріалу та зміни в ньому. Зберігати файли дуже важливо для викладача, адже це його нотатки, та збереження матеріалів. Зберігати файли дуже важливо для викладача, адже це його нотатки, та збереження матеріалів. Забезпечення зручного доступу до цих

матеріалів сприяє ефективній роботі та навчанню в освітньому середовищі.

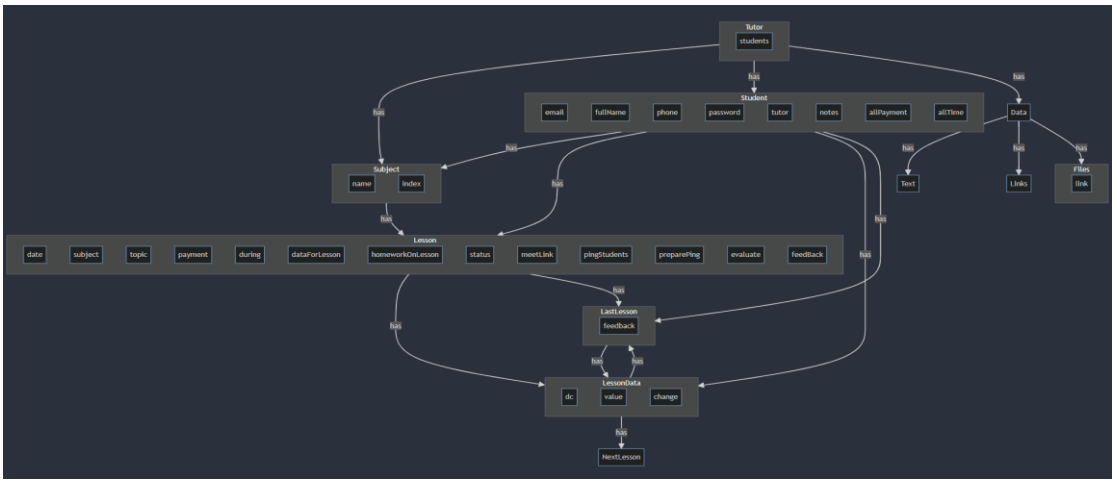


Рисунок 3.2 – Діаграма зв’язків таблиць в базі даних MongoDB

Структура бази даних розробленою для системи репетиторства спрямована на забезпечення ефективного, організованого та надійного зберігання інформації, пов'язаної з навчальним процесом. Використання різних схем та зв'язків дає можливість легко взаємодіяти з даними та враховувати їхні зміни, надаючи системі гнучкість та адаптованість до вимог користувачів.

Методи обробки даних включають в себе стандартні запити до бази даних, що дозволяє отримувати, оновлювати та видаляти інформацію. Використання індексів в базі даних сприяє підвищенню швидкодії операцій, забезпечуючи ефективність обробки великих обсягів даних.

Оглядаючи загальний контекст, структура бази даних створена з метою забезпечення оптимального функціонування системи та надання необхідної інформації для всіх учасників навчального процесу. Ця структура є ключовим елементом системи, який допомагає управляти та підтримувати ефективність у навчальному середовищі, роблячи процес взаємодії учнів та викладачів більш зручним та продуктивним.

База даних розроблена таким чином, щоб враховувати різні аспекти навчального процесу. Деталі профілю кожного студента та викладача, включаючи особисті дані, предмети, які вони викладають або вивчають, а

також зазначення тем занять, використовуються для створення повноцінного зображення участників системи.

Схема бази даних включає також інформацію про найближчі та минулі заняття, вартість та тривалість кожного з них, що надає можливість ефективно відстежувати та оцінювати навчальний процес. Важливою частиною є також збереження взаємодії між викладачами та учнями, включаючи відгуки, оцінки та робочі матеріали.

Іншим важливим аспектом є система автентифікації та безпеки, яка забезпечує захист конфіденційної інформації та обмеження доступу до важливих функцій. Це робить систему надійною та відповідною сучасним стандартам безпеки даних.

Основна ідея бази даних полягає в тому, щоб створити систему, яка легко масштабується та адаптується до змін у навчальному середовищі. Кожен об'єкт, від студента до викладача та заняття, має свою унікальну ідентифікацію та пов'язані дані, що робить можливим визначення зв'язків та взаємодії між ними.

Методи зберігання та обробки даних використовуються для ефективного використання ресурсів та забезпечення швидкого доступу до інформації. Застосування індексів дозволяє оптимізувати швидкість виконання запитів, а структура бази даних спроектована так, щоб зменшити дублювання даних та уникнути зайвої складності.

Додатково, впровадження механізмів резервного копіювання та відновлення даних гарантує стабільність та надійність бази даних у випадку непередбачуваних ситуацій. Забезпечення консистентності та цілісності даних є ключовим аспектом управління інформацією в системі.

3.4 Інтерфейс користувач. Розгляд інтерфейсу та його функціональність

В даному розділі детально розглядається інтерфейс користувача

програмного продукту, щоб надати повний огляд його функціональності та забезпечити зручність взаємодії для кінцевого користувача.

Предмети. На сторінці "Предмети" користувач може переглядати список доступних предметів, а також отримувати детальну інформацію про кожен предмет. Можливості включають вибір предметів для навчання та перегляд розкладу.

Календар занять. Сторінка "Календар занять" дозволяє користувачам переглядати та редагувати розклад у зручному календарному вигляді. Можливості включають визначення дат та часу занять, додавання нових подій та налаштування нагадувань.

Звіти. На сторінці "Звіти" користувач може переглядати статистику та звіти про свій академічний прогрес. Функціонал включає фільтрацію за періодами, графічне відображення даних та можливість завантаження звітів у різних форматах.

Лендінг. Лендінг-сторінка призначена для представлення ключових переваг та функцій програмного продукту. Тут користувач може долучитися до системи, отримати інформацію та ознайомитися з можливостями платформи.

Реєстрація/Логін. Сторінка "Реєстрація/Логін" дозволяє користувачам створити обліковий запис або увійти в систему. Забезпечується зручний та безпечний доступ до особистого кабінету.

Налаштування аккаунта. На цій сторінці користувач може настроювати особисті налаштування свого облікового запису, змінювати пароль, обирати мову інтерфейсу та налаштовувати оповіщення.

Інформація про аккаунт. Сторінка "Інформація про аккаунт" містить загальну інформацію про користувача, включаючи основні персональні дані та академічний прогрес.

Добавлення студента. Ця функціональність дозволяє викладачам або адміністраторам додавати нових студентів до системи, вказуючи їхні основні дані та параметри.

Цей розділ визначає ключові аспекти взаємодії користувача з системою, забезпечуючи зручний та ефективний інтерфейс для всіх учасників навчального процесу. За допомогою різноманітних сторінок, таких як "Предмети", "Календар занять", "Звіти" та інші, користувач може з легкістю організувати свою академічну діяльність, моніторити прогрес та взаємодіяти з платформою.

Лендінг-сторінка виступає важливим елементом для представлення основних можливостей платформи та привертання нових користувачів. Створення облікового запису та його подальше управління реалізовані на сторінці "Реєстрація/Логін" та "Налаштування аккаунта".

Завдяки можливості додавання нових студентів, система стає більш гнучкою та підходить для різних користувачів, включаючи викладачів та адміністраторів. Орієнтація на зручність та інтуїтивність інтерфейсу робить взаємодію з платформою приємною та продуктивною для всіх учасників освітнього процесу.

Розгляд інтерфейсу та його функціональності в розділі є критичним аспектом проекту з декількох причин:

- інтерфейс повинен бути привабливим та естетично приємним, забезпечуючи користувачам комфортну взаємодію;
- опис функціональності дозволяє визначити, наскільки легко користувачі можуть здійснювати необхідні операції та отримувати доступ до важливих ресурсів;
- розгляд функціональності визначає, чи задовольняють можливості системи потреби користувачів, враховуючи всі важливі аспекти їхнього взаємодії з платформою;
- розуміння інтерфейсу та його можливостей дозволяє оптимізувати взаємодію користувачів з системою, підвищуючи продуктивність та забезпечуючи задоволення від використання;
- опис різних сторінок та їхньої функціональності сприяє адаптації платформи до різних потреб користувачів, таких як студенти, викладачі,

адміністратори.

Усі ці аспекти спільно спрямовані на створення інтерфейсу, який не лише задовольняє технічні вимоги, але й стає дружнім та привабливим для кінцевих користувачів.

3.5 Опис методів управління розробкою проекту та ресурсами

Управління розробкою проекту та використанням ресурсів є важливою складовою ефективності та успіху будь-якої IT-ініціативи. Для досягнення цілей та максимізації результативності, використовуються різноманітні методи та стратегії.

Методології розробки - обрання методології розробки (наприклад, Scrum, Kanban, або Waterfall) визначає порядок виконання завдань, керування змінами та розподіл обов'язків у команді. Застосування гнучких методів може полегшити адаптацію до змін у вимогах та швидше впровадження нового функціоналу.

Етапи розробки – системне розбиття проекту на етапи дозволяє краще керувати процесом та планувати ресурси. Визначення кроків від ідеї до впровадження, а також оцінка часових рамок, є ключовим елементом успішної розробки.

Управління командою - ефективне управління роботою команди полягає в розподілі завдань, визначенні ролей та підтримці співпраці. Регулярні комунікації, зустрічі та ретроспективи сприяють вирішенню проблем та вдосконаленню процесів.

Управління витратами - слід постійно моніторити та оцінювати витрати, оптимізувати розподіл бюджету для максимальної ефективності. Контроль над фінансами дозволяє уникнути перевитрат та забезпечити стабільність проекту.

Використання інструментів управління проектами - використання спеціальних інструментів для управління задачами, контролю версій, спільної

роботи (наприклад, Jira, Trello, Git) допомагає автоматизувати та полегшити керування проектом.

Залучення та розвиток персоналу - забезпечення навчання та розвитку команди, впровадження ефективних практик роботи, а також створення сприятливого робочого середовища допомагає залучити та утримати висококваліфікованих фахівців.

Ризик-менеджмент - аналіз та управління ризиками є невід'ємною частиною процесу. Своєчасне виявлення та зменшення впливу можливих проблем дозволяючи уникнути серйозних перешкод та забезпечити стабільний розвиток проекту.

Звітність і комунікації - регулярні звітності про стан проекту, прогрес та досягнення допомагають забезпечити прозорість та взаєморозуміння між учасниками. Ефективна комунікація виключає непорозуміння та сприяє гармонійному співробітництву.

Фази тестування - додавання етапів тестування на кожному етапі розробки допомагає виявити та виправити помилки ще до введення нового функціоналу або релізу. Це забезпечує якість та стабільність продукту.

Оптимізація та масштабованість - постійне вдосконалення та оптимізація процесів розробки дозволяє підтримувати високу продуктивність. Планування масштабованості передбачає готовність системи до зростання обсягів даних та користувачів.

Управління змінами - гнучкість до змін у вимогах чи стратегії є необхідною умовою в сучасному розробництві. Ефективне управління змінами дозволяє швидко адаптуватися до нових умов та вдосконалювати продукт.

Використання кращих практик та стандартів - дотримання визнаних стандартів та кращих практик у галузі розробки забезпечує високу якість програмного продукту та дозволяє ефективно взаємодіяти з іншими системами.

Однією з найважливіших складових успіху є структурованість та організація. Ретельне визначення завдань та відповідальностей, чітке розподіл

ресурсів та етапів розробки, а також систематичний моніторинг процесів — це ключові елементи вдалих проектів.

Також, зосереджуючись на постійному навчанні та використанні інновацій, команда може забезпечити актуальність та конкурентоспроможність продукту на ринку. Ретельна робота над інтерфейсом користувача та забезпеченням його зручності розширює коло користувачів та поліпшує враження від використання продукту.

Найважливішим аспектом є, звісно ж, фокус на користувача. Задоволення їхніх потреб та створення продукту, який справляється із завданнями та надає значущі переваги, є ключовим для успіху будь-якого проекту.

Загалом, успіх управління розробкою проекту полягає в гармонійному поєднанні технічних аспектів, стратегічного планування та врахування потреб користувачів. Це вимагає від команди не лише технічної експертизи, але й здатності працювати разом для досягнення спільних цілей.

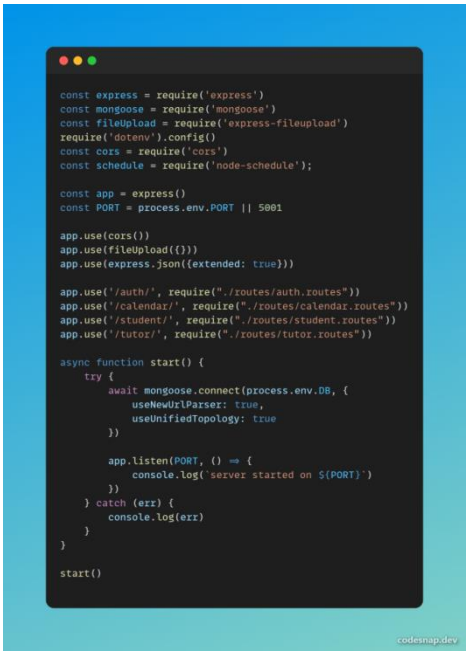
3.6 Розробка модулів серверної частини та їх функціональності

На цьому етапі розробки фокус зосереджений на детальній розробці окремих модулів, які складають основу програмного продукту. Кожен модуль має свою унікальну функціональність та завдання, спрямоване на вирішення конкретної задачі чи надання певної можливості користувачам.

Важливо визначити чіткі межі та взаємодії між модулями для ефективного інтегрування їх у єдиний функціональний блок. Крім того, під час розробки модулів враховується можливість майбутнього розширення та вдосконалення системи.

Кожен модуль проходить етапи від концепції та дизайну до реалізації та тестування, забезпечуючи високу якість та функціональність кожної частини програмного забезпечення. Такий підхід дозволяє ефективно керувати розробкою та впевнено впроваджувати новий функціонал у систему.

Отже, почнемо з вихідної точки сервера файлу index.js (рис. 3.3).



```
const express = require('express')
const mongoose = require('mongoose')
const fileUpload = require('express-fileupload')
require('dotenv').config()
const cors = require('cors')
const schedule = require('node-schedule');

const app = express()
const PORT = process.env.PORT || 5001

app.use(cors())
app.use(fileUpload({}))
app.use(express.json({extended: true}))

app.use('/auth/', require("./routes/auth.routes"))
app.use('/calendar/', require("./routes/calendar.routes"))
app.use('/student/', require("./routes/student.routes"))
app.use('/tutor/', require("./routes/tutor.routes"))

async function start() {
    try {
        await mongoose.connect(process.env.DB, {
            useNewUrlParser: true,
            useUnifiedTopology: true
        })

        app.listen(PORT, () => {
            console.log(`server started on ${PORT}`)
        })
    } catch (err) {
        console.log(err)
    }
}

start()
```

Рисунок 3.3 – Код для реалізації сервера в файлі index.js

Цей код представляє собою серверну частину веб-додатка, побудованого на платформі Node.js з використанням фреймворка Express та бази даних MongoDB. Починаючи з підключення необхідних залежностей, таких як Express для створення веб-додатків та mongoose для взаємодії з базою даних MongoDB, код також використовує різні бібліотеки, такі як express-fileupload для обробки файлових завантажень, dotenv для завантаження змінних середовища, cors для обробки CORS (Cross-Origin Resource Sharing) та node-schedule для роботи з плануванням подій за розкладом.

Далі в коді ініціалізується об'єкт Express, і встановлюється порт для сервера (зазвичай 5001). Якщо змінна середовища PORT не визначена, використовується значення за замовчуванням.

Після цього встановлюються middleware для обробки CORS, файлових завантажень та розпізнавання JSON-запитів, що дозволяє серверу обробляти різноманітні типи запитів та обмінюватися ресурсами між веб-сторінками.

Наступний крок - визначення маршрутів для обробки запитів. Кожен

маршрут вказує на відповідний файл, де містяться обробники для конкретних маршрутів, такі як автентифікація, календар, студенти та викладачі.

Підключення до бази даних MongoDB відбувається через mongoose. Після успішного з'єднання веб-сервер стартує та готується приймати запити на визначеному порту.

Отже, цей код визначає необхідні налаштування для веб-додатка, обробляє маршрути та забезпечує взаємодію з базою даних, готуючи сервер до обслуговування запитів користувачів.

Наступним кроком, я визначив бібліотеки, які мені потрібні в розробці та встановив їх в додаток (рис. 3.4).

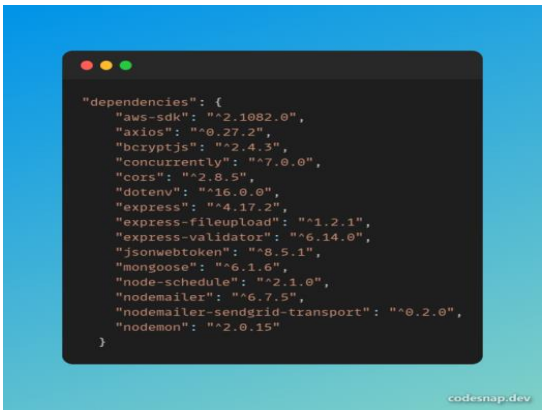


Рисунок 3.4 – Список залежностей в файлі package.json

Зараз, я трохи докладніше розповім про кожну з бібліотек:

1. aws-sdk - ця бібліотека надає SDK (набір інструментів для розробки) для взаємодії з Amazon Web Services (AWS). AWS SDK дозволяє вам взаємодіяти з різними сервісами AWS, такими як Amazon S3 для зберігання об'єктів, DynamoDB для бази даних NoSQL, і багатьма іншими;

2. axios - є бібліотекою для виконання HTTP-запитів. Вона дозволяє вам здійснювати асинхронні запити на сервери та обробляти їхні відповіді. Axios є популярним інструментом для взаємодії з API;

3. bcryptjs - використовується для хешування паролів. Вона забезпечує функції хешування паролів з використанням bcrypt, що є безпечним методом

для зберігання паролів, оскільки вони зберігаються у вигляді хешу, який важко відновити в оригінальний пароль;

4. concurrently - дозволяє вам одночасно запускати кілька команд Node.js. Це корисно, наприклад, коли ви хочете запустити як сервер, так і клієнт одночасно під час розробки;

5. cors (Cross-Origin Resource Sharing) - middleware для Express, яке дозволяє або блокує HTTP-запити на сервер з іншого домену. Використовується для розв'язання проблем заборони використання ресурсів з інших доменів в веб-браузерах;

6. dotenv - дозволяє завантажувати змінні середовища з файлу `.env` у вашому проекті. Це зручно для конфігурації параметрів середовища, таких як ключі API чи з'єднання з базою даних;

7. express - це веб-фреймворк для Node.js, який спрощує створення веб-додатків. Express надає широкий спектр можливостей для роботи з HTTP-запитами, розподіленням маршрутів, темплейтами та іншим;

8. express-fileupload - middleware для Express, яке спрощує обробку файлових завантажень. Воно дозволяє легко отримувати файли, які відправляються на сервер через HTTP-запити;

9. express-validator - це middleware для Express, яке допомагає валідувати та перевіряти дані, які приходять від користувачів через HTTP-запити;

10. jsonwebtoken - використовується для генерації та верифікації JSON Web Tokens (JWT). JWT використовується для автентифікації та передачі інформації між сторонами в безпечний спосіб;

11. mongoose - це ODM (Object Data Modeling) для MongoDB і Node.js. Він надає інтерфейс для взаємодії з базою даних MongoDB, забезпечуючи зручність роботи з об'єктами у вашому коді;

12. node-schedule - це бібліотека для планування подій в Node.js. Вона дозволяє вам запускати функції або виконувати код за розкладом, що може бути корисно для автоматизації завдань;

13. nodemailer - бібліотека для відправлення електронних листів з Node.js. Вона дозволяє вам надсилати електронні листи за допомогою різних транспортних методів, таких як SMTP;

14. nodemailer-sendgrid-transport - транспорт для Nodemailer, який дозволяє використовувати сервіс SendGrid для відправлення листів;

15. nodemon - це утиліта, яка дозволяє вам автоматично перезапускати сервер Node.js при зміні файлів у вашому проекті. Це великою мірою полегшує розробку, оскільки не потрібно вручну перезапускати сервер після змін.

