Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб-платформа для надання та отримання послуг
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-7

Ілля ДРОНОВ
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Наталія РУСАКОВА
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дронову Іллі Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Веб-платформа для надання та отримання послуг
Затверджена наказом по університету від № 397 Ст від 19.05.2025
2. Термін подання студентом роботи до екзаменаційної комісії 10.06.2025
3. Вихідні дані до роботи: розробка програмної веб системи для надання можливості користувачам знаходити менторів та продавати свої послуги в якості менторів; для розробки використовувати технології Next.js, FastAPI, у якості середовища розробки використати Cursor
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, впровадження програмного забезпечення, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2025	виконано
2	Створення специфікації ПЗ	22.05.2025	виконано
3	Проектування ПЗ	23.05.2025	виконано
4	Розробка ПЗ	27.05.2025	виконано
5	Тестування ПЗ	01.05.2025	виконано
6	Оформлення пояснювальної записки	03.06.2025	виконано
7	Підготовка презентації та доповіді	03.06.2025	виконано
8	Попередній захист	06.06.2025	виконано
9	Нормоконтроль, рецензування	05.06.2025	виконано
10	Здача роботи у електронний архів	07.06.2025	виконано
11	Допуск до захисту у зав. кафедри	09.06.2025	виконано

Дата видачі завдання «20» «травня» 2025р.

Здобувач  Ілля ДРОНОВ
(підпис)

Керівник роботи _____ доц. кафедри ПІ Наталія РУСАКОВА
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 90 стор., 17 рис., 18 джерел, 1 табл..

FASTAPI, NEXT.JS, POSTGRESQL, SOFTWARE SYSTEM, STRIPE, TALK.JS, WEB

Об’єкт розробки – повнофункціональна програмна система (frontend + backend), яка дозволяє користувачам надавати та отримувати послуги менторства, навчання та обміну навичками. Система включає модулі реєстрації, публікації оголошень, чат-спілкування, фінансових транзакцій та категоризації діяльності.

Мета розробки – створення зручної, безпечної та масштабованої веб-платформи, яка надає змогу користувачам як продавати свої знання та навички, так і отримувати нові. Один користувач може бути одночасно як ментором, так і учнем. Проєкт орієнтований на підвищення доступності якісних освітніх та консультаційних послуг.

Метод рішення – для клієнтської частини використано фреймворк Next.js з підтримкою аутентифікації через NextAuth (JWT + Google), Tailwind CSS для стилізації, інтеграцію з Talk.js для чатів та Stripe для оплати. Серверна частина реалізована за допомогою FastAPI (Python) з підтримкою REST API та інтеграції з платіжною системою Stripe. Для зберігання даних використовується PostgreSQL.

У результаті розробки створено масштабовану програмну систему, яка дозволяє користувачам ефективно взаємодіяти між собою, публікувати свої послуги, здійснювати оплату, вести переписку та отримувати освітні або консультаційні послуги. Рішення сприяє розвитку менторської культури, самореалізації та підвищенню кваліфікації користувачів.

ABSTRACT

FASTAPI, NEXT.JS, POSTGRESQL, SOFTWARE SYSTEM, STRIPE, TALK.JS, WEB

The object of development is a full-featured software system (frontend + backend) that allows users to provide and receive mentoring, training, and skills exchange services. The system includes modules for registration, announcement publishing, chat communication, financial transactions, and activity categorization.

The goal of the development is to create a convenient, secure, and scalable web platform that allows users to both sell their knowledge and skills and acquire new ones. One user can be both a mentor and a student. The project is focused on increasing the availability of quality educational and consulting services.

Solution method - for the client side, we used the Next.js framework with support for NextAuth authentication (JWT + Google), Tailwind CSS for styling, integration with Talk.js for chats, and Stripe for payment. The server side is implemented using FastAPI (Python) with support for REST API and integration with the Stripe payment system. PostgreSQL is used for data storage.

As a result of the development, a scalable software system has been created that allows users to effectively interact with each other, publish their services, make payments, correspond, and receive educational or consulting services. The solution promotes the development of a mentoring culture, self-realization, and professional development of users.

ЗМІСТ

Перелік скорочень 7
Вступ 8
1 Аналіз предметної галузі 9
1.1 Аналіз предметної галузі 9
1.2 Виявлення та вирішення проблем 13
1.3 Постановка задачі 14
1.3.1 Цільова аудиторія 16
1.3.2 Монетизація 16
2 Формування вимог до програмної системи 18
3 Архітектура та проектування ПЗ 20
3.1 UML проектування ПЗ 20
3.2 Проєктування архітектури ПЗ 21
3.3 Проєктування структури зберігання даних 24
3.5 Розробка UI/UX системи 30
4 Опис прийнятих програмних рішень 35
5 Тестування програмного забезпечення 41
5.1 Функціональне тестування 41
5.2 Інтеграційне тестування 42
5.2.1 Авторизація користувача 43
5.2.2 Створення оголошення 43
5.2.3 Отримання списку оголошень 43
6 Впровадження програмного забезпечення 45
6.1 Публікації 45
Висновки 46
Перелік джерел посилань 48

ПЕРЕЛІК СКОРОЧЕНЬ

HTTP – HyperText Transfer Protocol

REST – REpresentational State Transfer

SQL – Structured Query Language

AWS – Amazon Web Services

EC2 - Amazon Elastic Compute Cloud

K8S – Kubernetes

SC — Software Component (функціональність або вимога, що тестується)

ST — Software Testing (конфігурація середовища тестування)

ver — версія тест кейсу

ВСТУП

Сучасний світ стрімко розвивається в напрямку цифровізації, що охоплює різноманітні аспекти людської діяльності, включаючи сферу освіти, професійного розвитку та надання послуг. Зростає попит на інструменти, які дозволяють людям обмінюватися знаннями, навчати інших або самостійно отримувати нові навички. Особливої актуальності набувають онлайн-платформи, які об'єднують менторів та учнів у спільному цифровому просторі [1].

Актуальність даної роботи полягає у створенні програмної системи, яка дозволить користувачам легко знаходити виконавців або надавачів послуг, здійснювати взаємодію через зручний інтерфейс, оплачувати послуги онлайн, а також забезпечити безпечну й ефективну комунікацію між учасниками.

Метою цієї роботи є розробка надійної, масштабованої та безпечної веб-платформи, яка надає змогу користувачам як пропонувати, так і замовляти послуги освітнього чи консультаційного характеру. Система повинна забезпечувати можливість створення оголошень, здійснення фінансових транзакцій, ведення чату між користувачами, а також персоналізацію профілю та категоризацію послуг.

Для досягнення поставленої мети були визначені такі завдання: аналіз існуючих рішень у сфері онлайн-менторства, розробка архітектури програмної системи з використанням сучасного технологічного стеку, проектування бази даних, реалізація клієнтської та серверної частин, інтеграція з сервісами Stripe для обробки платежів та Talk.js для обміну повідомленнями, а також впровадження захищеної аутентифікації користувачів.

Сфера застосування результатів роботи охоплює онлайн-освіту, консалтинг, наставництво, кар'єрний коучинг та інші галузі, де є потреба в передачі знань і навичок. Запропонована система дозволить користувачам ефективно обмінюватися досвідом, розширити професійні зв’язки та покращити доступ до якісних послуг незалежно від географічного розташування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

У сучасних умовах динамічного розвитку цифрових технологій спостерігається зростання попиту на онлайн-платформи для надання та отримання знань, консультацій і послуг. Особливо після початку пандемії в 2020-му році та в умовах нестабільної економічної ситуації багато людей шукають можливості для віддаленого заробітку, навчання та професійного розвитку [1]. Це сприяє активному формуванню ринку менторських послуг і платформ для обміну навичками.

Менторство, онлайн-освіта та кар'єрні консультації стали невіддільною частиною сучасного підходу до самореалізації та розвитку. У відповідь на це виникає потреба у створенні зручних, інтуїтивно зрозумілих цифрових сервісів, які дозволяють швидко знайти наставника або учня, організувати безпечну комунікацію, провести оплату послуг та отримати зворотний зв’язок.

Організація взаємодії між ментором та учнем включає низку важливих аспектів: реєстрацію користувачів, створення профілів і оголошень, категоризацію навичок і послуг, пошук відповідних спеціалістів, комунікацію (наприклад, у форматі чату), фінансові транзакції та рейтингування. Впровадження автоматизованої платформи дозволяє оптимізувати всі ці етапи, зменшити ручну працю й мінімізувати ризики шахрайства або непорозумінь.

У рамках даної роботи увагу зосереджено як на клієнтській, так і на серверній частинах програмної системи. Клієнтська частина реалізована з використанням фреймворку Next.js та включає інтерфейс для реєстрації, перегляду й створення оголошень, чатів між користувачами, проведення оплат тощо. Серверна частина, розроблена за допомогою FastAPI, відповідає за реалізацію бізнес-логіки, обробку HTTP-запитів, взаємодію з базою даних і зовнішніми сервісами, такими як Stripe і Talk.js. Обидві частини взаємодіють через REST API, що забезпечує цілісність системи, її масштабованість та можливість інтеграції з іншими платформами й сервісами.

Крім того, API забезпечує можливості для збору та аналізу даних про активність користувачів, взаємодії між ментором та учнем, а також статистику фінансових операцій. Це критично важливо для подальшого вдосконалення сервісу, реалізації рекомендаційних механізмів та запуску маркетингових кампаній. З точки зору розвитку EdTech-сервісів, добре спроєктоване API дозволяє інтегрувати платформу з іншими цифровими екосистемами (наприклад, CRM, LMS чи платформами відеозв'язку), що сприяє створенню більш повноцінного та ефективного освітнього середовища.

Аналізуючи вже існуючі платформи, що використовуються для організації онлайн-менторства, навчання або надання послуг, можна виокремити кілька популярних рішень, які забезпечують ефективну взаємодію між користувачами, безпечні фінансові операції та зручне керування профілями. Однією з найвідоміших таких платформ є MentorCruise [2], яка надає широкий спектр функціональності для менторів і учнів. Система дозволяє створювати детальні профілі, вказувати спеціалізацію, доступність, ціни за сесію та формат спілкування. MentorCruise автоматизує процес підбору менторів, дозволяючи користувачам знаходити оптимального наставника за допомогою фільтрів і рекомендацій.

Функціонал MentorCruise також охоплює інтеграцію з календарем, оплату послуг через Stripe, а також ведення історії спілкування. Це забезпечує високий рівень сервісу, прозорість у відносинах і зручність для обох сторін. На рисунку 1.1 зображено інтерфейс профілю ментора на платформі MentorCruise.

Іншою популярною платформою є Superpeer [3] (див. рис. 1.2), яка орієнтована на проведення відеоконсультацій у форматі «один на один». Вона дозволяє створювати публічні профілі, вказувати вартість і тривалість сесій, а також бронювати час напряму через інтегрований календар. Особливістю Superpeer є повна автоматизація процесу: від бронювання до відеозустрічі та оплати, що робить її зручною як для експертів, так і для клієнтів. Superpeer також підтримує партнерські програми та продаж підписок, що дозволяє ментору монетизувати свою експертизу більш гнучко.

