ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмних засобів для підтримки навчання професійних оцінювачів кави»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Костянтин КОЗАЧОК
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Костянтин КОЗАЧОК

Керівник: Ігор ГАМАНЮК
Старший викладач кафедри ІПЗ

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Козачку Костянтину Юрійовичу_____

1. Тема кваліфікаційної роботи: «Розробка програмних засобів для підтримки навчання професійних оцінювачів кави»
керівник кваліфікаційної роботи старший викладач кафедри ІПЗ Ігор ГАМАНЮК, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про процес навчання професійних оцінювачів кави, методики проведення дегустацій (cupping), документація до фреймворків Django та React, опис побудови клієнт-серверної архітектури, приклади реалізації освітніх веб-застосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі навчання професійних оцінювачів кави та визначення вимог до програмного забезпечення.
2. Огляд та аналіз засобів реалізації застосунку для підтримки навчання професійних оцінювачів кави.
3. Проєктування архітектури застосунку для підготовки оцінювачів кави.

4. Реалізація та тестування функціональності застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма розгортання.
6. Діаграма пакетів.
7. Діаграма діяльності.
8. Екранні форми.
9. Демонстрація роботи застосунку.
10. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Аналіз предметної галузі навчання професійних оцінювачів кави та визначення вимог до програмного забезпечення	14.03-19.03.2025	
4	Огляд та аналіз засобів реалізації застосунку для підтримки навчання професійних оцінювачів кави	20.03-28.03.2025	
5	Проєктування архітектури застосунку для підготовки оцінювачів кави	29.03-12.04.2025	
6	Реалізація та тестування функціональності застосунку	14.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач(ка) вищої освіти _____ Костянтин КОЗАЧОК
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор ГАМАНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 1 табл., 23 рис., 17 джерел.

Мета роботи – удосконалення процесу навчання та оцінювання професійних дегустаторів кави на основі розробленого ПЗ, що сприяє підвищенню ефективності підготовки фахівців у цій сфері.

Об'єкт дослідження – процес навчання та професійної підготовки оцінювачів кави.

Предмет дослідження – програмні засоби, що забезпечують підтримку навчального процесу та моделювання практичного оцінювання кави з використанням цифрових технологій.

Короткий зміст роботи: В роботі проаналізовано предметну галузь та методики навчання професійних оцінювачів кави відповідно до стандартів SCA та Q-Grader. Проведено огляд існуючих програмних рішень у сфері дегустації кави(Barista Hustle, Tastify, CoffeeMind Academy) та визначено вимоги до функціональності майбутнього застосунку. Реалізовано застосунок, який включає модулі для теоретичного навчання, тестування, симуляції дегустації та аналітики. Застосунок забезпечує реєстрацію, особистий кабінет, проходження курсів, взаємодію з чат-ботом асистентом на базі OpenAI API, формування дегустаційних сесій та збереження оцінок. Для реалізації використано Django та Django REST Framework на бекенді, React на фронтенді, PostgreSQL для збереження даних. Проведено тестування функціоналу.

Сферою використання застосунку є організація самостійного навчання та практичного тренування професійних оцінювачів кави.

КЛЮЧОВІ СЛОВА: BACK-END, FRONT-END, PYTHON, DJANGO, RESTful API, OPENAI API, ОЦІНКА ЯКОСТІ КАВИ, ДЕГУСТАЦІЯ, ВЕБЗАСТОСУНОК.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment

API – Application Programming Interface

REST – Representational State Transfer

SPA – Single Page Application

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

JS – JavaScript

DRF – Django REST Framework

JWT – JSON Web Token

JSON – JavaScript Object Notation

SQL – Structured Query Language

ORM – Object-Relational Mapping

ВСТУП

Актуальність: підготовка професійних оцінювачів кави є важливою складовою сучасної індустрії кави, оскільки саме від їхньої експертної оцінки залежить якість продукції та визнання на міжнародному ринку. Зі зростанням популярності спеціальності Q-Grader та розвитку цифрових технологій постає потреба в ефективних інструментах для навчання та самопідготовки. Використання вебзастосунків у цьому процесі дозволяє організувати навчання в зручному форматі, забезпечити візуалізацію, інтерактивність та зворотний зв'язок. Це робить процес засвоєння теоретичних знань та практичних навичок більш доступним, персоналізованим і гнучким.