У процесі розробки веб-додатка я використав 4 основних маршрути, які будуть відповідати за визначені функціональні частини системи. Кожен з цих маршрутів відображатиме конкретний аспект взаємодії та функціональності в рамках додатка:

1. auth.routes.js. В маршруті `auth.routes.js` реалізовані функціональності, пов'язані з автентифікацією та управлінням користувачами (рис. 3.5). Цей маршрут буде відповідальний за вхід в систему, відновлення паролю, а також управління ролями та правами доступу**;**

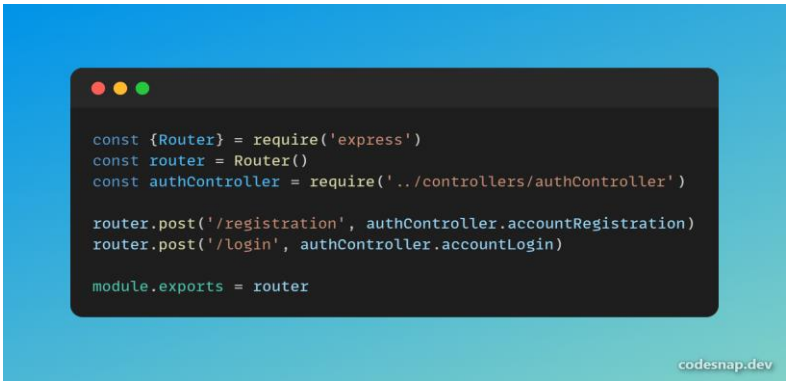


Рисунок 3.5 – Реалізація маршрутів для авторизації

2. calendar.routes.js. Маршрут `calendar.routes.js` відповідає за управління календарною частиною додатка. Це включає в себе додавання та редагування подій у календарі (рис. 3.6), отримання інформації про події**;**

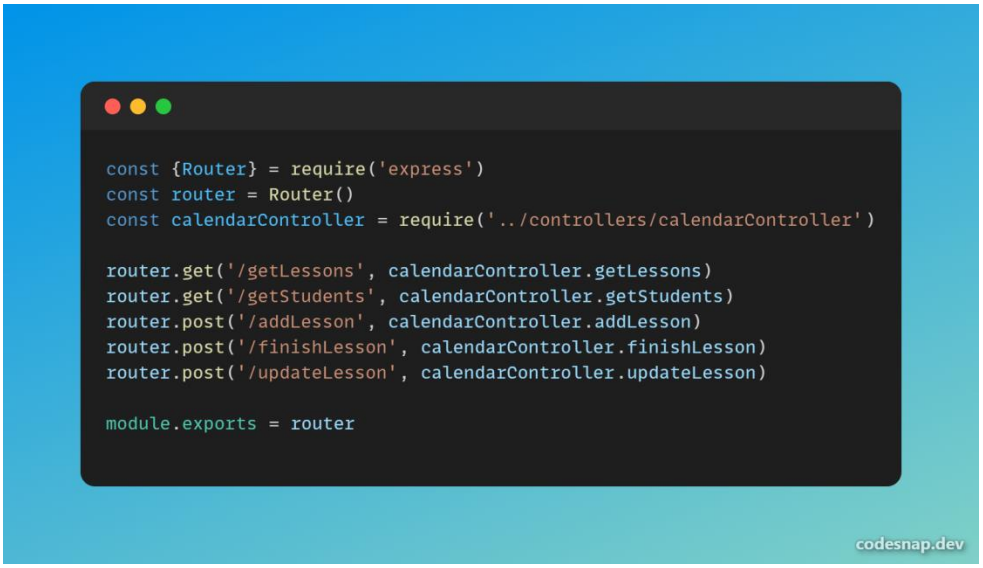


Рисунок 3.6 – Реалізація маршрутів для авторизації

3. student.routes.js. Маршрут `student.routes.js` використовується для взаємодії з інформацією, пов'язаною зі студентами. Він буде обробляти запити на отримання списку студентів (рис. 3.7), їхніх особистих даних;

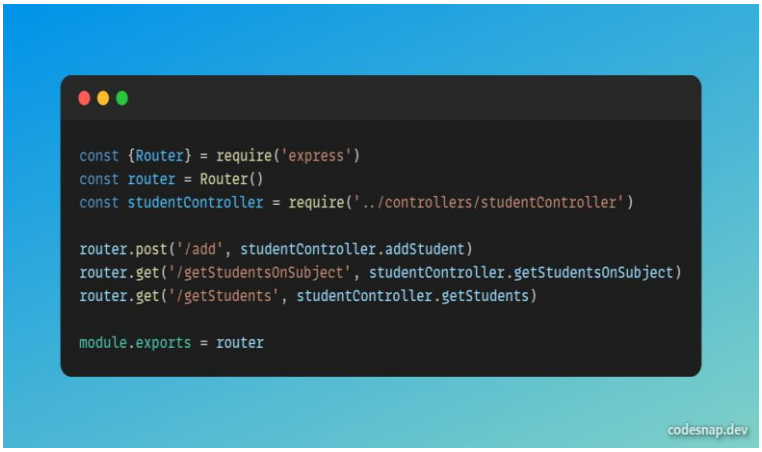


Рисунок 3.7 – Реалізація маршрутів для студентів

4. tutor.routes.js. Маршрут `tutor.routes.js` відповідальний за управління викладацькою інформацією та діяльністю викладачів. Він надасть можливість отримувати дані про викладачів (рис. 3.8), редагувати їх профілі та отримувати іншу важливу інформацію.

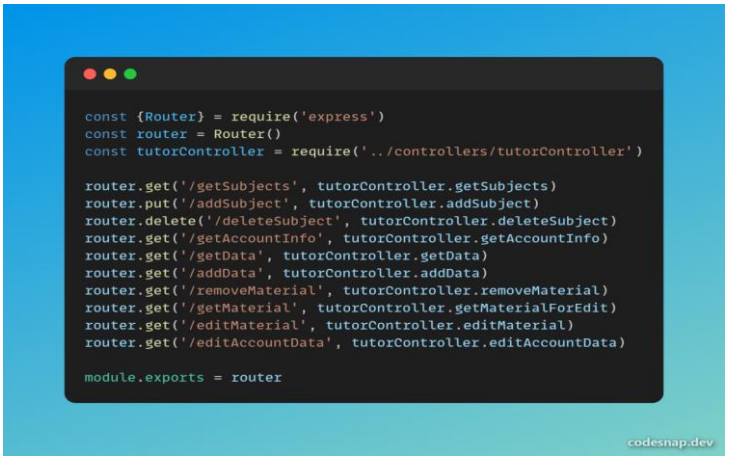


Рисунок 3.8 – Реалізація маршрутів для викладачів

Ці чотири маршрути є структурними складовими додатка, забезпечуючи відповідальність за окремі аспекти функціональності та полегшуючи розподілену роботу над різними частинами системи. Вони взаємодіють з базою даних, виконуватимуть операції з обробки даних та забезпечують ефективне функціонування веб-додатка.

Не можна не змовчати і про middleware. Middleware в контексті веб-розробки є шаром програмного забезпечення, який розташовується між клієнтом і сервером, або в процесі обробки запиту та відповіді. Вони грають важливу роль у фреймворках та бібліотеках, таких як Express.js для Node.js, Django для Python, або Ruby on Rails для Ruby. Ось деякі ключові аспекти та цілі використання middleware:

1. Обробка запитів та відповідей. Middleware дозволяє виконувати код при обробці кожного HTTP-запиту або відповіді. Це може бути корисним для перевірки даних, логування, обробки сесій, переадресації та інших дій, пов'язаних з обробкою запитів та відповідей.

2. Обробка інтерфейсів. Middleware може бути використаний для обробки даних, які приходять з користувацького інтерфейсу. Наприклад, перевірка та обробка даних форми перед їхнім збереженням у базі даних.

3. Аутентифікація та авторизація. Middleware є важливим

інструментом для реалізації механізмів аутентифікації та авторизації. Вони можуть перевіряти ідентифікаційні дані користувача, встановлювати його рівень доступу та виконувати дії в залежності від прав.

4. Логування та відслідковування. Middleware дозволяє вести журнал подій, таких як записи запитів, помилок чи іншої важливої інформації. Це значуще для відслідковування роботи програми, виявлення помилок та оптимізації її продуктивності.

5. Обробка сесій. Деякі middleware використовуються для роботи з сесіями, зберігаючи дані між різними HTTP-запитами в цілому сеансі взаємодії з користувачем.

6. Обробка помилок. Middleware може бути використаний для обробки помилок на різних етапах обробки запиту. Вони можуть перехоплювати та обробляти помилки, забезпечуючи гнучкість у їх обробці.

7. Підтримка статичних файлів. Middleware може використовуватися для обробки та надсилання статичних файлів (наприклад, зображень, таблиць стилів) безпосередньо до клієнта.

8. Кешування. Деякі middleware дозволяють реалізувати кешування для поліпшення продуктивності шляхом збереження копій ресурсів**.**

В цілому, використання middleware в веб-розробці дозволяє створювати модульні, легко управляються та розширюються додатки, забезпечуючи ефективну обробку запитів та відповідей. В проекті tutorTuls також використовуються middleware.

Функція getPasswordForStudent: ця функція створена для генерації випадкового паролю для студента (рис. 3.9). Вона використовується для створення унікальних паролів, які можуть бути надіслані студентам під час їх реєстрації чи інших сценаріїв. Процес генерації включає в себе визначення доступних символів (цифри, літери різного регістру та спеціальні символи) та випадковий вибір символів для створення паролю заданої довжини.

Згенерований пароль потім може бути використаний для входу в систему або подальшої обробки.

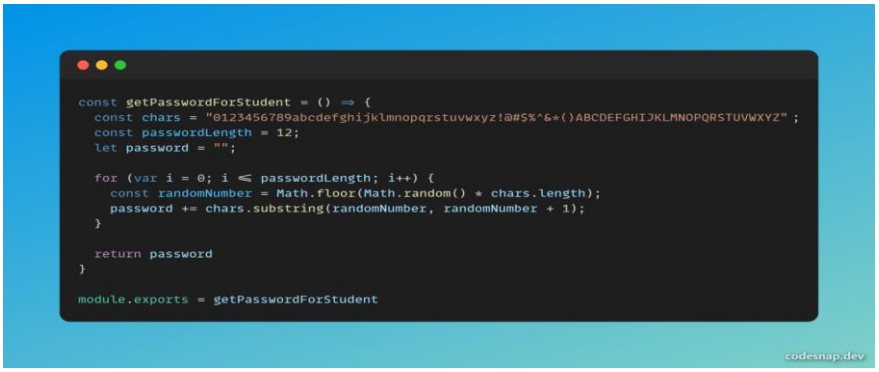


Рисунок 3.9 – Middleware для генерації паролю

Функція getUserId: ця функція використовується для отримання ідентифікатора користувача з JWT (JSON Web Token). JWT - це стандарт токенізації, який використовується для забезпечення безпеки в обміні даними між сторонами. У даному випадку, функція приймає токен як вхідний параметр і використовує його разкодовуючи за допомогою секретного ключа, щоб отримати ідентифікатор користувача (рис. 3.10).

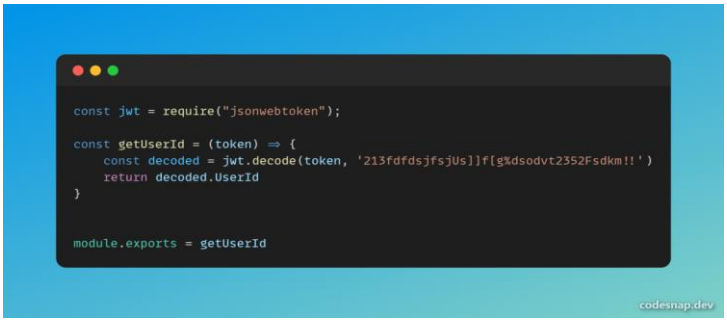


Рисунок 3.10 – Middleware для отримання id користувача

Обидві ці функції експортуються для використання в інших частинах програми, щоб можна було використовувати їх функціональність в інших модулях.

3.7 Розробка структури бази даних та методів зберігання даних

Розробка структури бази даних MongoDB включає в себе використання нереляційного (NoSQL) підходу, який дозволяє гнучко та ефективно зберігати та обробляти дані. MongoDB використовує формат документів у вигляді BSON (Binary JSON), що робить його особливо підходящим для роботи з динамічними та розширюваними данними.

В першу чергу потрібно розглянути моделі, які використовуються для взаємодії сервера з базою даних.

Даний код (рис. 3.11) є частиною моделі даних для MongoDB за допомогою бібліотеки Mongoose, яка є обгорткою над MongoDB для Node.js.

Рисунок 3.11 – Імпорт функціоналу mongoose

Тут використовуються різні об'єкти з Mongoose, такі як Schema, model, і Types. Schema використовується для визначення структури документа, model створює модель даних на основі схеми, а Types надає корисні типи даних.

У наступному рядку коду (рис. 3.12) визначається схема документа MongoDB.

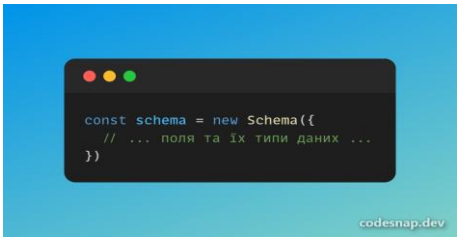


Рисунок 3.12 – Приклад створення нової схеми

Експорт моделі (рис. 3.13) завершує визначення моделі для колекції "Student" в базі даних MongoDB.

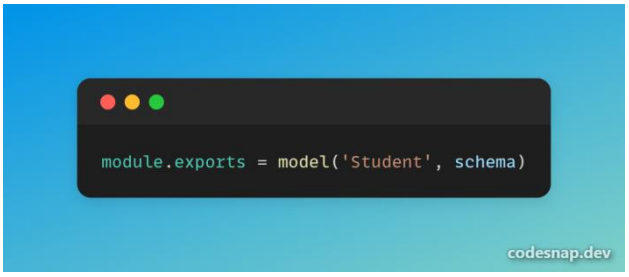


Рисунок 3.13 – Експорт схеми Student

Зараз я хочу розглянути поля для таблиці Student (рис. 3.14):

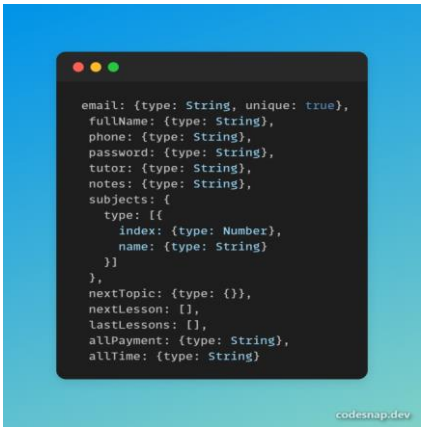


Рисунок 3.14 – Модель таблиці Student

- email, fullName, phone, password: загальні поля для інформації про студента, такі як електронна пошта, ім'я, телефон та пароль;
- tutor, notes: додаткові поля для інформації про репетитора та нотатки;
- subjects: масив об'єктів, які містять індекс та ім'я предметів;
- nextTopic: об’єкт, який містить дані про наступну тему;
- nextLesson, lastLessons: масиви, які представляють інформацію про майбутні, плановані та минулі уроки;

- allPayment, allTime: загальна інформація про всі оплати та час; Демострація схеми уроку (рис. 3.15):
- date: визначає дату майбутнього уроку у форматі рядка;
- subjec: вказує предмет, який буде вивчатися на майбутньому уроці;
- topic: описує тему, яку будуть вивчати на уроці;
- payment: вказує інформацію про оплату за урок;
- during: поля, яке визначає тривалість уроку;
- dataForLesson: представляє собою масив рядків, які містять додаткові дані для уроку;
- homeworkOnLesson: вказує домашнє завдання, яке доручено після уроку;
- status: логічне значення, яке вказує на статус уроку, наприклад, чи відбувся він чи ні;
- meetLink: поле, що містить посилання для віртуальної зустрічі, наприклад, Zoom-чи Google Meet-посилання;
- pingStudents та preparePing: логічні значення, які вказують на наявність попереджень чи повідомлень перед уроком. наприклад, pingStudents може вказувати на те, що студентів повідомлять перед уроком;
- evaluate: визначає оцінку за урок, яка може мати значення від "1" до "12";

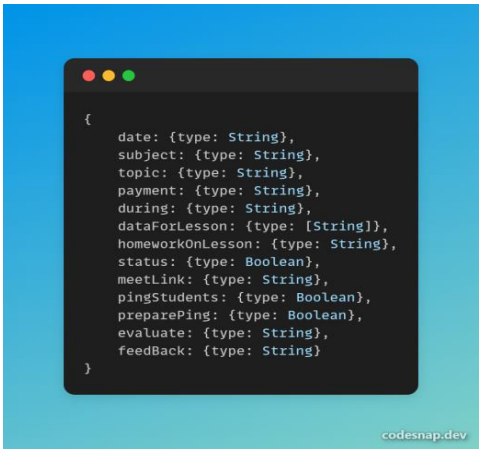


Рисунок 3.15 – Модель уроку

Також, потрібно розглянути таблцю Tutor (рис. 3.16):

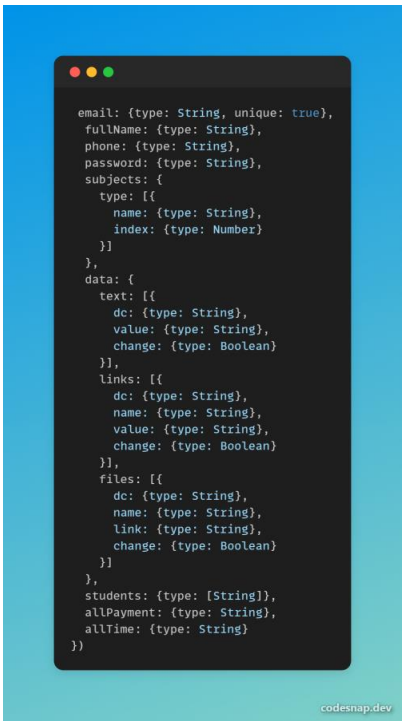


Рисунок 3.16 – Модель таблиці Tutor

- email: визначає електронну пошту користувача. Унікальність (unique) вказує на те, що кожна адреса повинна бути унікальною у системі;
- fullName: зберігає повне ім'я користувача;
- phone: містить номер телефону користувача;
- password: зберігає пароль користувача. Його тип String вказує на те, що він буде представлений у текстовому форматі;
- subjects: масив об'єктів, який містить дані про предмети, які викладає користувач. Кожен об'єкт включає name (назва предмету) та index (індекс предмету) ;
- data: об'єкт, який містить різні типи даних. Наприклад, text може включати масив об'єктів з текстовою інформацією та змінами, links - з посиланнями, а files - з інформацією про файли;
- students: масив рядків, що містить ідентифікатори або імена студентів, пов'язаних з цим користувачем;

- allPayment та allTime: загальна інформація про оплати та час, можливо, якісь загальні статистичні дані або сумарна інформація.