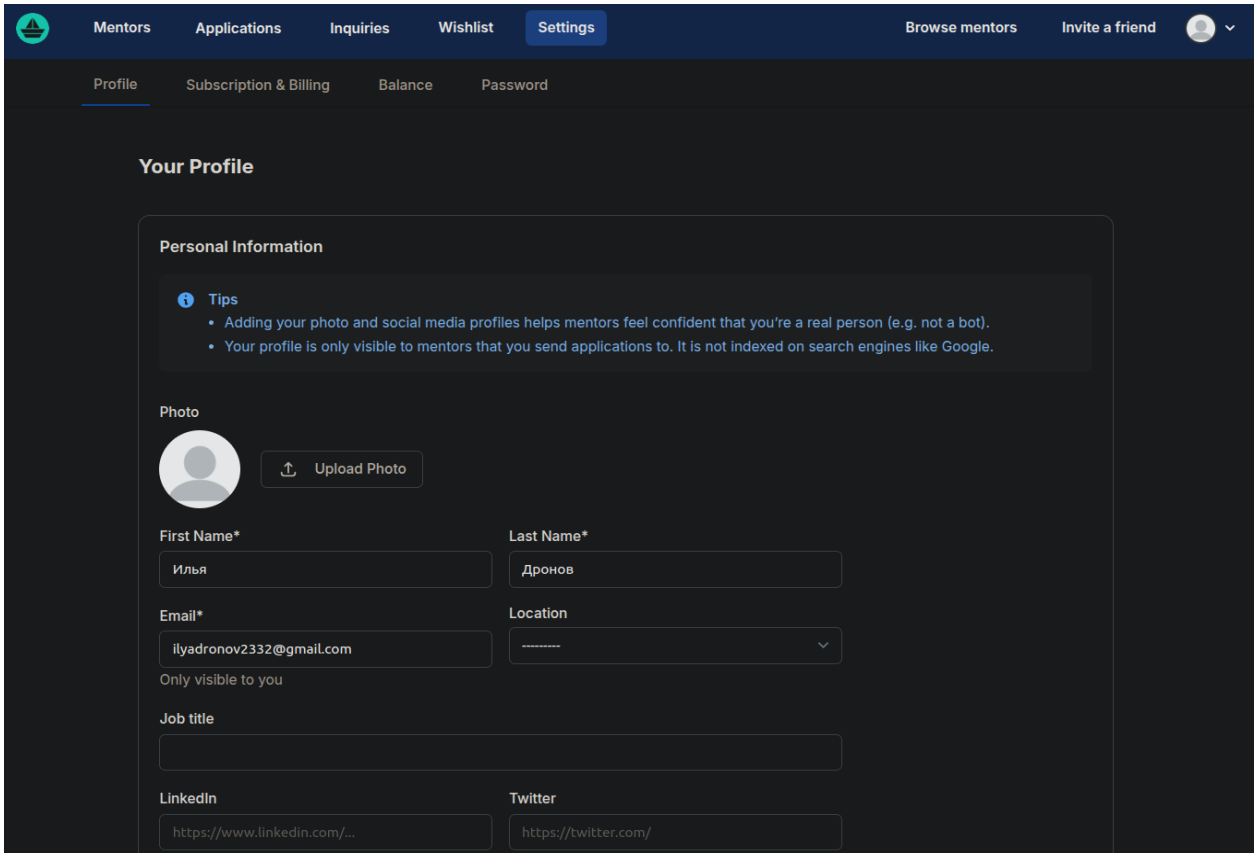

Рисунок 1.1 – Профіль учня на MentorCruise (за даними [2])

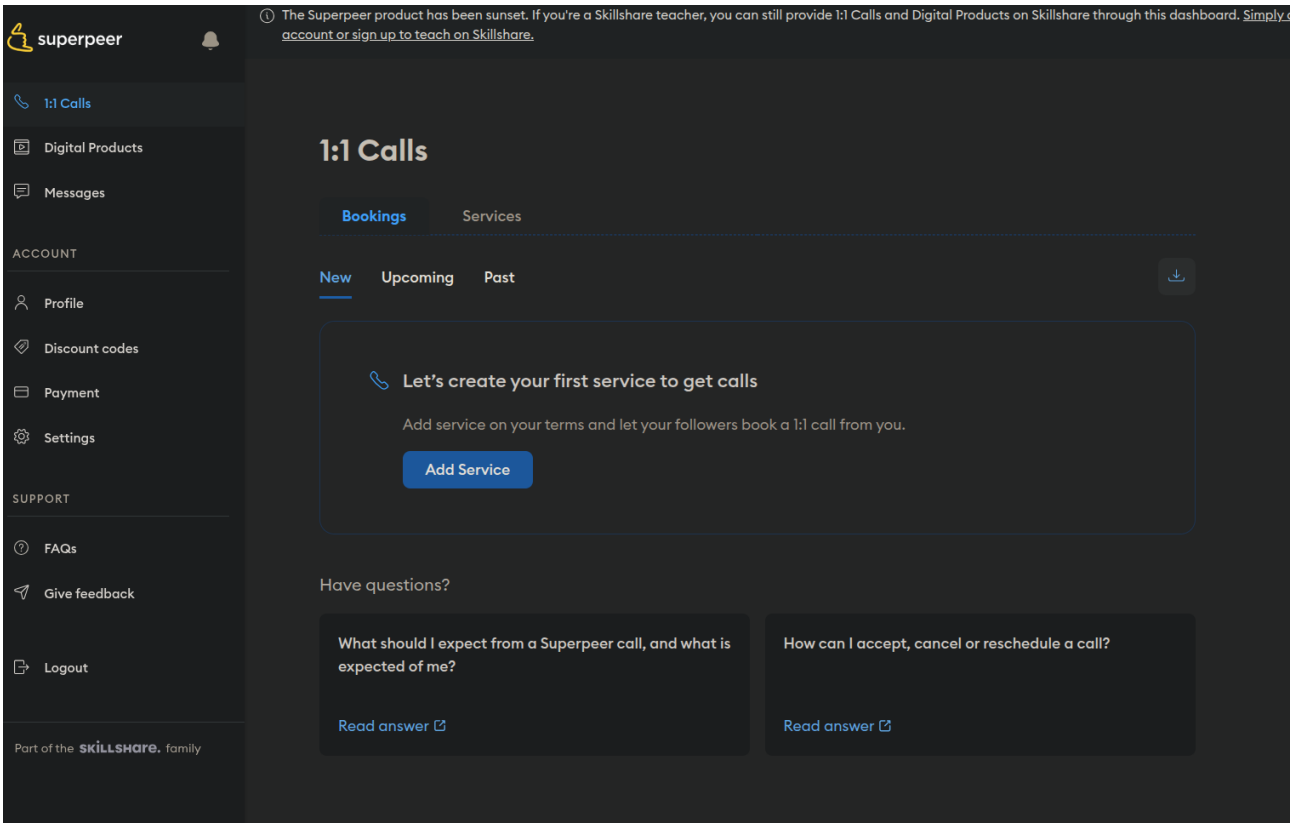

Рисунок 1.2 – Інтерфейс бронювання консультації на Superpeer (за даними [3])

Ще один важливий приклад — ADPList [4], платформа, що поєднує менторство з елементами волонтерства. Вона дозволяє експертам із різних галузей надавати безкоштовні сесії користувачам з усього світу. ADPList вирізняється сильною спільнотою, акцентом на доступність та підтримку початківців у професійному зростанні. Усі функції — від запису на сесію до спілкування — реалізовані через простий інтерфейс і API, що дозволяє швидко масштабувати платформу.

Приклад інтерфейсу вебсайту зображений на рисунку 1.3.

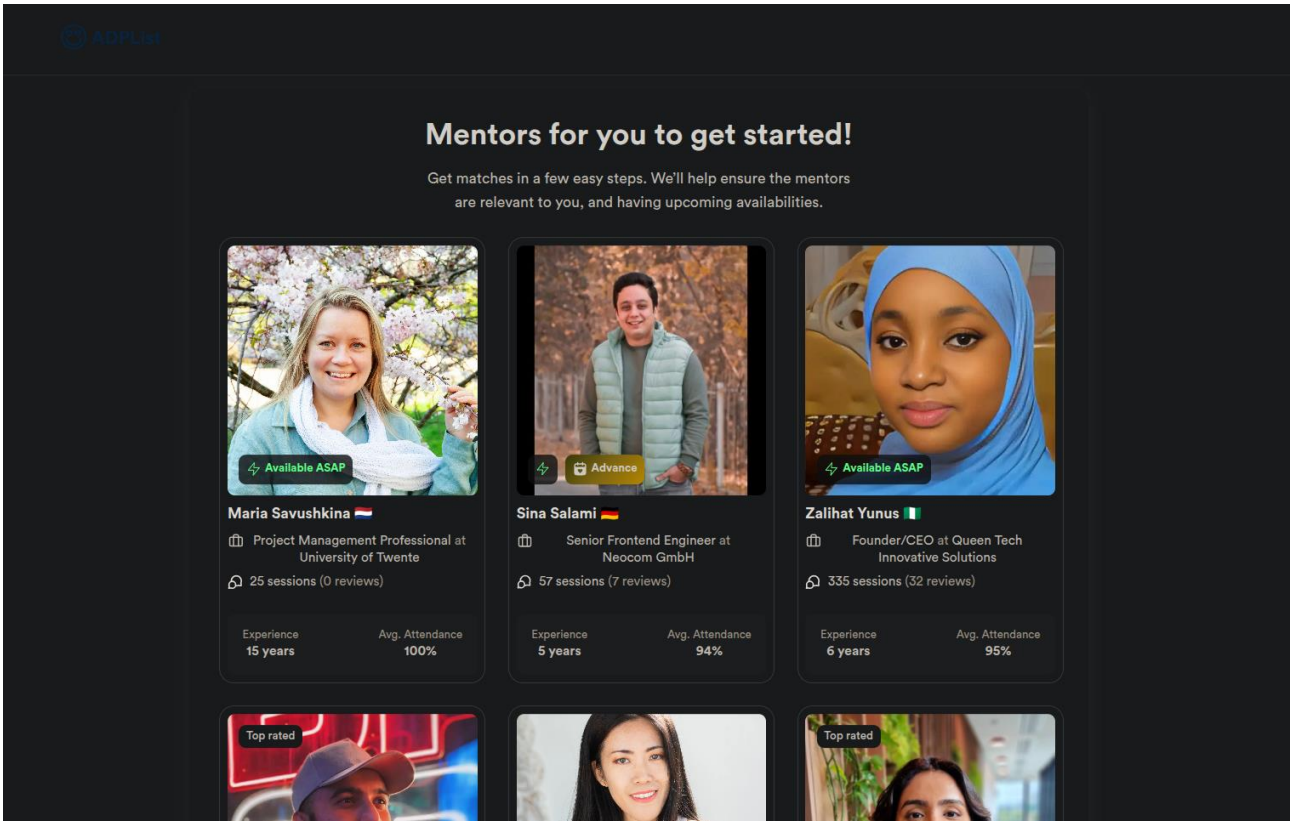

Рисунок 1.3 – Список доступних менторів на ADPList (за даними [4])

Таким чином, можна зробити висновок, що інтеграція таких технологічних рішень, як REST API, платіжні шлюзи та сервіси для відеозв’язку, відіграє критичну роль у розробці сучасних EdTech-платформ. Це дозволяє автоматизувати рутинні процеси, покращити досвід користувачів, забезпечити безпеку транзакцій і підвищити загальну ефективність платформи.

1.2 Виявлення та вирішення проблем

Існуючі платформи, такі як MentorCruise [2], Superpeer [3] чи ADPList [4], хоча й забезпечують широкий функціонал для менторів та учнів, виявляються обмеженими у гнучкості налаштування або адаптації під специфічні потреби окремих категорій користувачів. Часто системи орієнтовані лише на односторонній сценарій використання — або менторство, або навчання, — тоді як сучасні користувачі можуть одночасно виступати як наставники, так і учні. Крім того, інтерфейси таких систем зазвичай не дозволяють гнучко керувати категоріями послуг, мультирольовістю, чи глибоко кастомізувати профілі під конкретну спеціалізацію.

Ще однією поширеною проблемою є відсутність зручної внутрішньої комунікації. Не всі платформи мають вбудовані системи обміну повідомленнями з можливістю зберігання історії чату, індикаторами прочитання, підтримкою вкладень чи адаптацією під мобільні пристрої. В окремих випадках спілкування переноситься на сторонні платформи (наприклад, електронну пошту або Zoom), що ускладнює управління взаємодією й знижує рівень інтегрованості користувацького досвіду.

Окрема проблема полягає у недостатній прозорості фінансових транзакцій та обмежених варіантах монетизації. Багато платформ не дозволяють ментору гнучко керувати прайсингом, встановлювати тарифи за окремі типи консультацій (разові зустрічі, підписка, пакет сесій), або не мають підтримки локальних валют. Це обмежує потенціал масштабування систем у різних країнах, зокрема в Україні. Інтеграція з платіжними сервісами типу Stripe дає змогу вирішити цю проблему, але потребує детального налаштування бекенду, гарантування безпеки та прозорості процесу.

Крім того, недостатня увага приділяється персоналізації та аналітиці. У більшості систем відсутні інструменти для збору метрик ефективності — таких як кількість проведених сесій, середня оцінка менторів, рівень задоволеності учнів. Впровадження таких аналітичних засобів дозволило б покращити якість сервісу, а

також реалізувати рекомендаційні системи для кращого матчінгу між користувачами.

Ще одним вектором удосконалення може бути інтеграція штучного інтелекту. Наприклад, AI може використовуватись для автоматичного підбору менторів на основі запиту користувача, аналізу попереднього досвіду, рекомендацій та активності [5]. Також можлива реалізація AI-асистентів, які допомагатимуть користувачам у навігації системою, поясненні процесів або попередньому аналізі запитів.

Таким чином, вирішення вищезазначених проблем через гнучку архітектуру, інтеграцію сучасних технологій, поліпшену UX/UI взаємодію та інструменти аналітики є ключовими кроками у створенні конкурентоспроможної, масштабованої та ефективної платформи менторства.