Вебзастосунок виступає як універсальний інструмент для підтримки навчання професійних оцінювачів кави. Він об'єднує в собі курси, уроки, тести, інтерактивні симуляції дегустації та чат-бота асистента, що забезпечує комплексний підхід до підготовки. Цифрові інструменти дозволяють користувачам вивчати теоретичні основи оцінювання, закріплювати знання за допомогою тестування та практичних вправ, а також аналізувати результати дегустацій. Завдяки персоналізованому підходу, гнучкості у подачі матеріалу та інтеграції з OpenAI API, навчальний процес стає сучасним, зручним та ефективним.

Об'єктом дослідження є процес навчання та професійної підготовки оцінювачів кави.

Предметом дослідження є програмні засоби, що забезпечують підтримку навчального процесу та моделювання практичного оцінювання кави з використанням цифрових технологій.

Метою роботи є удосконалення процесу навчання та оцінювання професійних дегустаторів кави на основі розробленого ПЗ, що сприяє підвищенню ефективності підготовки фахівців у цій сфері.

Методи дослідження: першим етапом дослідження стало ознайомлення з професійними стандартами навчання кавових оцінювачів (Q-Grader, SCA), а також

аналіз існуючих рішень для підтримки навчального процесу в галузі дегустації кави. Це дозволило сформулювати функціональні та нефункціональні вимоги до майбутнього застосунку. Далі було проведено огляд стеку технологій, які відповідають визначеним вимогам. Для реалізації серверної частини було обрано фреймворк Django та мову програмування Python. Для фронтенду використано бібліотеку React. PostgreSQL обрано як об’єктно-реляційну систему управління базами даних. Для реалізації інтерактивного чат-бота застосовано OpenAI API. Всі компоненти об'єднано через REST API. Архітектуру програмного забезпечення спроєктовано з використанням UML-діаграм. Реалізацію функціоналу здійснено в кілька етапів із подальшим тестуванням, а інтерфейс користувача створено з урахуванням сучасних принципів зручності та доступності.

Наукова новизна роботи полягає у розробці програмного засобу, що поєднує в собі навчальний курс з теорії оцінювання кави, симуляцію процесу дегустації з можливістю введення оцінок та аналітику результатів. Особливістю є реалізація чат-бота на базі OpenAI API, який допомагає у навчанні користувача відповідно до тематики курсу та галузевих знань. У системі реалізовано автоматичну перевірку результатів тестування, що підвищує ефективність самостійного навчання.

Практична значущість результатів полягає у можливості використання створеного веб-застосунку для підготовки професійних дегустаторів кави в онлайн-форматі, що дозволяє проходити навчання у зручний час з більшості сучасних пристрої

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Професійні стандарти та методики навчання оцінювачів кави

Професійна діяльність оцінювачів кави є ключовою складовою сучасної кавової індустрії. Вона ґрунтується на суворих міжнародних стандартах, які визначають єдині підходи до органолептичного оцінювання якості кавового зерна, відбір зразків для дегустації, і навчання нових фахівців. Сьогодні професія кавового оцінювача визнана на міжнародному рівні, і її представники відіграють важливу роль у забезпеченні стабільної якості продукції по всьому світу.

Спеціалісти, що мають кваліфікацію Q-Grader, здатні оцінювати як арабіку, так і робусту, проводячи аналіз смакових властивостей, дефектів, балансу, аромату та інших характеристик кави[5]. Підготовка таких фахівців здійснюється відповідно до програми сертифікації, яка розроблена Інститутом якості кави (Coffee Quality Institute – CQI) у співпраці зі Specialty Coffee Association (SCA). Під час навчання кандидат проходить інтенсивний курс, який включає 19 тестів, що охоплюють сенсорний аналіз, фізичну класифікацію зерна, ідентифікацію дефектів, опис смакових профілів, а також здатність до об'єктивного порівняння зразків[2]. Усі тести проводяться в суворо стандартизованих умовах, що дозволяє гарантувати повторюваність результатів. Після успішного складання тестів фахівець отримує сертифікат Q-Grader, який діє протягом трьох років, після чого потребує оновлення.