3.8 Розробка окремих контролерів для управління даними

В розробці програмного забезпечення, розробка окремих контролерів відіграє важливу роль у забезпеченні ефективного управління даними. Контролери визначають логіку обробки запитів, взаємодіючи з базою даних та іншими компонентами системи. Цей процес не лише спрощує роботу з даними, але й сприяє створенню стійких, ефективних та масштабованих бекенд рішень. У даному контексті, вірно розроблені контролери є ключовим елементом для забезпечення функціональності та надійності веб-додатків.

Давайте розглянемо основний функціонал для авторизації:

```
async accountRegistration(req, res) {
  try {
    const {email, password, phone, fullName, subject, fullNameStudent, emailStudent,
phoneStudent, notice} = req.body

    const isUsed = await Tutor.findOne({email})
    const isUsedStudent = await Student.findOne({email: emailStudent})

    if (isUsed) {
      res.json({status: 'error', message: 'email is used'})
      return
    }
```

Отримання змінних студента:

```
    if (isUsedStudent) {
      res.json({status: 'error', message: 'student email is used'})
      return
    }

    const studentPassword = getPasswordForStudent();
```

```
const hashedPasswordStudent = await bcrypter.hash(studentPassword, 12)
const hashedPasswordTutor = await bcrypter.hash(password, 12)

const student = await new Student({
  email: emailStudent,
  fullName: fullNameStudent,
  phone: phoneStudent,
  notes: notice,
  password: hashedPasswordStudent,
  subjects: Array(subject),
})
```

Отримання змінних викладача:

```
const tutor = await new Tutor({
  email,
  fullName,
  phone,
  password: hashedPasswordTutor,
  subjects: Array(subject),
  students: [student.id],
  allPayment: '0',
  allTime: '0'
})

await tutor.save()
student.tutor = tutor.id
await student.save()
```

Створення листа:

```
let transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.GMAIL_EMAIL,
    pass: process.env.GMAIL_PASSWORD
  }
});
```

```
    await transporter.sendMail({
      from: process.env.GMAIL_EMAIL,
      to: emailStudent,
      subject: "Tutor tuls",
      html: `<p>Доброго дня. Ваш аккаунт студента створено. Ви можете зайти у свій
акаунт за допомогою паролю: <br> <b>${studentPassword}</b> </p>`
    });
```

Опрацьовування відповіді:

```
    res.json({status: 'user created'})
  } catch (err) {
    res.send({status: 'error'})
    console.log(err)
  }
}
```

Ця функція відповідає за обробку запитів на реєстрацію облікових записів користувачів, які можуть бути викладачами (Tutor) або студентами (Student) в системі:

1. Отримання даних з запиту - функція отримує дані з тіла HTTP-запиту, включаючи електронну пошту (email), пароль (password), телефон (phone), повне ім'я (fullName), предмет (subject), інформацію про студента (якщо це реєстрація студента).

2. Перевірка унікальності електронної пошти - використовуються запити до бази даних MongoDB, щоб перевірити, чи існує вже в системі викладач чи студент з вказаною електронною адресою.

3. Створення паролів та хешування - генеруються та хешуються паролі для викладача та студента за допомогою функції getPasswordForStudent та bcrypter.hash, відповідно.

4. Створення об'єктів Tutor та Student - створюються об'єкти моделей Tutor та Student з введеними даними. Додаються відносини між ними (викладач

має студента, студент належить викладачеві).

5. Відправка листа для підтвердження реєстрації студента - використовується nodemailer для надсилання електронного листа з тимчасовим паролем студента на вказану електронну адресу.

6. Збереження в базі даних - об'єкти викладача та студента зберігаються в базі даних за допомогою методів save.

7. Відповідь на клієнтський запит - залежно від успіху або невдачі, повертається відповідь у форматі JSON зі статусом операції.

8. Обробка помилок - у випадку виникнення помилок під час обробки запиту, вони виводяться в консоль, і клієнту надсилається відповідь із статусом помилки.

Ця функція взаємодіє з базою даних, генерує паролі, надсилає листи, і взагалі керує процесом реєстрації користувачів в системі.

Також важливим компонентом авторизації є логін користувача:

```
async accountLogin(req, res) {
  try {
    const {email, password} = req.body

    const tutor = await Tutor.findOne({email})
    const student = await Student.findOne({email})
```

Отримання кешованого токену для викладача:

```
    if (tutor) {
      bcrypter.compare(password, tutor.password, (err, resp) => {
        if (resp) {
          const jwtSecret = process.env.JWT_SECRET

          const token = jwtToken.sign({UserId: tutor.id},
            jwtSecret, {expiresIn: '1h'}
          )
          res.json({token, userId: tutor.id})
        } else {
```

```
          res.json({status: 'error', message: 'an incorrect password tutor'})
        }
       })
```

Отримання кешованого токену для студента:

```
    } else if (student) {
      bcrypter.compare(password, student.password, (err, resp) => {
        if (resp) {
          const jwtSecret = process.env.JWT_SECRET

          const token = jwtToken.sign({UserId: student.id},
            jwtSecret, {expiresIn: '1h'}
          )
          res.json({token, userId: student.id})
        } else {
          res.json({status: 'error', message: 'an incorrect password student'})
        }
      })
    } else {
      res.json({status: 'error', message: 'student and tutor is not defined'})
    }
```

Опрацьовування помилок:

```
   } catch (err) {
     res.json({status: 'error'})
     console.log(err)
   }
  }
}
```

Ця функція відповідає за обробку запитів на вхід в систему для користувачів – як викладачів (Tutor), так і студентів (Student). Давайте розглянемо її детальніше:

Функція приймає дані з тіла HTTP-запиту, включаючи електронну пошту (email) та пароль (password). Після цього виконується пошук користувача серед

викладачів та студентів в базі даних за допомогою методів Tutor.findOne та Student.findOne.

Якщо знайдено викладача (tutor), то порівнюється введений пароль з хешем пароля викладача, використовуючи `bcrypter.compare`. Якщо порівняння успішне, генерується токен за допомогою бібліотеки jwtToken, який містить ідентифікатор користувача (UserId) та підписаний секретний ключ. Цей токен повертається як відповідь на клієнтський запит, разом з ідентифікатором користувача.

Аналогічно, якщо знайдено студента (student), також виконується порівняння паролю та генерація токену.

У випадку, якщо користувач не знайдений серед викладачів чи студентів, повертається відповідь із статусом помилки.

Функція враховує можливі помилки в процесі входу в систему та повертає відповіді згідно з результатами виконання.

Важливим компонентом програми також є функції повернення списку студентів та повернення списку предметів.

```
async getLessons(req, res) {
  try {
    const {dateValue} = req.query

    const token = req.headers.authorization
    const userId = getUserId(token)
    const tutor = await Tutor.findOne({_id: userId})
    let students = await Student.find({tutor: userId})

    if (students?.length) {
      let lessons = []
```

Добавлення уроків для студента:

```
      students.forEach(student => {
        student.nextLesson.forEach(lesson => {
```

```
        if (!lesson.status) {
          lessons.push({lesson, fullName: student.fullName, id: student._id})
        }
       })
     })

     // console.log(lessons[0].lesson.date.slice(12, 15))

     // Sort by date lesson
     lessons = lessons.sort((a, b) => {
       return getDateFormats(a.lesson.date) > getDateFormats(b.lesson.date) ? 1 : -1
     })


     res.send({lessons})
   }
```

Опрацьовування помилок:

```
  } catch (e) {
    res.json({status: 'error'})
    console.log(e)
  }
}
```

Функція для отримання студента:

```
async getStudents(req, res) {
  try {
    const token = req.headers.authorization
    const userId = getUserId(token)
    let students = await Student.find({tutor: userId})

    if (students?.length) {
      res.send({students})
      return
    }
```

```
    res.send({status: 'error', message: 'students not found'})

  } catch (e) {
    res.json({status: 'error'})
    console.log(e)
  }
}
```

1. Async getLessons(req, res) - ця функція відповідає за отримання невиконаних уроків (запланованих, але ще не проведених) для викладача. Спочатку вона отримує дату з параметра запиту `dateValue` та ідентифікатор користувача з токену авторизації. Далі, шляхом пошуку викладача та його пов'язаних студентів в базі даних, функція формує масив уроків, які ще не мають статусу виконання. Ці дані сортуються за датою проведення уроку, і відправляються у відповідь клієнту.

2. Async getStudents(req, res) - ця функція служить для отримання списку студентів, які належать конкретному викладачу. Вона використовує ідентифікатор користувача з токену для знаходження викладача в базі даних та подальшого пошуку студентів, які належать цьому викладачеві. Якщо студенти знайдені, вони надсилаються у відповідь клієнту; в іншому випадку, повертається повідомлення про помилку.

Ці функції взаємодіють з базою даних, використовують ідентифікатор користувача для забезпечення безпеки, та надсилають відповіді клієнтам з урахуванням відповідних даних.

Необхідно також зазначати POST методи для добавлення уроку, та предмету.

```
async addLesson(req, res) {
  try {
    const {
      date,
      studentId,
      feedback,
```

```
  evaluate,
  subject,
  topic,
  dataForLesson,
  meetLink,
  homeworkOnLesson,
  during,
  payment,
  pingStudents,
  preparePing
} = req.body
```

Знаходження студента по id:

```
if (studentId, subject, evaluate, topic, meetLink, homeworkOnLesson, during, payment) {
  const student = await Student.findById(studentId)

  student?.nextLesson.push({
    date,
    subject,
    topic,
    payment,
    during,
    dataForLesson,
    homeworkOnLesson,
    meetLink,
    pingStudents,
    preparePing,
    evaluate,
    feedback,
    status: false
  })
```

Зберігання даних студента:

```
  await student.save()

  res.json({status: 'ok'})
```

```
    }
  } catch (e) {
    res.json({status: 'error'})
    console.log(e)
  }
}
```

Додавання студента:

```
async addSubject(req, res) {
  try {
    const {subject} = req.body
    const token = req.headers.authorization
    const userId = getUserId(token)

    const tutor = await Tutor.findById(userId)

    tutor.subjects.push(subject)

    await tutor.save()
    res.json({status: 'ok'})
  } catch (e) {
    res.json({status: 'error'})
    console.log(e)
  }
}
```

1. Async addLesson(req, res) - ця функція призначена для додавання нового уроку до розкладу конкретного студента. Вона отримує дані з тіла HTTP-запиту, такі як дата, ідентифікатор студента (studentId), звіт, оцінка, предмет, тема уроку, інші дані для уроку, посилання на зустріч, домашнє завдання тощо. Функція перевіряє наявність обов'язкових полів, а потім знаходить студента за його ідентифікатором, додає інформацію про новий урок у масив nextLesson студента та зберігає зміни в базі даних. Після успішного виконання повертає відповідь зі статусом "ok".

2. Async addSubject(req, res) - ця функція використовується для додавання

нового предмету до списку предметів викладача. Вона отримує дані про предмет з тіла HTTP-запиту та ідентифікатор викладача з токену авторизації. Функція знаходить викладача за його ідентифікатором, додає новий предмет до його списку предметів, зберігає зміни в базі даних та повертає відповідь із статусом "ok".

Ці функції враховують можливі помилки, використовують асинхронні методи для взаємодії з базою даних та повертають відповіді згідно з результатами виконання.

3.9 Розробка інтерфейсу користувача, та його функціональність

Розділ присвячений ключовим аспектам розробки інтерфейсу користувача (UI), його дизайну та функціональності у контексті створення веб-програмного забезпечення. Інтерфейс користувача визначає взаємодію між користувачем та програмою, визначаючи спосіб сприйняття та взаємодії з функціоналом.

На сторінці реєстрації (рис. 4.15) користувачеві пропонується заповнити різноманітні поля для створення облікового запису. Основні елементи сторінки включають:

1. Заголовок:

- заголовок на сторінці приваблює увагу користувачів та вказує на можливість безкоштовної проби продукту Tutor Tuls**.**

2. Введення користувача:

- є набір полів для введення основної інформації: ім'я та прізвище, email, номер телефону, пароль і його повторення**;**

- є текстова підказка, що нагадує користувачам про формат номеру телефону.

3. Вибір предмету:

- користувач може вибрати предмет, який він викладає, зі списку предметів**;**

- пояснення над полем вказує, що після реєстрації можна буде змінити обраний предмет.

4. Додавання першого студента:

- є блок, що дозволяє користувачеві додати першого студента за допомогою форми AddStudentForm**;**

- ця форма включає необхідні поля, які відповідають інформації.

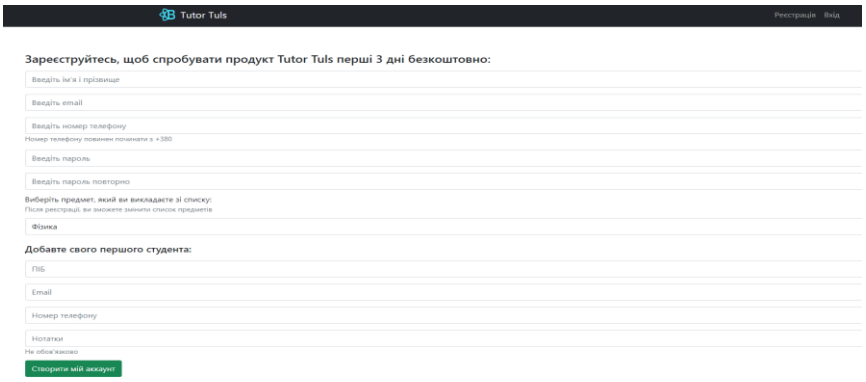


Рисунок 4.15 – Сторінка реєстрації

Основна функціональність і особливості сторінки логіну (рис. 3.17) включають в себе:

1. Введення облікових даних:

- користувач вводить свою електронну пошту та пароль для входу.

2. Валідація введених даних:

- використовується регулярний вираз для перевірки правильності введення електронної пошти;

- проводиться перевірка довжини паролю (повинен бути не менше 7 символів) ;

3. Відправлення запиту на сервер:

- користувач відправляє дані (електронну пошту та пароль) на сервер для аутентифікації.

4. Оновлення стану під час відправлення запиту:

- під час відправлення запиту на сервер відображається індикатор завантаження (Loader).

5. Обробка відповіді від сервера:

- обробляється відповідь від сервера після спроби входу;

- при успішній аутентифікації користувача отримується токен доступу;

- при успішній аутентифікації користувача отримується токен доступу.

6. Відображення повідомлень про помилки:

- використовується NotificationManager для виведення повідомлень про помилки при некоректних діях користувача.

Загалом, цей компонент відповідає за інтерфейс і логіку сторінки входу, надаючи користувачеві можливість увійти в систему "Tutor Tuls". Його ефективна робота є ключовою для забезпечення безпеки та зручності користувачів при використанні програмного забезпечення.

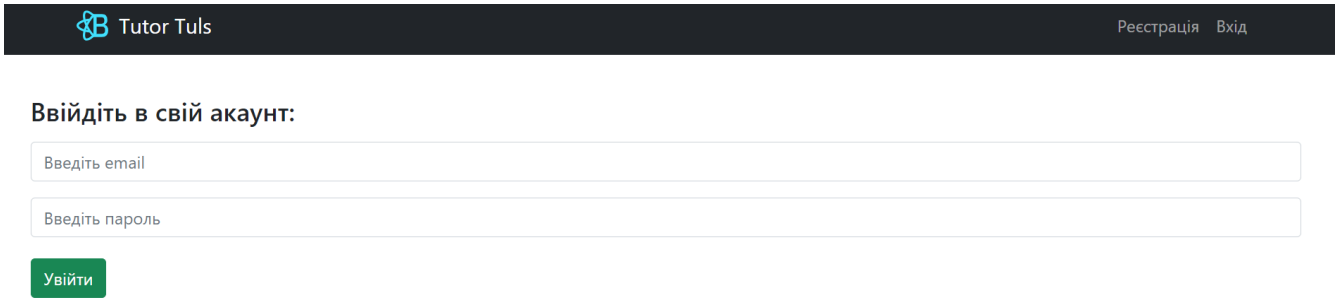


Рисунок 3.17 – Сторінка логіну

Після того, як користувач ввійшов в систему, він побачить перед собою сторінку предметів (рис 3.18).

