Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-платформи для надання послуг репетиторства з програмування «CodeMentor» з використанням фреймворку React.js

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Чендей Б.В.
	(підпис)	(прізвище та ініціали)
Керівник		Цуприк Г.Б.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю.М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М.Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення», ТНТУ, 2025. Сторінок: 72, рисунків: 13, таблиць: 1, формул: 0, додатків: 3, посилань: 36, частин:5

Ключові слова: веб-платформа, онлайн-освіта, програмування, репетиторство, спільне редагування коду, React.js.

Об’єкт дослідження – веб-платформа «CodeMentor» для організації репетиторських послуг з програмування з модулями спільного редагування коду та управління профілями.

Ціль роботи – створити масштабовану й зручну платформу для ефективної взаємодії студентів і менторів, покращити доступ до якісного навчання програмуванню.

Методи та технології – аналіз існуючих рішень, проєктування архітектури (UML-діаграми). Фронтенд на React.js, бекенд на Node.js + Express.js з REST API, PostgreSQL + MongoDB, WebSocket/WebRTC для реального часу; тести та CI/CD.

Результати й наукова новизна – реалізовано прототип із ключовими модулями: реєстрація, пошук ментора, спільний редактор коду, повідомлення. Запропоновано оптимізовану архітектуру інтеграції редактора коду для мінімізації затримок; гібридне зберігання даних для масштабованості; адаптивний підхід до навантаження.

Висновки й пропозиції розвитку – Платформа відповідає поставленій цілі й функціонує належно. Рекомендовано додати аналітику ефективності занять, інтеграції з зовнішніми сервісами, мобільну версію, покращити масштабування.

ABSTRACT

Bachelor's Qualification Thesis. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 "Software Engineering", TNTU, 2025. Pages: 72, figures: 13, tables: 1, formulas: 0, appendices: 3, references: 36, parts: 5.

Keywords**:** collaborative code editing, online education, programming, React.js, tutoring, web platform.

The object of the study is the "CodeMentor" web platform for organizing programming tutoring services, featuring modules for collaborative code editing and profile management.

The aim of the work is to create a scalable and user-friendly platform for effective interaction between students and mentors, and to improve access to quality programming education.

Methods and technologies – analysis of existing solutions, architecture design (UML diagrams). Frontend on React.js, backend on Node.js + Express.js with a REST API, PostgreSQL + MongoDB, WebSocket/WebRTC for real-time features; testing and CI/CD.

Results and scientific novelty – A prototype with key modules was implemented: registration, mentor search, a collaborative code editor, and messaging. An optimized architecture for integrating the code editor was proposed to minimize latency; a hybrid data storage solution for scalability; and an adaptive approach to load management.

Conclusions and suggestions for future development – The platform meets the stated goals and functions properly. It is recommended to add analytics for session effectiveness, integrations with external services, a mobile version, and to improve scalability.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (Інтерфейс програмування додатків).

ACID – Atomicity, Consistency, Isolation, Durability (Атомарність, Узгодженість, Ізольованість, Довговічність) – властивості транзакцій у базах даних.

AWS – Amazon Web Services.

CDN – Content Delivery Network (Мережа доставки контенту).

CI/CD – Continuous Integration / Continuous Deployment (Безперервна інтеграція / Безперервне розгортання).

CSRF – Cross-Site Request Forgery (Підробка міжсайтових запитів).

CSS – Cascading Style Sheets (Каскадні таблиці стилів).

DOM – Document Object Model (Об'єктна модель документа).

EdTech – Educational Technology (Освітні технології).

GDPR – General Data Protection Regulation (Загальний регламент про захист даних).

HTML – HyperText Markup Language (Мова розмітки гіпертексту).

HTTP – HyperText Transfer Protocol (Протокол передачі гіпертексту).

HTTPS – HyperText Transfer Protocol Secure (Захищений протокол передачі гіпертексту).

IDE – Integrated Development Environment (Інтегроване середовище розробки).

JSON – JavaScript Object Notation (Нотація об'єктів JavaScript).

JWT – JSON Web Token.

MOOC – Massive Open Online Courses (Масові відкриті онлайн-курси).

MVC – Model-View-Controller (Модель-Представлення-Контролер).

PWA – Progressive Web Applications (Прогресивні веб-додатки).

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах зростаючого попиту на ІТ-фахівців ефективне навчання програмуванню є надзвичайно важливим. Проте традиційні освітні підходи не завжди враховують індивідуальні особливості студентів, що ускладнює засвоєння матеріалу. У зв’язку з цим розробка веб-платформи, яка забезпечує зручну взаємодію між студентами та досвідченими репетиторами, є актуальним завданням у сфері освітніх технологій.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є розробка веб-платформи «CodeMentor» для надання послуг репетиторства з програмування з використанням сучасного фреймворку React.js. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

1. Проаналізувати стан досліджень в галузі онлайн-освіти та існуючі рішення для надання репетиторських послуг з програмування
2. Визначити функціональні та нефункціональні вимоги до веб-платформи CodeMentor
3. Спроектувати архітектуру веб-додатку з урахуванням сучасних підходів до розробки інтерфейсів
4. Реалізувати клієнтську частину з використанням React.js
5. Розробити серверну частину та інтегрувати її з клієнтською
6. Впровадити систему аутентифікації та авторизації користувачів
7. Реалізувати функціонал пошуку та проведення онлайн-занять
8. Протестувати розроблену веб-платформу та усунути виявлені недоліки

Практичне значення одержаних результатів. Розроблена веб-платформа CodeMentor дозволить підвищити доступність якісного навчання програмуванню за рахунок з'єднання студентів з досвідченими наставниками. Реалізовані інструменти для онлайн-занять з використанням сучасних технологій зроблять процес навчання інтерактивним та ефективним. Крім того, рішення може бути масштабоване та адаптоване для інших освітніх напрямків.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБ-ПЛАТФОРМ

1.1 Аналіз сучасних технологій розробки веб-додатків

Сучасна розробка веб-додатків охоплює різні підходи, архітектури та технології для створення інтерактивних платформ. Технологічний стек умовно поділяється на фронтенд, бекенд та інфраструктуру (БД, кешування, балансування тощо). Ефективна інтеграція цих компонентів є критичною для успіху проєкту. Актуальними є дослідження з використання високопродуктивних методів у моделюванні складних систем [1].

Основні компоненти сучасного веб-додатку включають:

1. Фронтенд – відповідає за інтерфейс та взаємодію з користувачем, включає HTML, CSS, JavaScript та фреймворки (React.js, Angular, Vue.js).
2. Бекенд – забезпечує бізнес-логіку, обробку даних та API, реалізується за допомогою Node.js, Python, PHP, Ruby, Java або інших серверних технологій.
3. База даних – зберігає інформацію, може бути реляційною (PostgreSQL, MySQL) або нереляційною (MongoDB, Redis).
4. Інфраструктура – включає серверне середовище, контейнеризацію (Docker), оркестрацію (Kubernetes), CDN та інші сервіси для забезпечення роботи додатку [2].

Фронтенд- розробка зосереджена на створенні інтерфейсу та взаємодії з користувачем. Основу складають HTML, CSS і JavaScript. HTML5 додав підтримку мультимедіа та графіки без плагінів, CSS3 — анімації, гнучкі макети та адаптивність, а JavaScript еволюціонував у потужну мову з підтримкою сучасних можливостей ES6+, як-от класи, модулі, async/await тощо [2].

Серед популярних фреймворків фронтенду вирізняються React.js, Angular і Vue.js.

React.js — JavaScript-бібліотека від Facebook для створення інтерфейсів на основі компонентів. Її головна перевага — віртуальний DOM, який мінімізує зміни

в реальному DOM завдяки алгоритму узгодження. Це забезпечує високу продуктивність, особливо у складних і динамічних додатках [3].

Компонентна архітектура React.js сприяє повторному використанню коду, полегшує тестування й підтримку. Із появою Hooks (useState, useEffect тощо) функціональні компоненти отримали всі можливості класових. React має розвинену екосистему: Redux — для управління станом, React Router — для навігації, Next.js — для SSR.

Angular — повноцінний фреймворк від Google на основі TypeScript, орієнтований на створення SPA. На відміну від React, Angular включає все необхідне з коробки: MVC-архітектуру, форми, валідацію, маршрутизацію, HTTP-запити. TypeScript забезпечує статичну типізацію, що знижує кількість помилок. Система залежностей Angular спрощує тестування та масштабування проєктів [4].

Vue.js — прогресивний фреймворк, який легко інтегрується в проєкти. Він поєднує переваги React і Angular: реактивність, компонентність і просту інтеграцію. Завдяки простоті та швидкому входженню Vue підходить для новачків і прототипів. Має власну екосистему: Vuex — для стану, Vue Router — для навігації, Nuxt.js — для SSR [5].

Бекенд відповідає за обробку запитів, роботу з базами даних, бізнес-логіку, авторизацію та API. Для його реалізації використовують різні мови й фреймворки залежно від потреб проєкту та команди.

Node.js став популярним завдяки можливості використовувати JavaScript і на клієнті, і на сервері. Його подійно-орієнтована, неблокуюча модель введення-виведення забезпечує обробку багатьох одночасних з'єднань, що особливо корисно для реального часу (WebSockets, API). Express.js — найпоширеніший фреймворк для Node.js, який спрощує розробку серверних додатків завдяки зручній роботі з HTTP-запитами, маршрутизацією та мідлварами [6].

Серед популярних бекенд-технологій — Python (Django, Flask), Ruby on Rails, PHP (Laravel, Symfony) та Java (Spring). Python вирізняється простотою й читаємістю, Ruby on Rails — швидкістю розробки, PHP — широкою підтримкою

та сучасними фреймворками, Java з Spring — надійністю для масштабних корпоративних рішень. Вибір залежить від потреб проєкту та досвіду команди [7].

У останні роки популярною стає архітектура мікросервісів, де додаток поділяється на незалежні сервіси, кожен з яких відповідає за певну функцію. Це дозволяє розробляти, розгортати та масштабувати сервіси окремо, підвищуючи гнучкість і стійкість системи. Мікросервіси взаємодіють через API або повідомлення (RabbitMQ, Apache Kafka). Цей підхід забезпечує масштабованість та стійкість, але збільшує складність системи та вимагає додаткових зусиль для комунікації, моніторингу та управління [8].

Для зберігання даних використовують як реляційні (PostgreSQL, MySQL, Microsoft SQL Server), так і нереляційні (MongoDB, Redis, Cassandra) бази даних. Реляційні СУБД з підтримкою ACID підходять для структурованих даних із чіткими зв’язками. PostgreSQL — потужна об'єктно-реляційна СУБД з підтримкою складних запитів і JSON. MySQL — швидка та проста у використанні, а Microsoft SQL Server — популярна в корпоративних середовищах [9].

Нереляційні (NoSQL) бази даних зберігають великі обсяги неструктурованих даних з високою продуктивністю та масштабованістю. MongoDB — документно-орієнтована база в форматі BSON, зручна для JSON-даних. Redis — швидке сховище в пам’яті для кешування і реального часу. Cassandra — розподілена база для обробки великих обсягів даних із високою доступністю. Вибір бази залежить від типу даних і вимог, а в сучасних системах часто застосовують "polyglot persistence" — використання різних баз для різних даних [10].

Важливим у веб-розробці є використання API для взаємодії між клієнтом і сервером та інтеграції зі сторонніми сервісами. Найпоширенішим є REST — простий і масштабований стиль, що використовує стандартні HTTP-методи (GET, POST, PUT, DELETE). RESTful API полегшує розробку, тестування та масштабування. GraphQL, розроблений Facebook, набирає популярність завдяки можливості клієнту точно запитувати потрібні дані, зменшуючи надлишкові запити, та підтримує складні взаємозв’язки і автоматичну документацію [11].

Розглянемо детальніше порівняння основних фреймворків для розробки клієнтської частини веб-додатків у вигляді таблиці 1.1

Таблиця 1.1 – Порівняння фреймворків для фронтенд-розробки

Характеристика	**React.js**	**Angular**	**Vue.js**
Розробник	Facebook	Google	Еван Ю
Рік створення	2013	2010 (AngularJS), 2016 (Angular 2+)	2014
Тип	Бібліотека	Фреймворк	Прогресивний фреймворк
Мова програмування	JavaScript, JSX	TypeScript	JavaScript
Архітектура	Компонентно-орієнтована	Компонентно-орієнтована, MVC	Компонентно-орієнтована, MVVM
Управління станом	Через власні рішення (useState, useReducer) або зовнішні бібліотеки (Redux, MobX)	Через сервіси та RxJS	Реактивна система та Vuex
Розмір (мін. gzip)	~42 КБ	~143 КБ	~33 КБ
Крива навчання	Помірна	Крута	Пологая
Масштабованість	Висока з додатковими бібліотеками	Висока за замовчуванням	Висока з екосистемою
Продуктивність	Висока завдяки віртуальному DOM	Висока з оптимізацією компілятора	Висока завдяки віртуальному DOM
Популярність (GitHub Stars)	>200K	>80K	>200K
Підтримка мобільної розробки	React Native	Ionic, NativeScript	Vue Native, NativeScript
Інструменти розробки	Create React App, Next.js	Angular CLI	Vue CLI, Nuxt.js
SSR (серверний рендеринг)	Через Next.js	Через Angular Universal	Через Nuxt.js
Тестування	Jest, React Testing Library	Karma, Jasmine	Jest, Vue Testing Library
Основні компанії-користувачі	Facebook, Instagram, Netflix	Google, Microsoft, IBM	Alibaba, Xiaomi, GitLab

Прогресивні веб-додатки (PWA) поєднують переваги веб-сайтів: офлайн-режим, push-повідомлення, доступ до апаратних функцій. Вони базуються на

Service Workers, Web App Manifest, HTTPS та адаптивному дизайні, працюють у будь-якому сучасному браузері без встановлення через магазини [12].

1.2 Огляд існуючих платформ для надання освітніх послуг

Сфера EdTech стабільно зростає завдяки попиту на гнучкі, доступні та персоналізовані освітні рішення.

Основні категорії освітніх платформ включають:

1. Масові відкриті онлайн-курси (MOOC) – Coursera, edX, Udacity – пропонують структуровані навчальні програми від університетів та компаній.
2. Платформи самостійного навчання програмуванню – Codecademy, freeCodeCamp, SoloLearn – фокусуються на інтерактивних вправах з миттєвим зворотним зв'язком.
3. Платформи репетиторства – Chegg Tutors, Preply, Wyzant – з'єднують студентів з індивідуальними викладачами для персоналізованого навчання.
4. Спеціалізовані платформи для професійного навчання – Pluralsight, Treehouse, DataCamp – пропонують поглиблені курси з конкретних технологій та навичок [13].

MOOC-платформи Coursera, edX і Udacity пропонують онлайн-курси від провідних університетів і компаній. Coursera співпрацює з понад 200 партнерами й має 4000+ курсів. edX, заснована Гарвардом і MIT, надає академічно якісні курси з можливістю отримання кредитів. Udacity фокусується на IT-програмах і пропонує "нанодипломи" для професійної підготовки.

Курси на Coursera, edX і Udacity включають відеолекції, інтерактивні завдання, проєкти та форуми. Лекції поділені на короткі сегменти, завдання перевіряють знання, а проєкти дають змогу застосувати матеріал. Форумна підтримка сприяє обміну досвідом. Монетизація здійснюється через продажи сертифікатів і преміум-підписки.

Codecademy, freeCodeCamp і SoloLearn зосереджені на практичному вивченні програмування через інтерактивні вправи. Codecademy забезпечує миттєвий фідбек у браузері, freeCodeCamp пропонує безкоштовні курси з проєктами та сертифікатами, а SoloLearn — короткі уроки з мобільною підтримкою та соціальними функціями.