Завдяки впровадженню сертифікації Q-Grader вдалося створити універсальний підхід до оцінювання кави незалежно від країни походження, що сприяє формуванню єдиної глобальної кавової спільноти. Такий підхід дозволяє виробникам, обжарювальникам, постачальникам та бариста працювати з чітким розумінням якості кавового продукту, ґрунтуючись на спільних критеріях та показниках[5].

Паралельно з цим, особливу роль у професійній підготовці фахівців відіграє модульна система навчання, затверджена Specialty Coffee Association. Вона орієнтована не лише на оцінювачів, але й на ширший спектр професіоналів у галузі кави. Навчальна програма Coffee Skills Program включає шість основних напрямів, що охоплюють ключові аспекти кавового ланцюга: від основ знань про каву до спеціалізованих навичок приготування напоїв.

До першого модуля входить вступ до світу кави, де слухачі дізнаються про її історію, географію вирощування, сорти та види обробки. Другий модуль присвячено сенсорному аналізу: тут учасники вчаться розрізняти смаки, аромати та тілесність кави, формуючи сенсорну карту. Третій напрям включає глибоке вивчення зеленого зерна – його ботанічні особливості, методи вирощування, сортування, дефекти та критерії якості. Наступний модуль фокусується на процесі обсмажування, де розглядаються режими термообробки, зміни в хімічному складі зерна та вплив температури на смак[3]. П’ятий модуль присвячено бариста-навичкам – підготовці еспресо, налаштуванню обладнання, роботі з молоком та подачі напоїв[4]. І нарешті, шостий модуль стосується управління кавовим бізнесом – від організації кав’ярень до роботи з постачальниками, клієнтами та аналітики продажів.

Завдяки комплексному охопленню різних аспектів кавової індустрії та чіткій структурі навчання, програми SCA дозволяють формувати висококваліфікованих фахівців, які можуть працювати в будь-якому сегменті ринку – від виробництва до обслуговування кінцевого споживача[1].

Таким чином, професійні стандарти та методики навчання оцінювачів кави, затверджені міжнародними організаціями, створюють надійну основу для підготовки компетентних спеціалістів, які здатні забезпечити якість і сталість у сфері кави. Вони враховують як об'єктивні методи сенсорного аналізу, так і практичні навички, необхідні для реальної роботи з кавовим продуктом[3]. Такий підхід гарантує не лише професіоналізм, але й відповідність найвищим вимогам сучасного кавового ринку.

1.2 Аналіз існуючих програмних рішень

На сучасному етапі розвитку цифрової освіти існує низка спеціалізованих онлайн-рішень, які спрямовані на підтримку професійного навчання у сфері оцінювання кави. Ці платформи надають доступ до структурованих навчальних матеріалів, інтерактивних курсів, симуляцій та аналітичних інструментів, які допомагають користувачам поглибити знання про каву, розвинути сенсорні навички та покращити практичні вміння.

Для аналізу було обрано три найбільш відомі приклади: Barista Hustle, Tastify та CoffeeMind Academy.Розглянемо їх детальніше.

1.2.1 Barista Hustle

Barista Hustle - це професійна онлайн-платформа, орієнтована на навчання барист та спеціалістів із приготування кави. Вона пропонує спеціалізовані курси з поглибленим контентом, створеним експертами галузі, та дозволяє користувачам опанувати знання й навички, необхідні для досягнення високого рівня майстерності в сфері кавової індустрії. Платформа використовується як новачками, так і досвідченими фахівцями, які прагнуть підвищити кваліфікацію або отримати міжнародну сертифікацію.

Основні функції Barista Hustle включають:

- Онлайн-курси: модулі з теоретичними знаннями та практичними порадами з приготування еспресо, молочних напоїв, екстракції, води, фільтрації, тощо.