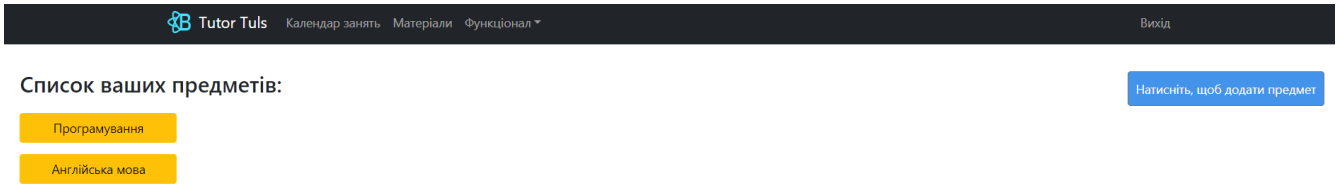


Рисунок 3.18 – Сторінка предметів

Якщо користувач нажме на предмет, він побачить перед собою сторінку предмету (рис.3.19), де будуть студенти, яким цей предмет добавлений, також користувач може видалити предмет. В інформації про студентів користувач побачить ім’я, email, телефон та кількість проведених уроків. Це дозволить викладачу отримати повні дані про студента, якими він зможе розпоряджатись у робочих цілях.

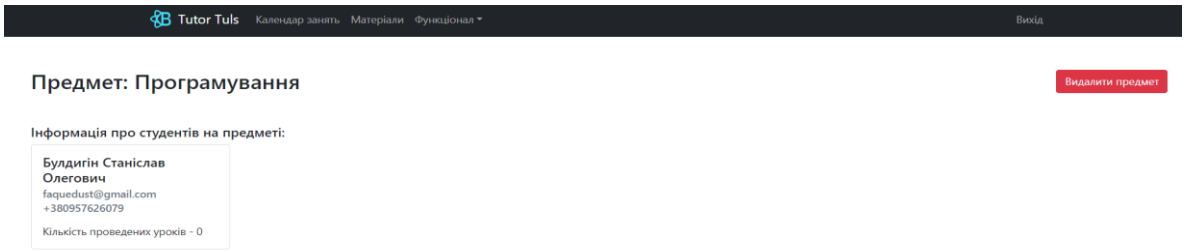


Рисунок 3.19 – Сторінка предмету

При натисканні в шапці сайту на «Календар занять» користувач побачить сторінку, на якій буде відображатись інформація про його уроки або їх відсутність (рис. 3.20)

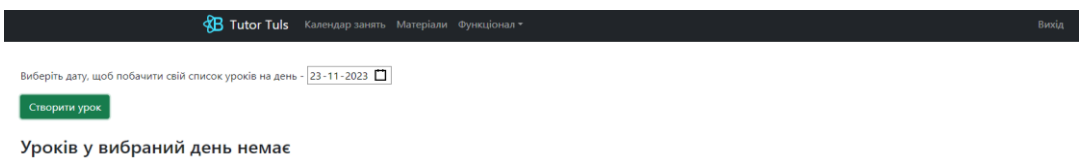


Рисунок 3.20 – Відсутність уроків в користувача

Користувач може натиснути на кнопку «Створити урок» та побачити модальне вікно з добавлення уроку (рис. 3.21). Також викладач бачить сьогоднішню дату і може вибрати іншу, щоб переглянути уроки які в нього будуть або були іншими днями. Натиснувши на урок який він побачить всю інформацію яку при створенні він заніс туди, і зможе нею користуватись.

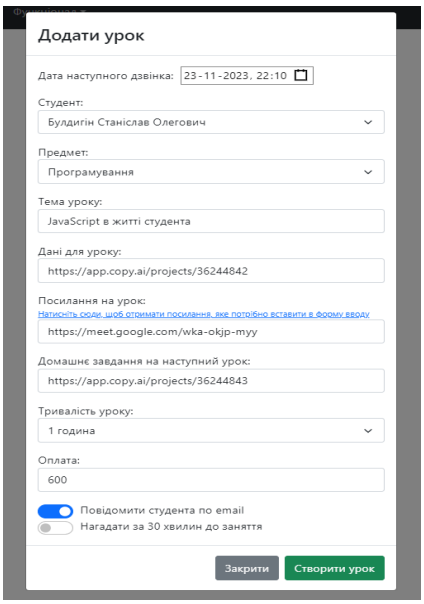


Рисунок 3.21 – Модальне вікно з створенням уроку

Ця сторінка відображає модальне вікно, яке дозволяє користувачеві додати новий урок у системі. Розглянемо основний функціонал:

1. Вибір дати та часу:

- користувач може обрати дату та час наступного уроку за допомогою віджета календаря.

2. Вибір студента:

- користувач може вибрати студента зі списку, який завантажується з сервера.

3. Вибір предмету:

- користувач може вибрати предмет зі списку, який також

завантажується з сервера.

4. Введення теми уроку:

- є поле для введення теми уроку. Якщи тема коротша за 2 символи, відображається помилка.

5. Введення додаткових даних для уроку:

- користувач може ввести додаткові дані для уроку, які відображаються як текстове поле.

6. Введення посилання на віртуальний клас (Meet):

- користувач може ввести посилання на віртуальний клас. Якщо посилання некоректне, відображається помилка.

7. Введення домашнього завдання:

- є поле для введення домашнього завдання на наступний урок. Якщо завдання коротше за 4 символи, відображається помилка.

8. Вибір тривалості уроку:

- користувач може вибрати тривалість уроку зі списку від 1 до 5 годин.

9. Введення оплати за урок:

- є поле для введення суми оплати за урок. Якщо введене значення не є числом, відображається помилка.

10. Опції повідомлення студента та нагадування:

- користувач може вибрати опції повідомлення студента по електронній пошті та нагадування за 30 хвилин до уроку за допомогою перемикачів.

11. Кнопки для взаємодії:

- "Закрити" дозволяє користувачеві закрити модальне вікно;

- "Створити урок" ініціює додавання нового уроку. Якщо є помилки у введених даних, вони відображаються, і користувач отримає відповідне

повідомлення.

Після того, як користувач створив урок, йому автоматично цей урок відобразиться в таблиці (рис. 3.22).

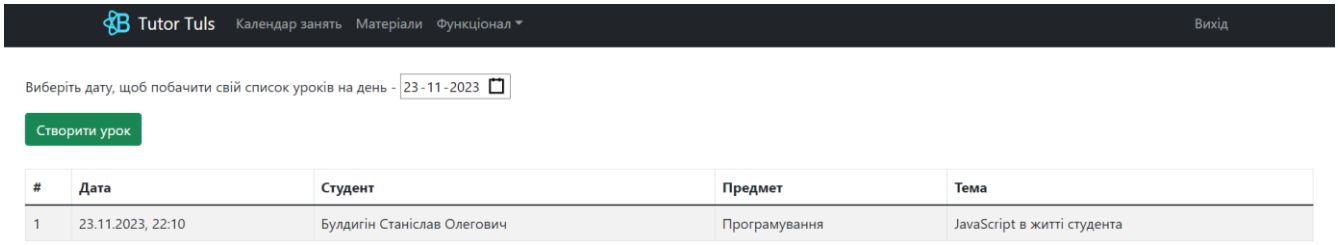


Рисунок 3.22 – Таблиця із заняттями

Якщо користувач натисне на заняття, він побачить інформацію про цей урок (рис. 3.23). В його доступі будуть кнопки «Закрити», «Оновити дані» та «Завершити урок». Колір кнопки закрити буде сірим, судячи з фідбеку користувачів, не викликатиме в користувача бажання натиснути цю кнопку, натомість зелені кнопки викликатимуть в користувача бажання ними користуватись.

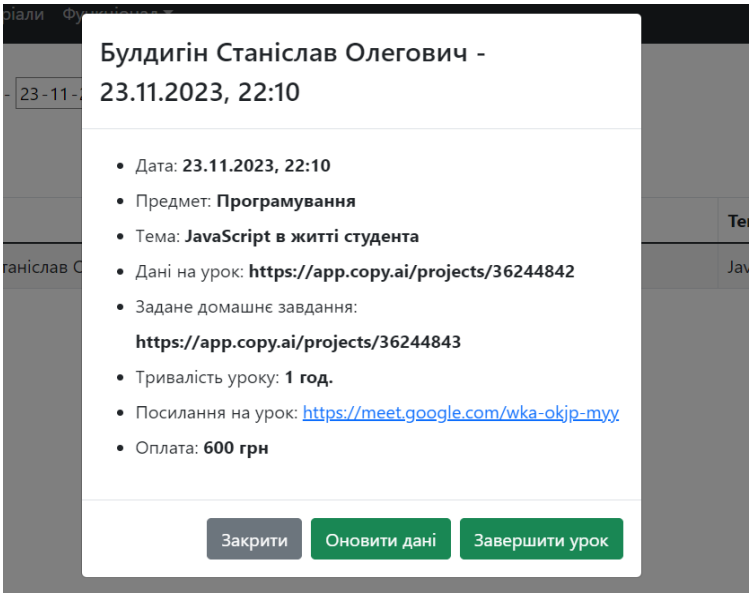


Рисунок 3.23 – Модальне вікне з заняттям

Після того як користувач натисне на кнопку «Матеріали», яка знаходиться в шапці сайту, він побачить сторінку з його матеріалами і нотатками (рис. 3.24). На сторінці він може перейти по збережному посиланню, відредагувати матеріал або видалити його.

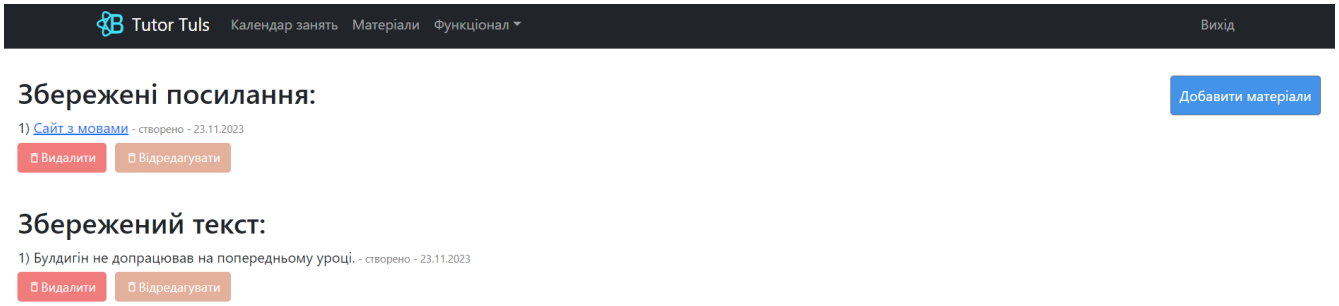


Рисунок 3.24 – Сторінка матеріалів

Також в нього є можливість натиснути на кнопку «Добавити матеріали», де він побачить модальне вікно з полями, в які користувач зберігає (рис. 3.25).

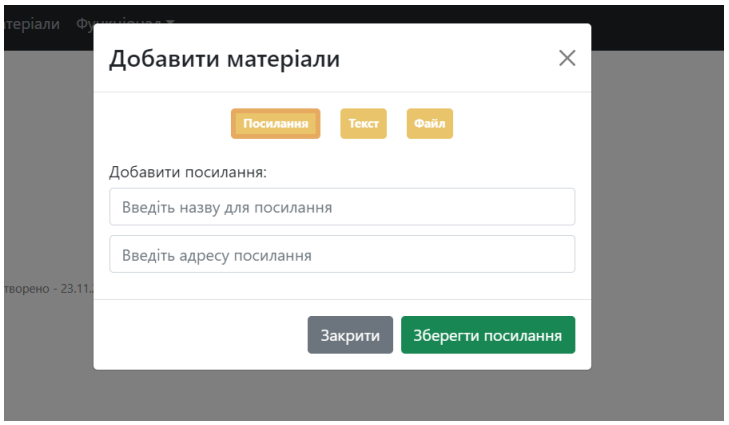


Рисунок 3.25 – Модальне вікно з добавленням матеріалом

Якщо користувач наведе мишу на кнопку «Функціонал», він побачить 3 різні пункти в дропдауні, це «Додати студента», «Інформація про аккаунт», «Звітність». Після того як викладач натисне на кнопку «Додати студента», він

побачить сторінку з додаванням студента (рис .3.26), схожу на сторінку яку він бачив при реєстрації.

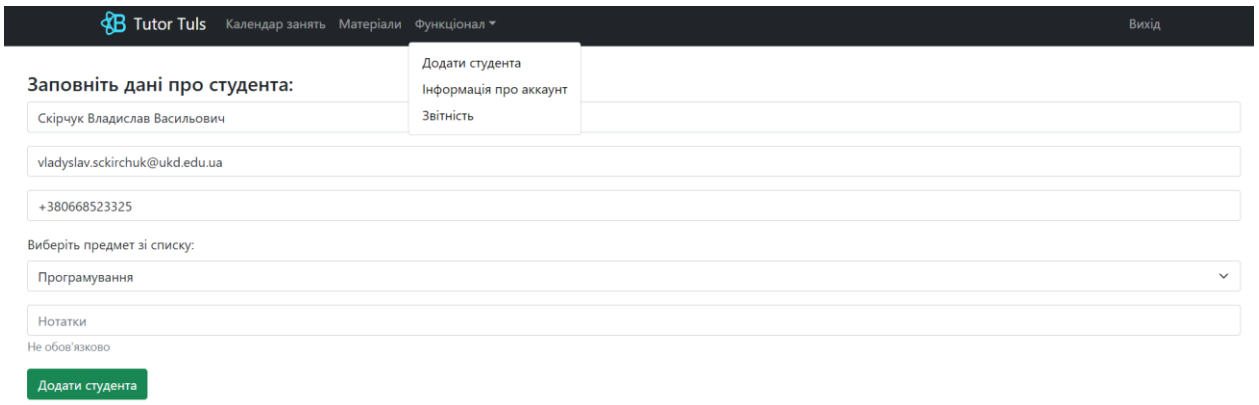


Рисунок 3.26 – Сторінка для додаванням студента

При відкритті сторінки «Інформація про аккаунт» користувач бачить перед собою сторінку з даними аккаунта (рис. 3.27).

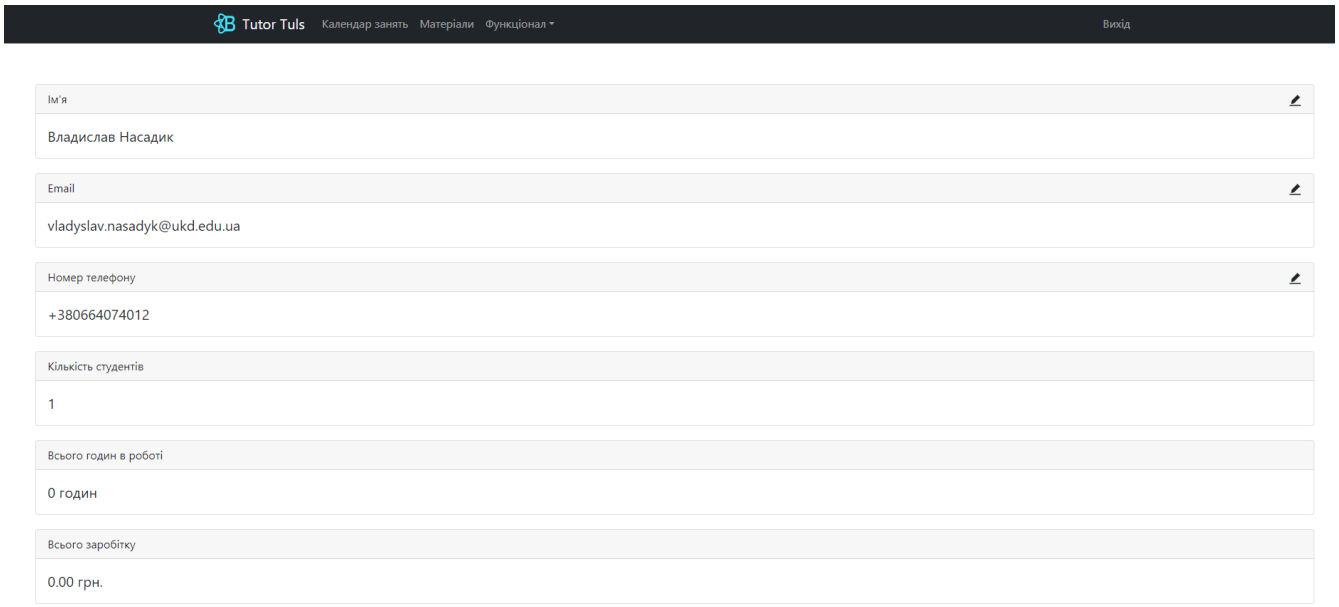


Рисунок 3.27 – Сторінка з інформацією про аккаунт

Давайте розглянемо її функціонал:

1. Відображення основних даних:

- показує основні дані викладача, такі як ім'я, email, та номер телефону;

- для кожного поля виводить кнопку редагування, яка дозволяє змінити відповідні дані.

2. Кількість студентів:

- виводить загальну кількість студентів, які призначені викладачу.

3. Загальна кількість годин в роботі:

- обчислює і виводить загальну кількість годин, які викладач витратив на уроки з усіх студентів.

4. Загальний заробіток:

- розраховує і виводить загальну суму заробітку викладача за всі уроки, враховуючи плату за кожен урок.

5. Можливість редагування:

- Забезпечує можливість редагувати основні дані викладача через модальне вікно, яке викликається при натисканні на кнопку редагування.

6. Завантаження даних:

- при завантаженні сторінки використовується асинхронний запит на сервер для отримання і відображення даних викладача та списку студентів.

7. Можливість редагування інших даних:

- забезпечує можливість редагування додаткових даних викладача, таких як ім'я, email, і номер телефону.

8. Можливість оновлення даних:

- після редагування даних в модальному вікні надсилає запит на сервер для оновлення інформації.

9. Обчислення та відображення інформації з уроків студентів:

- виводить загальну кількість годин та заробітку на основі уроків, які були проведені для кожного студента.