Платформи персоналізованого навчання та репетиторства — Chegg Tutors, Preply і Wyzant — з'єднують студентів з індивідуальними репетиторами в режимі реального часу. Chegg Tutors забезпечує цілодобовий доступ до репетиторів з різних дисциплін, Preply акцентує увагу на мовах та гнучкому розкладі, а Wyzant пропонує вибір викладачів за потребами, досвідом і бюджетом студента.

Монетизація зазвичай базується на погодинній оплаті з комісією платформи.

1.3 Вимоги до розробки веб-платформ у сфері освітніх послуг

Розробка освітніх веб-платформ вимагає врахування функціональних, нефункціональних і технічних вимог.

Функціональні вимоги

Функціональні вимоги визначають, що система має робити для користувачів. Для веб-платформи репетиторства з програмування ключовими є:

1. Реєстрація та аутентифікація з розподілом ролей (студент, ментор, адміністратор), верифікація електронної пошти та авторизація через соцмережі. Для менторів — перевірка професійних навичок.
2. Управління профілями з інформацією про навички, досвід, освіту, портфоліо; для менторів — спеціалізація і доступність, для студентів — рівень знань і цілі. Можливість редагування даних.
3. Пошук і фільтрація менторів за мовами програмування, технологіями, досвідом, вартістю, рейтингом і часовим поясом із зручним інтерфейсом.

Календар та система бронювання занять з відображенням доступності менторів, можливістю вибору зручного часу та отримання підтверджень. Ключові компоненти системи бронювання:

1. Інтерактивний календар для вибору дат і часу
2. Автоматична перевірка доступності та запобігання конфліктам
3. Система нагадувань про заплановані заняття

Інтегроване середовище для проведення онлайн-занять, що включає:

1. Текстовий чат для обміну повідомленнями і посиланнями
2. Віртуальну дошку та спільне редагування коду

Нефункціональні вимоги

Нефункціональні вимоги описують якісні характеристики системи, які впливають на користувацький досвід та загальну ефективність платформи:

Продуктивність системи, що забезпечує:

1. Швидкий час відгуку (менше 1 секунди)
2. Обробку достатньої кількості одночасних користувачів
3. Стабільну роботу функцій реального часу (редагування коду)

Масштабованість архітектури для адаптації до зростання користувачів і навантаження без втрати продуктивності, через горизонтальне масштабування, хмарні сервіси та оптимізацію баз даних.

Безпека даних і захист конфіденційної інформації користувачів відповідно до стандартів (GDPR, CCPA). Це включає:

1. Шифрування даних та захищені з'єднання (HTTPS)
2. Безпечне зберігання паролів (хешування)
3. Захист від поширених типів атак (SQL-ін'єкції, XSS, CSRF)

Доступність системи з мінімальним часом простою (99,9%+), а також стратегії резервного копіювання і відновлення даних для запобігання втратам та мінімізації впливу збоїв.

Юзабіліті та зручність користувацького інтерфейсу, що забезпечує:

1. Інтуїтивно зрозумілу навігацію
2. Адаптивний дизайн для різних пристроїв

3. Доступність для користувачів з обмеженими можливостями

Технічні вимоги

Технічні вимоги визначають технологічний стек та архітектурні рішення, що будуть використовуватися для розробки веб-платформи:

Використання сучасних фреймворків та бібліотек для розробки інтерактивного та динамічного користувацького інтерфейсу. Для проєкту CodeMentor обрано React.js завдяки таким перевагам:

1. Декларативний підхід до створення інтерфейсу
2. Компонентна архітектура для повторного використання елементів
3. Багата екосистема бібліотек і розширень

Архітектура клієнт-сервер з фронтендом і бекендом, що спілкуються через RESTful API або GraphQL, забезпечує гнучкість, незалежне масштабування та підтримку мобільних додатків у майбутньому.

Використання оптимальних баз даних для різних типів даних:

1. Реляційні (PostgreSQL) для структурованої інформації
2. Нереляційні (MongoDB) для динамічного контенту
3. Кешування частих запитів для підвищення продуктивності

Інтеграція з сторонніми сервісами для забезпечення різноманітної функціональності:

1. Конференції (Twilio, Zoom API)
2. Платежі (Stripe, PayPal)
3. Автентифікація (OAuth, Firebase Auth) та сповіщення

Розгортання на хмарній інфраструктурі (AWS, Google Cloud, Azure) для забезпечення:

1. Масштабованості залежно від навантаження
2. Високої доступності та надійності
3. Оптимізації витрат на обслуговування

Впровадження CI/CD (Continuous Integration/Continuous Deployment) для автоматизації процесів тестування, збірки та розгортання змін, що підвищує якість коду та швидкість виведення нових функцій на ринок.

1.4 Обґрунтування вибору технологій для реалізації проєкту

Вибір технологій для розробки CodeMentor базується на аналізі вимог і сучасних трендів, щоб забезпечити ефективну розробку, підтримку, масштабування та відповідність потребам освітньої платформи для репетиторства з програмування.

При виборі технологій ми орієнтувались на три основні критерії:

1. Продуктивність і швидкодія
2. Можливість масштабування
3. Наявність активної спільноти розробників і документації

Для фронтенду обрано React.js через компонентну архітектуру, що дозволяє створювати повторно використовувані UI-компоненти для складних інтерфейсів (профілі, календар, чат, редактор коду). Віртуальний DOM підвищує продуктивність, оптимізуючи оновлення інтерфейсу. Екосистема (Redux, Context API, React Router, Material-UI, Tailwind CSS) забезпечує інструменти для управління станом, навігації та створення стильного UI.

Для бекенду обрано Node.js з Express.js через єдину мову (JavaScript) для фронтенду і бекенду, що спрощує розробку та підтримку. Node.js ефективно обробляє асинхронні операції завдяки подійно-орієнтованій архітектурі, ідеально для API та великої кількості запитів.

Для зберігання даних обрано комбінацію реляційної та нереляційної бази даних. Основні переваги такого підходу:

1. PostgreSQL для зберігання структурованих даних (користувачі, заняття, транзакції)
2. MongoDB для зберігання динамічного контенту (чати, повідомлення в реальному часі)
3. Оптимальна продуктивність для різних типів даних і операцій

Для комунікації в реальному часі між користувачами платформи використано WebSockets із Socket.io, що забезпечує двонаправлений зв’язок для чату та

спільного редагування коду. Інтегроване середовище розробки (IDE) створюється на основі Visual Studio Code і бібліотеки Yjs для підтримки колаборативного редагування з підсвічуванням синтаксису та автодоповненням.

Для безпеки додатку використовуватимуть JWT для аутентифікації, bcrypt для хешування паролів, HTTPS та захист від поширених атак. Платіжна система інтегрується зі Stripe для безпечної обробки платежів і відповідності стандартам PCI DSS. Додаток розгортається на AWS з Docker і Kubernetes, що забезпечує масштабування, високу доступність. Обраний стек (React.js, Node.js, PostgreSQL, MongoDB, WebSockets, WebRTC) відповідає потребам платформи CodeMentor.

РОЗДІЛ 2. АНАЛІЗ ТА ПРОЄКТУВАННЯ ВЕБ-ПЛАТФОРМИ

2.1 Формування вимог до веб-платформи

При розробці веб-платформи CodeMentor для надання послуг репетиторства з програмування необхідно чітко сформулювати вимоги, які визначатимуть функціональність, якість та технічні аспекти системи. Ці вимоги є основою для подальшого проєктування архітектури, структури бази даних та інтерфейсу користувача. Вимоги були сформовані на основі аналізу потреб цільової аудиторії, дослідження існуючих аналогічних платформ та врахування сучасних тенденцій у розробці освітніх веб-сервісів.

2.1.1 Функціональні вимоги

Функціональні вимоги визначають основні можливості системи для задоволення потреб користувачів. Для CodeMentor ключові з них — реєстрація та авторизація з розподілом ролей (студент, репетитор, адміністратор), підтримка входу через email, пароль і соцмережі, а також впровадження двофакторної автентифікації та відновлення паролю через email [14].

Основні компоненти цієї вимоги:

1. Реєстрація нових користувачів з обов'язковою верифікацією електронної пошти
2. Різні рівні доступу для студентів, репетиторів та адміністраторів
3. Процес верифікації репетиторів для підтвердження їхніх навичок та кваліфікації

Управління профілями має забезпечувати створення та редагування особистої інформації, включаючи досвід, освіту, фото та контактні дані. Профілі

репетиторів містять навички, досвід, портфоліо, погодинну ставку, доступність за днями та часовий пояс для зручного планування.

Пошук і фільтрація репетиторів дозволяють студентам знаходити викладачів за технологіями, досвідом, ціною, рейтингом, доступністю та терміновістю консультацій, з різними варіантами сортування та зручним відображенням для швидкого порівняння.

Ключові аспекти:

1. Розширена система фільтрів для точного пошуку репетиторів
2. Інтелектуальне ранжування результатів пошуку на основі рейтингу та відгуків
3. Можливість збереження улюблених репетиторів та історії пошуків

Система бронювання та розкладу — забезпечує зручне планування занять. Вона включає інтерактивний календар із реальним відображенням доступності репетиторів, вибір дат і підтримку різних типів занять із налаштуванням тривалості та періодичності.

Середовище для проведення онлайн-занять має забезпечувати всі необхідні інструменти:

1. Інтегроване середовище розробки (IDE) для спільного програмування в реальному часі
2. Віртуальна дошка для пояснення концепцій та алгоритмів

Система оплати має забезпечувати зручні та безпечні фінансові транзакції з підтримкою банківських карт, електронних гаманців і переказів у різних валютах. Вона повинна відповідати стандартам PCI DSS, автоматично генерувати рахунки та підтримувати різні моделі оплати (погодинна, пакетна, абонемент) [15].

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якість роботи системи та обмеження її функціонування, що важливо для задоволення користувачів і стабільної роботи платформи.

CodeMentor має забезпечувати швидке завантаження сторінок (до 2 секунд) і відгук на дії користувача не більше 0,5 секунди. Особливо важлива мінімальна затримка у компонентах реального часу, наприклад, у спільному редагуванні коду, для комфортної взаємодії.

Основні вимоги включають:

1. Швидкий час відгуку інтерфейсу користувача (менше 2 секунд)
2. Оптимізована робота з базами даних для швидкого отримання інформації
3. Ефективне використання кешування для частих запитів

Безпека даних — ключова вимога через обробку персональних і фінансових даних. Система має відповідати стандартам GDPR та PCI DSS, забезпечувати шифрування даних (HTTPS), надійне хешування паролів і захист від атак (SQL-ін’єкції, XSS, CSRF, DDoS).

Масштабованість архітектури потрібна для стабільної роботи при зростанні користувачів, з підтримкою вертикального (потужність серверів) та горизонтального (додавання серверів) масштабування.

Важливими аспектами є:

1. Модульна структура системи для незалежного масштабування компонентів
2. Використання балансувальників навантаження для розподілу запитів
3. Мікросервісна архітектура для ізоляції функціональних модулів

Юзабіліті та доступність інтерфейсу критичні для комфортного користування: інтерфейс має бути простим, з логічною навігацією, адаптивним для різних пристроїв і браузерів.

2.2 Проєктування архітектури системи

Архітектура веб-платформи CodeMentor побудована на принципах розробки з використанням мікросервісної моделі, що забезпечує чітке розділення клієнтської та серверної частин для ефективного виконання функціональних і нефункціональних вимог.

Основні переваги обраної архітектури:

1. Гнучкість розробки та масштабування окремих компонентів
2. Можливість використання різних технологій для різних мікросервісів
3. Висока стійкість системи до відмов окремих компонентів [16].

Архітектура системи складається з наступних ключових компонентів:

Клієнтська частина (Front-end) — односторінковий додаток (SPA) на React.js, що забезпечує швидкі переходи без перезавантаження. Використовуються Redux для управління станом, React Router для маршрутизації, WebSockets і WebRTC для комунікації в реальному часі. Стилізація — через Material-UI для сучасного адаптивного дизайну [17].

Серверна частина (Back-end) побудована на мікросервісній архітектурі. API Gateway маршрутизує запити до мікросервісів, відповідає за аутентифікацію, авторизацію та первинну валідацію.

Основні мікросервіси включають:

1. Сервіс аутентифікації та управління користувачами – відповідає за реєстрацію, авторизацію, управління профілями та налаштуваннями користувачів
2. Сервіс пошуку та бронювання – забезпечує функціональність пошуку репетиторів за різними критеріями та управління бронюваннями занять
3. Сервіс проведення занять – відповідає за функціонування середовища для онлайн-навчання, включаючи спільне редагування коду та чат
4. Платіжний сервіс – забезпечує обробку платежів, формування рахунків та управління фінансовими транзакціями

5. Сервіс повідомлень та нотифікацій – відповідає за комунікацію між користувачами та надсилання сповіщень через різні канали [18].

На рисунку 2.1 представлена загальна архітектурна схема системи CodeMentor.

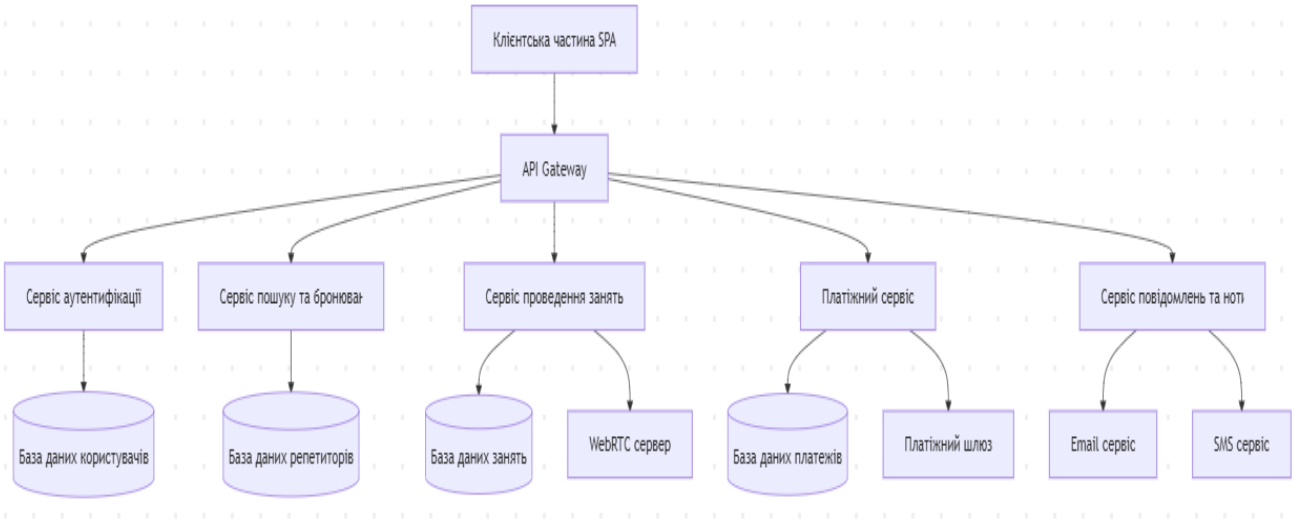


Рисунок 2.1 — Архітектурна схема системи CodeMentor

Комунікація між мікросервісами реалізована за допомогою двох основних підходів:

1. Синхронна комунікація через REST API для прямих запитів, що вимагають негайної відповіді
2. Асинхронна комунікація через Message Queue (RabbitMQ) для подій, які не вимагають миттєвої обробки [19].

Для доступності кожен мікросервіс розгортається в кількох екземплярах за балансувальником навантаження. Запроваджено автоматичне масштабування за навантаженням та механізм Circuit Breaker для запобігання каскадним відмовам.