1.3 Постановка задачі

Задача даної роботи полягає у створенні комплексної програмної системи, що поєднує функціональні можливості як для менторів, так і для учнів. Платформа складатиметься з клієнтської та серверної частин, які взаємодіють через REST API, забезпечуючи повноцінний цикл реєстрації, створення профілів, публікації послуг, комунікації між користувачами та фінансових операцій. Основною метою є автоматизація рутинних дій, зменшення навантаження на користувачів та забезпечення зручного та безпечного цифрового простору для обміну знаннями.

Клієнтська частина системи повинна забезпечувати інтуїтивно зрозумілий інтерфейс для:

- реєстрації користувачів як менторів або учнів (або обох одночасно);
- створення та редагування оголошень про послуги;
- перегляду профілів інших користувачів;
- спілкування через чат;
- бронювання сесій та поповнення балансу через Stripe.

Серверна частина, реалізована на основі FastAPI, повинна відповідати за:

- зберігання даних про користувачів, послуги, категорії, сесії та

транзакції;

- авторизацію та аутентифікацію користувачів за допомогою JWT або Google OAuth;
- обробку HTTP-запитів від клієнтської частини та забезпечення стабільного API;
- взаємодію з платіжною системою Stripe;
- забезпечення безпеки та валідації введених даних.

До основних вимог, які має виконувати система, належать:

- безпечне зберігання персональних даних користувачів і історії замовлень;
- підтримка фільтрації та сортування оголошень за категоріями, ціною, рейтингом;
- ефективна обробка запитів та швидкий доступ до потрібної інформації;
- реалізація внутрішнього чату через інтеграцію з Talk.js;
- сповіщення користувачів про події (бронювання, оплата, нові повідомлення) через email або push-нотифікації.

На практиці це означає створення універсального API, яке дозволяє гнучко керувати всіма аспектами взаємодії користувачів у системі. Особлива увага приділяється зручності створення та керування профілями, бронюванням послуг та збереженню історії комунікації між сторонами. Це сприятиме формуванню довіри між учасниками та підвищенню їхньої залученості.

Система також включає модуль управління оплатами, що забезпечує прозорість фінансових операцій. Stripe інтегрується з серверною частиною для автоматизації процесу оплати й дотримання стандартів безпеки.

Таким чином, дана робота спрямована на розробку повноцінної веб-платформи для взаємодії менторів і учнів, що дозволяє реалізувати весь цикл: від пошуку послуг до їх оплати й отримання. У процесі розробки передбачається застосування сучасних технологій frontend та backend-розробки, що забезпечить масштабованість, безпеку та високу якість користувацького досвіду.

1.3.1 Цільова аудиторія

Цільовою аудиторією платформи є широке коло користувачів, які мають потребу в індивідуальному навчанні, наставництві або професійному консультуванні. До основних груп користувачів відносяться:

- люди, які хочуть розвивати свої навички або перекваліфіковуватись;
- фахівці-початківці, які потребують підтримки більш досвідчених менторів;
- студенти, які прагнуть отримати прикладне бачення професійної сфери;
- практикуючі ментори, що хочуть поділитися досвідом і монетизувати свої знання;
- компанії, які впроваджують менторство для розвитку внутрішніх кадрів;
- люди різного віку, статі, освіти та професійного рівня;
- користувачі з досвідом використання навчальних платформ;
- користувачі, які вперше знайомляться з індивідуальним навчанням онлайн.

Таким чином, **розроблена платформа** орієнтована на максимально широке коло користувачів, які прагнуть ефективної, доступної та персоналізованої взаємодії в освітньому середовищі.

1.3.2 Монетизація

Застосунок використовує комбіновану модель монетизації, яка дозволяє отримувати дохід від проведених сесій без створення фінансового бар’єру для входу в систему.

Основне джерело доходу — фіксована комісія, яка утримується з кожної оплаченої менторської сесії. Користувачі (учні) сплачують вартість консультації через інтегровану платіжну систему (Stripe), а платформа автоматично утримує

відсоток перед переказом коштів ментору. Це забезпечує прозорість та зручність оплати.

Для підвищення довіри та якості на платформі діє процес верифікації менторських профілів, який наразі є безкоштовним. Надалі можлива інтеграція з зовнішніми базами або підтримка офіційної сертифікації, що відкриває потенційні можливості для монетизації.

У перспективі планується створення корпоративного тарифу для командного доступу (наприклад, для HR-відділів або освітніх організацій), який включатиме аналітику, внутрішню модерацію, створення груп користувачів та розширену статистику успішності.

Обрана модель забезпечує баланс між прибутковістю платформи та її доступністю для широкого кола користувачів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Програмна система, призначена для організації взаємодії між ментором і учнем, потребує реалізації надійної, безпечної та масштабованої архітектури. Система повинна включати клієнтську частину, що забезпечує зручний інтерфейс користувача, і серверну частину, що реалізує бізнес-логіку через REST API. Основними користувачами системи є ментори, учні та адміністратори, кожен із яких повинен мати доступ до специфічного набору функцій згідно зі своєю роллю.

Для реалізації серверної частини обрано FastAPI (Python) (разом з ORM SQLAlchemy) — високопродуктивний фреймворк для розробки API, який забезпечує швидкість, асинхронність і підтримку OpenAPI (Swagger) [6, 7]. Клієнтська частина реалізована за допомогою Next.js, з використанням Tailwind CSS для стилізації, NextAuth для аутентифікації (JWT + Google), та інтеграцій із зовнішніми сервісами: Stripe (оплата) і Talk.js (чат) [8, 9].

Система повинна реалізовувати наступні функціональні можливості:

- реєстрація користувачів з підтримкою кількох ролей (ментор / учень / обидва);
- створення, редагування та видалення оголошень про послуги;
- пошук послуг за категоріями, назвою, ціною;
- перегляд профілів інших користувачів;
- бронювання сесій;
- комунікація між ментором та учнем (інтеграція з Talk.js);
- поповнення внутрішнього балансу Stripe;
- email-сповіщення користувачів про підтвердження верифікації, нові повідомлення тощо;
- адмін-панель для підтвердження верифікацій менторів.

Користувацькі ролі та рівні доступу:

- учні можуть переглядати оголошення менторів або створювати оголошення для пошука ментора, спілкуватися, залишати відгуки;
- ментори можуть публікувати послуги або шукати учнів за їх

оголошеннями, створювати сесії;

- адміністратори мають доступ до підтвердження особистих даних менторів, що буде дозволяти їм створювати нові оголошення;

Вимоги до безпеки та зберігання даних:

- аутентифікація через JWT і Google OAuth (NextAuth);
- захищена передача даних через HTTPS;
- обмеження доступу до ресурсів відповідно до ролей;
- збереження чутливої інформації з шифруванням;
- резервне копіювання бази даних.

Вимоги до інфраструктури:

- сховище: PostgreSQL — основна реляційна база даних;
- хостинг: AWS EC2;
- зовнішні сервіси: Stripe, Talk.js AWS S3 (для зберігання статичних файлів користувачів).

Вимоги до API:

- архітектура REST;
- документація через Swagger/OpenAPI;
- CRUD-операції для оголошень, профілів, сесій, транзакцій;
- підтримка пагінації, фільтрації та сортування.

Можливе розширення в майбутньому:

- інтеграція з іншими платіжними системами (наприклад, LiqPay або Monobank для локального ринку);
- підтримка групових сесій або курсів;
- впровадження рекомендованої системи (AI-підбір ментора/учня);
- розширена аналітика для менторів і адміністраторів.

Після проведеного аналізу можна зробити висновок про необхідність інтеграції широкого спектру сучасних технологій, які забезпечать високу продуктивність, безпеку, зручність та масштабованість.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПЗ

3.1 UML проектування ПЗ

В процесі проєктування програмної системи для надання і отримання послуг було розроблено use case діаграми для представлення функціональностей, які система надає різним типам користувачів, таким як адміністратор, та користувач. Так як користувачі можуть виступати як в якості учнів, так і в якості менторів, обидві сутності були поєднані в більш загальну.

Функціональність застосунку представлено на рисунку 3.1.

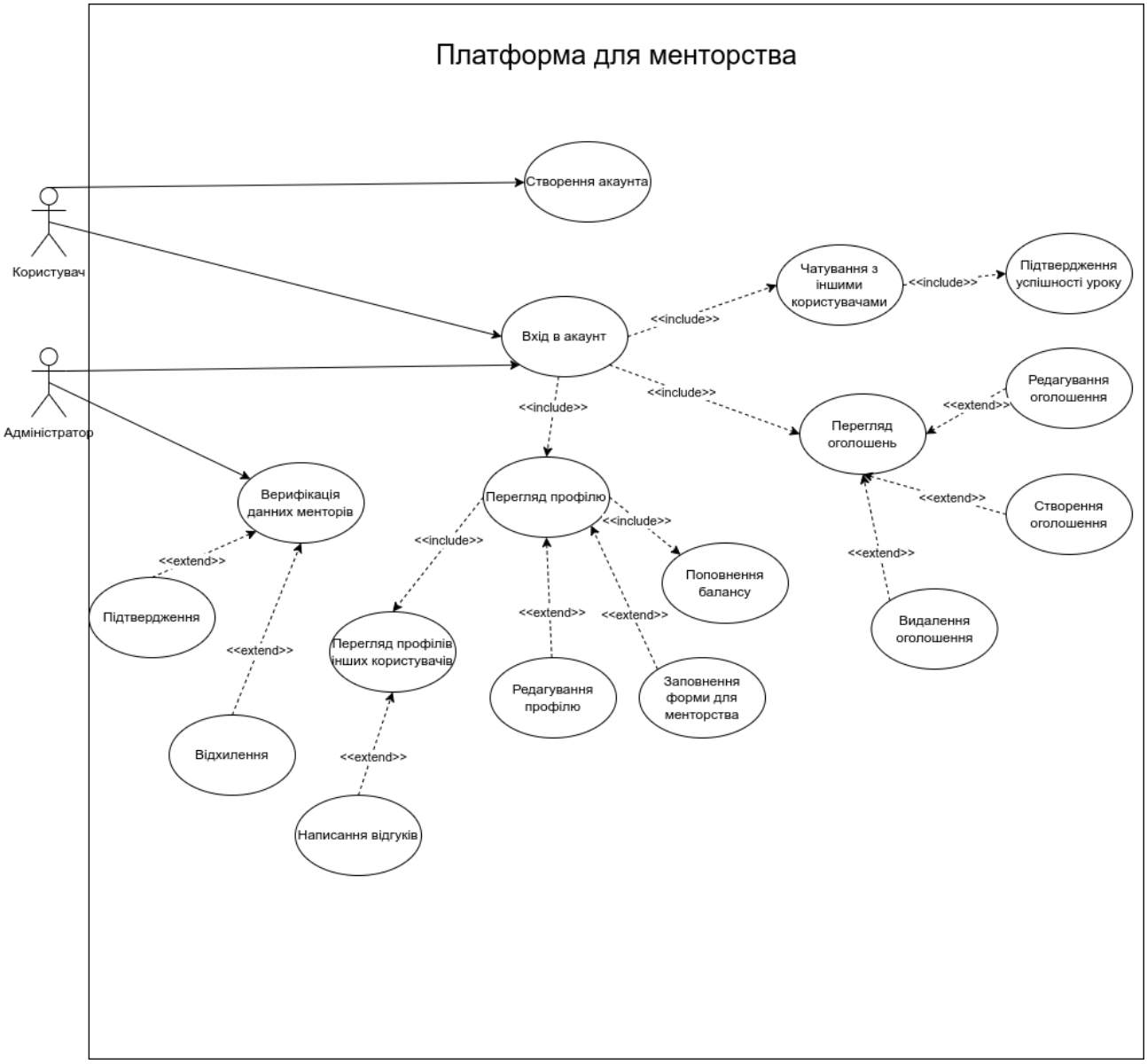

Рисунок 3.1 - Use case діаграма застосунку (рисунок виконано самостійно)

Адміністратор має такі можливості, як:

– логін;
– перегляд поданих даних на верифікацію;
– підтвердження чи відхилення цих даних з вказанням причини.

Користувачі мають такі можливості:

– логін та реєстрація;
– перегляд, створення, редагування та видалення оголошень;
– перегляд профілю, його редагування;
– поповнення внутрішнього балансу;
– заповнення даних для статусу ментора;
– переписка з іншими користувачами;
– підтвердження закінчення сесій, початих іншими користувачами.

Таким чином, було визначено функціональність адміністраторів та користувачів застосунку

3.2 Проєктування архітектури ПЗ

Для розробки програмного забезпечення платформи для надання та отримання послуг було обрано трьохрівневу архітектуру (three-tier architecture), яка є перевіреним підходом для створення масштабованих, підтримуваних і модульних веб-систем. Така архітектура дозволяє ефективно розділити обов’язки між різними частинами системи, спростити розробку, тестування та подальше розширення функціоналу. Рисунок схеми представлено на рисунку 3.2.