Після того, як користувач перейде на сторінку звітності, він побачить

перед собою можливість переглянути 3 різних звіти, такі як «Звіт про прогрес студентів», «Звіт вільних та зайнятих годин» та «Фінансовий звіт». Давайте розглянемо кожен з них.

Звіт про прогрес студентів призначений для користувачів, які хочуть отримати зведену інформацію про навчальний прогрес конкретного студента. Користувачеві надається зручний та легкий у сприйнятті засіб перегляду середньої оцінки студента та деталей його останніх уроків (рис. 3.28).

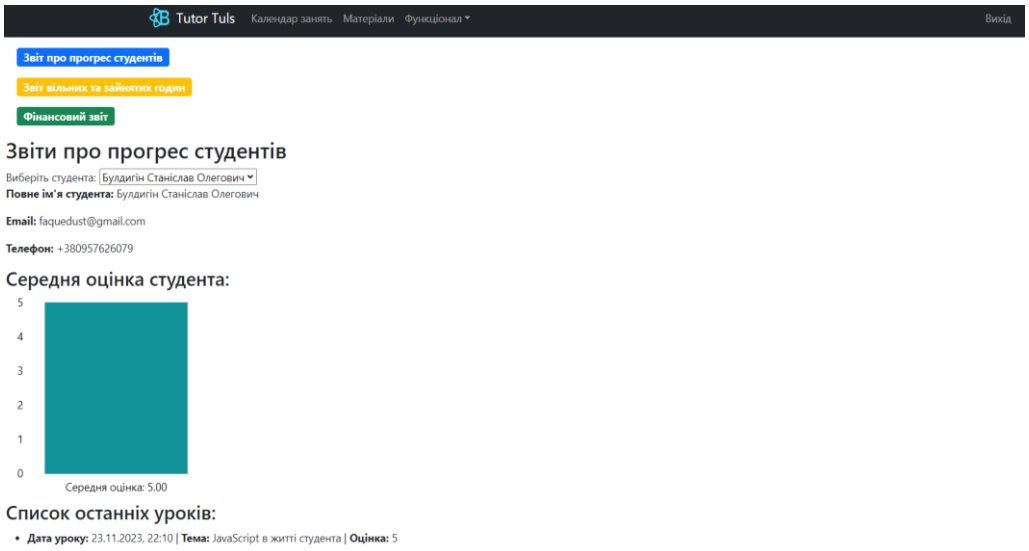


Рисунок 3.28 – Інтерфейс звіту про прогрес студентів

На початку сторінки вказана основна особиста інформація про студента, така як повне ім'я, електронна пошта і телефонний номер. Після цього слідує блок, де представлена середня оцінка студента, яка відображається як числове значення та графік. Графік дозволяє користувачеві швидко зрозуміти динаміку успішності студента.

Далі на сторінці розміщений перелік останніх уроків студента, де для кожного уроку вказана дата, тема та отримана оцінка. Цей перелік надає детальний огляд навчальної діяльності студента.

Важливо відзначити, що якщо у студента немає оцінок, то замість середньої оцінки виводиться повідомлення "Немає оцінок". Це спрощує сприйняття інформації та робить сторінку корисною для вивчення прогресу

будь-якого студента, навіть якщо він тільки розпочав свій навчальний шлях.

Звіт вільних та зайнятих годин розроблений для відображення інформації про графік вільних та зайнятих годин викладача на сьогоднішній день (рис. 3.29). Користувачеві пропонується чіткий та зручний спосіб перегляду часових інтервалів, коли викладач доступний для проведення уроків та коли його робочий час зайнятий.

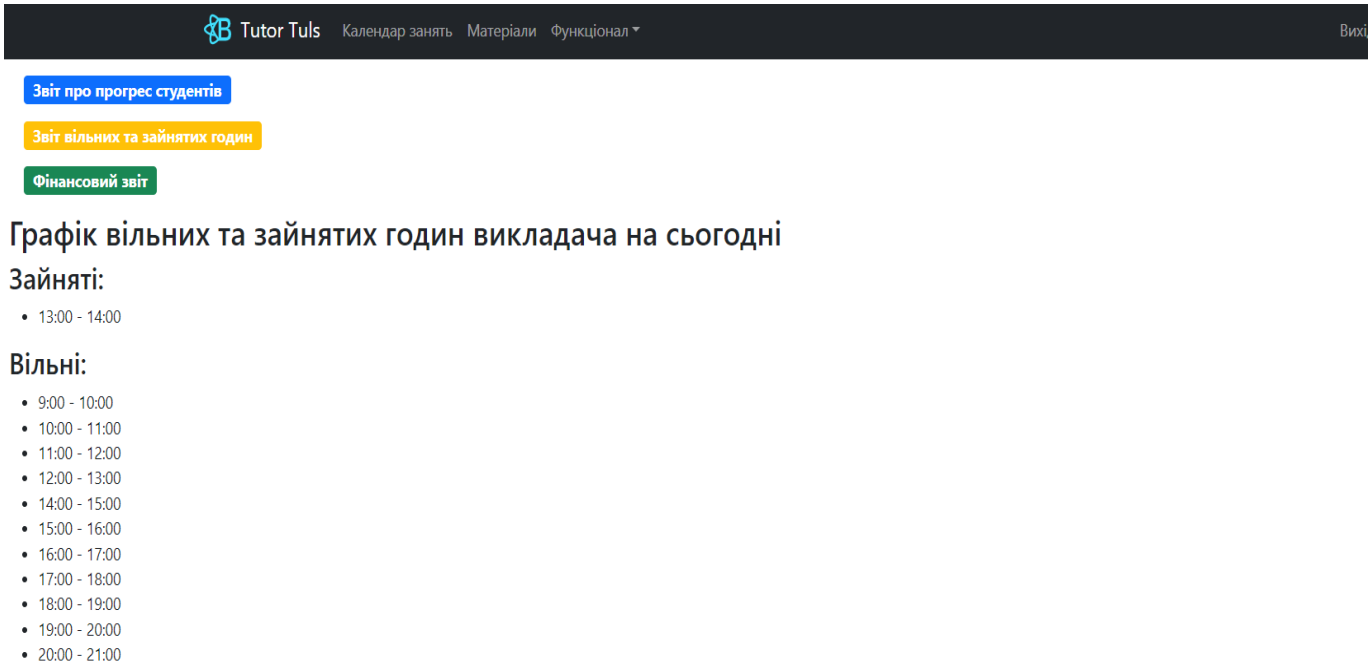


Рисунок 3.29 – Звіт вільний та зайнятих годин

На початку сторінки виводиться заголовок "Графік вільних та зайнятих годин викладача на сьогодні", що вказує користувачеві на тематику звіту.

Далі сторінка розділена на дві частини: "Зайняті" та "Вільні". В розділі "Зайняті" виводяться годинні інтервали, коли викладач вже має заплановані уроки на сьогодні. Для кожного годинного інтервалу вказується час початку і закінчення зайнятого періоду.

У розділі "Вільні" виводяться годинні інтервали, коли викладач вільний і може приймати нові уроки. Також для кожного годинного інтервалу вказується час початку і закінчення вільного періоду.

Цей звіт допомагає користувачеві легко зорієнтуватися в розкладі

викладача та визначити оптимальний час для запланування нових уроків.

Фінансовий звіт створений для відображення звіту про фінанси (рис. 3.30). Користувач може вибрати період (тиждень, місяць, рік або два роки) і отримати інформацію про загальні зароблені кошти від кожного студента за обраний період. Також, для кожного студента виводиться топ-3 уроки з найвищою оплатою.

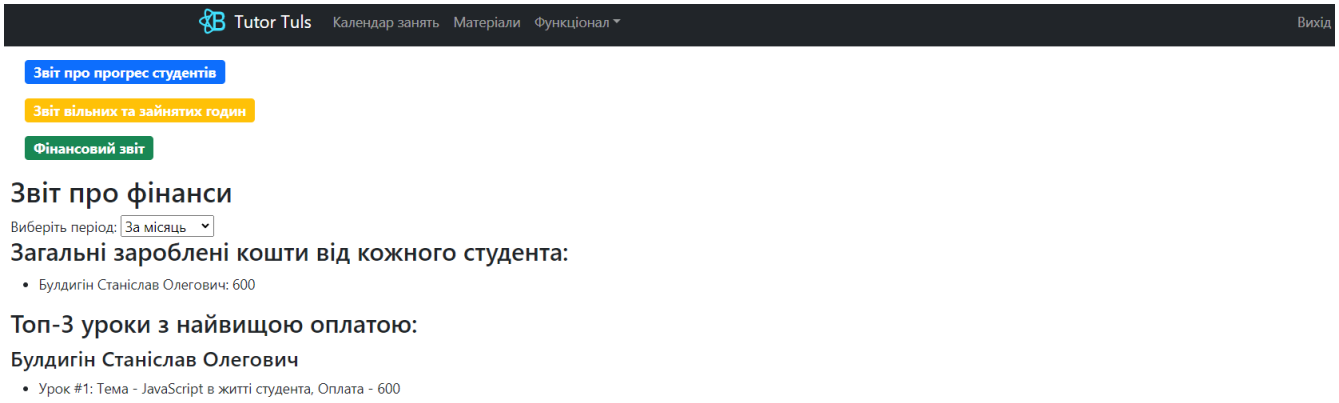


Рисунок 3.30 – Фінансовий звіт

На початку сторінки виводиться заголовок "Звіт про фінанси", що вказує на тематику звіту.

Під заголовком розташований випадаючий список, де користувач може обрати бажаний період для аналізу - тиждень, місяць, рік або два роки. Після вибору періоду, сторінка автоматично перезавантажується, враховуючи обраний період.

Далі слідують два блоки з інформацією:

1. "Загальні зароблені кошти від кожного студента" - список студентів та сума їх зароблених коштів за обраний період. Для кожного студента вказується його повне ім'я та сума зароблених коштів.

2. "Топ-3 уроки з найвищою оплатою" - для кожного студента виводиться три уроки з найбільшою оплатою. Для кожного уроку вказується його тема та

сума оплати.

Цей звіт допомагає викладачеві та адміністраторам платформи аналізувати фінансові результати та визначати найбільш прибуткових студентів та уроки.

Проект включає різноманітні інтерфейси користувача для відображення та управління різними аспектами платформи. Ці інтерфейси користувача забезпечують зручний та інтуїтивно зрозумілий доступ до ключової інформації та функціоналу для ефективного взаємодії з платформою.

3.10 Забезпечення безпеки веб-програми та захист від потенційних загроз

Безпека веб-програми та захист від потенційних загроз є невід'ємною частиною розробки і використання будь-якого веб-застосунку. У проекті реалізовано низку заходів для забезпечення безпеки та захисту від потенційних загроз. Деякі з них включають:

1. Аутентифікація та авторизація:

- використання механізмів аутентифікації для перевірки ідентифікації користувачів;

- встановлення рівнів доступу та авторизації для контролю прав доступу користувачів до функцій та ресурсів.

2. Шифрування даних:

- використання шифрування для захисту інформації, передається між клієнтом та сервером;

- застосування безпечних протоколів шифрування, таких як HTTPS, для забезпечення захисту під час передачі даних.

3. Захист від введення даних:

- валідація та фільтрація введених даних для запобігання атак на введення, таких як SQL-ін'єкції та Cross-Site Scripting (XSS).

4. Захист від CSRF-атак:

- використання механізмів захисту від атак типу Cross-Site Request Forgery (CSRF) для запобігання неконтрольованому виконанню дій від імені автентифікованих користувачів.

5. Моніторинг та журналювання:

- впровадження систем моніторингу та журналювання для вчасного виявлення та реагування на потенційні загрози.

6. Оновлення та захист від відомих ураз:

- регулярні оновлення програмного забезпечення для усунення відомих ураз та використання найновіших патчів безпеки.

7. Обмеження доступу до ресурсів:

- використання принципів обмеження доступу та захисту ресурсів від несанкціонованого використання.

Ці заходи забезпечують комплексний підхід до безпеки веб-програми та дозволяють уникнути багатьох типових загроз, що можуть виникнути під час експлуатації веб-застосунку.

Важливо відзначити, що коректне налаштування Cross-Origin Resource Sharing (CORS) є необхідною складовою забезпечення безпеки веб-програми. CORS визначає політику, яка регулює, як веб-застосунок може спілкуватися з ресурсами на інших доменах. Правильна конфігурація CORS дозволяє обмежити доступ до ресурсів тільки для визначених джерел, що запобігає можливим атакам, таким як Cross-Site Request Forgery (CSRF) та Cross-Site Scripting (XSS).

Забезпечення безпеки веб-програми — це невід'ємна частина розробки, оскільки невірна конфігурація може призвести до серйозних наслідків, таких як

виток конфіденційної інформації, втрата цілісності даних та можливість атак на систему. Врахування вищезазначених аспектів забезпечення безпеки дозволяє створювати надійні та стійкі веб-застосунки, які відповідають сучасним стандартам безпеки та забезпечують захист від різноманітних загроз.

3.11 Управління проектом та задачами

У високотехнологічному світі розробка та управління програмними проектами вимагають вдосконалених методів та стратегій. У цьому контексті різні методології та інструменти для керування процесами стають невід'ємною частиною робочого процесу. Дві з найпоширеніших методологій, які забезпечують ефективне керування розробкою програмного забезпечення, - Scrum і Kanban.

Scrum - це гнучкий методологічний підхід до розробки програмного забезпечення, який базується на принципах прозорості, інспекції та адаптації. В Scrum процес розробки розділений на короткі ітерації, називані спринтами, зазвичай тривалістю від одного до чотирьох тижнів. Кожен спринт завершується випуском потенційно готового продукту. Робочі завдання зазвичай поділяються на короткі робочі періоди, називані скрам-зборами, що сприяє прозорості та ефективному керуванню завданнями. Для роботи по Scrum технології широко використовується інструмент Jira. Нижче можна переглянути декілька процесів в Jira:

1. Створення продуктового backlog - в Jira можна легко створювати та управляти списком функцій та завдань, які повинні бути включені в продуктовий backlog. Кожна задача може мати призначену приоритетність, оцінку складності, та інші важливі характеристики.

2. Планування спринтів - Jira дозволяє команді визначити обсяг робіт на майбутній спринт, вибрати завдання для включення та визначити обсяг робіт.

3. Візуалізація робочого процесу - картки задач в Jira можуть представляти різні стани робочого процесу, такі як "В розробці," "На рецензії," "Готово до тестування," і т.д. Це допомагає команді візуально відстежувати прогрес кожного завдання.

4. Створення звітів - Jira надає різноманітні можливості створення звітів, таких як звіти по швидкості, діаграми виконання, графіки вартості тощо. Ці засоби дозволяють оцінити продуктивність команди та здійснювати узагальнені аналізи.

5. Відстеження багів - коли баг знайдений або виявлений під час тестування, його можна легко відстежувати в Jira, призначати відповідальних та відслідковувати стан виправлення.

6. Планування випусків - Jira дозволяє зручно планувати релізи, визначаючи, які завдання та функції входять в кожен випуск, і надає інструменти для моніторингу прогресу.

Загалом, Jira стає ефективним інструментом для впровадження Scrum-методології, надаючи зручний інтерфейс та широкі можливості конфігурації для потреб команди.

Kanban - це метод управління процесами, який базується на візуальному відображенні потоку роботи. Замість фіксованих ітерацій, як у Scrum, Kanban надає гнучкий підхід до роботи над завданнями, де завдання пересуваються через дошку (зазвичай розглядається як чотири колонки: "Спроектовано", "В процесі", "Перевірено", "Готово") по мірі їхнього виконання. Кожне завдання обробляється і видаляється, що дозволяє враховувати нові задачі, призначати пріоритети та ефективно виконувати роботу. Для роботи по Kanban технології широко використовується інструмент Trello. Нижче можна переглянути декілька процесів в Trello:

1. Створення дошок - Trello надає можливість створювати різні дошки, кожна з яких може представляти проект, завдання або інший вид робочого

потоку. Наприклад, для реалізації Kanban-системи можна створити дошку для кожного проекту чи процесу.

2. Створення списків завдань - у межах кожної дошки можна створювати списки завдань, які відображають етапи робочого процесу. Наприклад, "Не розпочато," "В роботі," "На перевірці," "Завершено.".

3. Створення карток завдань - кожне завдання представляється карткою, на якій можна вказати назву, опис, теги, строки виконання та інші параметри. Картки можна пересувати між списками для відображення прогресу.

4. Призначення та терміни - карткам можна призначати відповідальних, вказувати терміни виконання, що дозволяє ефективно визначати пріоритети та слідкувати за строками.

5. Коментарі та вкладені файли - Trello надає можливість обговорення кожного завдання через коментарі. Також, користувачі можуть долучати файли, що полегшує обмін інформацією.

6. Візуалізація процесу - завдяки практиці Kanban, Trello дозволяє візуально відстежувати потік роботи, сприяючи швидкому виявленню завдань, що потребують уваги.

7. Широкі можливості розширення - Trello підтримує різноманітні розширення та інтеграції, що дозволяє розширити його функціонал і впроваджувати інші інструменти для спрощення роботи. Trello відмінно підходить для впровадження Kanban-методології завдяки своїй простоті та зручності використання, дозволяючи командам ефективно визначати, виконувати та відстежувати завдання в реальному часі.

3.12 Опис процесу інтеграції програмного забезпечення

Процес інтеграції програмного забезпечення та впровадження його в робоче середовище є критичним етапом у розробці та забезпеченні успішного

функціонування програми. Цей процес включає декілька ключових етапів:

1. Деплой на хостинг:

- вибір хостинг-провайдера - обирається хостинг-провайдер, який відповідає вимогам за потужністю, безпекою та іншими факторами;

- створення облікового запису - реєстрація облікового запису на обраному хостингу для отримання доступу до серверних ресурсів;

- завантаження програмного коду - розміщення програмного коду на сервері, що може бути здійснено через FTP, SSH або інші протоколи.

2. Організація сервера:

- встановлення залежностей - установка необхідного програмного забезпечення та залежностей, таких як веб-сервер (наприклад, Apache, Nginx), база даних (наприклад, MySQL, PostgreSQL) та інші;

- конфігурація веб-сервера - налаштування веб-сервера для обробки запитів та направлення їх до програмного забезпечення;

- налаштування бази даних - створення бази даних та конфігурація для ефективної взаємодії з програмою.