2.3 Розробка структури бази даних

Для ефективного зберігання та управління даними веб-платформи CodeMentor розроблено комплексну структуру бази даних, що враховує особливості предметної області та забезпечує оптимальну продуктивність операцій з даними. Відповідно до вимог проєкту та обраної архітектури, було прийнято рішення використовувати поліглотний підхід до зберігання даних, комбінуючи реляційну базу даних PostgreSQL для структурованих даних та MongoDB для неструктурованих або напівструктурованих даних [20].

Основні переваги такого підходу:

1. Оптимальне використання сильних сторін різних типів баз даних для різних видів даних
2. Гнучкість схеми даних для модулів, які потребують частих змін структури
3. Можливість горизонтального масштабування для частин системи з високим навантаженням [21].

Реляційна база даних PostgreSQL використовується для зберігання основних структурованих даних системи, де важлива цілісність даних, транзакційність та складні зв'язки між сутностями. Концептуальна модель реляційної бази даних представлена на (рисунку 2.2.)

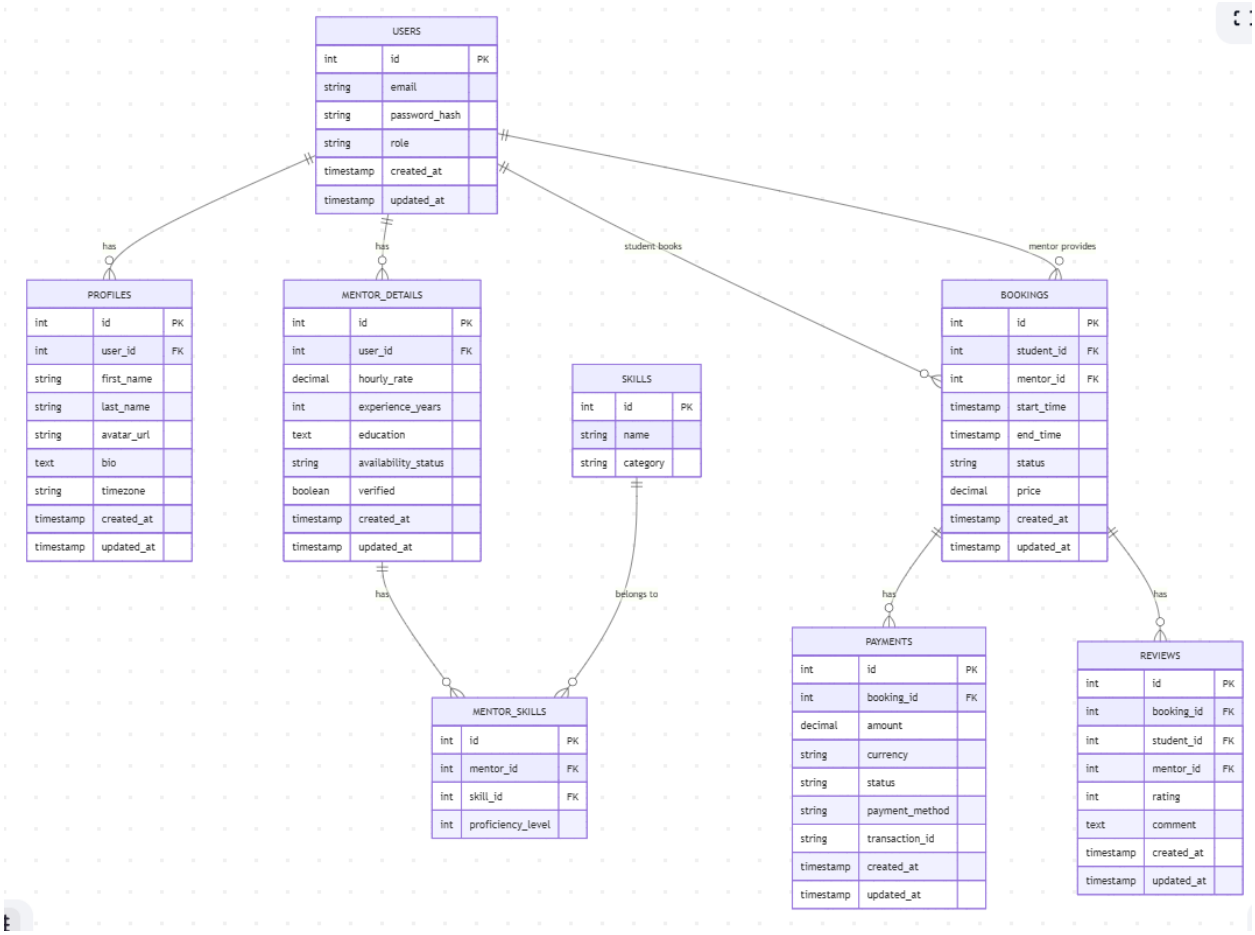


Рисунок 2.2 — Концептуальна модель реляційної бази даних CodeMentor

Ключові сутності реляційної бази даних та їх основні атрибути:

1. Users — зберігає основну інформацію про користувачів системи:

– id (PK) — унікальний ідентифікатор
– email — електронна пошта (унікальна)
– password_hash — хеш паролю
– role — роль (student, mentor, admin)
– created_at, updated_at — мітки часу створення та оновлення

2. Profiles — містить детальну інформацію про профілі користувачів:

– id (PK) — унікальний ідентифікатор
– user_id (FK) — зв'язок з таблицею Users
– first_name, last_name — ім'я та прізвище
– avatar_url — посилання на аватар
– bio — біографія або опис

– timezone — часовий пояс

– created_at, updated_at — мітки часу

3. MentorDetails — зберігає специфічну інформацію про репетиторів:

– id (PK) — унікальний ідентифікатор

– user_id (FK) — зв'язок з таблицею Users

– hourly_rate — погодинна ставка

– experience_years — роки досвіду

– education — інформація про освіту

– availability_status — статус доступності

– verified — статус верифікації

– created_at, updated_at — мітки часу

4. Skills та MentorSkills — зберігають інформацію про навички програмування та їх зв'язок з репетиторами:

– Skills: id (PK), name, category

– MentorSkills: id (PK), mentor_id (FK), skill_id (FK), proficiency_level

5. Bookings — містить інформацію про бронювання занять:

– id (PK) — унікальний ідентифікатор

– student_id, mentor_id (FK) — зв'язки з таблицею Users

– start_time, end_time — час початку та завершення заняття

– status — статус бронювання (pending, confirmed, completed, cancelled)

– price — вартість заняття

– created_at, updated_at — мітки часу

6. Payments — зберігає інформацію про платежі:

– id (PK), booking_id (FK), amount, currency, status, payment_method, transaction_id

7. Reviews — містить відгуки та рейтинги:

– id (PK), booking_id, student_id, mentor_id (FK), rating, comment, created_at, updated_at

2.3.1 Структура нереляційної бази даних

MongoDB використовується для зберігання даних, які мають менш структурований характер, часто змінюються або потребують високої швидкості запису та читання [22].

Основними колекціями в MongoDB є:

Messages — зберігає повідомлення між користувачами:

```
{
    _id: ObjectId,
        conversation_id: String,
            sender_id: String,
                receiver_id: String,
                    content: String,
                        attachments: [
                            {
                                type: String, // "file", "image", "code"
                                url: String,
                                name: String
                            }
                        ],
                            read: Boolean,
                                created_at: Date
      }
```

Лістинг 2.1 — Модель Messages для зберігання повідомлень між користувачами

Notifications — зберігає сповіщення для користувачів:

```
{
    _id: ObjectId,
        user_id: String,
            type: String, // "booking", "message", "system"
                title: String,
                    message: String,
                        related_id: String, // id пов'язаного об'єкта
                            read: Boolean,
                                created_at: Date
      }
```

Лістинг 2.2 — Модель Notifications для зберігання сповіщень користувачів

2.4 Проєктування інтерфейсу користувача

Інтерфейс користувача веб-платформи CodeMentor розроблено з урахуванням принципів UX/UI дизайну та специфічних потреб різних типів користувачів системи. Основними цілями при проєктуванні інтерфейсу були:

1. Забезпечення інтуїтивно зрозумілої та зручної навігації
2. Створення візуально привабливого дизайну з дотриманням принципів Material Design
3. Оптимізація користувацьких сценаріїв для різних типів користувачів
4. Забезпечення адаптивності для різних пристроїв (ПК, планшети, смартфони) [23].

2.4.1 Дизайн інтерфейсу для студентів

Інтерфейс для студентів спроєктовано з акцентом на пошук відповідних репетиторів, управління заняттями та зручне навчання. Ключові екрани та їх функціональність.

Середовище проведення заняття – ключовий компонент інтерфейсу, який включає:

1. Інтегроване середовище розробки з підтримкою різних мов програмування
2. Чат для обміну текстовими повідомленнями та файлами [24].

Головна сторінка для студентів містить швидкий доступ до основних функцій: пошук репетиторів, перегляд запланованих занять, продовження незавершених уроків. (див. рис. 2.3)

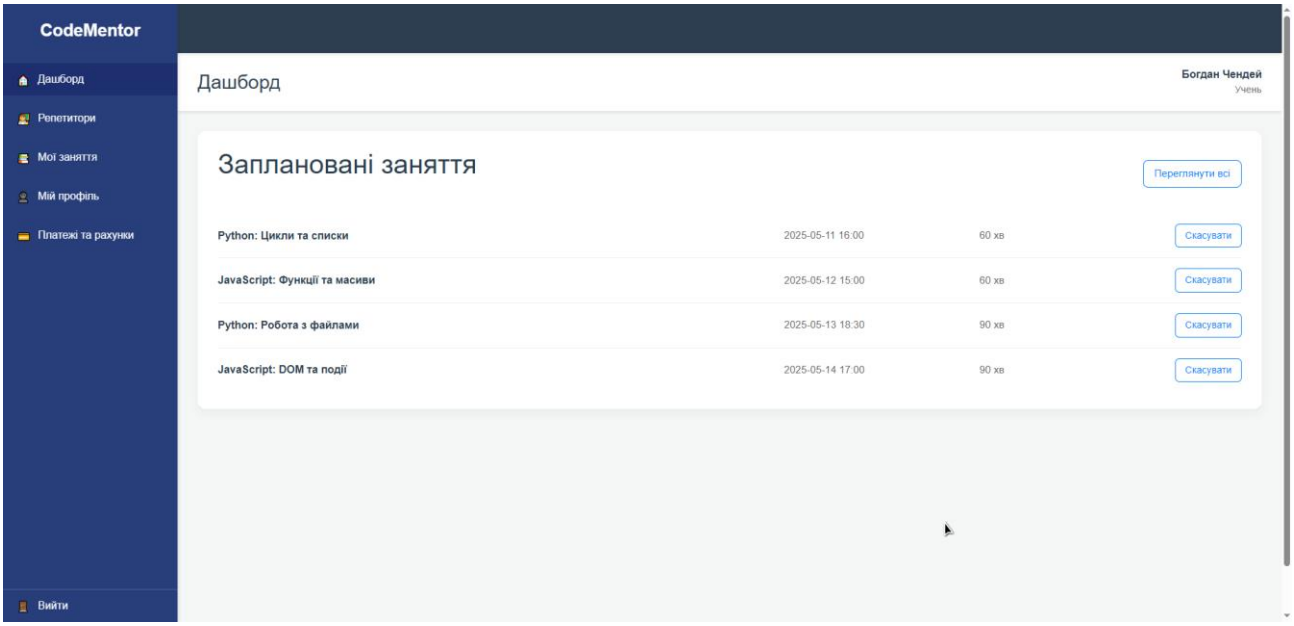


Рис. 2.3 — Дизайн панелі користувача

Профіль репетитора включає інформацію про кваліфікацію, досвід, а також можливість зв’язку зі студентом.

Сторінка управління заняттями дозволяє переглядати заплановані уроки, скасовувати.

2.4.2 Дизайн адміністративної панелі

Адміністративна панель забезпечує керування платформою та моніторинг активності користувачів. Вона включає дашборд із основною статистикою, кількістю користувачів, реєстрацій, занять, а також сповіщення про важливі події, як-от скарги, проблеми з оплатою та запити на верифікацію (див. рис. 2.4).

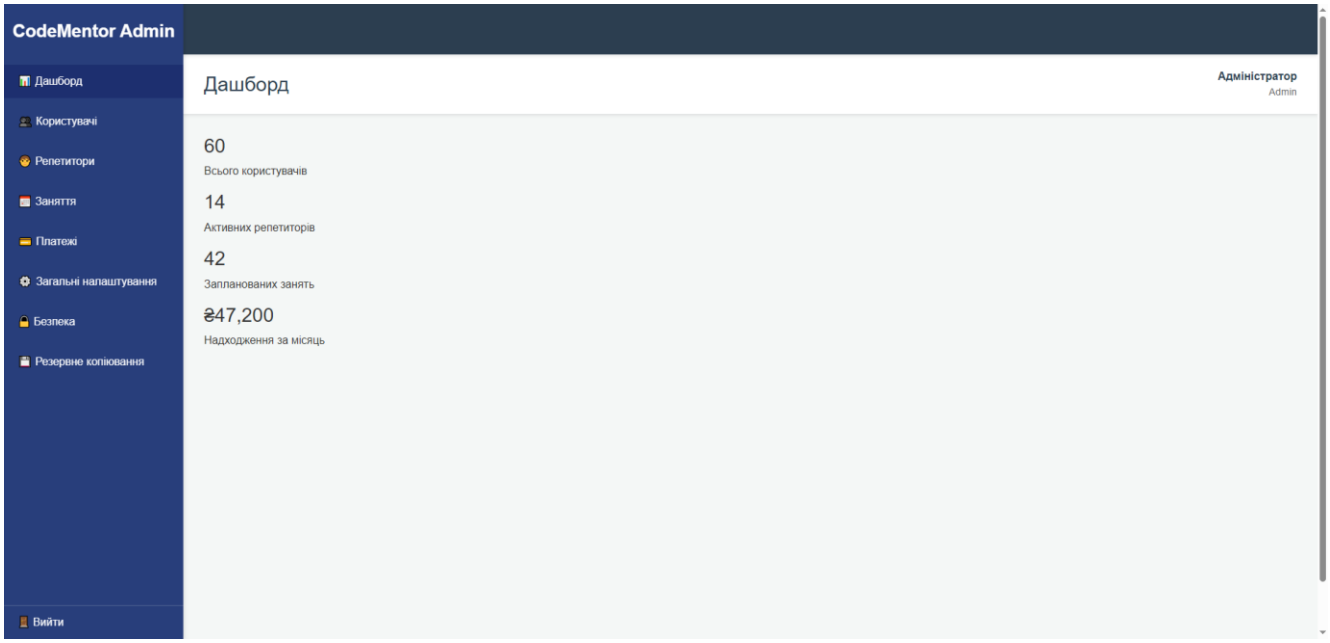


Рис. 2.4 — Дизайн адміністративної панелі

Адміністративна система дозволяє переглядати та редагувати профілі користувачів, верифікувати репетиторів, блокувати акаунти і надавати підтримку. Верифікація репетиторів включає перевірку документів, підтвердження навичок, запис співбесід і прийняття рішень із зворотним зв’язком.

Система моніторингу якості відстежує рейтинги, модерує відгуки, виявляє проблемних користувачів та реагує на скарги.

Панель фінансів забезпечує управління платежами, виплатами, поверненнями і комісіями з можливістю фільтрації та звітності.

Налаштування платформи дозволяють керувати комісіями, правилами скасування занять, тарифами, політиками конфіденційності, контентом і автоматичними сповіщеннями.