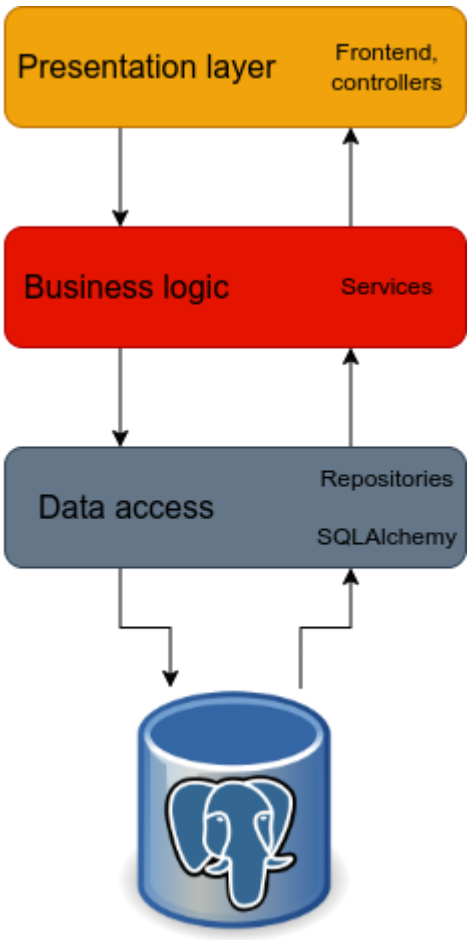

Рисунок 3.2 – Схема архітектури програмного забезпечення (рисунок виконано самостійно)

Вибір саме цієї архітектури зумовлений її зрозумілою структурою та перевагами в масштабуванні. Рівні клієнта, бізнес-логіки та даних можуть розвиватися незалежно один від одного. Наприклад, при зростанні кількості користувачів, можна масштабувати окремо API або базу даних без впливу на інші компоненти. Це також дозволяє різним розробникам або командам паралельно працювати над окремими частинами системи. Такий підхід забезпечує ізоляцію логіки, покращує безпеку, оскільки frontend не взаємодіє безпосередньо з базою даних, та спрощує тестування кожного шару окремо.

Система поділена на три основні рівні: презентаційний, бізнес-логіки та доступу до даних. У клієнтській частині використовується Next.js у поєднанні з Tailwind CSS для реалізації динамічного, адаптивного інтерфейсу, через який користувачі можуть реєструватися, створювати оголошення, здійснювати

бронювання, вести комунікацію та оплачувати послуги. Презентаційний рівень взаємодіє з сервером через REST API, дотримуючись принципів безстановості, уніфікації інтерфейсу та чіткого маршрутизаційного поділу.

Серверна частина реалізована за допомогою FastAPI — асинхронного фреймворку Python, який забезпечує високу швидкодію та підтримку автоматичної генерації документації через Swagger. Вона включає контролери, які приймають HTTP-запити та передають їх до відповідних сервісів бізнес-логіки. RESTful архітектура дозволяє ефективно керувати ресурсами системи через стандартні HTTP-методи. Кожен ресурс — користувач, сесія, послуга, транзакція — ідентифікується через унікальний URI, що полегшує масштабування та зрозумілу інтеграцію з клієнтськими додатками.

На рівні бізнес-логіки обробляються всі вхідні дані — від реєстрації користувача, до логіки бронювання, підтвердження оплат, генерації токенів Stripe та взаємодії з чатом (Talk.js). Тут реалізована авторизація за ролями, валідація, логіка розсилки email-повідомлень, обробка відгуків, рейтингів та іншої взаємодії між ментором і учнем. Усі дії із сутностями проходять через сервіси, що інкапсулюють логіку взаємодії з рівнем даних.

Рівень доступу до даних забезпечує взаємодію з PostgreSQL базою даних. Для зберігання інформації про користувачів, послуги, категорії, бронювання та транзакції створено окремі таблиці зі зв’язками відповідно до ER-моделі. Взаємодія із СУБД організована через абстракції, які дозволяють легко модифікувати логіку запитів, не впливаючи на решту системи. Репозиторії інкапсулюють всі CRUD-операції, а ORM (SQLAlchemy) дозволяє працювати з даними через Python-об’єкти, спрощуючи підтримку та розвиток коду.

Система буде розгорнута у хмарному середовищі з підтримкою автоматичного CI/CD. Для цього буде використано AWS EC2 [10]. У майбутньому передбачається можливість переходу на мікросервісну архітектуру, що забезпечить ще більшу масштабованість, гнучкість і незалежне оновлення окремих частин системи. Також буде впроваджено докеризацію з підтримкою багатосервісної структури, що дозволить автоматизувати деплой нових версій, спростити

обслуговування та зменшити ризики при оновленнях. Діаграма розгортання розгортання представлена на рисунку 3.3.

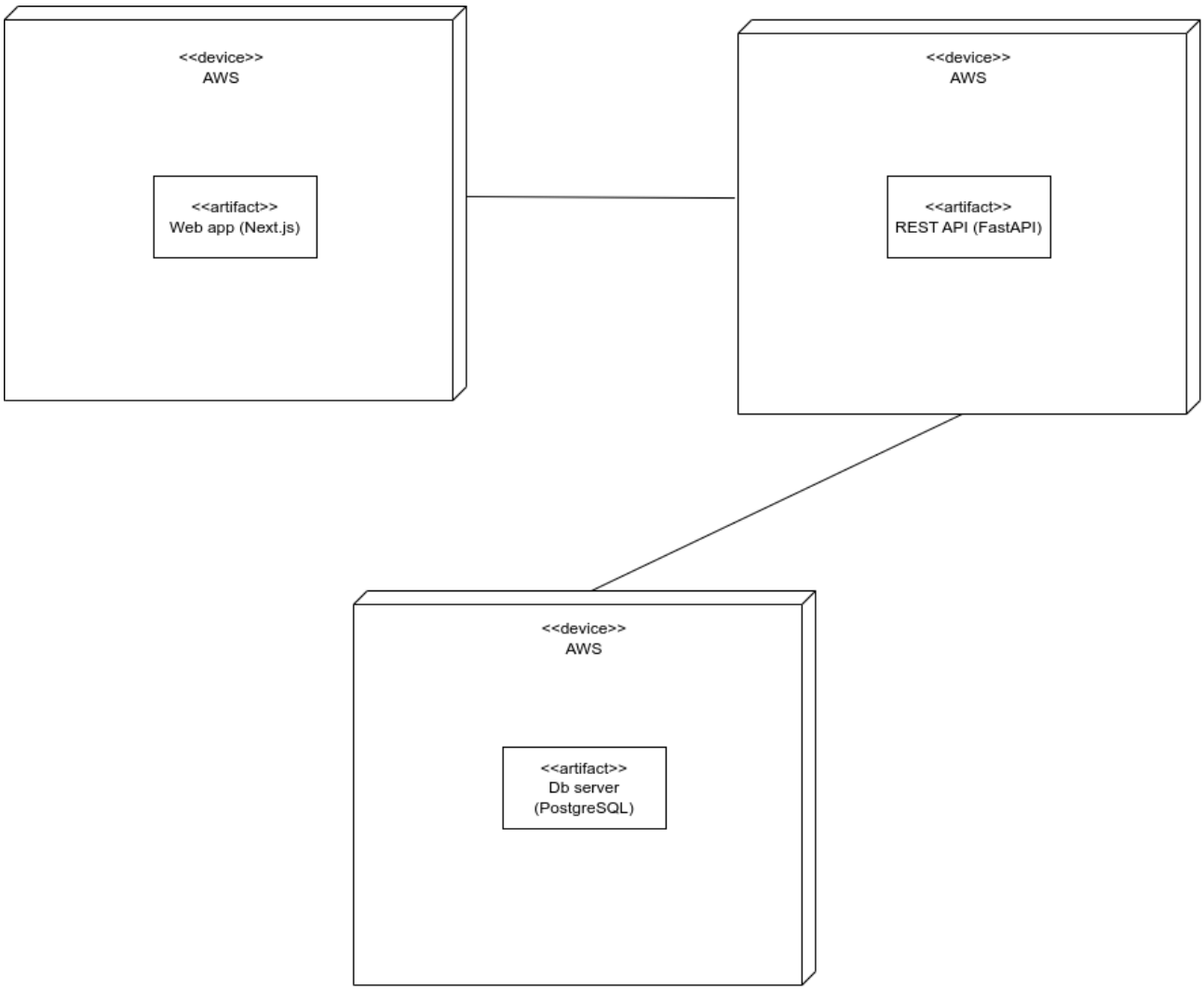

Рисунок 3.3 – Діаграма розгортання застосунку (рисунок виконано самостійно)

Такий підхід до архітектурного проєктування гарантує модульність, високу доступність, ефективність використання ресурсів і готовність системи до масштабування та інтеграції нових функцій у майбутньому.

3.3 Проєктування структури зберігання даних

У рамках проєктування структури зберігання даних програмної системи важливим аспектом є вибір надійної, масштабованої та безпечної системи управління базами даних. З огляду на ці критерії, для зберігання даних було обрано

PostgreSQL — об’єктно-реляційну СУБД з відкритим вихідним кодом, яка відзначається стабільністю, продуктивністю, підтримкою складних запитів і сучасних механізмів безпеки.

PostgreSQL широко використовується в різних сферах, включаючи фінансові, освітні, медичні та технологічні системи, і є ідеальним вибором для платформи менторства, яка оперує великою кількістю чутливих даних — профілів користувачів, історій взаємодії, фінансових транзакцій, повідомлень у чаті тощо. PostgreSQL гарантує дотримання принципів транзакційності (ACID), що забезпечує надійність кожної операції — від оплати послуги до створення профілю користувача [12, 13].

Крім того, PostgreSQL має потужні механізми безпеки, включаючи контроль доступу, шифрування з’єднань, підтримку SSL та гнучке керування правами користувачів. Це особливо важливо для веб-платформи, яка зберігає персональні дані, реквізити платіжних операцій та інші конфіденційні записи.

Базу даних планується розгорнути у хмарному середовищі, що забезпечить гнучкість, масштабованість та доступність. Платформи на кшталт DigitalOcean або AWS надають інструменти для автоматичного резервного копіювання, моніторингу продуктивності, керування інцидентами та масштабування інфраструктури у разі зростання навантаження. Це дозволить уникнути простоїв і гарантувати безперервну роботу застосунку навіть при підвищеній кількості одночасних користувачів.

Вибір PostgreSQL також пов’язаний із високою сумісністю з ORM-інструментами, зокрема SQLAlchemy у Python-екосистемі. Це спрощує розробку серверної частини, дозволяючи працювати з базою даних через об’єктну модель, уникати помилок у SQL-запитах та зосередитися на реалізації бізнес-логіки.

У системі було спроєктовано логічну модель бази даних, яка враховує основні сутності, необхідні для функціонування платформи. До них належать:

- користувачі (user);
- профілі (mentor/mentee);
- послуги (services);

- категорії (categories);
- бронювання сесій (bookings);
- інвойси (invoices);
- чат-переписки (conversations);
- повідомлення (messages);
- відгуки (reviews).

Ця модель дозволяє ефективно зберігати та обробляти всі критично важливі для платформи дані. Вона враховує зв’язки між сутностями (один-до-багатьох, багато-до-багатьох), забезпечує швидкий доступ до інформації, а також підтримує оптимізацію через індекси, обмеження цілісності та каскадні видалення.

У подальших версіях платформи розглядається впровадження реплікації бази даних, яка дозволить забезпечити ще більшу стійкість до збоїв і розподілення навантаження на читання між декількома серверами. Це відкриває шлях до побудови кластерної інфраструктури, що є важливим кроком для систем із потенційно великою кількістю одночасних запитів.

ER-діаграма (див. рис. 3.4) відображає структуру зв’язків між основними сутностями програмної системи та демонструє логічні взаємозв’язки між об’єктами бази даних. Вона є критично важливою для розуміння внутрішньої організації даних, допомагає ефективно реалізувати відношення між таблицями та забезпечити цілісність даних у процесі їх обробки.

Логічна модель системи (див. рис. 3.5) є основою для реалізації структури зберігання даних і гарантує масштабованість, узгодженість та безпеку інформації в межах розроблюваної веб-платформи.