- Сертифікація: можливість проходження іспитів і отримання сертифікатів після завершення курсів.

- Відеоуроки: візуальна подача матеріалу з демонстрацією кавових технік.

- Завдання і тести: перевірка засвоєння матеріалу після кожного модуля.

- Спільнота користувачів: обговорення тем, підтримка, обмін досвідом між учасниками.

- Доступ до блогу та статей: регулярне оновлення контенту з галузевими новинами й дослідженнями.

Переваги:

- Висока якість навчального контенту, розробленого професіоналами кавової індустрії.

- Орієнтованість на практичне застосування знань.

- Можливість сертифікації після проходження курсу.

- Інтуїтивно зрозумілий інтерфейс та зручність у користуванні.

- Актуальність матеріалів - оновлення відповідно до сучасних стандартів.

Недоліки:

- Переважно англомовний інтерфейс і матеріали, що може бути бар'єром для деяких користувачів.

- Платна підписка: безкоштовний доступ обмежений.

- Відсутність повноцінної мобільної версії додатку.

- Вузька спеціалізація: зосередженість виключно на барист і неохоплення ширшого спектра дегустаційних навичок

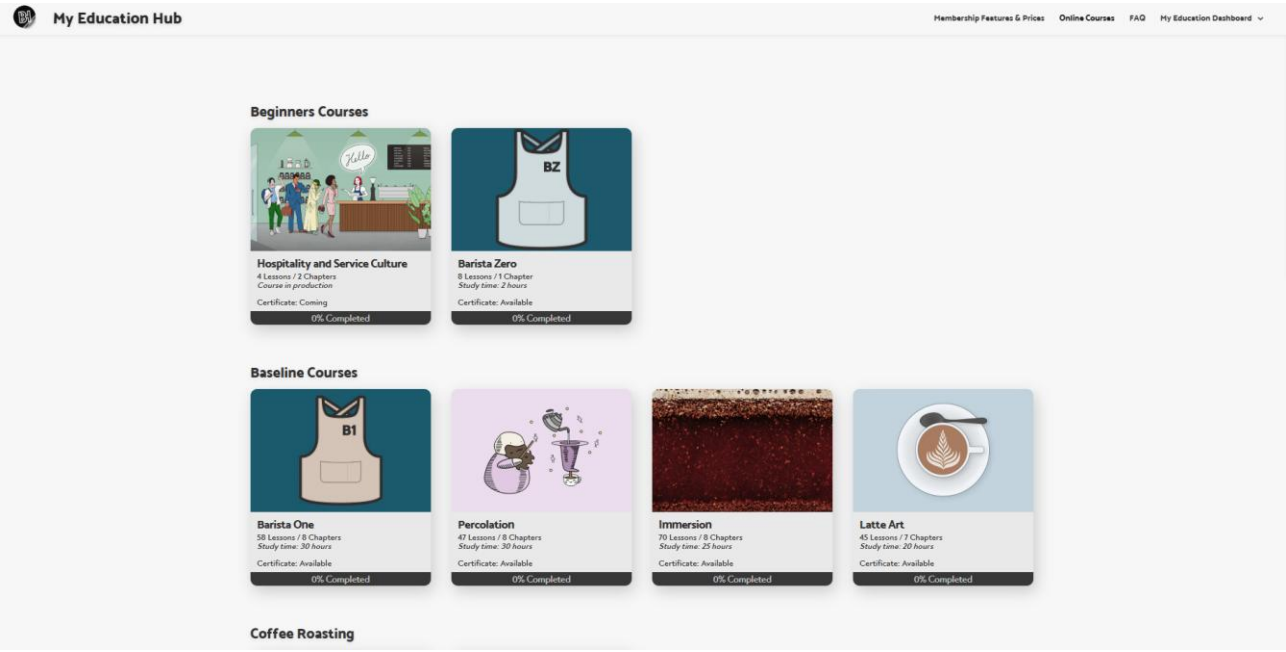


Рис. 1.1 Приклад сторінки доступних курсів Barista Hustle

Рис. 1.2 Приклад сторінки уроку Barista Hustle

1.2.2 Tastify

Tastify - це спеціалізований веб-сервіс, створений для підтримки процесу дегустації кави. Платформа була розроблена компанією Coffee Enterprises і надає дегустаторам можливість фіксувати, аналізувати та візуалізувати результати оцінювання смакових характеристик кави. Tastify широко використовується в професійних дегустаційних сесіях, на змаганнях, а також в освітніх цілях під час навчання Q-Grader'ів.