Система підтримки обробляє звернення користувачів, управляє тикетами та аналізує часті запити для покращення сервісу.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ

3.1 Розробка клієнтської частини

Клієнтська частина веб-платформи CodeMentor розроблена як одностороінковий додаток (SPA) з використанням React.js, що забезпечує високу інтерактивність і динамічне оновлення вмісту без перезавантаження сторінки [25].

На рисунку 3.1 зображено загальну структуру клієнтської частини платформи.

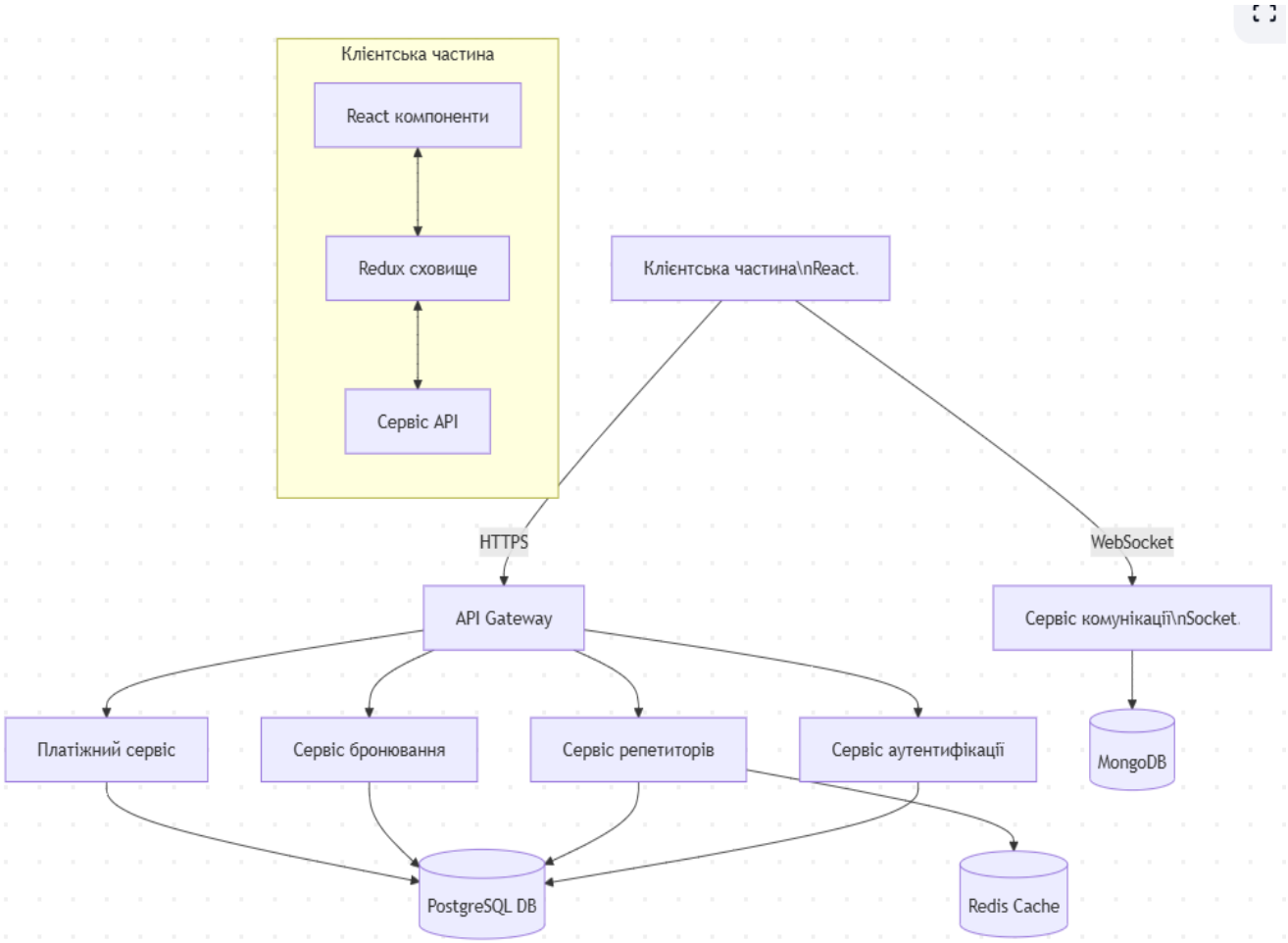


Рис. 3.1 — Архітектура клієнтської частини веб-платформи

Для управління станом додатку використовується Redux, для маршрутизації — React Router, а для створення стильного інтерфейсу — бібліотека Material-UI.

3.1.1 Управління станом клієнтського застосунку

Для управління глобальним станом застосунку та збереження даних про заплановані заняття було реалізовано контекст AppContext, який доступний у будь-якому компоненті завдяки хуку useAppContext. Даний підхід дозволяє централізовано керувати списком занять, зберігати їх у localStorage між сеансами користувача та забезпечує гнучкість у роботі з цими даними на стороні клієнта [26].

У наступному прикладі наведено реалізацію контексту:

```
import React, { createContext, useState, useContext, useEffect }  from 'react';

export const AppContext = createContext();

export const AppProvider = ({ children }) = > {
    const [scheduledLessons, setScheduledLessons] = useState(() = > {
        const saved = localStorage.getItem('scheduledLessons');
        return saved ? JSON.parse(saved) : [] ;
    });

    useEffect(() = > {
        localStorage.setItem('scheduledLessons', JSON.stringify(scheduledLessons));
    }, [scheduledLessons]);

    const addLesson = (lesson) = > {
        setScheduledLessons(prev = > [le sson, ...prev]);
    };

    const removeLesson = (id) = > {
        setScheduledLessons(prev = > prev.filter(lesson = > lesson.id != = id));
    };

    return (
        < AppContext.Provider value = { { scheduledLessons, addLesson, removeLesson
} } >
    {children}
    < / AppContext.Provider>
        );
};

export const useAppContext = () = > useContext(AppContext);
```

Лістинг 3.1 — Реалізація контексту для управління розкладом занять

Такий підхід спрощує керування даними на клієнтській стороні, зменшує потребу в додаткових запитах до сервера та покращує взаємодію користувача з інтерфейсом.

3.1.2 Інтеграція з API бекенда

Для взаємодії з сервером реалізовано сервіс-шар, що інкапсулює логіку HTTP-запитів, спрощує обробку помилок і забезпечує стабільну та централізовану роботу з API.

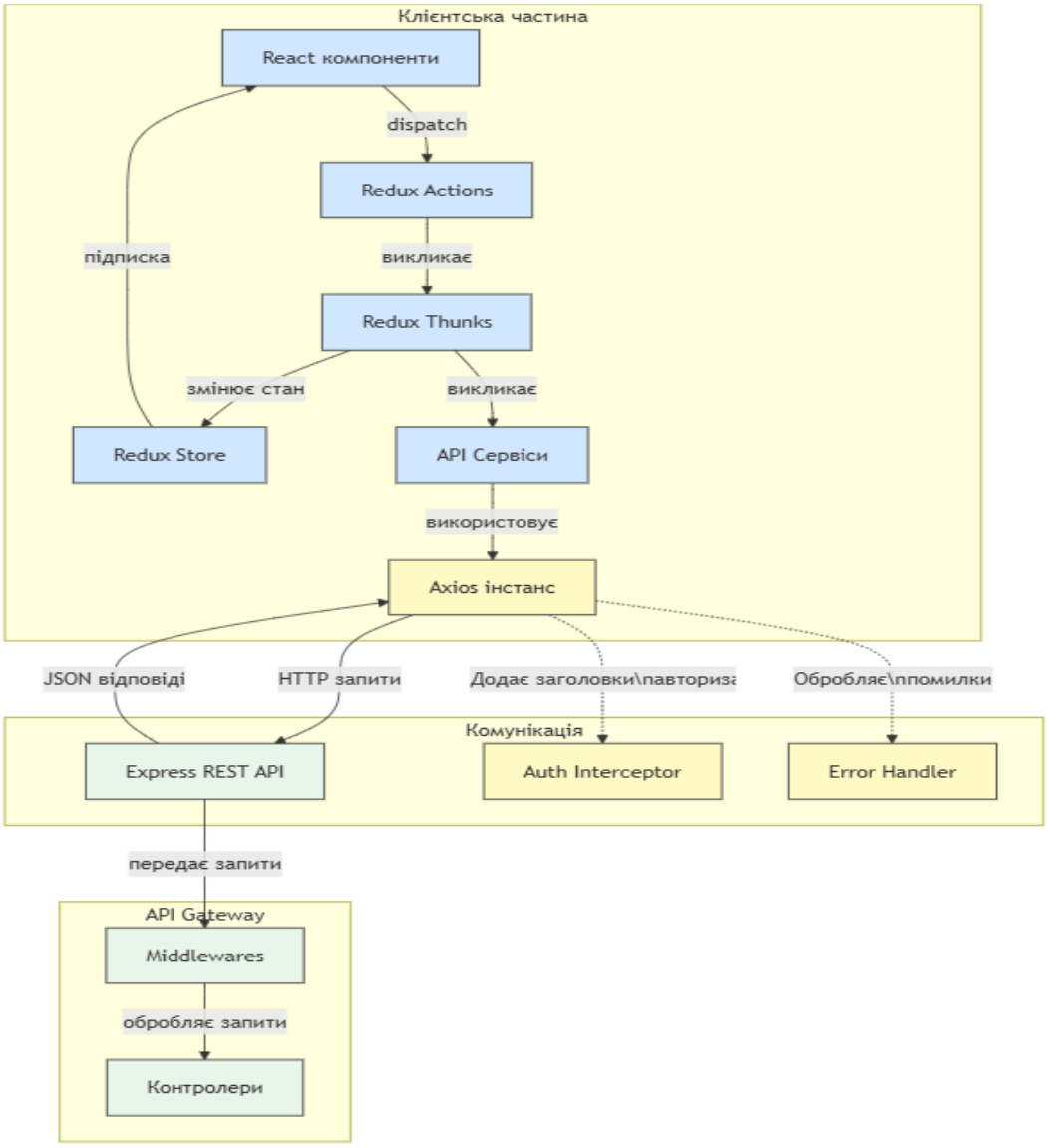


Рис 3.2 — Інтеграція клієнтської частини з API бекенду

Цей підхід дозволяє централізувати логіку взаємодії з сервером та спростити інтеграцію API в компонентах [27].

Нижче представлений приклад реалізації сервісу для роботи з API репетиторів.

```
import mentorsService from '../../services/mentorsService';
import {
    FETCH_MENTORS_REQUEST,
    FETCH_MENTORS_SUCCESS,
    FETCH_MENTORS_FAILURE,
    FETCH_MENTOR_DETAILS_REQUEST,
    FETCH_MENTOR_DETAILS_SUCCESS,
    FETCH_MENTOR_DETAILS_FAILURE
} from '../types';

// Дії для отримання списку репетиторів
export const fetchMentors = (filters) => async (dispatch) => {
    dispatch({ type: FETCH_MENTORS_REQUEST });

    try {
        const mentors = await mentorsService.getMentors(filters);
        dispatch({
            type: FETCH_MENTORS_SUCCESS,
            payload: mentors
        });
    } catch (error) {
        dispatch({
            type: FETCH_MENTORS_FAILURE,
            payload: error.message || 'Помилка при отриманні списку репетиторів'
        });
    }
};

// Дії для отримання деталей конкретного репетитора
export const fetchMentorDetails = (mentorId) => async (dispatch) => {
    dispatch({ type: FETCH_MENTOR_DETAILS_REQUEST });

    try {
        const mentor = await mentorsService.getMentorById(mentorId);
        dispatch({
            type: FETCH_MENTOR_DETAILS_SUCCESS,
            payload: mentor
        });
    } catch (error) {
        dispatch({
            type: FETCH_MENTOR_DETAILS_FAILURE,
            payload: error.message || 'Помилка при отриманні деталей репетитора'
        });
    }
};

export const fetchMentorReviews = (mentorId, page, limit) => async (dispatch) => {
      };
```

Лістинг 3.2 — Redux actions для роботи з репетиторами

Керування станом застосунку та обробки подій, пов’язаних із репетиторами, були створені відповідні Redux actions, які забезпечують зміну стану сторінки згідно з діями користувача.

3.2 Розробка серверної частини

Серверна частина веб-платформи CodeMentor реалізована з використанням Node.js та Express.js, організована за принципами RESTful API. Архітектура сервера побудована з дотриманням модульного підходу та патерну MVC (Model-View-Controller) [28].

3.2.1 Реалізація API

Для організації API був створений комплекс маршрутів (routes) та контролерів (controllers), які обробляють HTTP-запити та повертають відповіді у форматі JSON (див. рис. 3.3)

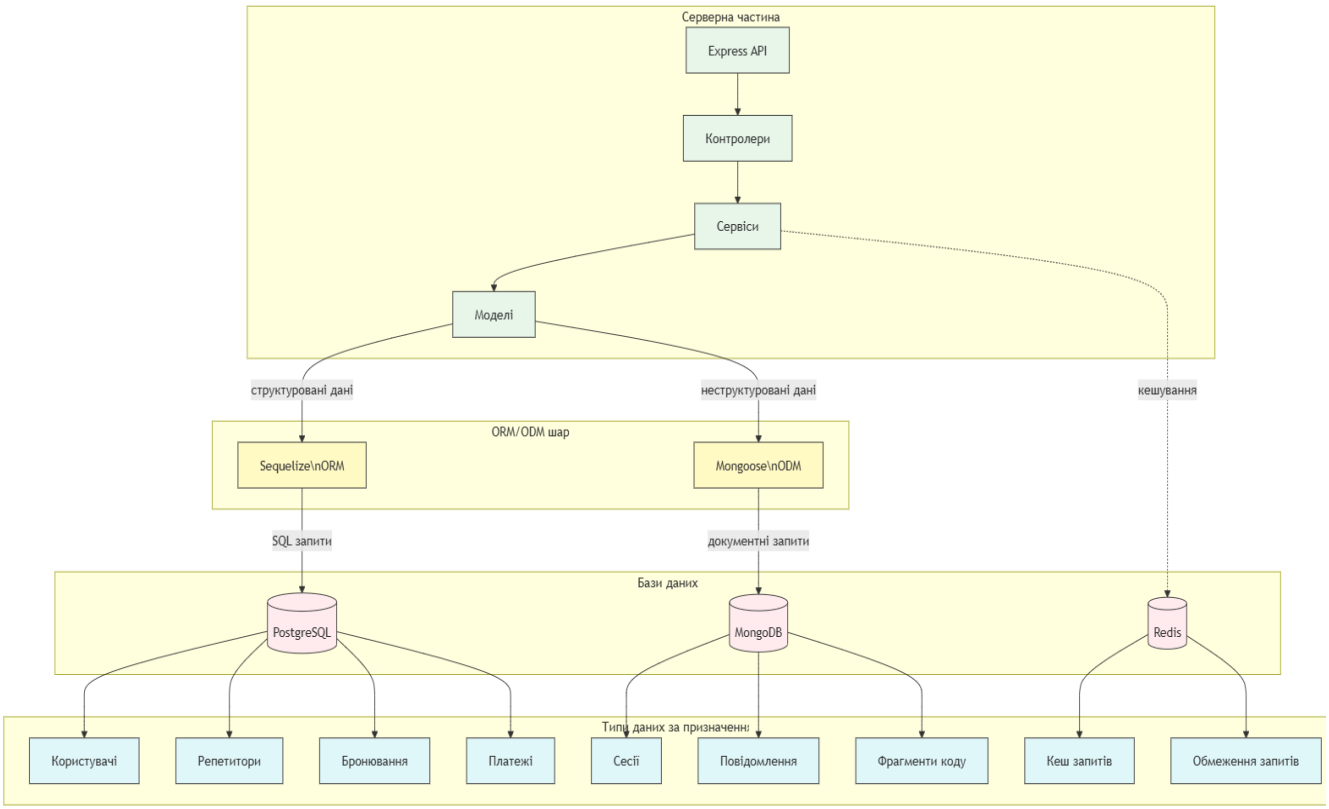


Рис. 3.3 — Інтеграція серверної частини з базами даних

Нижче представлений приклад реалізації маршрутів для роботи з репетиторами.

```
// routes/mentors.js
const express = require('express');
const router = express.Router();
const mentorsController = require('../controllers/mentorsController');
const authMiddleware = require('../middleware/authMiddleware');

router.get('/', mentorsController.getMentors);
router.get('/:id', mentorsController.getMentorById);
router.get('/:id/reviews', mentorsController.getMentorReviews);

router.get('/:id/availability', authMiddleware.authenticate,
mentorsController.getMentorAvailability);
router.post('/:id/book', authMiddleware.authenticate,
mentorsController.bookMentorSlot);

router.put('/:id/profile',
    authMiddleware.authenticate,
    authMiddleware.authorize(['mentor', 'admin']),
    mentorsController.updateMentorProfile
);

router.post('/:id/availability',
    authMiddleware.authenticate,
    authMiddleware.authorize(['mentor', 'admin']),
    mentorsController.setMentorAvailability
);

      module.exports = router;
```

Лістинг 3.3 — Маршрути для API репетиторів

Для кожного маршруту визначено відповідний контролер, який містить логіку обробки запитів. Нижче наведено приклад реалізації контролера для роботи з репетиторами.