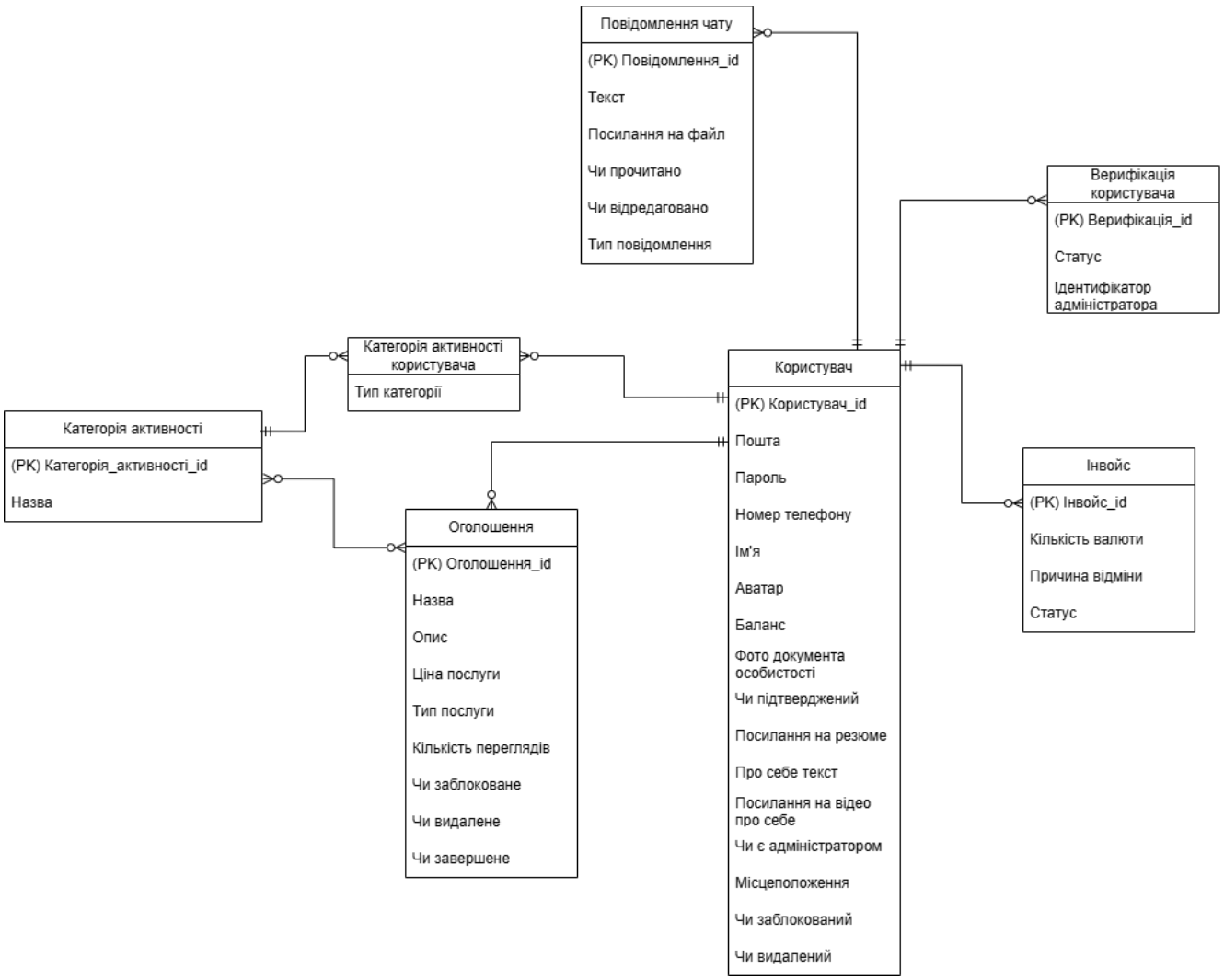

Рисунок 3.4 – ER-діаграма застосунку (рисунок виконано самостійно)

ER-діаграма наведена з метою візуалізації логічної структури бази даних застосунку та відображення взаємозв’язків між основними сутностями системи. Вона дозволяє чітко уявити, як пов’язані між собою користувачі, оголошення, повідомлення чату, інвойси, категорії активностей та інші компоненти. Таке графічне представлення є важливим етапом проєктування, оскільки допомагає уникнути помилок на рівні моделювання даних, забезпечити цілісність та узгодженість даних, а також спрощує подальшу реалізацію системи та супровід її бази даних.

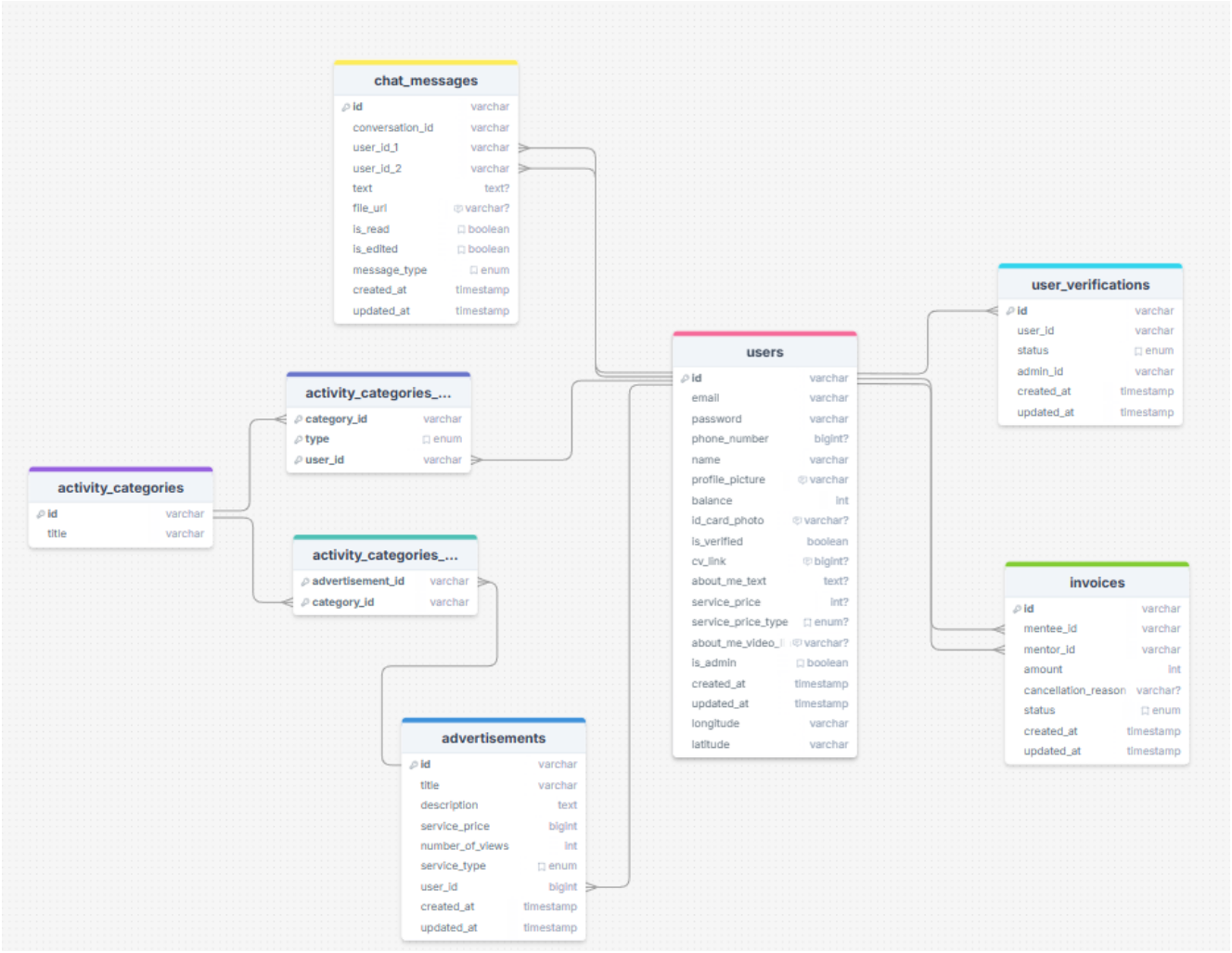

Рисунок 3.5 – Логічна модель бази даних застосунку (рисунок виконано самостійно)

Логічна модель складається з основних таблиць: **users** (користувачі), **advertisements** (оголошення), **transactions** (транзакції), **chat_conversations** та **chat_messages** (чат), **user_verifications** (верифікації), а також категорій діяльності (**activity_categories**) і зв’язків між ними. Таблиці пов’язані між собою зовнішніми ключами: користувачі створюють оголошення, надсилають повідомлення, проходять верифікацію й здійснюють транзакції. Модель забезпечує цілісне зберігання даних для реалізації основного функціоналу платформи.

3.4 Приклади найцікавіших алгоритмів та методів

Одним з ключових функціональних модулів платформи є система обробки платежів. Її реалізація здійснена за допомогою інтеграції з платіжною системою **Stripe [17]**, що забезпечує безпечне поповнення внутрішнього балансу користувачів. Алгоритм включає створення сесії оплати та обробку вебхука після завершення транзакції.

Користувач, який бажає придбати певну кількість кредитів (внутрішньої валюти платформи), ініціює запит до backend-сервісу. Система генерує сесію оплати через Stripe API, з вказанням типу оплати (картка), ціни, кількості, а також метаданих — електронної пошти користувача та суми поповнення. Після створення сесії, користувачу повертається `session_id`, за допомогою якого відбувається перенаправлення на платіжну сторінку:

```
checkout_session = stripe.checkout.Session.create(
    payment_method_types=["card"],
    mode="payment",
    line_items=[{
        "price": self.stripe_price_ids[credits_amount],
        "quantity": 1,
    }],
    success_url=f"{settings.WEB_URL}/en/profile",
    cancel_url=f"{settings.WEB_URL}/en/profile",
    metadata={
        "app_email": current_user.email,
        "credits_amount": credits_amount,
    },
)
```

Після успішної оплати Stripe надсилає POST-запит на заздалегідь вказаний webhook-ендпоінт. Система перевіряє підпис запиту, після чого виконується обробка події `checkout.session.completed`. З метаданих витягується електронна пошта та кількість кредитів. Далі з бази даних отримується користувач, його баланс збільшується, і зміни зберігаються:

```
if event["type"] == "checkout.session.completed":
    session = event["data"]["object"]
    customer_email = session["metadata"]["app_email"]
    credits_amount = session["metadata"]["credits_amount"]

    user = await
self.user_repository.get_user_by_email(customer_email)
    user.balance += int(credits_amount)
    await self.user_repository.save(user)
```

Для перевірки легітимності запиту використовується цифровий підпис (`stripe-signature`) та секретний ключ вебхука. У разі помилок (наприклад, недійсна підпис або пошкоджений payload) сервер повертає відповідні коди помилок (400):

```
    event = stripe.Webhook.construct_event(payload, sig_header,
endpoint_secret)
```

Цей алгоритм демонструє, як платформа забезпечує безпечну транзакційну логіку, автоматичне оновлення балансу та інтеграцію з сучасними фінансовими сервісами, дотримуючись високих стандартів захисту даних.

3.5 Розробка UI/UX системи

Естетика або зовнішній вигляд програмної системи є складовою зручності використання. Графічний інтерфейс дозволяє користувачеві інтуїтивно та легко взаємодіяти з додатком, тому при розробці дизайну було дотримано сучасних практик: зручна навігація, чітка структура інтерфейсу, адаптивність до різних пристроїв, контрастна кольорова гама, читабельна типографіка, узгоджені візуальні елементи, а також відображення станів системи та підказок [14].

Після відкриття додатку неавторизований користувач бачить форму входу, де може обрати авторизацію за допомогою Google або ввести пошту та пароль

(рисунок 3.6). Також передбачено зручне посилання для переходу на форму реєстрації.