3. Підготовка робочого середовища:

- виконання Міграцій (за необхідності - запуск міграцій для створення таблиць та необхідних структур у базі даних;

- налаштування оточення - встановлення конфігураційних параметрів, таких як змінні середовища, шляхи до ресурсів, ключі доступу тощо;

- тестування - проведення тестів для перевірки функціональності та виявлення можливих проблем.

4. Запуск:

- запуск застосунку - запуск програмного забезпечення для перевірки його працездатності в реальному середовищі.

Ці етапи дозволяють забезпечити ефективну інтеграцію програмного

забезпечення та його успішне впровадження в робоче середовище. Ретельне тестування та моніторинг важливі для забезпечення стабільності та безпеки роботи застосунку.

3.13 Плани для моніторингу веб-програми після впровадження

Плани для моніторингу та підтримки веб-програми після впровадження є важливою частиною життєвого циклу розробки. Ось деякі ключові аспекти цих планів:

1. Моніторинг продуктивності:

- моніторинг використання ресурсів - слід встановити систему моніторингу, яка відстежує використання ресурсів сервера (рис. 3.31), таких як CPU, пам'ять, дисковий простір;
- профілювання запитів - моніторинг продуктивності бази даних (рис. 4.31) для виявлення повільних запитів та оптимізації їх.

2. Забезпечення безпеки:

- регулярні оновлення - план оновлень для всіх компонентів програми, включаючи фреймворки, бібліотеки та серверне забезпечення;
- моніторинг уразливостей - використання інструментів для автоматичного виявлення та виправлення потенційних уразливостей.

3. Логування та аналіз помилок:

- система логування - реалізація системи логування, яка реєструє події та помилки для подальшого аналізу;
- моніторинг помилок - використання інструментів для моніторингу та аналізу логів для оперативного виявлення та вирішення проблем.

4. Технічна підтримка:

- система заявок - підтримка системи заявок для відстеження та вирішення технічних питань користувачів;
- план резервного копіювання - регулярне створення та перевірка

резервних копій для запобігання втраті даних.

5. Оновлення та розвиток:

- план оновлень функцій - розробка плану оновлень для додавання нових функцій та покращень;
- збереження сумісності - випробування та збереження сумісності з новими версіями браузерів та інших залежностей.

6. Моніторинг безпеки зв'язку (CORS):

- конфігурація CORS - забезпечення правильної конфігурації CORS для управління безпекою взаємодії між різними джерелами на веб-сторінці;
- моніторинг запитів - відстеження та аналіз HTTP-запитів для виявлення потенційних проблем з безпекою.

Ці пункти складають комплексний план моніторингу та підтримки, спрямований на забезпечення ефективної та безпечної роботи веб-програми після впровадження. Після впровадження веб-програми, важливо продовжувати моніторити її продуктивність та забезпечувати безпеку відповідно до визначених планів. Основний акцент повинен бути зроблений на постійному вдосконаленні системи та її функціоналу, а також на оперативній реакції на виникаючі проблеми. Додатково, важливо підтримувати зв'язок з користувачами, забезпечуючи їхній доступ до технічної підтримки та інформаційних ресурсів.

Плани для моніторингу та підтримки веб-програми після впровадження є важливою частиною життєвого циклу розробки. Моніторинг продуктивності включає в себе встановлення системи моніторингу для відстеження використання ресурсів сервера, таких як CPU, пам'ять, дисковий простір, а також профілювання запитів для оптимізації їхньої продуктивності. Забезпечення безпеки передбачає регулярні оновлення всіх компонентів програми та моніторинг уразливостей для автоматичного виявлення та виправлення потенційних проблем.

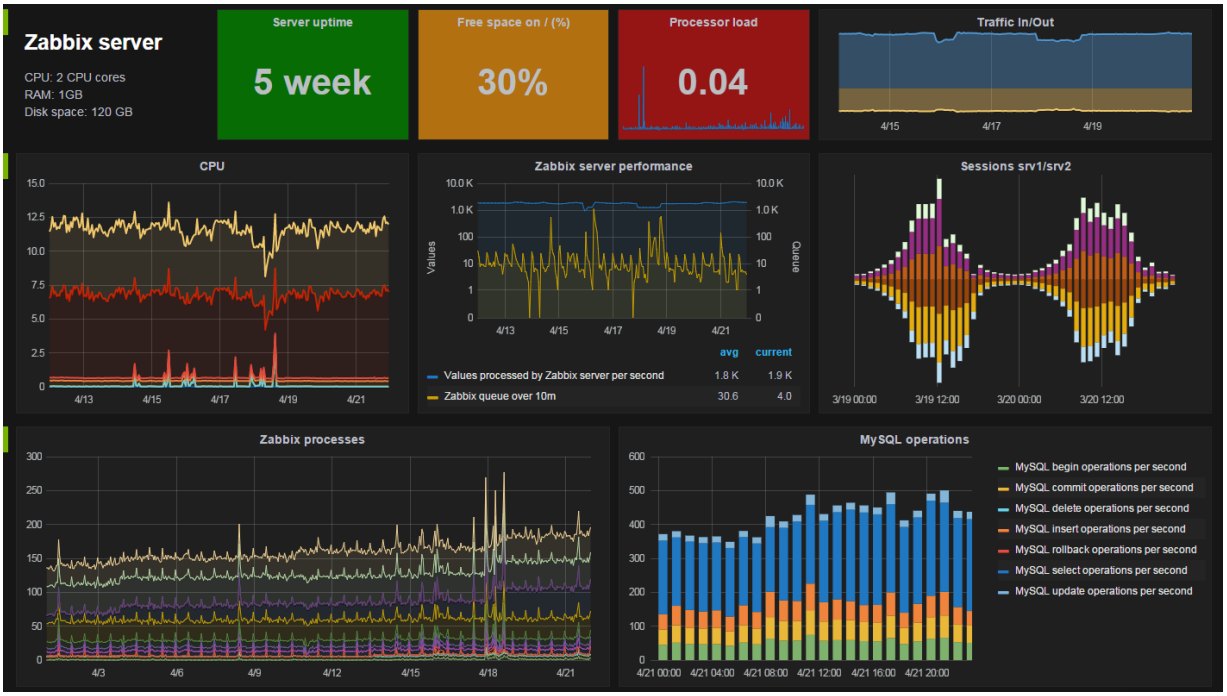


Рисунок 3.31 – Приклад моніторингу сервера

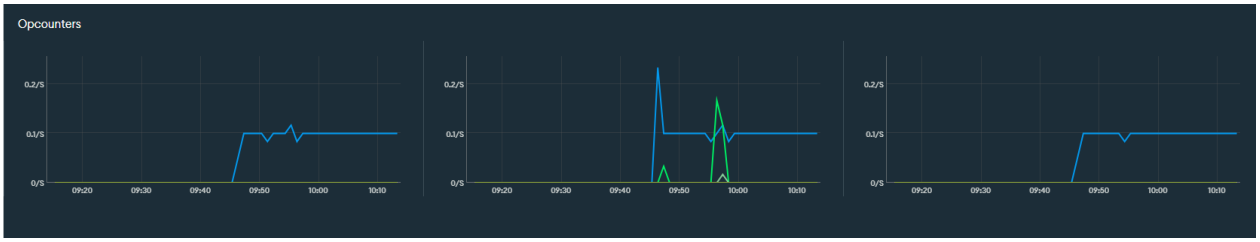


Рисунок 3.32 – Моніторинг ресурів бази даних MongoDB

3.14 Аналіз продуктивності. Оцінка ефективності веб-програми

При розгляді продуктивності веб-програми обов'язково звертаємо увагу на кілька ключових аспектів, що визначають її ефективність. По-перше, швидкість завантаження сторінок для користувача на різних пристроях та в різних регіонах є важливим показником. Важливо мінімізувати час завантаження, використовуючи стратегії кешування та оптимізацію запитів.

Другий аспект — ефективне використання ресурсів, таких як процесор, оперативна пам'ять та мережевий трафік. Оптимізація коду та ресурсів дозволяє зменшити використання системних ресурсів та підтримує стабільну роботу

реалізованої веб-програми.

Третій аспект — час відповіді на HTTP-запити та реакція сервера на різні види запитів. Оптимізація операційного часу сервера та бази даних покращує загальний досвід користувача.

Додатково, важливо враховувати оптимізацію зображень та інших ресурсів, використання стратегій кешування та моніторинг користувацького досвіду для покращення взаємодії з інтерфейсом.

Ще одним кроком у забезпеченні продуктивності є проведення тестів навантаження для визначення максимальних можливостей програми при великому обсязі запитів та користувачів. Такий підхід дозволяє виявити можливі буттівлі та забезпечити оптимальну роботу в різних умовах.

Після впровадження веб-програми в робоче середовище важливо систематично виконувати моніторинг та підтримку. Моніторинг здійснюється для виявлення можливих аномалій, помилок чи проблем з продуктивністю. Це може включати в себе відслідковування журналів помилок, аналіз логів використання ресурсів сервера, та моніторинг доступності веб-сайту.

Підтримка передбачає вчасне виправлення виявлених проблем, вдосконалення функціоналу та додавання нових можливостей.

Додатково, рекомендується проводити аудит безпеки періодично, а також вдосконалювати заходи захисту від потенційних загроз.

3.15 Аналіз користувацького досвіду (UX) та ітеративне поліпшення інтерфейсу

Після впровадження веб-програми в робоче середовище важливо систематично аналізувати користувацький досвід (UX) для забезпечення задоволення користувачів та оптимізації ефективності використання продукту. Аналіз UX включає в себе спостереження за тим, як користувачі взаємодіють з програмою, збір зворотного зв'язку, тестування взаємодії та оцінку загального

враження від використання веб-застосунку.

Ітеративне поліпшення інтерфейсу означає постійне вдосконалення дизайну та функціоналу з урахуванням виявлених слабких місць. Це може включати в себе зміни в розташуванні елементів, додавання нових функцій або вдосконалення існуючих. Такий підхід дозволяє підтримувати веб-програму актуальною та відповідати змінюючимся потребам користувачів.

Зокрема, рекомендується використовувати зворотний зв'язок від користувачів, проводити тестування взаємодії з різними аудиторіями, виявляти та усувати проблеми доступності та вдосконалювати елементи дизайну для оптимальної зручності використання.

Всі ці заходи спрямовані на постійне покращення користувацького досвіду та забезпечення того, що веб-програма відповідає очікуванням та потребам користувачів.

Крім того, важливо вивчати дані щодо продуктивності веб-програми, зокрема її швидкості та ресурсозбереження. Моніторинг продуктивності дозволяє вчасно виявляти можливі проблеми та оптимізувати роботу програми. Використання аналітики та інструментів моніторингу допомагає визначити обсяг використовуваних ресурсів, час завантаження сторінок та інші параметри продуктивності.

Зокрема, аналіз ефективності може включати в себе оцінку швидкості завантаження, відгуків користувачів щодо продуктивності та реакції на різні дії. Це допомагає виявляти можливі обмеження та робити налагодження для забезпечення оптимального функціонування.

Плани для моніторингу та підтримки включають систематичне вдосконалення та виправлення помилок, а також реагування на нові вимоги та тенденції в індустрії веб-розробки. Постійне оновлення та оптимізація є ключем до забезпечення довгострокової стабільності та розвитку веб-програми.

З аналізом користувацького досвіду (UX) пов'язане постійне ітеративне поліпшення інтерфейсу. Збір фідбеку від користувачів, аналіз їхньої взаємодії з

програмою та реакції на різні функціональні можливості є важливим етапом у розробці. Відповідно до зібраної інформації вносяться зміни та доповнення, спрямовані на покращення користувацького досвіду.

Ітеративний підхід передбачає послідовні цикли розробки, тестування та вдосконалення, де кожна ітерація додає новий функціонал або оптимізує існуючий. Це може включати зміни у дизайні, розширення можливостей, усунення недоліків чи відповідь на нові вимоги користувачів.

Одним із способів отримання фідбеку та визначення слабких місць є проведення тестувань з реальними користувачами та аналіз їхніх реакцій. Це сприяє виявленню проблем, які можливо не враховані на етапі розробки та дозволяє здійснювати точне вирішення вимог користувачів.

В цілому, поєднання безпеки, продуктивності та вдосконалення користувацького досвіду створює надійну та ефективну веб-програму, готову задовольняти потреби користувачів і відповідати високим стандартам веб-розробки.

Для забезпечення високої ефективності веб-програми важливо проводити аналіз продуктивності. Моніторинг швидкості відгуку системи, оптимізація використання ресурсів та визначення можливих проблем допомагають у підтримці стабільної роботи програми. Крім того, оцінка продуктивності дозволяє виявити можливості для вдосконалення та оптимізації роботи програми.

Важливим етапом у впровадженні веб-програми є інтеграція програмного забезпечення в робоче середовище. Деплой на хостинг, організація сервера та налагодження взаємодії з іншими складовими системи допомагають забезпечити готовність програми до активного використання. Цей процес включає в себе кроки забезпечення безпеки, інтеграцію з базами даних та іншими сервісами, а також тестування для визначення правильності роботи всіх компонентів.

Після впровадження, моніторинг та підтримка грають ключову роль у

забезпеченні стабільності та доступності веб-програми. Реагування на можливі проблеми, вдосконалення функціоналу, а також врахування фідбеку користувачів визначають успішність та довгостроковий успіх програмного продукту.

З огляду на дизайн лендінгу (рис. 3.33) можливі декілька шляхів для його покращення. Зокрема, можна звернутися до таких аспектів:

1. Кольорова палітра та контрастність. Важливо вибрати гармонійні кольори, які відповідають бренду та забезпечують і гарантують достатню контрастність для зручного читання.

2. Типографіка. Звернути увагу на шрифти, розміри, колір, відступи тексту та його розташування.

3. Графічні елементи. Потрібно вставити якісні графічні елементи, такі як зображення, ілюстрації або відео, що відображають і демонструють сутність продукту чи послуги.

4. Впорядкування інформації. Організувати контент логічно, розмістіть важливі елементи у видимих місцях. Використати структуровані блоки (сектори) тексту та заголовки.

5. Адаптивний дизайн. Забезпечити, щоб лендінг був адаптивним для різних пристроїв. Переконатися, що він добре виглядає на мобільних пристроях, планшетах та настільних комп'ютерах.

6. Використання викликів до дії (Call-to-Action). Розмістити чіткі виклики до дії, щоб відвідувачі могли легко здійснити необхідні кроки (наприклад, реєстрація, покупка тощо).

7. Тестування користувацького досвіду. Провести тестування з використанням фокус-груп або отримати фідбек від реальних користувачів для визначення слабких місць та можливостей поліпшення.

Здійснення зазначених покращень допоможе не лише поліпшити естетику лендінгу, але й підвищить його ефективність у привертанні та утриманні відвідувачів.

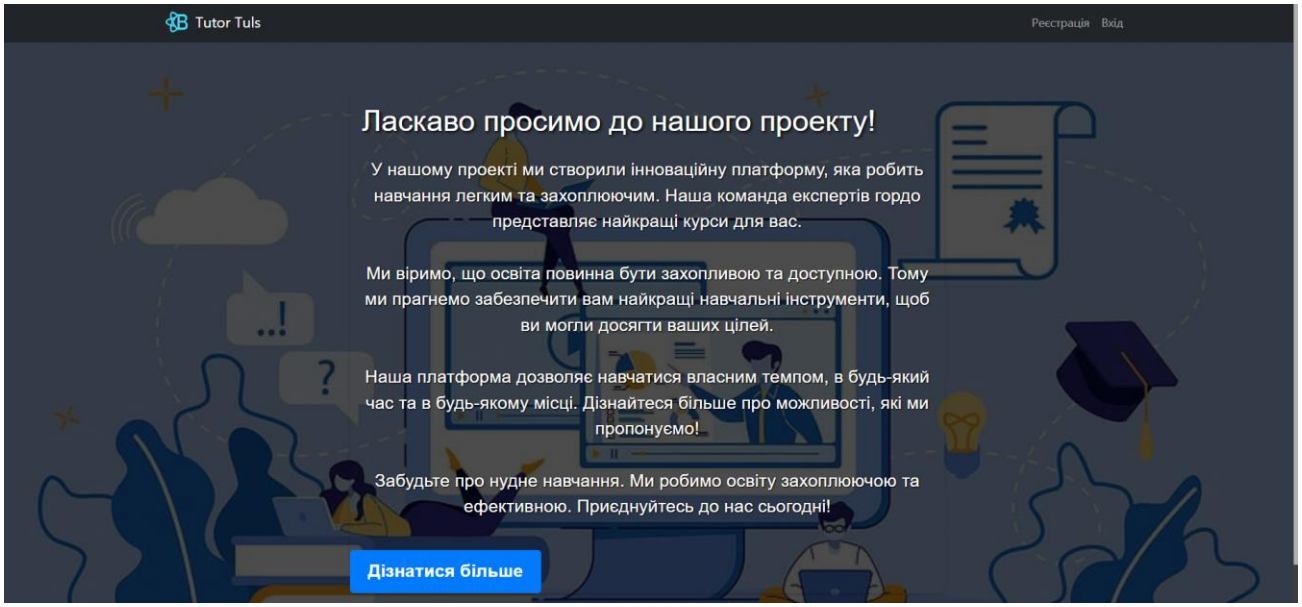


Рисунок 3.33 – Лендінг

Після аналізу фідбеку користувачів, досліджень та випуску мінімально життєздатного продукту (MVP) можна визначити певні аспекти UX, які можна покращити:

1. Взаємодія інтерфейсу. Переконатися, що інтерфейс є інтуїтивно зрозумілим і зручним у використанні. Враховувати отримані відгуки щодо навігації та доступності основних функцій.

2. Швидкість та продуктивність. Визначити можливості для оптимізації швидкості завантаження та відгуків системи. Зменшення затримок і покращення загальної продуктивності позитивно вплине на задоволення користувачів.

3. Відповідь на пропозиції користувачів. Якщо юзеру висловлюють конкретні пропозиції або побажання, розглянути їхню реалізацію. Це може включати додавання нових функцій, поліпшення інтерфейсу чи виправлення можливих недоліків.

4. Адаптивний дизайн. Впевнитися, що продукт добре пристосовується до різних пристроїв і розмірів екранів. Забезпечити, щоб користувачі могли легко взаємодіяти з продуктом на мобільних телефонах, планшетах та настільних комп'ютерах.