3.2.2 Розробка системи авторизації та аутентифікації

Система авторизації використовує JWT для безпечної автентифікації без зберігання сесій на сервері. Реалізовано реєстрацію, вхід, відновлення паролю, підтвердження пошти та двофакторну аутентифікацію через одноразові коди. Паролі шифруються з використанням bcrypt [29].

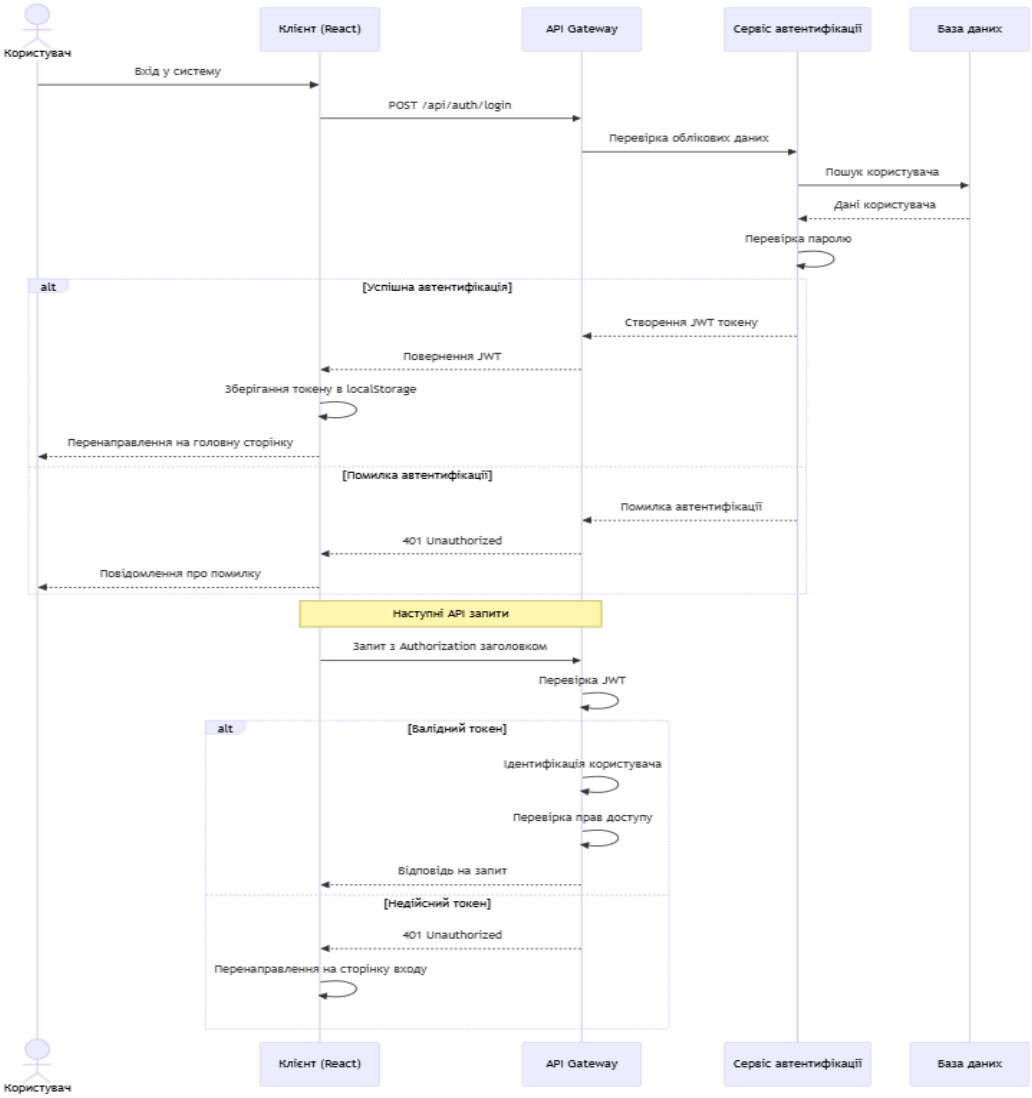


Рис. 3.4 — Процес автентифікації та авторизації

Для захисту приватних маршрутів API та перевірки прав доступу користувачів було реалізовано middleware, яке відповідає за аутентифікацію вхідних запитів на сервер.

```
exports.authenticate = async (req, res, next) => {
    try {
        const token = req.headers.authorization.split(' ')[1];
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = await User.findById(decoded.id).select('-password');
        next();
    } catch (error) {
        res.status(401).json({ message: 'Необхідна авторизація' });
    }

};
```

Лістинг 3.4 — Middleware для аутентифікації запитів

Для захисту маршрутів реалізовано контроль доступу на основі ролей (RBAC), що розмежовує права студентів, репетиторів і адміністраторів та дотримується принципу найменших привілеїв.

3.2.3 Інтеграція з базою даних

Для зберігання даних використано комбінований підхід з двома типами баз даних: PostgreSQL для структурованих даних та MongoDB для неструктурованого вмісту.

Для збереження інформації про репетиторів у базі даних MongoDB була створена відповідна модель. Вона описує структуру документа, визначає обов’язкові поля та типи даних, що зберігаються.

```
const MentorSchema = new mongoose.Schema({
    user: { type: mongoose.Schema.Types.ObjectId, ref:  'User' },
    hourlyRate: Number,
    skills: [{ type: mongoose.Schema.Types.ObjectId, ref:  'Skill' }],
    education: String,
    verified: { type: Boolean, default: false },
    averageRating: { type: Number, default: 0 }
      });
```

Лістинг 3.5 — Модель репетитора для MongoDB

Для роботи з PostgreSQL використано ORM Sequelize для зручної взаємодії з базою та управління міграціями. Моделі відображають основні сутності з встановленими зв’язками. MongoDB зберігає дані чату, сесій кодування та неструктурований контент, забезпечуючи гнучкість і швидкість операцій. Для цілісності даних впроваджено транзакції, особливо в платежах і бронюваннях. Комунікація в реальному часі реалізована через WebSocket та WebRTC, з основним компонентом Socket.IO сервером [30].

Для реалізації функціоналу реального часу в застосунку було налаштовано сервер Socket.IO, що забезпечує двонаправлену комунікацію між клієнтом і сервером.

```
const io = require('socket.io')(server);

io.on('connection', (socket) => {

    socket.on('join-session', sessionId => {
        socket.join(`session:${sessionId}`);

});
    socket.on('chat-message', data => {

        io.to(`session:${data.sessionId}`).emit('new-message', data);

    });
});
```

Лістинг 3.6 — Налаштування Socket.IO сервера

Середовище спільного кодування створено на базі CodeMirror та алгоритмів operational transformation для синхронного редагування коду без конфліктів.

Система обміну повідомленнями підтримує форматований текст із підсвічуванням коду, що полегшує обговорення під час занять.

3.3 Реалізація системи оплати та бронювання занять

Система бронювання занять дозволяє студентам резервувати часові слоти у репетиторів з урахуванням їхньої доступності. Реалізовано алгоритм, який враховує часові пояси обох сторін для коректного відображення розкладу.

Така система значно спрощує процес планування та координації між студентами та репетиторами. Завдяки автоматичному врахуванню різниці в часових поясах, користувачі можуть бути впевнені у точності відображеного розкладу, незалежно від свого географічного розташування, що робить процес бронювання максимально зручним та ефективним для обох сторін взаємодії.

Для реалізації функціоналу бронювання занять було розроблено відповідну функцію.

```
exports.bookSession = async (req, res) => {
    try {
        const { mentorId, startTime, duration } = req.body;
        // Логіка перевірки доступності та створення бронювання
        const booking = await Booking.create({
            student: req.user.id,
            mentor: mentorId,
            startTime: new Date(startTime),
            duration
        });
        res.status(201).json({ success: true, data: booking });
    } catch (error) {
        res.status(400).json({ success: false, message: error.message });
    }
};
```

Лістинг 3.7 — Функція бронювання заняття

Система оплати інтегрована з платіжними шлюзами Stripe і PayPal, забезпечуючи безпечні транзакції без зберігання даних карт на сервері. Реалізовано автоматичний розподіл коштів між репетитором і платформою з урахуванням комісії. Впроваджено гнучку політику скасування бронювань з різними умовами повернення коштів.

3.4 Тестування та оптимізація

Для забезпечення якості та стабільності коду впроваджено багаторівневу стратегію тестування: модульні, інтеграційні та end-to-end тести. Модульні тести написані з використанням Jest і охоплюють ключові компоненти системи. Такий підхід дозволяє виявляти помилки на ранніх етапах, підвищує надійність і відмовостійкість платформи, забезпечуючи стабільну роботу після оновлень [31].

Для перевірки роботи API реалізовано автоматизовані модульні тести, які перевіряють функціональність окремих компонентів. Це допомагає виявляти помилки на ранніх етапах і забезпечує стабільність системи.

Одним із прикладів протестованих методів є функція bookSession.

```
exports.bookSession = async (req, res) => {
    try {
        const { mentorId, startTime, duration } = req.body;
        // Логіка перевірки доступності та створення бронювання
        const booking = await Booking.create({
            student: req.user.id,
            mentor: mentorId,
            startTime: new Date(startTime),
            duration
        });
        res.status(201).json({ success: true, data: booking });
    } catch (error) {
        res.status(400).json({ success: false, message: error.message });
    }

};
```

Лістинг 3.8 — Приклад модульного тесту для API

Інтеграційне тестування проводилось для перевірки взаємодії між різними модулями системи, особливо для критичних процесів, таких як авторизація, бронювання та платежі. End-to-end тестування з використанням Cypress дозволило перевірити функціональність системи в умовах, наближених до реальних.

Для оптимізації продуктивності впроваджено кешування на різних рівнях: клієнтське кешування з використанням Service Worker, серверне кешування з Redis для частих запитів та CDN для статичних ресурсів. Це дозволило значно знизити час відгуку системи та підвищити пропускну здатність серверів.

Для моніторингу системи в реальному часі впроваджено рішення на базі ELK Stack (Elasticsearch, Logstash, Kibana), що дозволяє відстежувати продуктивність, виявляти вузькі місця та оперативно реагувати на інциденти.

Проведено оптимізацію баз даних з використанням індексів та денормалізації для часто використовуваних запитів. Для масштабування впроваджено шардинг MongoDB колекцій з високим навантаженням.

РОЗДІЛ 4. АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

4.1 Аналіз отриманих результатів розробки

Розроблена веб-платформа CodeMentor успішно реалізує поставлені цілі та відповідає визначеним функціональним і нефункціональним вимогам. У результаті розробки отримано комплексну систему для надання послуг репетиторства з програмування, що має наступні ключові характеристики:

Повнофункціональний інтерфейс користувача, реалізований як одностороінковий додаток (SPA) на базі React.js, забезпечує швидку навігацію та високу інтерактивність. Інтерфейс адаптований для різних пристроїв (комп'ютери, планшети, смартфони) і оптимізований для забезпечення позитивного користувацького досвіду. Ключовий елемент інтерфейсу — середовище проведення онлайн-занять, яке поєднує інтегроване середовище розробки для спільного програмування, чат та віртуальну дошку [32].

Серверна частина на базі мікросервісної архітектури з Node.js та Express.js забезпечує масштабованість, надійність і ефективне використання ресурсів платформи.

Система управління даними, що використовує поліглотний підхід з комбінуванням PostgreSQL для структурованих даних та MongoDB для неструктурованих, дозволяє ефективно організувати зберігання та доступ до різних типів інформації. Впровадження кешування з використанням Redis значно підвищує швидкодію системи при частих запитах.

Система комунікації на базі WebSocket і WebRTC забезпечує якісну взаємодію користувачів у реальному часі, підтримуючи синхронне спільне програмування з мінімальними затримками.

Система оплати, інтегрована з провідними платіжними шлюзами, забезпечує безпечну і обробку фінансових транзакцій. Реалізовано гнучкі механізми розподілу коштів між репетиторами та платформою, а також систему повернення платежів у випадку скасування занять.

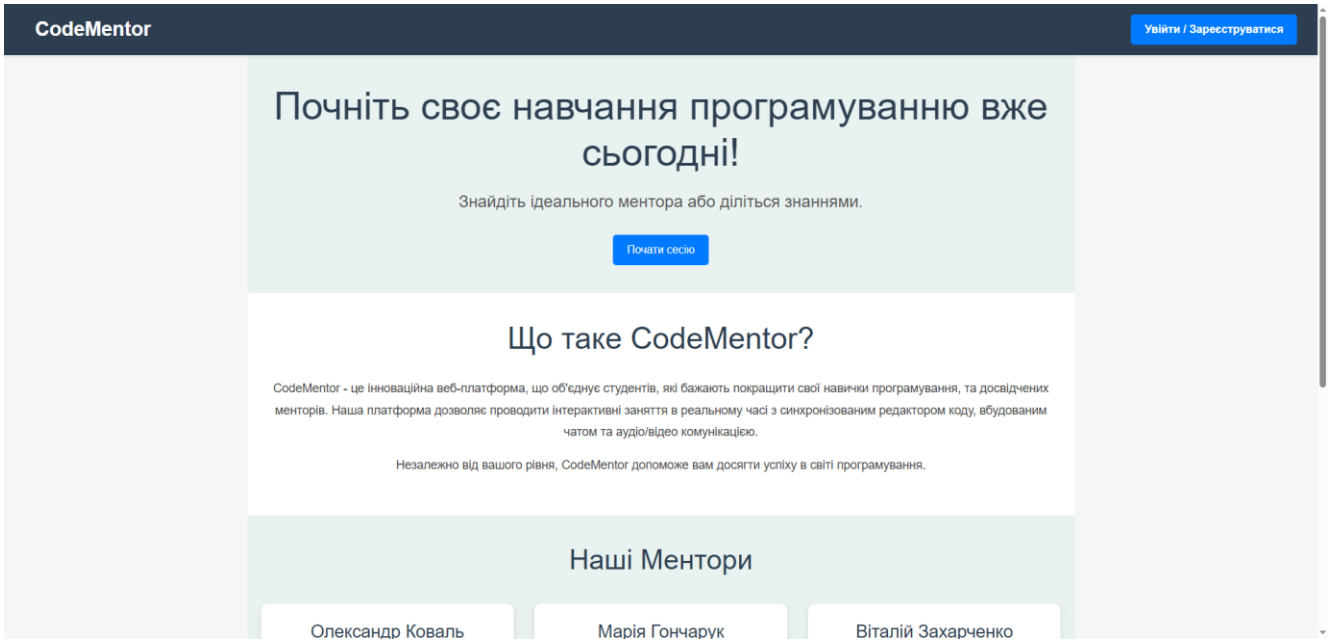


Рис. 4.1 — Скріншот головної сторінки платформи CodeMentor

Порівняння результатів із початковими вимогами показало успішну реалізацію всіх ключових функцій, а також високий рівень продуктивності, безпеки та масштабованості. Важливим досягненням є ефективне середовище для онлайн-занять з програмування, що поєднує всі необхідні інструменти в одному інтерфейсі.