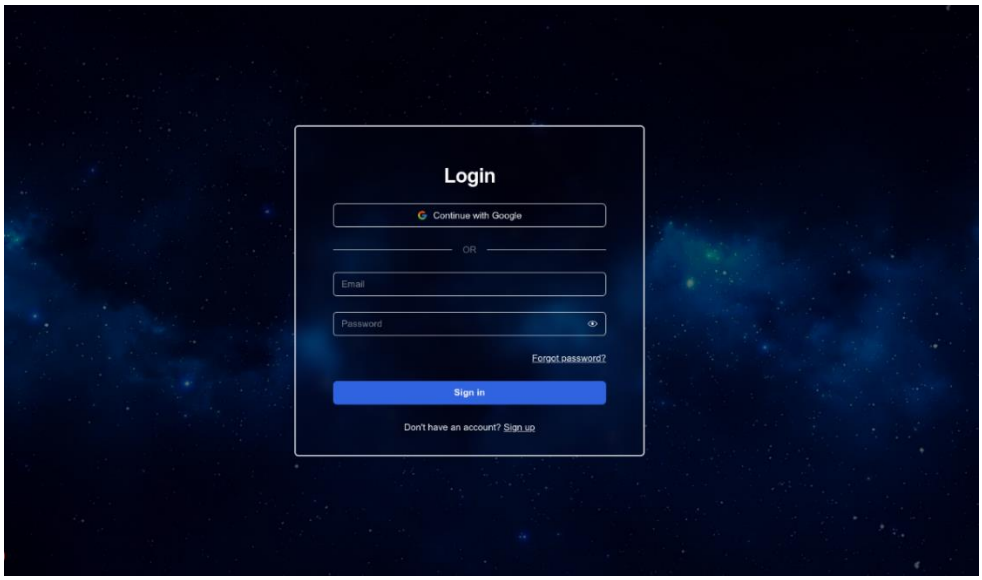

Рисунок 3.6 – Сторінка входу (рисунок виконано самостійно)

На сторінці реєстрації користувач вводить email, ім’я, номер телефону та пароль або використовує Google-реєстрацію. Всі нові користувачі створюються в системі з роллю "учень" без можливості вибору ролі під час реєстрації (рисунок 3.7).

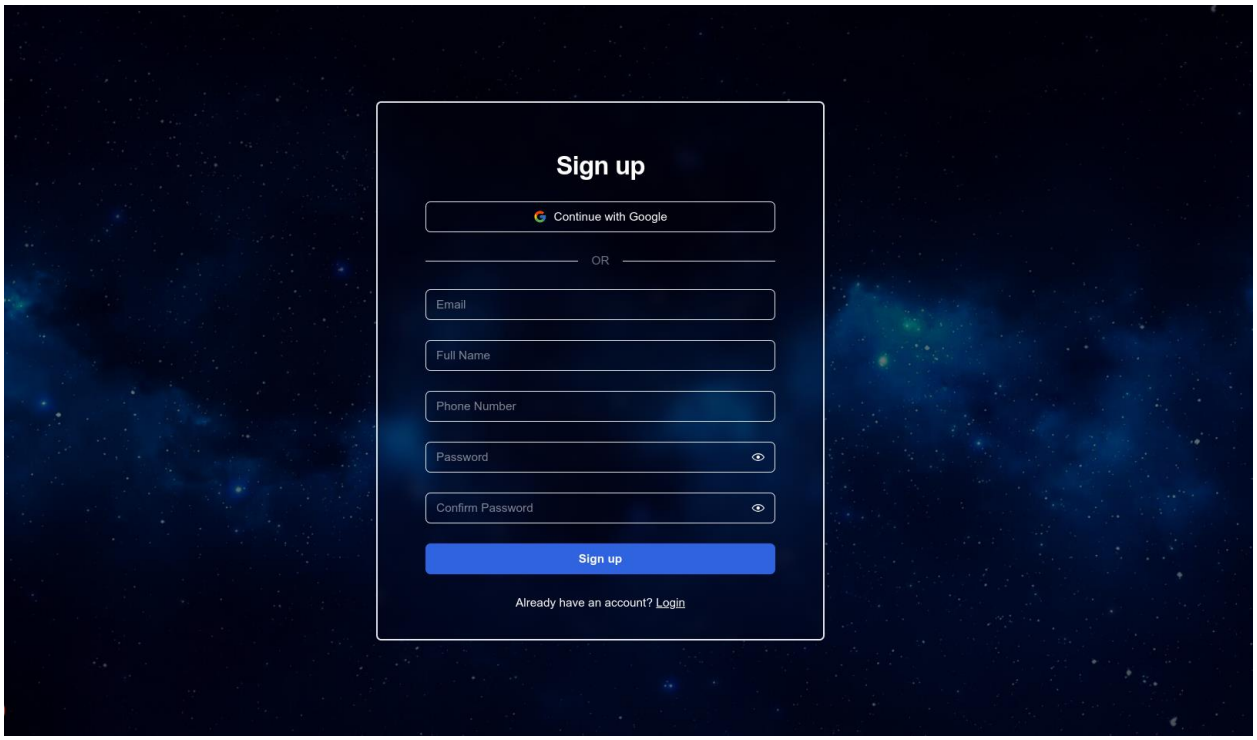

Рисунок 3.7 – Форма реєстрації користувача (рисунок виконано самостійно)

Після входу користувач потрапляє до особистого кабінету, де доступна форма для заповнення основної інформації: зображення профілю, ім’я, номер телефону, вибір категорій оголошень (рисунок 3.8).

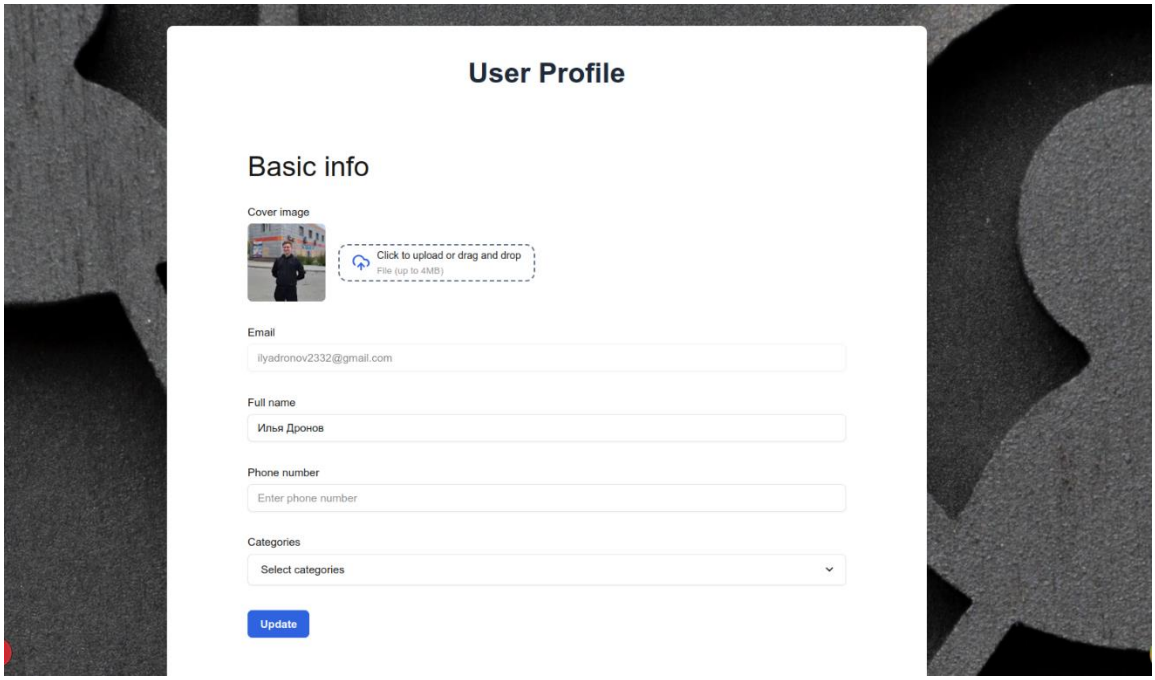

Рисунок 3.8 – Заповнення особистого профілю (рисунок виконано самостійно)

У разі подачі заявки на менторство, користувач заповнює додаткову форму, де вказує текст "про себе", завантажує резюме у PDF-форматі, додає відеопрезентацію, вказує вартість послуги, тип ціни та відповідні категорії (рисунок 3.9). Також завантажується фотографія документа, що посвідчує особу, як елемент модерації та верифікації.

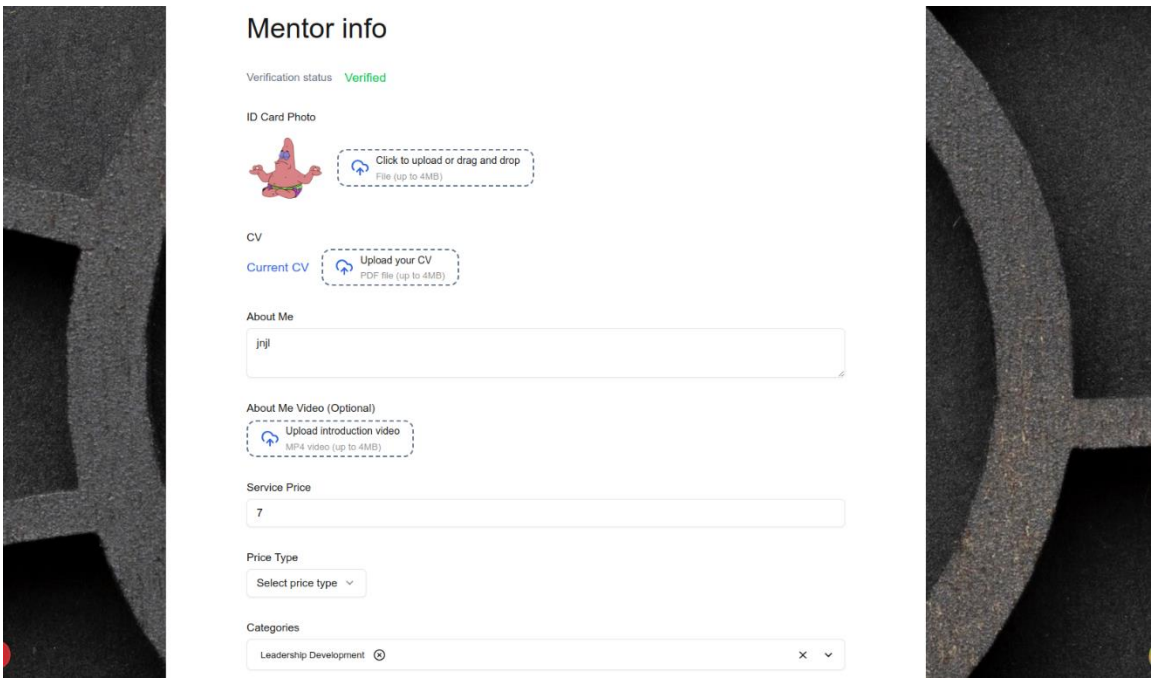

Рисунок 3.9 – Форма створення менторського профілю (рисунок виконано самостійно)

Система включає функціональність обміну повідомленнями — реалізований вбудований чат між користувачами, що дозволяє обговорювати майбутні сесії, узгоджувати умови співпраці або ставити уточнюючі запитання (рисунок 3.10).

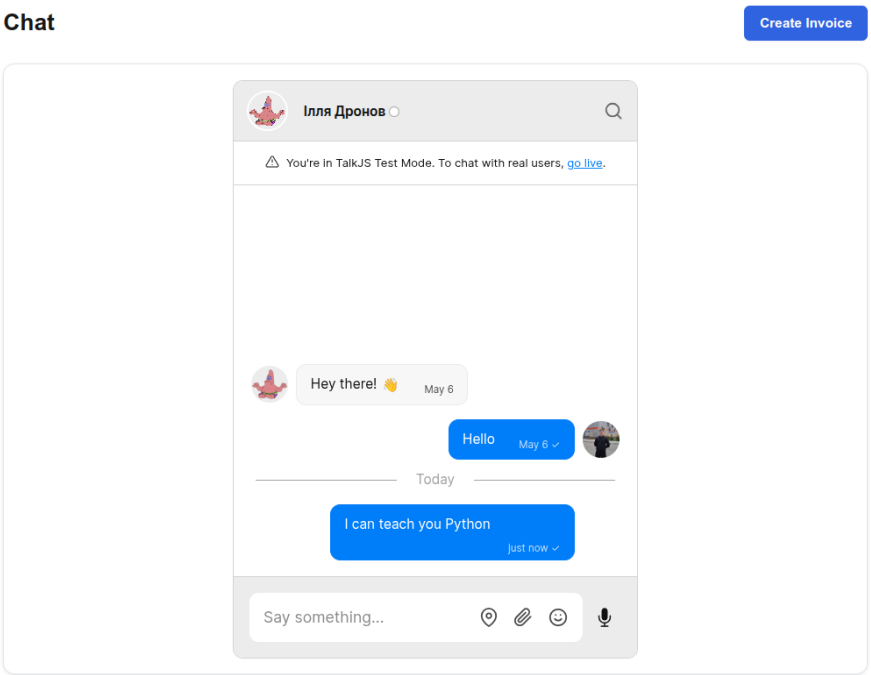

Рисунок 3.10 – Чат між користувачами (рисунок виконано самостійно)

У вкладці інвойсів користувачі можуть переглядати свої рахунки за менторські сесії, бачити деталі угоди, статус оплати та вартість послуги (рисунок 3.11).