Основні функції Tastify включають:

- Створення дегустаційних протоколів: користувачі можуть створювати цифрові SCA-форми оцінювання кави.

- Візуалізація оцінок: побудова графіків смакових характеристик у вигляді “flavor wheel” або павутинних діаграм.

- Порівняння зразків: зручне зіставлення оцінок різних зразків кави.

- Хмарне збереження даних: результати дегустацій доступні в особистому кабінеті.

- Можливість роботи в команді: запрошення інших учасників до сесії для колективної оцінки.

- Звіти у PDF: експорт результатів для подальшого аналізу або надання клієнтам.

Переваги:

- Професійна орієнтація: відповідає стандартам SCA та зручно інтегрується у навчальні та сертифікаційні процеси[1].

- Наглядність даних: чітка графічна візуалізація результатів дегустації.

- Збереження історії дегустацій: можливість аналізу змін оцінок у часі.

- Дружній інтерфейс, адаптований для сенсорних екранів.

Недоліки:

– Обмежений функціонал без підписки: деякі функції доступні лише у платній версії.

– Відсутність мобільного додатку.

– Вузький фокус: платформа не охоплює навчання теоретичним аспектам чи приготуванню кави, а лише дегустацію.

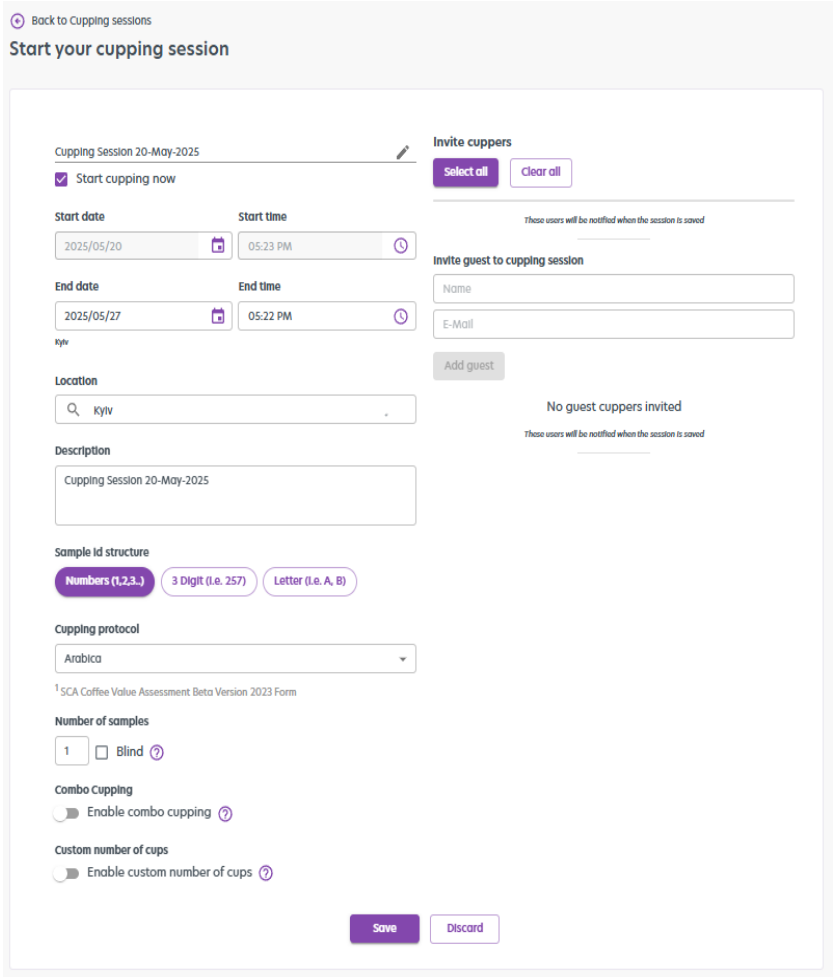


Рис. 1.3 Приклад сторінки створення дегустаційної сесії Tastify

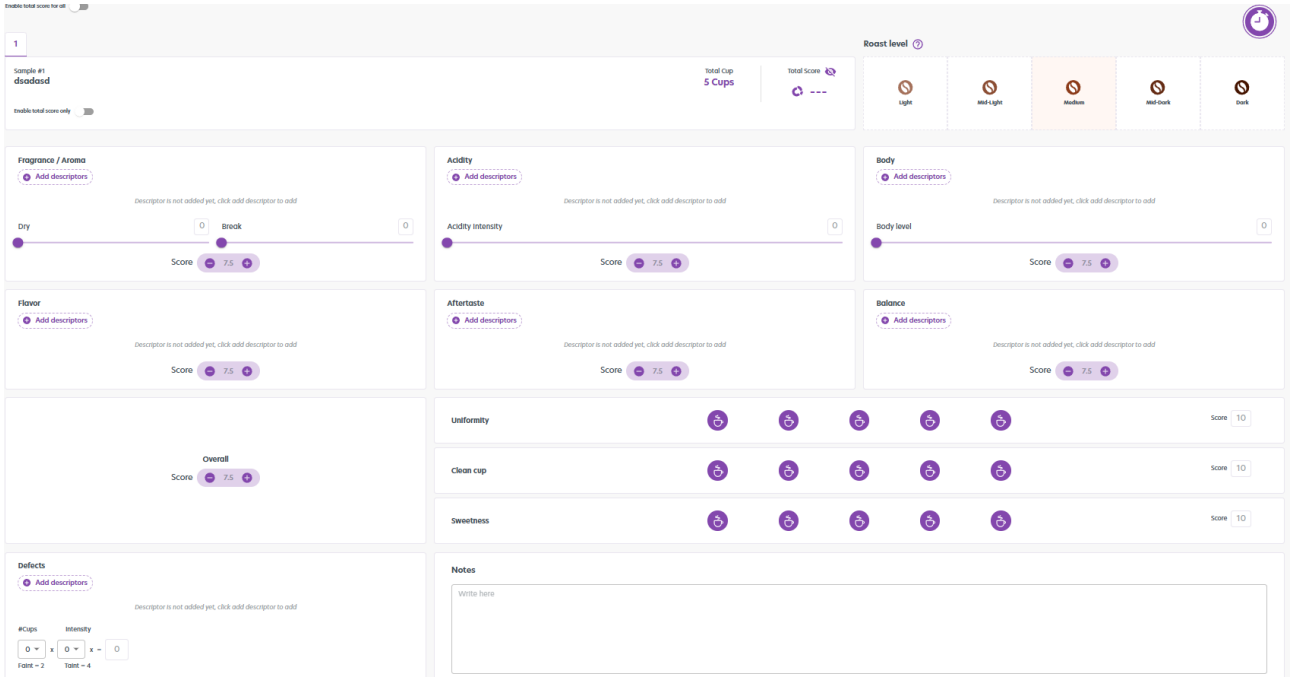


Рис. 1.4 Приклад сторінки дегустаційної сесії Tastify

1.2.3 CoffeeMind Academy

CoffeeMind Academy - це навчальна онлайн-платформа, створена данським інститутом CoffeeMind, який спеціалізується на підготовці кавових спеціалістів. Академія пропонує комплексні курси для тих, хто прагне поглибити знання в галузі сенсорного аналізу, обсмажування кави та підвищення загального рівня кавової освіти. Платформа тісно співпрацює з SCA та орієнтована на формування наукового підходу до процесу навчання[1].

Основні функції CoffeeMind Academy включають:

- Онлайн-курси з відеолекціями: навчальні матеріали записані з експертами галузі.

- Сертифіковані модулі: курси узгоджені з програмами SCA.