4.2 Оцінка ефективності розробленої платформи

Для оцінки ефективності платформи проведено тестування за участю студентів та програмістів, а також технічні перевірки продуктивності, безпеки і зручності. Результати підтвердили високу якість системи за основними показниками:

Швидкодія системи відповідає вимогам — середній час завантаження сторінок становить 1,3 секунди. Критичні компоненти, як середовище занять, працюють стабільно навіть при високому навантаженні. Кешування даних знизило час відгуку API на 68% для повторних запитів [33].

Масштабованість системи. Тестування під навантаженням показало, що система здатна ефективно обробляти до 1000 одночасних користувачів без істотного зниження продуктивності. Мікросервісна архітектура дозволяє легко масштабувати окремі компоненти при зростанні кількості користувачів.

Безпека системи. Проведений аудит безпеки не виявив критичних вразливостей у системі. Впроваджені механізми захисту (шифрування даних, безпечне зберігання паролів, двофакторна аутентифікація, захист від CSRF та XSS атак) забезпечують високий рівень захисту персональних даних та фінансових транзакцій користувачів.

Порівняльний аналіз з існуючими аналогами показує, що розроблена платформа має ряд конкурентних переваг, зокрема:

1. Більш розвинене середовище для спільного програмування з можливістю виконання коду в реальному часі
2. Гнучка система пошуку та фільтрації репетиторів за спеціалізацією
3. Адаптивний інтерфейс, оптимізований для використання на різних пристроях

4.3 Перспективи розвитку веб-платформи

Аналіз ринку освітніх онлайн-платформ та тенденцій у сфері навчання програмуванню дозволяє визначити ряд перспективних напрямків для подальшого розвитку платформи CodeMentor:

Розширення освітнього компоненту. Інтеграція структурованих навчальних матеріалів, інтерактивних завдань та тестів дозволить перетворити платформу з сервісу для індивідуальних занять на комплексне освітнє середовище. Впровадження системи управління навчанням (LMS) забезпечить можливість створення та проходження повноцінних курсів з програмування.

Використання штучного інтелекту для персоналізації навчання, аналізу коду та рекомендацій підвищує ефективність процесу. Також AI може автоматично підбирати репетиторів за навичками студентів і стилем викладання.

Розробка нативних мобільних додатків для iOS та Android розширить доступ до платформи, дозволяючи студентам займатися з будь-якого місця. Додатки можуть містити спеціальні функції для мобільного навчання програмуванню.

4.4 Рекомендації щодо просування та розширення функціоналу

На основі проведеного аналізу ринку та результатів розробки, можна сформулювати наступні рекомендації щодо просування та розширення функціоналу платформи CodeMentor:

Рекомендується спочатку запустити бета-версію платформи з обмеженою кількістю користувачів для тестування та збору зворотного зв’язку. Після доопрацювання провести повноцінний запуск із маркетинговою кампанією, орієнтованою на студентів і професіоналів програмування.

Пріоритетні напрямки розширення функціоналу:

1. Інтеграція системи автоматизованої перевірки коду, що дозволить студентам отримувати зворотний зв'язок навіть без участі репетитора
2. Розробка інструментів для створення інтерактивних навчальних матеріалів репетиторами
3. Впровадження системи гейміфікації для підвищення мотивації студентів
4. Створення відкритого API для інтеграції з іншими освітніми платформами та інструментами для розробників
5. Розробка функціоналу групових занять для навчання команд та проведення воркшопів.

РОЗДІЛ 5. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

5.1 Роль центральної нервової системи в трудовій діяльності людини.

Центральна нервова система (ЦНС) є головною координуючою структурою людського організму, що забезпечує інтеграцію та взаємодію фізіологічних, психоемоційних та когнітивних процесів, необхідних для виконання трудової діяльності. Вона включає головний та спинний мозок, які разом формують єдину систему управління всім організмом, зокрема й трудовою активністю людини.

У сучасних умовах науково-технічного прогресу, де праця часто має інтелектуально-інформаційний характер, особливо зростає роль ЦНС як регулятора професійної поведінки та ефективності. Саме вона забезпечує здатність людини до сприйняття інформації, її аналізу, прийняття рішень, планування дій, моторної координації, контролю та корекції результатів діяльності. Центральна нервова система функціонує як центр керування, що миттєво реагує на зміну зовнішніх і внутрішніх умов, адаптуючи організм до нових вимог виробничого процесу.

Завдяки роботі ЦНС забезпечується гармонійна взаємодія між сенсорними (аналізаторами) та моторними (руховими) функціями. Наприклад, у процесі керування технікою чи використання комп’ютерного обладнання, людина повинна одночасно сприймати візуальну інформацію, аналізувати її, приймати рішення та координувати рухи рук. Усе це здійснюється завдяки злагодженій роботі нейронних мереж головного мозку, зокрема кори великих півкуль, підкіркових структур, мозочка та спинного мозку.

Навантаження на центральну нервову систему в процесі трудової діяльності залежить від складності завдань, темпу роботи, рівня відповідальності, ступеня автоматизації виробництва, психоемоційного клімату в колективі та інших факторів. Особливо важливою є здатність ЦНС до швидкої адаптації, стресостійкості та тривалого підтримання уваги при монотонних або рутинних видах діяльності, які характерні для операторських, офісних або інженерних спеціальностей.

Порушення в роботі центральної нервової системи, викликані перевтомою, стресом, недостатнім відпочинком або впливом несприятливих чинників виробничого середовища (таких як шум, вібрація, освітлення, температура, монотонність), безпосередньо впливають на працездатність працівника. Це може проявлятися у вигляді зниження концентрації уваги, уповільнення реакцій, помилок у діях, зниження продуктивності праці та навіть виникнення небезпечних ситуацій, що загрожують здоров’ю або життю.

Особливо вразливою є ЦНС у молодих спеціалістів, які ще не мають сформованих механізмів адаптації до психофізіологічних навантажень, а також у працівників, які виконують відповідальну, напружену або рутинну роботу в умовах дефіциту часу чи підвищеної відповідальності. Здатність центральної нервової системи до тривалого збереження працездатності залежить від низки чинників: індивідуальних фізіологічних особливостей, професійного досвіду, тренованості нервових процесів, наявності умов для відпочинку.

З фізіологічної точки зору, діяльність центральної нервової системи у процесі праці відбувається циклічно й включає певні фази, які тісно пов’язані з рівнем активації, працездатності та втоми організму. Традиційно виділяють три основні фази функціонального стану ЦНС під час трудової діяльності: фазу оптимальної активації, фазу нестійкої працездатності та фазу гальмування (перевтоми).

Фаза оптимальної активації характеризується повною мобілізацією психофізіологічних ресурсів, високою швидкістю психічних процесів, стійкою увагою, доброю пам’яттю та високим рівнем працездатності. У цей період людина здатна вирішувати складні завдання, приймати зважені рішення, зосереджено працювати над об'єктами з високими вимогами до точності та швидкості. Саме ця фаза є найбільш бажаною впродовж робочого дня і має бути підтримана через раціональну організацію праці.

Фаза нестійкої працездатності настає після тривалої або інтенсивної роботи, коли активність ЦНС починає знижуватися. У цій фазі спостерігаються періоди зниження уваги, уповільнення реакцій, частіші помилки в діях. Проте при короткочасному відпочинку, зміні виду діяльності або активізації зовнішніх

стимулів (наприклад, через емоційне підбадьорення чи зміну обстановки), працездатність ще може тимчасово відновитися.

Фаза гальмування або перевтоми є завершальною у циклі праці. Вона супроводжується пригніченням функцій ЦНС, домінуванням процесів гальмування в корі головного мозку, що призводить до різкого зниження працездатності, апатії, неуважності, емоційної нестабільності. Перебування працівника в цій фазі створює підвищений ризик виробничих травм та професійних помилок. На цьому етапі відновити функціональну активність можливо лише шляхом повноцінного відпочинку або сну.

Розуміння фазової природи функціонування ЦНС дозволяє ефективніше планувати графік праці, чергування навантажень, а також упроваджувати профілактичні заходи для підтримання стабільного психофізіологічного стану працівника. Важливо уникати перенавантаження, тривалого перебування в умовах стресу, порушення режиму сну — факторів, що сприяють передчасному переходу в фазу гальмування.

Для зниження функціонального навантаження на ЦНС в умовах трудової діяльності передбачаються різні заходи: раціональна організація праці та відпочинку, оптимізація робочого місця, ергономічне проєктування робочих процесів, чергування розумової та фізичної активності, профілактика емоційного вигорання. Важливо також враховувати біоритми організму, забезпечувати повноцінний сон і збалансоване харчування, що позитивно впливають на відновлювальні можливості [34].

Урахування ролі центральної нервової системи у трудовій діяльності є важливим для забезпечення безпеки праці та збереження здоров’я працівників. В умовах розумової або монотонної роботи стабільний стан ЦНС допомагає підтримувати увагу, швидкість реакції та уникати помилок. Раціональна організація праці, відпочинку та психологічна підтримка сприяють профілактиці перевтоми та підвищенню ефективності трудового процесу.

5.2 Шляхи збереження працездатності та підвищення продуктивності праці.

Раціональне використання трудових ресурсів, збереження працездатності працівників та підвищення продуктивності праці є одними з головних умов ефективного функціонування будь-якої організації. Особливо це актуально в галузях, пов’язаних із високими інтелектуальними та психоемоційними навантаженнями, таких як ІТ-сфера. Працездатність працівника є результатом злагодженої роботи фізіологічних систем організму, насамперед центральної нервової, а також впливу зовнішніх факторів виробничого середовища. Одним із ключових шляхів підтримки високої працездатності є раціональна організація праці, яка передбачає оптимальне поєднання робочих і відновлювальних процесів. Правильне планування робочого дня, регламентовані перерви, чергування видів діяльності та контроль за навантаженням допомагають уникнути передчасної втоми, зниження уваги та помилок у виконанні завдань. Рекомендується використовувати науково обґрунтовані режими праці та відпочинку, що відповідають характеру виконуваної роботи.

Наступним важливим чинником є створення сприятливого мікроклімату на робочому місці. Температурний режим, вологість повітря, рівень шуму, освітленість та вентиляція мають відповідати нормативним вимогам. Несприятливі умови навколишнього середовища викликають додаткове навантаження на організм, що призводить до швидшого настання втоми. Зокрема, при роботі з комп’ютерною технікою важливе значення має організація робочого місця відповідно до принципів ергономіки, що дозволяє зменшити навантаження на зір, опорно-руховий апарат та нервову систему. Крім того, велике значення мають соціально-психологічні чинники, такі як морально-психологічний клімат у колективі, стиль керівництва, система мотивації. Дружня атмосфера, справедливий розподіл обов’язків, визнання досягнень працівників сприяють позитивному емоційному настрою, що безпосередньо впливає на загальний рівень продуктивності [35].

Ще одним важливим напрямом підтримання працездатності є профілактика перевтоми та своєчасне відновлення функціонального стану організму. Перевтома виникає внаслідок тривалого або інтенсивного навантаження без достатнього відпочинку й проявляється у вигляді зниження уваги, координації рухів, швидкості мислення, а також погіршення емоційного стану. Для її запобігання доцільно впроваджувати активні форми відпочинку, фізичні вправи під час перерв, гімнастику для очей і спини, особливо при тривалій роботі за комп'ютером. У сучасних умовах цифровізації та стрімкого розвитку технологій особливої ваги набуває адаптація робочих процесів до змін у характері праці. Важливо враховувати індивідуальні особливості працівників, рівень їхньої підготовки, фізичний і психоемоційний стан. Систематичний моніторинг працездатності, гнучкий підхід до формування графіків роботи, впровадження здоров'язбережувальних технологій є ключовими елементами стратегії підтримання високої продуктивності та професійного довголіття працівників.

У практиці охорони праці ефективними є також заходи з підвищення психоемоційної стійкості працівників. Сюди належать тренінги з управління стресом, створення умов для неформального спілкування, підтримка доброзичливої атмосфери в колективі. Працівник, який відчуває психологічний комфорт, менш схильний до втоми та професійного вигорання, а його продуктивність залишається стабільною впродовж тривалого часу. Окрему увагу слід приділяти мотивації до праці — як матеріальній, так і нематеріальній. Підвищення рівня зацікавленості в результатах роботи, можливість професійного розвитку, кар'єрного зростання, визнання досягнень сприяють формуванню внутрішньої мотивації, яка є потужним стимулом для підвищення ефективності праці. Працівники, які мають чітке бачення своїх перспектив, виявляють більшу ініціативу, відповідальність та відданість роботі [36].

Отже, підвищення продуктивності праці та збереження працездатності працівників є багатофакторним процесом, що вимагає комплексного підходу: від оптимізації умов праці до формування сприятливого соціального середовища. Впровадження сучасних методів організації працею.

ВИСНОВКИ

Отже, у цій роботі ми розробили веб-платформу "CodeMentor" для надання послуг репетиторства з програмування з використанням сучасного фреймворку React.js. Проаналізувавши сучасні технології веб-розробки, ми дослідили існуючі платформи для надання освітніх послуг та визначили функціональні й нефункціональні вимоги до системи, які стали фундаментом для подальшого проєктування. Враховуючи визначені вимоги, ми спроєктували архітектуру веб-платформи на основі мікросервісного підходу, що забезпечує гнучкість, масштабованість та стійкість до відмов. Для зберігання даних розробили комбіновану структуру бази даних, використовуючи PostgreSQL для структурованих даних (користувачі, заняття, транзакції) та MongoDB для динамічного контенту (чати, повідомлення в реальному часі), що дозволило оптимізувати продуктивність для різних типів даних і операцій.

При розробці клієнтської частини платформи ми використали React.js, що дозволило створити динамічний інтерфейс користувача з швидкими переходами між сторінками без повного перезавантаження. Ми реалізували систему базових компонентів, які забезпечують уніфікований дизайн та спрощують розробку інтерфейсу. Особлива увага була приділена розробці інтерактивних елементів, таких як календар для бронювання занять, система пошуку з фільтрацією та середовище для спільного програмування.

Серверна частина веб-платформи була реалізована з використанням Node.js та Express.js, організована за принципами RESTful API та патерном MVC. Ми розробили комплексну систему аутентифікації та авторизації на основі JSON Web Tokens, що забезпечує безпечний механізм автентифікації без необхідності зберігання стану сесії на сервері. Важливим компонентом стала система комунікації в реальному часі, реалізована з використанням технологій WebSocket та WebRTC. Для проведення онлайн-занять ми створили інтегроване середовище, яке поєднує редактор коду та віртуальну дошку, що забезпечує ефективне навчання

програмуванню в режимі реального часу. Ми реалізували систему бронювання занять, та інтегрували платіжні шлюзи Stripe та PayPal для безпечної обробки транзакцій.