5. Забезпечення конфіденційності і безпеки. Якщо користувачі виносять питання про конфіденційність і безпеку, забезпечити ясне розуміння того, як їхні дані зберігаються та обробляються.

6. Виклики до дії. Враховувати фідбек щодо викликів до дії на сайті. Якщо користувачі нечітко розуміють, як взаємодіяти чи як вони можуть скористатися послугою, розглянути вдосконалення цих аспектів.

7. Моніторинг великих навантажень. Якщо кількість користувачів росте, переконатися, що система може витримати великі навантаження. Оптимізувати серверну частину для забезпечення стабільної роботи при збільшенні трафіку.

Проведення аналізу фідбеку та постійне вдосконалення UX - ключові кроки у розвитку продукту, що дозволяють досягти високої задоволеності користувачів і підтримувати конкурентоспроможність.

3.16 Розгляд сумісності з різними браузерами та пристроями

Забезпечення сумісності з різними браузерами та пристроями, а також можливості масштабування програми є важливими аспектами в розробці веб-програмного забезпечення.

Щоб забезпечити сумісність програма була оптимізована для браузерів, таких як Google Chrome, Mozilla Firefox, Safari, Microsoft Edge тощо. Це включає перевірку коректності відображення, правильне виконання функцій та відсутність помилок на кожному браузері. Важливо враховувати технічні відмінності між браузерами та застосовувати стандарти, які підтримуються у всіх веб-переглядачах.

Щодо пристроїв, важливо враховувати різницю у розмірах екранів, операційних системах та можливостях введення (наприклад, тачскрін). Респонсивний дизайн дозволяє адаптувати інтерфейс до різних розмірів, забезпечуючи зручну взаємодію на мобільних телефонах (рис 3.35), планшетах

(рис.3.34) та настільних комп'ютерах.

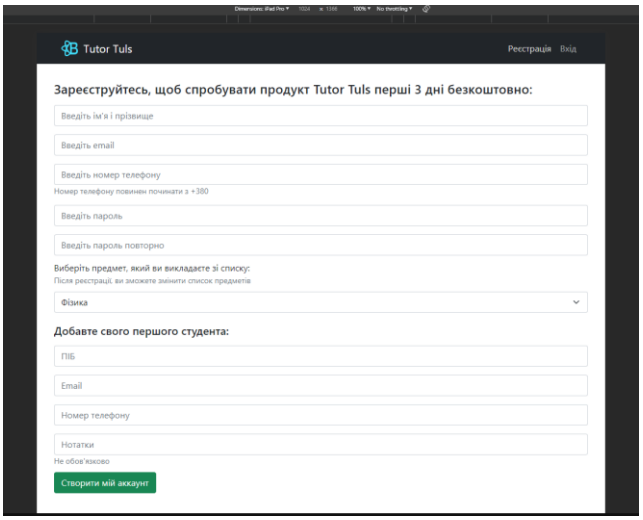


Рисунок 3.34 – Відображення сторінки реєстрації на IPad Pro

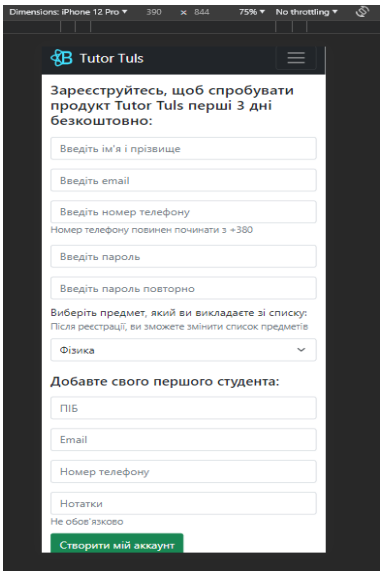


Рисунок 3.35 - Відображення сторінки реєстрації на IPhone 12 Pro

Масштабованість програми стосується її здатності збільшувати обсяг і функціональність з ростом числа користувачів та обсягу даних. Для цього важливо правильно організувати серверну інфраструктуру, використовувати ефективні бази даних, кешування і т.д. Це забезпечить стабільну роботу продукту навіть при великих навантаженнях.

Одним із рішень є використання облачних сервісів, які дозволяють легко

масштабувати ресурси в залежності від потреб проекту. Також слід враховувати можливість оптимізації фронтенду та бекенду для ефективної роботи на різних конфігураціях серверів.

Усі ці аспекти взаємодіють для створення продукту, який надає спільне враження для різних категорій користувачів та здатний зростати разом із збільшенням його популярності та обсягів.

Аналіз продуктивності є критичним етапом для забезпечення ефективності та стабільності веб-програми. Важливо вимірювати швидкість завантаження сторінок, час відгуку та використання ресурсів для забезпечення позитивного користувацького досвіду.

Використання інструментів аналізу продуктивності, таких як Google Lighthouse чи PageSpeed Insights, дозволяє отримати інформацію щодо різних аспектів продуктивності, включаючи оптимізацію зображень, кешування ресурсів, компресію файлів та інше.

Оцінка ресурсозбереження включає в себе вивчення використання пам'яті, процесорного часу та інших ресурсів під час роботи веб-застосунку. Оптимізація цих аспектів допомагає покращити продуктивність, зменшити завантаження сервера та забезпечити більш якісний сервіс для користувачів.

Для ефективного моніторингу та підтримки веб-програми після впровадження, команда розробників повинна використовувати системи моніторингу, які надають інформацію про стан серверів, відслідковують помилки та сповіщають про них. Крім того, слід впроваджувати системи резервного копіювання даних для запобігання втраті інформації.

Важливим елементом є також швидке реагування на фідбек користувачів після MVP (мінімально життєздатний продукт). Вивчення думок і зауважень користувачів та впровадження поліпшень в інтерфейсі та функціоналі дозволяє підтримувати високий рівень задоволеності користувачів.

Таким чином, взаємодія всіх цих аспектів в плануванні, розробці та підтримці веб-програмного забезпечення забезпечує високий ступінь

ефективності, надійності та задоволення користувачів.

3.17 Обгрунтування можливих напрямків для подальшого розвитку

Можливі напрямки для подальшого розвитку та покращень веб-програми можуть бути орієнтовані на декілька ключових аспектів:

1. Додавання нових функцій, які розширять можливості веб-програми та зроблять її більш універсальною для різних категорій користувачів.

2. Вдосконалення алгоритмів та оптимізація коду для зменшення використання ресурсів та оптимізації фінансових витрат на інфраструктуру.

3. Вдосконалення системи безпеки для захисту від нових загроз, врахування вимог щодо захисту персональних даних та дотримання стандартів безпеки.

4. Введення можливостей інтеграції з популярними зовнішніми сервісами та платформами для розширення функціональних можливостей.

5. Проведення аналізу поведінки користувачів для виявлення найбільш та найменш використовуваних функцій та оптимізація їх для покращення використання.

Ці напрямки можуть бути обрані в залежності від конкретних потреб користувачів, стратегії бізнесу та обмежень ресурсів. Розвиток веб-програми повинен бути орієнтованим на забезпечення високої якості обслуговування та відповіді на зростаючі потреби користувачів.

Впровадження зазначених стратегій є критичним для невперервної еволюції та успіху веб-програми. Забезпечуючи високий стандарт якості, ми створюємо надійну та зручну для використання платформу, яка відповідає потребам наших користувачів.

Стабільність та безпека є основоположними аспектами довгострокового функціонування веб-програми. Акцент на зменшенні ризиків інцидентів та впровадженні передових заходів безпеки дозволяє нам надійно захищати дані

користувачів і забезпечувати стійкість до потенційних загроз.

Ефективне використання ресурсів і оптимізація швидкості завантаження сторінок грають важливу роль у забезпеченні відмінного користувацького досвіду. Масштабованість і гнучкість технічної інфраструктури дозволяють легко адаптуватися до зростання обсягу користувачів та змін до програми.

Відкритість для фідбеку користувачів та їхніх потреб відображається у взаємодії з комуніті, спрямованій на активне обговорення та впровадження покращень. Це створює позитивний круговорот інновацій, що сприяє зростанню та розвитку.

Зважаючи на динаміку веб-ринку, наші напрямки розвитку спрямовані на забезпечення конкурентоспроможності та вдосконалення функціональності веб-програми. Додаткові можливості для користувачів, такі як розширення функціоналу та підтримка нових технологій, викликають попит і підтримують інтерес до продукту.

Впровадження ініціатив по розширенню мобільної сумісності та підтримки різних браузерів стає ключовим елементом нашої стратегії. Це дозволяє нашим користувачам безперешкодно використовувати програму на будь-яких пристроях та з будь-яких браузерів, що підвищує загальну доступність та комфорт використання.

Планування для майбутнього також включає оцінку та удосконалення продуктивності, зосередження на оптимізації завантаження сторінок та ефективному використанні ресурсів. Це спрямовано на забезпечення найвищого рівня задоволення від використання програми та виконання потреб користувачів.

Все це відбувається в контексті постійного аналізу користувацького досвіду та вдосконалення інтерфейсу з урахуванням відгуків користувачів. Процес ітеративного вдосконалення UX веде до створення ще зручнішого та привабливішого середовища для користувачів.

Узагальнюючи, найважливішим є постійне вдосконалення та адаптація до

нових викликів у світі веб-технологій. Шлях до успіху лежить в поєднанні технічної експертизи, уважного вислуховування користувачів та оперативного реагування на зміни в галузі.

3.18 Способи отримання фідбеку від користувачів

Отримання фідбеку від користувачів включає в себе кілька ефективних стратегій, спрямованих на забезпечення відкритого та діалогового взаємодії з аудиторією. Серед найважливіших способів:

1. Форми зворотнього зв'язку на веб-сайті. Розміщення форм зворотнього зв'язку або опитувань на веб-сайті дозволяє користувачам висловлювати свої думки, зауваження та пропозиції безпосередньо. Важливо забезпечити зручний інтерфейс для заповнення та надати можливість анонімності для тих, хто це бажає.

2. Соціальні мережі. Активна присутність у соціальних мережах дозволяє збирати фідбек через коментарі, приватні повідомлення та опитування. Важливо систематично слідкувати за згадками вашої програми та відповідати на питання чи зауваження користувачів.

3. Електронна пошта. Надання можливості користувачам надсилати свої думки через електронну пошту сприяє приватності та може призвести до більш детального обговорення. Важливо відповідати на кожен отриманий лист та надавати конкретні відповіді.

4. Аналітика використання. Використання аналітичних інструментів дозволяє відстежувати поведінку користувачів на веб-сайті та дізнатися, як вони взаємодіють з програмою. Це надає об'єктивні дані, які можуть бути використані для вдосконалення користувацького досвіду.

Щоб впроваджувати фідбек користувачів, важливо:

- систематично аналізувати фідбек - регулярно переглядати та аналізувати отриманий фідбек, визначати ключові тенденції та проблеми;

- впроваджувати поліпшення - відкрито комунікувати з користувачами щодо впровадження їхніх ідей та корекції виявлених недоліків;

- інформувати про зміни - у випадку впровадження конкретних змін на основі фідбеку, важливо інформувати користувачів та висловлювати вдячність за їхні внески.

Intercom (рис. 3.36) та Zendesk (рис. 3.37) є платформами для надання служби підтримки та взаємодії з користувачами, але вони мають деякі відмінності, які роблять їх специфічними для різних використань та потреб бізнесів.

Intercom:

1. Живий чат. Одна з ключових особливостей Intercom - це можливість спілкуватися з користувачами в реальному часі через чат на веб-сайті.

2. Система повідомлень. Дозволяє відправляти автоматизовані повідомлення користувачам на основі їхньої активності або інших умов.

3. CRM-функції. Є можливості для управління відносинами з клієнтами, збору даних та аналізу користувацької активності.

4. Продажі та маркетинг. Додаткові інструменти для автоматизації процесів продажів та маркетингу.

Zendesk:

1. Широкий функціонал. Надає широкий спектр інструментів для служби підтримки, включаючи тікет-систему, базу знань, чат та інші.

2. Множина каналів підтримки. Zendesk підтримує обробку запитів з різних каналів, таких як електронна пошта, чат, соціальні мережі.

3. Багаторівнева система тікетів. Велика підтримка для управління різними типами та пріоритетами тікетів.

4. Аналітика та звіти. Надає інструменти для вимірювання ефективності служби підтримки та аналізу задоволеності клієнтів.

Вибір між Intercom та Zendesk залежить від конкретних потреб бізнесу. Intercom може бути більш відповідним для тих, хто акцентує на прямому спілкуванні з користувачами та маркетингових можливостях.

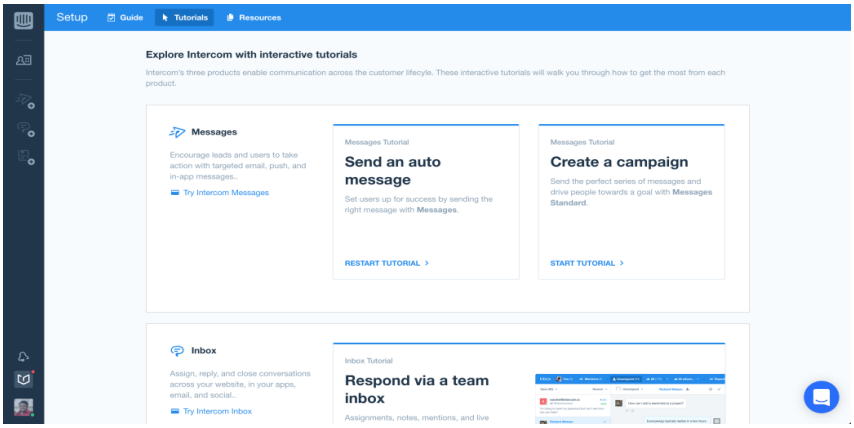


Рисунок 3.36 – Інтерфейс Intercom

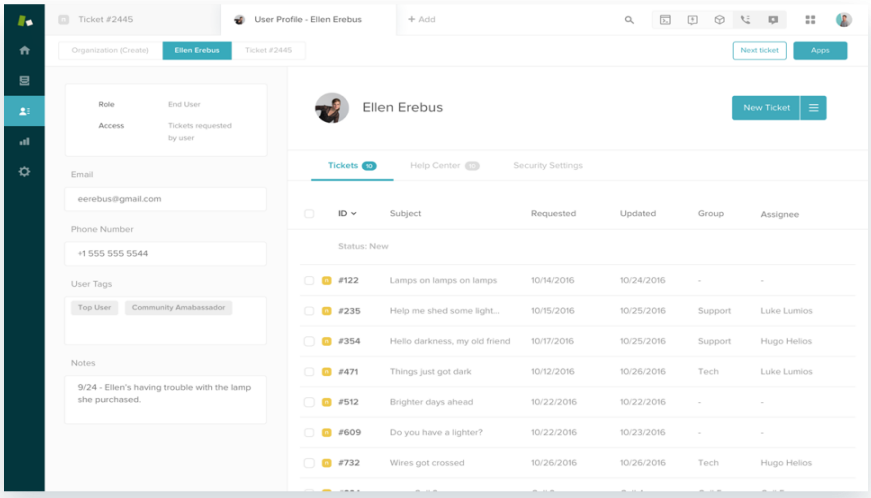


Рисунок 3.37 – Інтерфейс Zendesk

У той час як Zendesk частіше використовується для вирішення проблем підтримки клієнтів та управління зверненнями. Важливо підтримувати інтеграцію між цими платформами для забезпечення ефективного спілкування та спільної роботи команди.

3.19 Виконання вимог законодавства щодо захисту даних

Виконання вимог законодавства та політики щодо захисту даних та приватності користувачів є важливою частиною розробки та управління веб-програмою.

Вимоги законодавства та політики:

1. Загальний регламент з охорони даних (GDPR). Відповідність цього регламенту передбачає введення механізмів для забезпечення доступу користувачів до їхніх особистих даних та можливості контролю над ними. Це включає право на забвання, підтримку точності даних і механізми перенесення. Зобов'язуємося не передавати особисту інформацію третім сторонам без явної згоди користувача, забезпечуючи високий ступінь конфіденційності.

2. Конфіденційність та нерозголошення. Політика конфіденційності визначає правила обробки та зберігання даних, забезпечуючи їх безпеку від доступу третіх сторін. Всі зусилля спрямовані на збереження конфіденційності особистої інформації. Жодна інформація не розголошується без належного повідомлення та згоди.

3. Безпека даних. Застосування шифрування для захисту конфіденційної інформації в процесі передачі та зберігання. Ми використовуємо сучасні технології та методи, щоб гарантувати безпеку даних від несанкціонованого доступу чи втрати. Забезпечення безпеки даних - наш пріоритет.

4. Повідомлення про порушення безпеки. Забезпечення оперативного та якісного вирішення можливих порушень безпеки. Це включає реалізацію процедур, які надають швидке та точне повідомлення про будь-які можливі загрози безпеці даних. Транспарентність та оперативність - ключові аспекти управління безпекою.

5. Згода користувача. Збір та документування явної згоди користувачів на обробку їхніх даних. Ми ретельно враховуємо вимоги до згоди, щоб забезпечити правову легітимність обробки даних. Згода отримується перед

збором будь-якої особистої інформації, зробивши процес прозорим та зрозумілим для користувачів.

Реалізація заходів:

1. Введення згоди через інтерфейс. У користувальницькому інтерфейсі реалізовано чіткий та доступний механізм для збору згоди на обробку особистих даних. Користувачі отримують інформацію про цілі та обсяг обробки даних перед тим, як давати свою згоду.

2. Шифрування даних у передачі. Для забезпечення конфіденційності введених користувачем даних ми використовуємо сучасні алгоритми шифрування під час їхньої передачі через мережу.

3. Захист даних на сервері. Всі отримані дані зберігаються на серверах з високим рівнем безпеки. Використання механізмів шифрування та регулярне оновлення системи забезпечують найвищий стандарт безпеки.

4. Автоматизована система виявлення порушень. Реалізована система автоматичного виявлення та реагування на можливі порушення безпеки. Заходи передбачають ідентифікацію та ліквідацію проблем на ранніх етапах для запобігання серйозним наслідкам.

5. Система відповідального ведення журналу. Щоденний журнал подій і дій, пов'язаних з безпекою даних, ведеться автоматично. Це дозволяє оперативно реагувати на будь-які аномалії та забезпечує повну прозорість управління безпекою.