- Інтерактивні завдання: завдання для самоперевірки та аналізу засвоєного матеріалу.

- Курси з сенсорного аналізу, обсмажування, бізнесу в кавовій індустрії.

- Можливість участі в живих онлайн-сесіях або проходження матеріалу у власному темпі.

- Доступ до презентацій і навчальних матеріалів після завершення курсу.

Переваги:

- Академічний підхід: навчання базується на наукових дослідженнях і сучасних методиках.

- Сертифікація: після завершення курсу користувач отримує сертифікат, що визнається в кавовій спільноті.

- Гнучкість: можливість навчатися у будь-який час, у власному темпі.

- Глибокий фокус на сенсорику: особлива увага приділяється розвитку сенсорних навичок.

Недоліки:

- Англомовний контент: відсутність перекладів може створити бар'єри для деяких користувачів.

- Орієнтація на професіоналів: для новачків деякі теми можуть бути складними.

- Відсутність практичної дегустації: через онлайн-формат користувач не має змоги безпосередньо дегустувати зразки в рамках курсу.

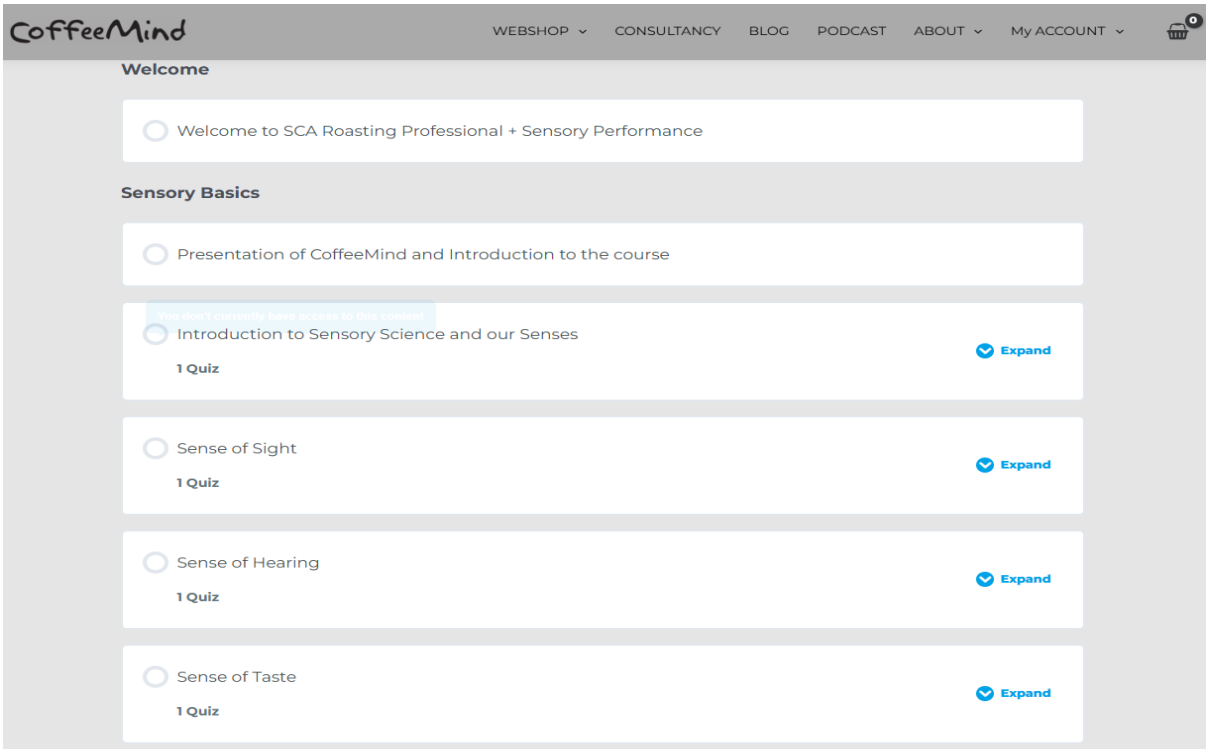


Рис. 1.5 Приклад сторінки курсу CoffeeMind Academy

Кожен з розглянутих застосунків має свої особливості у структурі навчального матеріалу, підходах до оцінювання та рівні інтерактивності. Деякі платформи орієнтовані на самостійне навчання з доступом до теоретичних матеріалів, тоді як інші більше акцентують увагу на практичних вправах, зокрема — дегустаціях кави. Особливу увагу варто приділити способам оцінювання та аналізу результатів, які можуть бути як автоматизованими, так і виконуватись вручну.