Для забезпечення якості коду та стабільності системи ми впровадили комплексну стратегію тестування, що включає модульні, інтеграційні та end-to-end тести. Також провели оптимізацію продуктивності, впровадивши кешування на різних рівнях: клієнтське кешування з використанням Service Worker, серверне кешування з Redis для частих запитів та CDN для статичних ресурсів.

На основі проведеного аналізу ми визначили перспективні напрямки розвитку платформи CodeMentor, включаючи розширення освітнього компоненту, впровадження елементів штучного інтелекту для персоналізації навчання, розробку мобільного додатку та інтеграцію з популярними платформами для розробників. Розроблена платформа має ряд конкурентних переваг перед існуючими аналогами, включаючи більш розвинене середовище для спільного програмування, гнучку систему пошуку репетиторів та інтегрований механізм верифікації спеціалістів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Остапчук О., Цуприк Г. Технічні особливості взаємодії між клієнтом та сервером у реальному часі. Матеріали X науково-технічної конференції, інформаційні моделі, системи та технології Тернопільського національного технічного університету імені Івана Пулюя. Тернопіль: ТНТУ, 2022. С. 124–124.

2. Казмірчук В., Цуприк Г. Розробка автоматизованої інформаційної системи обробки даних. Матеріали VIII науково-технічної конференції „Інформаційні моделі, системи та технології. Тернопіль: ТНТУ, 2020. С. 148–148.

3. Пасіка Д., Цуприк Г. Розробка односторінкового веб-застосунку з використанням VUE. JS і REACT: порівняльний аналіз продуктивностіб користувацького досвіду та доцільності. Матеріали XI науково-технічної конференції „Інформаційні моделі, системи та технології”. Тернопіль: ТНТУ, 2023. С. 218–219.

4. Typescriptlang. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 02.05.2025).

5. 2. Vuejs. Docs. URL: https://vuejs.org/guide/introduction.html (дата звернення: 30.04.2025).

6. Кантелон М., Хартер М., Ганнон Т., Морган Н. Дж. Node.js у дії. Київ: Видавнича група «Діалектика», 2018. 360 с.

7. Флауерс, М. Ruby on Rails: швидка розробка веб-додатків М. Флауерс. Харків : Видавництво «Фоліо», 2020. 350 с.

8. RabbitMQ: Messaging that just works URL: https://www.rabbitmq.com/documentation.html (дата звернення: 03.05.2025).

9. Microsoft SQL Server Documentation URL: https://learn.microsoft.com/sql/(дата звернення: 07.04.2025)

10. Google Web Fundamentals. URL: https://developers.google.com/web (дата звернення: 01.03.2025)

11. Сущик, А. М. Метод створення документації для REST API на основі тестів: 121 Інженерія програмного забезпечення А. М. Сущик. Київ: КПІ ім. Ігоря Сікорського, 2019. 130 с

12. Шустер, О. Прогресивні веб-додатки: ключові аспекти та переваги для бізнесу О. Шустер. Київ: FreshTech, 2024. 15 с.

13. Codecademy. Learn to Code. URL: https://www.codecademy.com/ (дата звернення: 20.03.2025).

14. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 02.03.2025)

15. PCI Security Standards Council. Payment Card Industry Data Security Standard (PCI DSS). Version 3.2.1. 2018. URL: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf (дата звернення: 05.04.2025).

16. Express.js Documentation. URL: https://expressjs.com/en/guide/routing (дата звернення: 02.03.2025)

17. WebSocket API URL:https://developer.mozilla.org/docs/Web/API/Web Socket (дата звернення: 12.04.2025)

18. Ньюмен, С. Building Microservices: Designing Fine-Grained Systems С. Ньюмен. Boston: O'Reilly Media, 2015. 300 с.

19. Доссот, Д., Йоханссон, Л. RabbitMQ Essentials Second Edition Д. Доссот, Л. Йоханссон: Packt Publishing, 2020. 154 с.

20. Зеленюк А., Крропп А. Apache Kafka in Action: From basics to production А. Зеленін, А. Крропп. Нью-Йорк: Manning, 2025. 370 с.

21. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ current (дата звернення: 09.03.2025)

22. MongoDB Documentation. URL: https://docs.mongodb.com (дата звернення: 09.03.2025)

23. Material Design Guidelines URL: https://material.io/design (дата звернення: 08.03.2025)

24. DigitalOcean Tutorials. URL: https://www.digitalocean.com/community/tutorials (дата звернення: 23.03.2025)

25. Redux.js Documentation. URL: https://redux.js.org/introduction/getting-started (дата звернення: 28.03.2025)

26. WebRTC Documentation. URL: https://webrtc.org/getting-started/overview (дата звернення: 18.04.2025)

27. Socket.IO Documentation. URL: https://socket.io/docs/v4/tutorial/introduction (дата звернення: 22.04.2025)

28. Stripe Documentation. URL: https://stripe.com/docs (дата звернення: 22.04.2025)

29. Гупта У. Secure User Authentication with JWT, Bcrypt, and Node.js URL: https://medium.com/%40utkarsh.gupta0311/secure-user-authentication-with-jwt-bcrypt-and-node-js-78c7bb2d86a1 (дата звернення: 29.03.2025)

30. OWASP Foundation. OWASP Top 10 Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 25.03.2025)

31. Di Francesco, H. The Jest Handbook: Advanced JavaScript Testing Patterns Х. Ді Франческо. 2025. 100 с.

32. BairesDev. (2023). React Single Page Application. URL: https://www.bairesdev.com/blog/react-spa-single-page-application/ (дата звернення: 19.05.2025).

33. Regmi, S., & Phakami Pun, C. (2024). GPT Semantic Cache: Reducing LLM Costs and Latency via Semantic Embedding Caching. arXiv. URL: https://arxiv.org/abs/2411.05276 (дата звернення: 20.05.2025).

34. Бедрій Я.І. Основи охорони праці : навч. посіб. 4-е вид. перероб. і доп. Тернопіль : Навчальна книга – Богдан, 2018. 240 с

35. Безпека життєдіяльності та охорона праці : підруч. / В.В. Сокуренко, О.М. Бандурка та ін. – Харків : ХНУВС, 2021. 308 с.

36. ДСТУ ISO 9241-5:2004 Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та до робочої пози (ISO 9241-5:1998, IDT).

ДОДАТКИ

ДОДАТОК А — Лістинг

1. МОДЕЛІ ДАНИХ

// Модель користувача (студент або ментор)

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  role: { type: String, enum: ['student', 'mentor', 'admin'], default:
'student' },
  avatar: { type: String, default: '/images/default-avatar.png' },
  isVerified: { type: Boolean, default: false },
  createdAt: { type: Date, default: Date.now }
});
```

// Хешування паролю перед збереженням

```
UserSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password, 10);
  next();
});
```

// Модель профілю ментора

```
const MentorProfileSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true
},
  specialization: { type: String, required: true },
  skills: [{ type: String, required: true }],
  hourlyRate: { type: Number, required: true },
  experience: { type: Number, required: true }, // років досвіду
  rating: { type: Number, default: 0 },
  availability: {
    monday: [{ start: String, end: String }],
    tuesday: [{ start: String, end: String }],
    wednesday: [{ start: String, end: String }],
    thursday: [{ start: String, end: String }],
    friday: [{ start: String, end: String }],
```

```
    saturday: [{ start: String, end: String }],
    sunday: [{ start: String, end: String }]
  }
});
```

// Модель бронювання заняття

```
const BookingSchema = new mongoose.Schema({
  student: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  mentor: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  startTime: { type: Date, required: true },
  endTime: { type: Date, required: true },
  topic: { type: String, required: true },
  status: {
    type: String,
    enum: ['pending', 'confirmed', 'completed', 'cancelled'],
    default: 'pending'
  },
  price: { type: Number, required: true },
  sessionUrl: { type: String },
  createdAt: { type: Date, default: Date.now }
});
```

// Модель відгуку

```
const ReviewSchema = new mongoose.Schema({
  booking: { type: mongoose.Schema.Types.ObjectId, ref: 'Booking', required:
true },
  student: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  mentor: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  rating: { type: Number, required: true, min: 1, max: 5 },
  comment: { type: String, required: true },
  createdAt: { type: Date, default: Date.now }
});
```

// 2. КОНТРОЛЕРИ

// AuthController - реєстрація та авторизація

```
const authController = {
  // Реєстрація нового користувача
  register: async (req, res) => {
    try {
      const { email, password, firstName, lastName, role } = req.body;

      // Перевірка, чи email вже використовується
      const existingUser = await User.findOne({ email });
      if (existingUser) {
        return res.status(400).json({ message: 'Користувач з таким email вже
існує' });
      }

      // Створення нового користувача
      const user = new User({
        email,
        password,
        firstName,
        lastName,
        role
      });

      await user.save();
```

// Створення токена для авторизації

```
    const token = jwt.sign(
        { id: user._id, role: user.role },
        process.env.JWT_SECRET,
        { expiresIn: '24h' }
      );
          res.status(201).json({
        success: true,
        token,
        user: {
          id: user._id,
          email: user.email,
          firstName: user.firstName,
          lastName: user.lastName,
          role: user.role
        }
      });
    } catch (error) {
      res.status(500).json({ message: 'Помилка при реєстрації користувача'
});
    }
  },
```

// Авторизація користувача

```
  login: async (req, res) => {
    try {
      const { email, password } = req.body;
```

// Пошук користувача за email

```
      const user = await User.findOne({ email });
      if (!user) {
        return res.status(401).json({ message: 'Неправильний email або
пароль' });
      }

      // Перевірка паролю
      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) {
        return res.status(401).json({ message: 'Неправильний email або
пароль' });
      }

      // Створення токена для авторизації
      const token = jwt.sign(
        { id: user._id, role: user.role },
        process.env.JWT_SECRET,
        { expiresIn: '24h' }
      );

      res.status(200).json({
        success: true,
        token,
        user: {
          id: user._id,
          email: user.email,
          firstName: user.firstName,
          lastName: user.lastName,
          role: user.role
        }
      });
    } catch (error) {
      res.status(500).json({ message: 'Помилка при авторизації' });
    }
  }
};
```

// Отримання списку всіх менторів з фільтрацією

```
  getMentors: async (req, res) => {
    try {
```

```
            const { skills, minPrice, maxPrice, minRating, search, page = 1, limit
= 10 } = req.query;
            const skip = (page - 1) * limit;

            // Базовий запит - знайти всіх активних менторів
            let query = { role: 'mentor', isVerified: true };

            // Додавання пошуку за ім'ям/прізвищем
            if (search) {
              const searchRegex = new RegExp(search, 'i');
              query.$or = [
                { firstName: searchRegex },
                { lastName: searchRegex }
              ];
            }
```

// Отримання ідентифікаторів користувачів-менторів

```
          const mentorUsers = await User.find(query).select('_id');
            const mentorIds = mentorUsers.map(user => user._id);
            let profileQuery = { user: { $in: mentorIds } };
```

// Додаткові фільтри

```
            if (skills) {
              const skillsArr = Array.isArray(skills) ? skills : [skills];
              profileQuery.skills = { $in: skillsArr };
            }

            if (minPrice) {
              profileQuery.hourlyRate = profileQuery.hourlyRate || {};
              profileQuery.hourlyRate.$gte = Number(minPrice);
            }

            if (maxPrice) {
              profileQuery.hourlyRate = profileQuery.hourlyRate || {};
              profileQuery.hourlyRate.$lte = Number(maxPrice);
            }

            if (minRating) {
              profileQuery.rating = { $gte: Number(minRating) };
            }
```

// Отримання профілів менторів з пагінацією

```
      const mentorProfiles = await MentorProfile.find(profileQuery)
        .populate('user', 'firstName lastName avatar')
        .sort({ rating: -1 })
        .skip(skip)
        .limit(Number(limit));
```

// Підрахунок загальної кількості

```
      const total = await MentorProfile.countDocuments(profileQuery);

      res.status(200).json({
        success: true,
        mentors: mentorProfiles,
        pagination: {
          total,
          page: Number(page),
          limit: Number(limit),
          pages: Math.ceil(total / limit)
        }
      });
    } catch (error) {
      res.status(500).json({ message: 'Помилка при отриманні списку
менторів' });
    }
  },
```

// Отримання профілю ментора за id

```
  getMentorById: async (req, res) => {
    try {
      const { id } = req.params;

// Отримання даних користувача
      const user = await User.findOne({ _id: id, role: 'mentor' })
        .select('firstName lastName avatar');

      if (!user) {
        return res.status(404).json({ message: 'Ментора не знайдено' });
      }

      const profile = await MentorProfile.findOne({ user: id });

      if (!profile) {
        return res.status(404).json({ message: 'Профіль ментора не знайдено'
});
      }
```

// Отримання відгуків

```
      const reviews = await Review.find({ mentor: id })
        .populate('student', 'firstName lastName avatar')
        .sort({ createdAt: -1 })
        .limit(5);

      res.status(200).json({
        success: true,
        mentor: {
          user,
          profile,
          reviews,
          reviewsCount: await Review.countDocuments({ mentor: id })
        }
      });
    } catch (error) {
      res.status(500).json({ message: 'Помилка при отриманні профілю
ментора' });
    }
  }
};
```

// BookingController - управління бронюваннями занять

```
const bookingController = {
  // Створення нового бронювання
  createBooking: async (req, res) => {
    try {
      const { mentorId, startTime, endTime, topic } = req.body;
      const studentId = req.user.id; // з JWT токена
      const mentor = await User.findOne({ _id: mentorId, role: 'mentor' });
      if (!mentor) {
        return res.status(404).json({ message: 'Ментора не знайдено' });
      }
      const mentorProfile = await MentorProfile.findOne({ user: mentorId });
      if (!mentorProfile) {
        return res.status(404).json({ message: 'Профіль ментора не знайдено'
});
      }
```

// Перевірка доступності ментора в зазначений час

```
      const conflictingBooking = await Booking.findOne({
        mentor: mentorId,
        status: { $in: ['pending', 'confirmed'] },
        $or: [
```

```
            { startTime: { $lt: endTime, $gte: startTime } },
            { endTime: { $gt: startTime, $lte: endTime } },
            { startTime: { $lte: startTime }, endTime: { $gte: endTime } }
          ]
        });

        if (conflictingBooking) {
          return res.status(400).json({ message: 'Обраний час вже
заброньований' });
        }

        const start = new Date(startTime);
        const end = new Date(endTime);
        const durationMinutes = (end - start) / (1000 * 60);
```

// Розрахунок ціни

```
        const price = (mentorProfile.hourlyRate / 60) * durationMinutes;

        // Створення бронювання
        const booking = new Booking({
          student: studentId,
          mentor: mentorId,
          startTime: start,
          endTime: end,
          topic,
          price,
          status: 'pending'
        });

        await booking.save();

        res.status(201).json({
          success: true,
          booking
        });
      } catch (error) {
        res.status(500).json({ message: 'Помилка при створенні бронювання' });
      }
    },
```

// Підтвердження бронювання ментором

```
    confirmBooking: async (req, res) => {
      try {
        const { id } = req.params;

        // Пошук бронювання
        const booking = await Booking.findById(id);
        if (!booking) {
          return res.status(404).json({ message: 'Бронювання не знайдено' });
```

```
            }

            if (booking.mentor.toString() !== req.user.id) {
              return res.status(403).json({ message: 'Немає прав для підтвердження
цього бронювання' });
            }

            // Перевірка статусу
            if (booking.status !== 'pending') {
              return res.status(400).json({ message: 'Можна підтвердити тільки
очікуючі бронювання' });
            }

            // Оновлення статусу
            booking.status = 'confirmed';