3.20 Розробка тестових сценаріїв та їх виконання

Розробка тестових сценаріїв та їх виконання — це критичний етап в процесі розробки веб-програми, який спрямований на перевірку її функціональності, стабільності та відповідності вимогам. Розробка тестових сценаріїв включає в себе створення детальних планів тестування для різних компонентів програми, враховуючи сценарії взаємодії з користувачем.

Виконання тестів включає в себе активне тестування кожного елемента програми на відповідність визначеним критеріям. Це включає у себе як ручне тестування, так і використання автоматизованих тестів для ефективної перевірки функцій.

Цей процес є важливою частиною розробки, оскільки він дозволяє виявити та виправити помилки та дефекти до впровадження продукту в реальне середовище. Послідовне та завершене виконання тестів гарантує високу якість програмного забезпечення та задоволення користувачів високопродуктивним та стабільним продуктом.

Під час тестування веб-програми виявив ряд багів, які були успішно виправлені для покращення якості та стабільності продукту. Один із багів виявився у функціоналі зміни особистих даних, де відсутня була перевірка на коректність введених даних, що призводило до некоректного відображення інформації. Для виправлення цього багу було внесено необхідні перевірки та валідацію даних перед збереженням.

Ще однією проблемою була пов'язана з відображенням списку уроків, де при великій кількості записів частина їх не відображалася коректно. Цю проблему було вирішено оптимізацією виведення списку та покращенням алгоритмів відображення.

Також виявлено та виправлено баг, пов'язаний із відправкою запитів на сервер, де в окремих випадках вони можливо були втрачені через невірне управління асинхронними запитами. Це було виправлено шляхом додаткового контролю за асинхронними викликами та усуненням можливих точок втрати даних.

Всі ці виправлення спрямовані на покращення загального функціоналу та взаємодії з користувачем, забезпечуючи стабільну та ефективну роботу веб-програми.

3.21 Вивчення та інтеграція нових технологій

Під час вивчення та інтеграції нових технологій веб-програми було приділено увагу особливо важливим аспектам, таким як обробка запитів, середовище виконання на стороні клієнта та сервера, а також управління дозволами для запитів з різних джерел.

Була використана бібліотека axios для взаємодії з сервером. Вона надає зручний та ефективний інтерфейс для виконання HTTP-запитів та обробки відповідей. Це дозволило оптимізувати обмін даними між клієнтом та сервером, забезпечуючи швидкість та надійність взаємодії.

Middleware використовувалося для обробки запитів на сервері перед тим, як вони досягають обробника запиту. Це дозволяє виконувати певні операції чи перевірки на етапі обробки запиту перед тим, як сервер відповідає на нього. Такий підхід сприяє покращенню безпеки, ефективності та обробки помилок.

Крім того, важливою частиною інтеграції була робота з CORS (Cross-Origin Resource Sharing), що забезпечує безпечний обмін ресурсами між веб-сайтами, розташованими на різних доменах. Правильна конфігурація CORS дозволяє обробляти запити з інших джерел та забезпечує високий рівень безпеки в обміні даними між клієнтом та сервером.

3.22 Розробка резервних планів та стратегій відновлення

Процес розробки резервних планів та стратегій відновлення веб-програми передбачав створення системи, що забезпечує найвищий рівень надійності та доступності.

В першу чергу були визначені критичні для функціонування програми дані та компоненти. Забезпечено регулярні резервні копії баз даних та інших важливих ресурсів, щоб у разі необхідності можна було відновити весь функціонал.

При розробці стратегій відновлення враховувалися можливі сценарії аварійного відновлення, такі як втрата даних, відмова серверів або інші критичні проблеми. Визначалися терміни відновлення та максимальні допустимі втрати даних.

Важливим елементом стала імплементація моніторингу та системи автоматичного виявлення проблем, що дозволяє вчасно виявляти потенційні проблеми та вживати заходів для їх вирішення.

Усі резервні плани та стратегії відновлення періодично перевірялися та оновлювалися відповідно до змін в інфраструктурі програми та її вимог.

Git відіграє ключову роль у розробці та управлінні вихідним кодом веб-програми. Система контролю версій Git дозволяє ведення історії змін в проекті та спрощує співпрацю розробників.

Кожен розробник може створювати власні гілки для роботи над конкретними функціями чи виправленнями безпеки. Зміни можна об'єднувати та вирішувати конфлікти при необхідності. Git також надає можливість повертатися до попередніх версій коду у випадку необхідності.

Колективна робота над проектом стає більш ефективною завдяки можливості взаємодії з віддаленими репозиторіями, такими як GitHub чи GitLab. Це спрощує обмін змінами між розробниками, рецензування коду та відслідковування проблем.

3.23 Оцінка впливу на навколишнє середовище

Оцінка впливу на навколишнє середовище включає в себе оцінку різних аспектів, щоб забезпечити екологічну сталість діяльності. На першому етапі визначаються фактори, пов'язані із споживанням ресурсів, включаючи електроенергію для серверів та інфраструктури, обсяги викидів CO2 під час розробки та експлуатації, а також управління відходами.

Після ідентифікації цих факторів впливу проводиться оцінка

ефективності та приймаються стратегії для зменшення екологічного відбитку. Це може включати перехід до використання відновлюваних джерел енергії, оптимізацію алгоритмів для зменшення споживання ресурсів та управління відходами.

Забезпечення екологічної сталості веб-програми стає не тільки актом соціальної відповідальності, але й допомагає забезпечити довгострокову ефективність проекту в умовах зростаючої уваги до екологічних питань у сучасному світі.

Додатково, впровадження ефективних стратегій для зменшення впливу на навколишнє середовище дозволяє не лише знизити витрати на енергопостачання та ресурси, але і покращити імідж проекту, роблячи його привабливішим для екологічно свідомих користувачів та інвесторів.

Окрім того, періодична оцінка впливу на навколишнє середовище може служити основою для подальшого вдосконалення стратегій, адаптації до нових технологій та вимог, що сприяє динамічному розвитку екологічних ініціатив.

Загалом, здійснення оцінки впливу на навколишнє середовище є важливим етапом для будь-якого проекту, орієнтованого на сталість та довгостроковий успіх.

3.24 Розробка стратегії масштабування та росту

Стратегія масштабування та росту важлива для забезпечення ефективного розвитку та адаптації веб-програми до зростаючих потреб користувачів та ринку. Розробка такої стратегії включає в себе кілька ключових етапів:

1. Аналіз інфраструктури. Здійснюється докладний аналіз інфраструктури проекту, визначаються її можливості та обмеження. Враховуються параметри масштабування, такі як обсяг трафіку, обробка даних та пропускна здатність серверів.

2. Вибір технологій масштабування. Обираються технології, які дозволяють ефективно масштабувати систему. Це може включати в себе використання облачних сервісів, контейнеризацію та інші техніки.

3. Горизонтальне та вертикальне масштабування. Розробляється стратегія для обрання між горизонтальним та вертикальним масштабуванням. Горизонтальне зазвичай полягає в додаванні нових екземплярів серверів, тоді як вертикальне — у збільшенні ресурсів і потужності існуючих серверів.

4. Запасна інфраструктура. Враховуються можливості резервного копіювання та запасної інфраструктури для забезпечення безперебійної роботи системи під час масштабування.

5. Стратегії росту користувачів. Розробляються стратегії для збільшення користувачської бази, включаючи маркетингові та рекламні кампанії, партнерства та розширення функціоналу для привертання нових користувачів.

6. Моніторинг та оптимізація. Встановлюється система моніторингу для постійного відстеження продуктивності та взаємодії з користувачами. Оптимізуються процеси для забезпечення ефективності та швидкості системи.

7. Підтримка росту команди. Розглядається стратегія збільшення команди розробників та підтримки для ефективної роботи над новими функціями та масштабуванням.

Стратегія масштабування та росту є необхідною для забезпечення довгострокового успіху веб-програми в умовах змінного ринкового середовища та росту конкуренції. Зацікавленість у постійному покращенні та адаптації до нових технологій та трендів є ключовим фактором. Першочерговим завданням є гнучкість та швидкість реакції на зміни в запитах користувачів та технологічному ландшафті.

Стратегічне масштабування включає в себе оптимізацію існуючих процесів та ресурсів для ефективного використання потенціалу. Постійне вдосконалення кодової бази, використання новітніх технологій, та розвиток

інфраструктури - це основні компоненти стратегії, що спрямовані на підтримання високої продуктивності та швидкості веб-програми.

Важливою частиною стратегії є також партнерства та інтеграції з іншими платформами, що дозволяють розширити аудиторію та функціонал.

В сучасному інтернет-середовищі важливим елементом успіху веб-програми є ефективна стратегія масштабування та росту, спрямована на забезпечення сталого розвитку та задоволення потреб користувачів. В процесі масштабування слід звертати увагу на оптимізацію та гнучкість існуючих структур, щоб вони могли ефективно пристосовуватися до зростання обсягів трафіку та розширення функціоналу.

Додатково, розвиток мережевих партнерств і інтеграція з іншими системами та сервісами можуть розширити можливості веб-програми та покращити її конкурентоспроможність. Збалансована стратегія підтримки інфраструктури та оновлення технічної бази є ключем до стійкого функціонування та забезпечення високої якості обслуговування.

Паралельно з цим, постійний аналіз ринкових тенденцій і вимог користувачів дозволяє оперативно реагувати на зміни та адаптувати продукт до нових умов. Стратегічне вивчення конкуренції, виявлення слабких місць та підтримка стабільності виступають ключовими чинниками в управлінні масштабуванням.

У випадку виявлення труднощів або проблем, необхідно оперативно розробляти резервні плани та стратегії відновлення, щоб забезпечити найменший вплив на користувачів та забезпечити стабільну роботу веб-програми.

Такий комплексний підхід дозволяє досягти не тільки технічної стійкості, але й надійності веб-програми в умовах постійних змін та зростаючих вимог користувачів.

Загалом, стратегія масштабування та росту є стратегічним керованим

процесом, спрямованим на створення стійкої, конкурентоздатної та інноваційної веб-програми.

Висновок до розділу 3

Розділ 3 надає вичерпний огляд розробки проекту, починаючи від загальної архітектури та закінчуючи стратегіями масштабування та росту веб-програми. Визначені ключові аспекти, що включають структуру баз даних, інтерфейс користувача, засоби забезпечення безпеки та управління розробкою проекту. Кожен підрозділ ретельно розкриває функціональність та особливості відповідного аспекту розробки.

Зокрема, було описано модулі серверної частини, контроллери для управління даними, інтерфейс користувача та методи забезпечення безпеки. Особлива увага приділена аналізу продуктивності, користувацького досвіду та сумісності з різними браузерами та пристроями.

Цей розділ є важливим кроком у впровадженні розробленої веб-програми, забезпечуючи обґрунтований підхід до різних аспектів розробки та демонструючи готовність проекту до подальшого масштабування.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було надано обґрунтування та контекст для проведення дослідження та зазначено необхідність створення нового програмного забезпечення для оптимізації освітнього наставництва.

В ході роботи над кваліфікаційною роботою було сформульовано вимоги до майбутнього проекту та до методів аналізу ефективності, здійснено аналіз технологій популярних конкурентів. Було розглянуто дослідження, проведені для описаних платформ та застосовано методи порівняння аналогів з проектом кваліфікаційної роботи. Також описано переваги та недоліки обраних веб-інструментів, та необхідні для їх розробки фреймворки.

Визначені технічні вимоги до програмного забезпечення, які включають не лише функціональні аспекти, але й вимоги до продуктивності та масштабованості. Особлива увага приділена огляду заходів забезпечення безпеки використовуваних технологій, щоб гарантувати захищеність користувачів та даних у процесі використання програмного забезпечення.

Визначені ключові аспекти, що містять структуру баз даних, інтерфейс користувача, засоби забезпечення безпеки та управління розробкою проекту. Кожен підрозділ ретельно розкриває функціональність та особливості відповідного аспекту розробки.

Зокрема, було описано модулі серверної частини, контролери для управління даними, інтерфейс користувача та методи забезпечення безпеки. Особлива увага приділена аналізу продуктивності, користувацького досвіду та сумісності з різними браузерами та пристроями.

Результатом роботи став інструмент, який сприяє покращенню взаємодії між учасниками освітнього процесу і полегшує їхню роботу. Важливим аспектом дослідження був аналіз ефективності та користності розробленого програмного забезпечення. Проведені тести та оцінки підтвердили позитивний вплив програми на якість навчання та зручність робочого процесу.

Впровадження програми сприятиме покращенню процесів взаємодії між вчителями та студентами, спростить процес оцінювання успішності та зробить навчання більш ефективним.

Загалом, результати дослідження свідчать про потенціал впровадження розробленого програмного забезпечення для поліпшення якості освіти та навчання.

Подальший розвиток та вдосконалення програми можуть сприяти подальшому покращенню освітнього наставництва та сприяти більш зручному користуванню.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Smashing Magazine: веб-сайт. URL: https://www.smashingmagazine.com/ (дата звернення 12.09.2023)

2. A List Apart: веб-сайт. URL: https://alistapart.com/ (дата звернення 13.09.2023)

3. CSS-Tricks: веб-сайт. URL: https://css-tricks.com/ (дата звернення 13.09.2023)

4. SitePoint: веб-сайт. URL: https://www.sitepoint.com/ (дата звернення 22.10.2023)

5. Medium: веб-сайт. URL: https://medium.com/ (дата звернення 21.10.2023)

6. React.js Documentation: веб-сайт. URL: https://reactjs.org/ (дата звернення 11.10.2023)

7. Redux Documentation: веб-сайт. URL: https://redux.js.org/ (дата звернення 11.10.2023)

8. Vue.js Documentation: веб-сайт. URL: https://vuejs.org/ (дата звернення 11.10.2023)

9. You Don't Know JS (book series) by Kyle Simpson: веб-сайт. URL: https://github.com/getify/You-Dont-Know-JS (дата звернення 03.11.2023)

10. JavaScript.info: веб-сайт.URL: https://javascript.info/ (дата звернення 03.11.2023)

11. Mozilla Developer Network (MDN) - JavaScript Guide: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення 14.11.2023)

12. JavaScript Design Patterns by Addy Osmani: веб-сайт. URL: https://addyosmani.com/resources/essentialjsdesignpatterns/book/ (дата звернення 15.11.2023)

13. MongoDB University: веб-сайт. URL: https://university.mongodb.com/ (дата звернення 03.10.2023)

14. MongoDB Documentation: веб-сайт. URL: https://docs.mongodb.com/manual/crud/ (дата звернення 03.10.2023)

15. Дуглас Крокфорд. JavaScript: The Good Parts: довідник. O'Reilly Media, 2008. 176 с.

16. Андрей Мезин. React и Redux функциональная веб-разработка: довідник. "Питер", 2018. 336 с.

17. Стоян Стефанов. React Up and Running: довідник. O'Reilly Media, 2016. 250 с.

18. Джеремі Вілкен. Angular in Action: довідник. Manning Publications, 2015. 192 с.

19. Каллум Макрей. Vue.js: Up and Running: довідник. O'Reilly Media, 2018. 174 с.

20. Маріо Касціаро. Node.js Design Patterns: довідник. Packt Publishing, 2014. 454 с.

21. Крістіна Чодоров. MongoDB: The Definitive Guide: довідник. O'Reilly Media, 2018. 352 с.

22. Джейсон Крол. Web Development with MongoDB and Node.js: довідник. O'Reilly Media, 2014. 400 с.

23. Адді Османі. Learning JavaScript Design Patterns: довідник. O'Reilly Media, 2017. 180 с.

24. Responsive Web Design: довідник. A Book Apart, 2011. 150 с

25. Eloquent JavaScript by Marijn Haverbeke: довідник. No Starch Press, 2018. 456 с.

26. Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin: довідник. Prentice Hall, 2008. 464 с.

27. JavaScript: The Definitive Guide by David Flanagan: довідник. O'Reilly Media, 2020. 706 с.

28. JavaScript: The Good Parts by Douglas Crockford: довідник. O'Reilly Media, 2008. 176 с.

29. Exploring ES6: Upgrade to the Next Version of JavaScript by Axel Rauschmayer: довідник. O'Reilly Media, 2017. 324 с.

30. Learning React by Alex Banks and Eve Porcello: довідник. O'Reilly Media, 2017. 352 с.

31. React Up and Running by Stoyan Stefanov: довідник. O'Reilly Media, 2016. 250 с.

32. Vue.js: Up and Running by Callum Macrae: довідник. O'Reilly Media, 2018. 174 с.

33. Node.js Design Patterns by Mario Casciaro: довідник. Packt Publishing, 2014. 454 с.

34. Web Development with MongoDB and Node.js by Jason Krol: довідник. Packt Publishing, 2017. 320 с.

35. Learning JavaScript Design Patterns by Addy Osmani: довідник. O'Reilly Media, 2012. 254 с.

36. Design Patterns. Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: довідник. Addison-Wesley, 1994. 395 с.

37. Head First Design Patterns by Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra: довідник. O'Reilly Media, 2004. 694 с.

38. Responsive Web Design by Ethan Marcotte: довідник. A Book Apart, 2011. 150 с.

39. Smashing Book 5. Real-Life Responsive Web Design: довідник. Smashing Magazine, 2015. 412 с.

40. JavaScript: The Good Parts by Douglas Crockford: довідник. O'Reilly Media, 2008. 176 с.

41. Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript by David Herman: довідник. Addison-Wesley, 2012. 240 с.

метадані

Заголовок

Розробка та дослідження нового програмного забезпечення для вдосконалення та автоматизації процесів освітнього наставництва

Автор
Насадика В.В.

Науковий керівник / Експерт
кандидат технічних наук Сергій Ващишак

підрозділ
King Danylo University

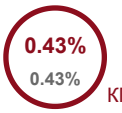
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	3
Інтервали	A→	0
Мікропробіли	·	47
Білі знаки	ß	2118
Парафрази (SmartMarks)	a	4

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

КП 2

25
Довжина фрази для коефіцієнта подібності 2

16558
Кількість слів

133867
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	Розроблення вебсистеми для менеджменту Школи "Академія новітніх технологій "Хеврон" [redacted] 1/24/2024 National Forestry University of Ukraine (ЦДН НЛТУ України)	13	0.08 %
2	Інформаційно-аналітичний геопортал походів УПА [redacted] 1/9/2024 National Forestry University of Ukraine (ЦДН НЛТУ України)	11	0.07 %
3	https://embo.com.ua/uk/blog/odnostorinkovi-spa-i-bagatostorinkovi-pwa/	11	0.07 %