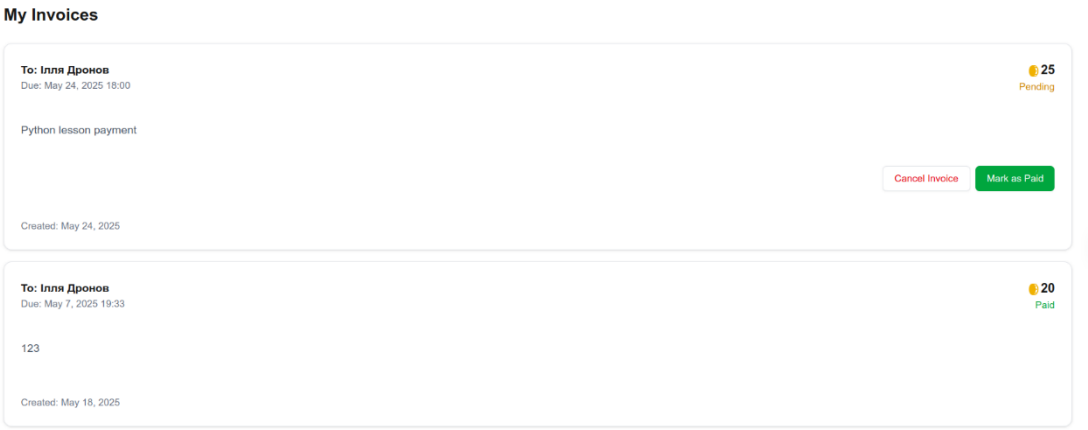

Рисунок 3.11 – Інвойс на оплату послуги (рисунок виконано самостійно)

Таким чином, інтерфейс системи спрямований на зручність, швидкість взаємодії, інтуїтивну логіку та приємне візуальне враження, що відповідає сучасним вимогам до UX/UI дизайну.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

Для реалізації веб-додатку було обрано стек технологій, що дозволяє забезпечити високу гнучкість, масштабованість та підтримку сучасних принципів UI/UX і архітектури.

Клієнтська частина додатку реалізована за допомогою фреймворку React та бібліотеки Next.js. Це дозволяє ефективно поєднувати серверний та клієнтський рендеринг, покращувати швидкодію сторінок та забезпечити пошукову оптимізацію (SEO). Для керування формами та валідації використовувались react-hook-form, zod, shadcn/ui [15] для компонування інтерфейсів, та TailwindCSS для стилізації.

Приклад компонента, що відповідає за фільтрацію та відображення постів у системі, включає використання фільтрації за типом послуги, сортуванням, назвою, діапазоном цін, а також пагінацію. Всі дані отримуються за допомогою Axios-запитів з використанням токена авторизації. Результати відображаються у вигляді карток із базовою інформацією про оголошення: назва, опис, ціна, тип послуги, кількість переглядів, користувач, дата публікації та категорії. Частина коду компонента:

```
export default function Home() {
  const { data: session } = useSession();
  const [posts, setPosts] = useState<Post[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(1);
  const lng = useParams().lng;

  const form = useForm<FilterValues>({
    resolver: zodResolver(filterSchema),
    defaultValues: {
      title: '',
      min_price: '',
      max_price: '',
      service_type: 'ALL',
      sort_field: 'created_at',
      sort_order: 'desc',
    },
  });

  const fetchPosts = async (page: number, filters: FilterValues) => {
    if (!session?.accessToken) return;

    try {
```

```
      const queryParams = new URLSearchParams({
        page: page.toString(),
        per_page: '9',
        sort_field: filters.sort_field,
        sort_order: filters.sort_order,
      });

      if (filters.title) queryParams.append('title', filters.title);
      if (filters.min_price) queryParams.append('min_price',
filters.min_price);
      if (filters.max_price) queryParams.append('max_price',
filters.max_price);
      if (filters.service_type && filters.service_type !== 'ALL')
        queryParams.append('service_type', filters.service_type);

      const response = await
axiosInstance.get<PaginatedResponse>(`/posts/?${queryParams}`, {
        headers: {
          Authorization: `Bearer ${session.accessToken}`,
        },
      });

      setPosts(response.data.items);
      setTotalPages(response.data.total_pages);
    } catch (error) {
      if (error instanceof AxiosError) {
        toast({
          title: 'Error',
          description: error.response?.data.message,
          variant: 'destructive',
        });
      } else {
        toast({
          title: 'Error',
          description: 'Failed to fetch posts',
          variant: 'destructive',
        });
      }
    } finally {
      setIsLoading(false);
    }
  };

  const onSubmit = (values: FilterValues) => {
    setCurrentPage(1);
    fetchPosts(1, values);
  };

  useEffect(() => {
    fetchPosts(currentPage, form.getValues());
  }, [currentPage, session?.accessToken]);

  if (isLoading) {
    return (
      <div className="container mx-auto py-10 flex justify-center">
        <p>Loading...</p>
      </div>
```

```
    );
  }

  return (
    <div className="container mx-auto py-10">
      <div className="mb-8">
        <Form {...form}>
          <form onSubmit={form.handleSubmit(onSubmit)}
className="space-y-6">
            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-
cols-6 gap-4">
              <FormField
                control={form.control}
                name="title"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Title</FormLabel>
                    <FormControl>
                      <Input placeholder="Search by title" {...field}
/>
                    </FormControl>
                  </FormItem>
                )}
              />

              <FormField
                control={form.control}
                name="min_price"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Min Price</FormLabel>
                    <FormControl>
                      <Input type="number" placeholder="Min price"
{...field} />
                    </FormControl>
                  </FormItem>
                )}
              />
// ... Other form fiels
}
```

Зовнішній вигляд цієї сторінки наведено на рисунку 4.1

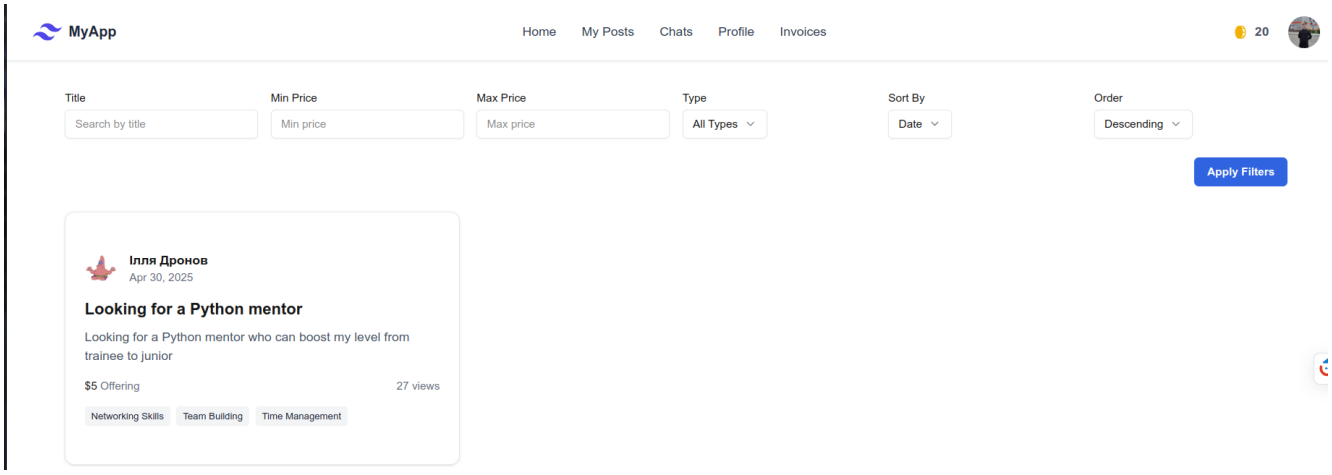

Рисунок 4.1 – Сторінка зі списком доступних постів (рисунок виконано самостійно)

Форма створення нового оголошення реалізована з підтримкою багатопольових форм, валідацією всіх полів, підтримкою мультівибору категорій, типу послуги (пошук або надання менторства) та ціни. Дані надсилаються через Axios POST/PUT запити з Bearer токеном авторизації. Код наведений в додатку В.

Зовнішній вигляд цієї форми наведено на рисунку 4.2

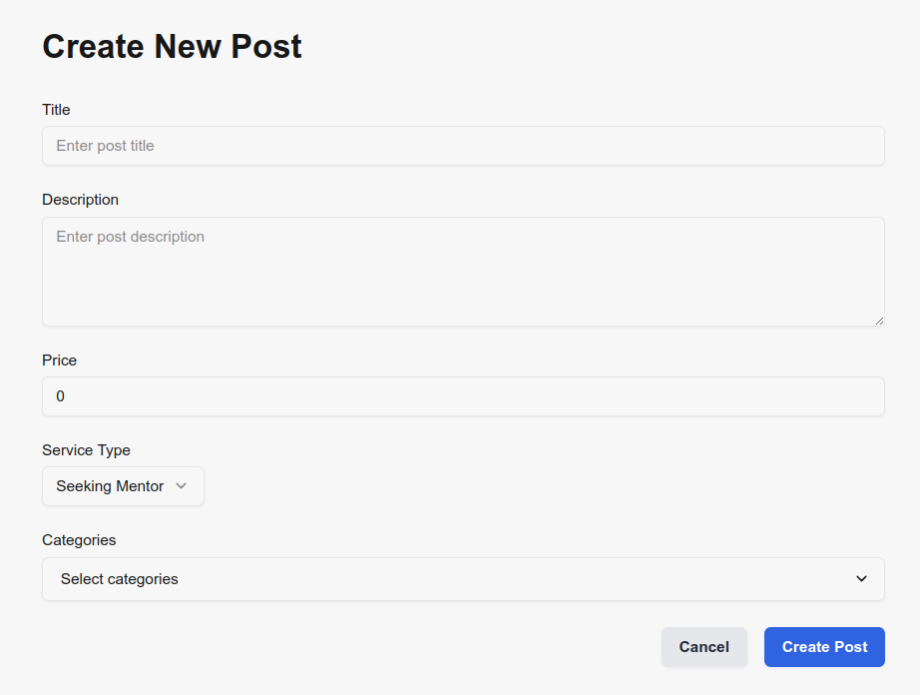

Рисунок 4.2 – Форма створення нового поста (рисунок виконано самостійно)

Для роботи з ORM у backend було використано SQLAlchemy. Це потужна бібліотека для роботи з реляційними базами даних у Python. SQLAlchemy дозволяє використовувати декларативний стиль моделювання сутностей за допомогою класів Python, підтримує складні зв'язки між таблицями, та інтегрується з FastAPI через асинхронну сесію.

Клас моделі LessonInvoice представляє рахунок за менторську сесію, з вказанням менті, ментора, суми, статусу, дати створення та оновлення. Для ідентифікації використовується UUID:

```
class LessonInvoice(Base):
    __tablename__ = "lesson_invoices"
    id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
primary_key=True, default=uuid.uuid4)
    mentor_id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
nullable=False)
    mentee_id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
nullable=False)
    amount: Mapped[int] = mapped_column(Integer, nullable=False)
    status: Mapped[InvoiceStatus] = mapped_column(String(1),
default=InvoiceStatus.PENDING.value,
server_default=InvoiceStatus.PENDING.value)
    created_at: Mapped[datetime] =
mapped_column(TIMESTAMP(timezone=True), default=lambda:
datetime.now(timezone.utc))
    updated_at: Mapped[datetime] =
mapped_column(TIMESTAMP(timezone=True), default=lambda:
datetime.now(timezone.utc), onupdate=lambda: datetime.now(timezone.utc))
```

Модель Post зберігає інформацію про оголошення: назва, опис, ціна, тип послуги, кількість переглядів, автор публікації, зв'язок із категоріями та дату створення/оновлення.

```
class Post(Base):
    __tablename__ = "posts"

    id: Mapped[uuid.UUID] = mapped_column(UUID(as_uuid=True),
primary_key=True, default=uuid.uuid4)
    title: Mapped[str] = mapped_column(String(200), nullable=False)
    description: Mapped[str] = mapped_column(Text, nullable=False)
    service_price: Mapped[float] = mapped_column(default=0)
    number_of_views: Mapped[int] = mapped_column(default=0)
    service_type: Mapped[ServiceTypes] = mapped_column(String(2),
nullable=False)
    user_id: Mapped[uuid.UUID] = mapped_column(ForeignKey("users.id",
ondelete="CASCADE"), nullable=False)
    created_at: Mapped[datetime] =
mapped_column(TIMESTAMP(timezone=True), default=lambda:
datetime.now(timezone.utc))
```

```
        updated_at: Mapped[datetime] =
mapped_column(TIMESTAMP(timezone=True), default=lambda:
datetime.now(timezone.utc), onupdate=lambda: datetime.now(timezone.utc))

        user: Mapped["User"] = relationship("User")
        categories: Mapped[list["ActivityCategoryPost"]] =
relationship("ActivityCategoryPost", back_populates="post")
```

Для взаємодії з клієнтською частиною система реалізує REST API. Приклад основних ендпоінтів для аутентифікації:

```
    @router.post("/auth/login")
    async def credentials_login(login_data: UserLoginInput, user_service:
UserService = Depends(get_user_service)) -> LoginResponse:
        return await user_service.authenticate_user(login_data)

    @router.post("/auth/login/google")
    async def google_login(token: TokenData, user_service: UserService =
Depends(get_user_service)) -> LoginResponse:
        return await user_service.google_login(token)

    @router.post("/auth/sign-up")
    async def register_user(sign_up_data: UserSignUpInput, user_service:
UserService = Depends(get_user_service)) -> LoginResponse:
        return await user_service.register_user(sign_up_data)

    @router.post("/auth/forgot-password/request")
    async def forgot_password(background_tasks: BackgroundTasks, email:
str = Body(..., embed=True), user_service: UserService =
Depends(get_user_service)) -> None:
        return await user_service.forgot_password(email,
background_tasks)
```

Ці ендпоїнти підтримують як класичну авторизацію за логіном/паролем, так і авторизацію через Google. Також реалізована реєстрація та запит на відновлення паролю.