            // Генерація URL для проведення заняття
            booking.sessionUrl = `https://codementor.com/session/${booking._id}`;
                  await booking.save();

            res.status(200).json({
              success: true,
              booking
            });
          } catch (error) {
            res.status(500).json({ message: 'Помилка при підтвердженні бронювання'
});
          }
        }
      };
```

// Створення нового відгуку

```
  createReview: async (req, res) => {
    try {
      const { bookingId, rating, comment } = req.body;
      const studentId = req.user.id; // з JWT токена
```

// Перевірка наявності бронювання

```
            const booking = await Booking.findById(bookingId);
            if (!booking) {
              return res.status(404).json({ message: 'Бронювання не знайдено' });
            }
               if (booking.student.toString() !== studentId) {
              return res.status(403).json({ message: 'Немає прав для створення
відгуку до цього бронювання' });
            }
```

// Перевірка статусу бронювання

```
            if (booking.status !== 'completed') {
              return res.status(400).json({ message: 'Можна залишати відгуки
тільки до завершених занять' });
            }
```

// Перевірка, чи вже існує відгук

```
            const existingReview = await Review.findOne({ booking: bookingId });
            if (existingReview) {
              return res.status(400).json({ message: 'Відгук до цього заняття вже
існує' });
            }
```

// Створення відгуку

```
            const review = new Review({
              booking: bookingId,
              student: studentId,
              mentor: booking.mentor,
              rating,
              comment
            });

            await review.save();
```

// Оновлення рейтингу ментора

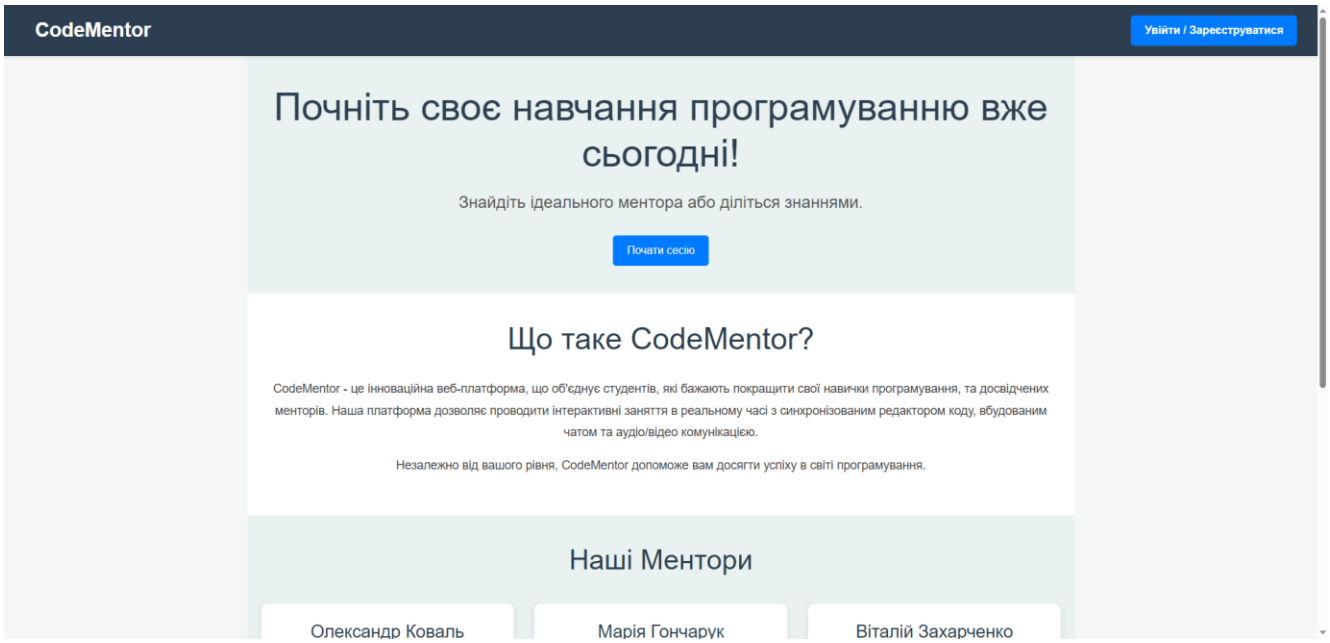
```
            const mentorReviews = await Review.find({ mentor: booking.mentor });
            const totalRating = mentorReviews.reduce((sum, review) => sum +
review.rating, 0);
            const avgRating = totalRating / mentorReviews.length;

            await MentorProfile.findOneAndUpdate(
              { user: booking.mentor },
              { rating: avgRating.toFixed(1) }
            );

            res.status(201).json({
              success: true,
              review
            });
          } catch (error) {
            res.status(500).json({ message: 'Помилка при створенні відгуку' });
          }
        }
      };
```

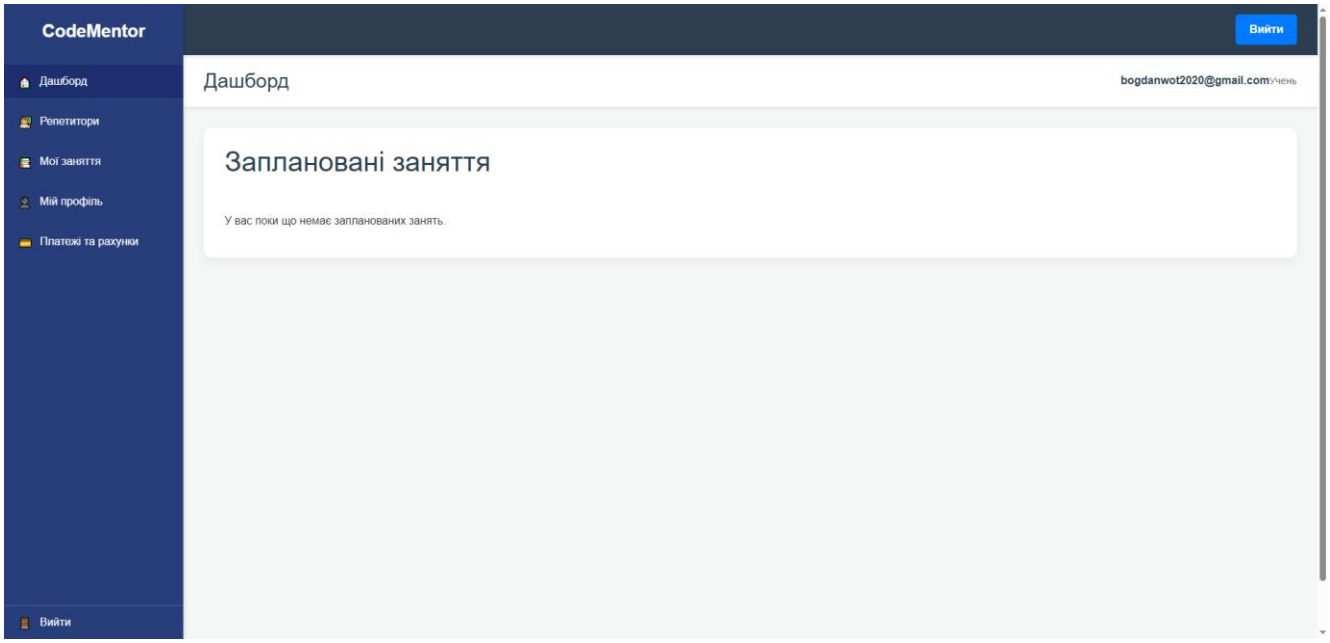
```
// 3. МАРШРУТИ API

const express = require('express');
const router = express.Router();

// Маршрути для авторизації

router.post('/auth/register', authController.register);
router.post('/auth/login', authController.login);

// Маршрути для менторів

router.get('/mentors', mentorController.getMentors);
router.get('/mentors/:id', mentorController.getMentorById);

// Маршрути для бронювань (захищені middleware)

router.post('/bookings', authMiddleware, bookingController.createBooking);
router.put('/bookings/:id/confirm', authMiddleware,
bookingController.confirmBooking);

// Маршрути для відгуків (захищені middleware)

router.post('/reviews', authMiddleware, reviewController.createReview);

// Middleware для перевірки авторизації

function authMiddleware(req, res, next) {
  try {
    // Отримання токену з заголовка
    const token = req.headers.authorization.split(' ')[1];

    // Перевірка токену
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
```

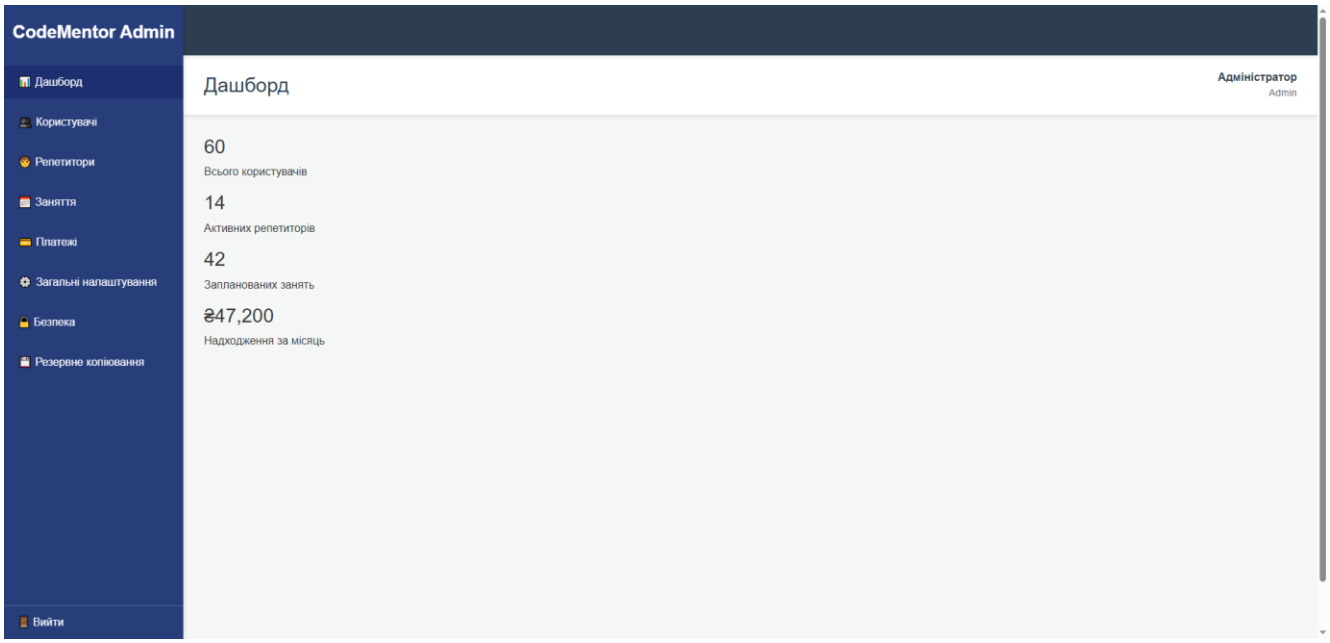
// 4. ОСНОВНИЙ КОД СЕРВЕРА

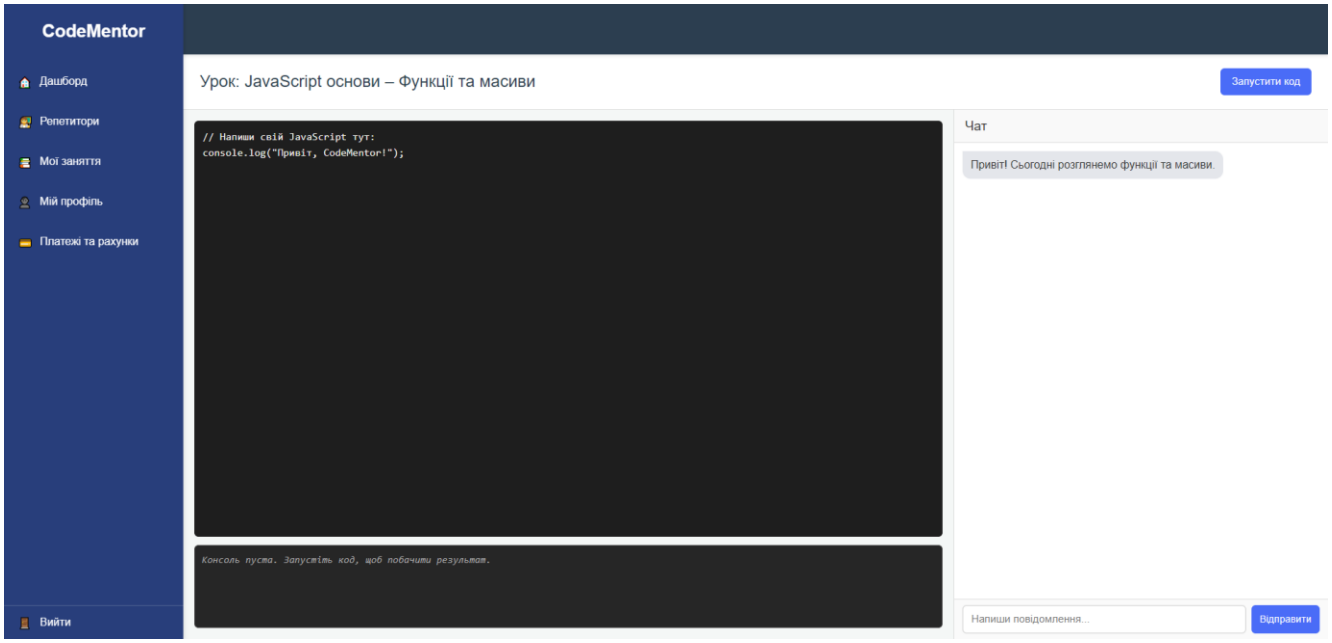
```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const http = require('http');
const socketIo = require('socket.io');
const jwt = require('jsonwebtoken');
const app = express();
const server = http.createServer(app);
const io = socketIo(server);

app.use(express.json());
app.use(cors());

mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log('MongoDB підключено'))
  .catch(err => console.error('Помилка підключення до MongoDB:', err));
app.use('/api', router);

io.on('connection', (socket) => {
  console.log('Користувач підключений:', socket.id);


  socket.on('join-session', (sessionId, token) => {
    try {

      const decoded = jwt.verify(token, process.env.JWT_SECRET);

      socket.join(`session:${sessionId}`);

      io.to(`session:${sessionId}`).emit('user-joined', {
        userId: decoded.id,
        socketId: socket.id
      });
    } catch (error) {
      socket.emit('error', { message: 'Помилка авторизації' });
    }
  });


  socket.on('code-update', (data) => {
    socket.to(`session:${data.sessionId}`).emit('code-update', data);
  });

  socket.on('disconnect', () => {
    console.log('Користувач від\'єднався:', socket.id);
  });
});
const PORT = process.env.PORT || 5000;
server.listen(PORT, () => {
  console.log(`Сервер запущено на порту ${PORT}`);
});
```

ДОДАТОК Б — Результат роботи






CodeMentor Admin
Дашборд
Користувачі
Репетитори
Заняття
Платежі
Загальні налаштування
Безпека
Резервне копіювання
Вийти
Дашборд
Адміністратор
Admin
60
Всього користувачів
14
Активних репетиторів
42
Запланованих занять
₴47,200
Надходження за місяць


CodeMentor
Дашборд
Репетитори
Мої заняття
Мій профіль
Платежі та рахунки
Вийти
Урок: JavaScript основи – Функції та масиви
Запустити код
// Напиши свій JavaScript тут:
console.log("Привіт, CodeMentor!");
Консоль пуста. Запустіть код, щоб побачити результат.
Чат
Привіт! Сьогодні розглянемо функції та масиви.
Напиши повідомлення...
Відправити

ДОДАТОК В — Диск із кваліфікаційною роботою бакалавра