Таким чином, описані програмні рішення дозволяють реалізувати надійну, сучасну, безпечну систему для взаємодії менторів та учнів на платформі.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Функціональне тестування

Щоб перевірити відповідність системи встановленим вимогам, було проведено функціональне тестування. Для цього були створені тестові випадки, які охоплюють ключову функціональність системи. Тестовий випадок являє собою набір кроків, умов та параметрів, що дозволяють перевірити правильність роботи певної функції. Приклад такого тесту для перевірки можливості додавання страви до системи наведено в таблиці 1.

Таблиця 5.1 - Тестовий випадок для функції додавання нового оголошення (таблиця виконана самостійно)

Інформація про тестовий випадок			
Ідентифікатор тестового випадку		SC01 ver1.0	
Власник тесту		Дронов Ілля Олександрович	
Місцезнаходження тесту(шлях)		/home/user/Desktop/tests/test-1.doc	
Дата останнього перегляду		22.05.2025	
Технічна вимога, що тестується		SC101	
Конфігурація засобів тестування		ST01	
Мета тесту		Виявити спроможність системи додавати нове оголошення	
Методика тестування			
Налаштування прогону тесту	Авторизований користувач із заповненим профілем.		
Крок	Дія	Очікуваний результат	Відмітка
1.	Перейти на сторінку «My posts»	Перехід на сторінку, де наявний список з постами користувача і кнопка «Add post»	Пройдено
2.	Натиснути на кнопку «Add post»	Перехід на сторінку з формою для додавання нового поста	Пройдено
3.	Ввести потрібні дані (назва, опис, категорії, тип поста)	Перехід на сторінку, де наявний список з постами користувача, доданий пост повинен з’явитися в списку	Пройдено

Кінець таблиці 5.1

Крок	Дія	Очікуваний результат	Відмітка
	та натиснути кнопку «Add new post»		
Очистка після прогону тесту	Видалити з бази даних створений пост		
Результати тесту			
Тестувальник: Дронов. І.О.		Дата прогону тесту: 22.05.2025	Результат тесту: Пройдено

Було також розроблено тестові сценарії для інвойсів, верифікацій, користувачів, а також для редагування, видалення вказаних сутностей та перегляду кожної з основних сторінок веб-платформи. Усі сценарії були успішно пройдені під час тестування системи. Проведення функціонального тестування є необхідним етапом перевірки, адже першочерговим завданням є впевнитися, що система реалізує всі функції, визначені у вимогах до програмного забезпечення.

5.2 Інтеграційне тестування

На підсумковому етапі розробки веб-сервісу було здійснено інтеграційне тестування з використанням фреймворку `pytest` [16], що дозволяє автоматизувати процес перевірки взаємодії між різними частинами backend-системи.

Це тестування використовує реальні стартові дані та базу даних PostgreSQL у тестовому середовищі. Кожний з тестів реалізовано через HTTP-запити до RESTful API з валідним JWT-токеном.

Наведено три тестові сценарії, які були зреалізовані за допомогою pytest.

5.2.1 Авторизація користувача

```
import pytest
from httpx import AsyncClient

@pytest.mark.asyncio
async def test_login_user(async_client: AsyncClient):
    response = await async_client.post("/auth/login", json={
        "email": "user@example.com",
        "password": "user123"
    })
    assert response.status_code == 200
    assert "access_token" in response.json()
```

5.2.2 Створення оголошення

```
@pytest.mark.asyncio
async def test_create_post(async_client: AsyncClient, access_token:
str):
    headers = {"Authorization": f"Bearer {access_token}"}
    response = await async_client.post("/posts/", json={
        "title": "Test mentorship",
        "description": "Helping with Python",
        "service_price": 50,
        "service_type": "P",
        "category_ids": ["categ-id-1"]
    }, headers=headers)

    assert response.status_code == 200
    assert response.json()["title"] == "Test mentorship"
```

5.2.3 Отримання списку оголошень

```
@pytest.mark.asyncio
async def test_get_posts(async_client: AsyncClient, access_token:
str):
    headers = {"Authorization": f"Bearer {access_token}"}
    response = await async_client.get("/posts/", headers=headers)
    assert response.status_code == 200
    assert isinstance(response.json()["items"], list)
```

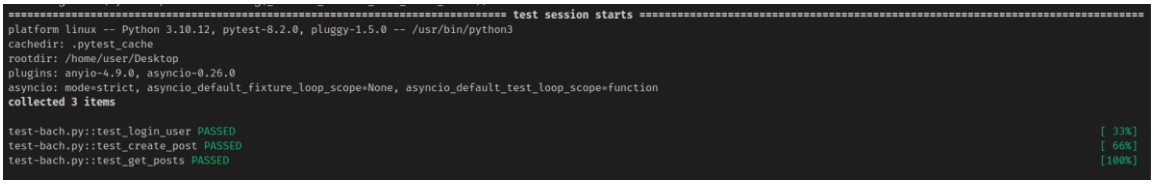

Рисунок 5.1 – Успішний результат виконання тестів (рисунок виконаний самостійно)

Усі тести пройшли успішно, що засвідчує про стабільну роботу API-шару та чітку реакцію на запити з аутентифікованих користувачів.

6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗБЕЧЕННЯ

6.1 Публікації

У межах дослідження спрямованого на розробку програмної системи для підтримки надання та отримання послуг, була підготовлена публікація для участі у VIII Всеукраїнській студентській науковій конференції «ЕКСПЕРИМЕНТАЛЬНІ ТА ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ В КОНТЕКСТІ СУЧАСНОЇ НАУКИ» [11].

Назва публікації: «Програмна система для надання та отримання начальних послуг». Повний текст публікації розміщено в додатку А до документа.

У публікації представлено огляд веб-системи “Knowlity” — сучасної менторської платформи, розробленої для підтримки індивідуальних менторів і їхніх учнів. Система забезпечує зручне управління профілями, планування сесій, безпечну комунікацію та фінансові транзакції. Завдяки використанню сучасних технологій, таких як Next.js, TailwindCSS, FastAPI, JWT та Google Sign-In, платформа вирізняється адаптивним інтерфейсом, гнучкою архітектурою і високим рівнем автоматизації, що робить її ефективним інструментом для онлайн-менторства.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто значного прогресу в розробці як клієнтської, так і серверної частин програмної системи для надання та отримання послуг у форматі менторства та обміну навичками. На початковому етапі було проведено аналіз предметної галузі, зокрема огляд існуючих рішень, таких як MentorCruise, Superpeer, ADPList та інших, що дало змогу виявити ключові виклики, з якими стикаються сучасні платформи — від нестачі гнучкості у сценаріях використання до недостатньої інтеграції засобів комунікації та оплати. Отримані результати стали підґрунтям для формулювання вимог до майбутньої системи.

Було визначено як функціональні, так і нефункціональні вимоги до програмної системи, серед яких — підтримка багаторольової взаємодії користувачів (менторів, учнів, адміністраторів), можливість створення та бронювання послуг, проведення платежів, ведення чатів, а також забезпечення безпеки й стабільності роботи системи. На основі цих вимог було спроєктовано загальну архітектуру програмного забезпечення.

У ході проєктування архітектури було реалізовано трьохрівневу модель системи з чітким розділенням презентаційного рівня (Next.js + Tailwind), рівня бізнес-логіки (FastAPI), та рівня доступу до даних (PostgreSQL). Архітектура забезпечує масштабованість, незалежне оновлення компонентів і зручність подальшої підтримки. Всі взаємодії реалізовано у форматі REST API з документуванням через Swagger, що значно спрощує інтеграцію та розширення системи.

Для зберігання даних обрано PostgreSQL, як надійну, транзакційно захищену систему управління базами даних. Вона забезпечує ACID-властивості, що критично важливо для обробки фінансових операцій та персональних даних користувачів. Було побудовано логічну модель сутностей, яка охоплює всі основні компоненти системи: користувачів, послуги, сесії, транзакції, повідомлення та відгуки. Розгортання системи планується у хмарному середовищі з підтримкою CI/CD, що

забезпечить гнучкість, надійність і швидке реагування на зміни.

Обраний стек технологій та архітектурні рішення дозволяють сформувати стабільну основу для подальшого розвитку проєкту, його масштабування та запуску в реальному середовищі з високими вимогами до продуктивності, безпеки та користувацького досвіду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Journals Indianapolis. The Impact of COVID-19 on Formal Mentorship. URL: https://journals.indianapolis.iu.edu/index.php/advancesinsocialwork/article/view/27605 (дата звернення 08.05.2025).

2. MentorCruise. Connecting Mentors & Mentees. URL: https://mentorcruise.com/ (дата звернення 08.05.2025).

3. Superpeer. Superpeer | Home to your community with livestreams, live courses, 1:1s and more! URL: https://superpeer.com/register (дата звернення 08.05.2025).

4. ADPList. Learn from the world's best mentors for free. URL: https://adplist.org (дата звернення 08.05.2025).

5. The Influence of AI in Modern Mentorship: Challenges and Opportunities. URL: https://www.pushfar.com/article/the-influence-of-ai-in-modern-mentorship-challenges-and-opportunities/ (дата звернення 08.05.2025).

6. Here is the reason why SQLAlchemy is so popular. URL: https://towardsdatascience.com/here-is-the-reason-why-sqlalchemy-is-so-popular-43b489d3fb00/ (дата звернення 08.05.2025).

7. What Is FastAPI: The Future of Modern Web Development. URL: https://www.simplilearn.com/what-is-fastapi-article (дата звернення 08.05.2025).

8. Mastering Next.js: Best Practices for Clean, Scalable, and Type-Safe Development. URL: https://medium.com/@PedalsUp/mastering-next-js-best-practices-for-clean-scalable-and-type-safe-development-626257980e60 (дата звернення 08.05.2025).

9. Unlocking E-commerce: Stripe Payment Gateway with Python. URL: https://medium.com/@nikhilwani05/stripe-payment-gateway-with-python-caad352d008c (дата звернення 08.05.2025).

10. Best practices for developing and deploying cloud infrastructure with the AWS CDK. URL: http://docs.aws.amazon.com/cdk/v2/guide/best-practices.html (дата звернення 08.05.2025).

11. Дронов І.О. Програмна система для надання та отримання навчальних послуг. VIII Всеукраїнська мультидисциплінарна студентська наукова конференція «Експериментальні та теоретичні дослідження в контексті сучасної науки». м. Львів. 2025. – С. 125-127.

12. Advantages of PostgreSQL. URL: https://www.cybertec-postgresql.com/en/postgresql-overview/advantages-of-postgresql/ (дата звернення 08.05.2025).

13. PostgreSQL Transaction. URL: https://neon.tech/postgresql/postgresql-tutorial/postgresql-transaction (дата звернення 08.05.2025).

14. How to Design Stunning Next.js UIs. URL: https://brisktechsol.com/nextjs-ui-designer/ (дата звернення 24.05.2025).

15. React Hook Form. URL: https://ui.shadcn.com/docs/components/form (дата звернення 24.05.2025).

16. Інтеграційні тести на Python з використанням pytest та FastAPI. Частина друга. URL: https://dou.ua/forums/topic/47957/ (дата звернення 24.05.2025).

17. Stripe documentation. URL: https://docs.stripe.com/ (дата звернення 27.05.2025).

18. Посилання на github репозиторій з вихідним кодом проекту: https://github.com/NureDronovIllia/bachelor-work