Щоб полегшити порівняння між платформами та визначити їх сильні та слабкі сторони, нижче наведено зведену таблицю аналізу ключових характеристик кожного з досліджених інструментів. Таблиця враховує тип матеріалів, рівень інтерактивності, формат навчання, механізми оцінювання, а також наявність функціоналу аналізу дегустацій та особливості доступу до ресурсів.

Таблиця 1.1

Зведена таблиця аналізу існуючих застосунків для навчання оцінювачів кави та їх порівняння

Показник	**Barista Hustle**	**Tastify**	**CoffeeMind Academy**
Тип матеріалів	Відеолекції, тести, статті	Панелі оцінювання, дегустаційні інструменти	Курси з сенсорики, відео та тести
Інтерактивність	Середня	Низька	Висока
Формат навчання	Курси з теорії та практики	Інструменти для капінгу	Модульне навчання з квізами

Продовження таблиці 1.1

Зведена таблиця аналізу існуючих застосунків для навчання оцінювачів кави та їх порівняння

Оцінювання	Квізи	Автоматичний підрахунок балів	Квізи
Аналіз дегустацій	Відсутній	Присутній	Опосередковано
Доступність	Передплата	Онлайн доступ із обліковим записом	Онлайн доступ через сайт

1.3 Визначення вимог до застосунку

На основі аналізу професійних стандартів, методик навчання, сучасних цифрових рішень і функціональних особливостей платформи, було сформульовано перелік функціональних та нефункціональних вимог до розроблюваного застосунку для навчання та оцінювання професійних дегустаторів кави.

Функціональні вимоги:

- Реєстрація та авторизація користувачів із перевіркою даних.
- Ведення особистого кабінету з відображенням прогресу користувача.
- Надання теоретичних навчальних матеріалів з оцінки якості кави.
- Проведення інтерактивного тестування для перевірки знань.
- Модуль симуляції дегустації з можливістю введення оцінок.
- Автоматична перевірка результатів навчання та тестів.
- Надання зворотного зв’язку щодо результатів.
- Адміністративна панель для управління контентом і користувачами.
- Відображення статистики та аналітики результатів.
- Інтеграція чат-бота асистента на основі OpenAI API для допомоги в навчанні.

Нефункціональні вимоги:

- Забезпечення безпеки даних і шифрування при авторизації.
- Завантаження сторінок менше ніж за 2 секунди.
- Адаптивний інтерфейс для мобільних пристроїв, планшетів та ПК.
- Підтримка сучасних браузерів (Chrome, Firefox, Safari, Edge).

1.4 Інформаційна модель предметної галузі

Інформаційна модель предметної галузі — це абстрактне уявлення ключових сутностей та їх взаємозв'язків у рамках системи підготовки професійних оцінювачів кави. Метою побудови такої моделі є формалізація логіки взаємодії між користувачами, навчальними об'єктами, тестами, результатами й іншими компонентами. Це дає змогу ефективно спроєктувати як структуру бази даних, так і бізнес-логіку програмного забезпечення.

У центрі інформаційної моделі знаходиться студент, який взаємодіє із системою через проходження тестів, створених на основі білетів. Студент має базову інформацію (ім'я, прізвище) та пов'язаний зі звітом (Report), у якому зберігаються результати його відповідей. Звіт, у свою чергу, формується оцінювачем (Evaluator) — кваліфікованим фахівцем, який перевіряє відповіді студентів та надає оцінки за кожен білет чи питання.

Кожен білет (Ticket) складається з певного набору питань (Question), що належать до конкретної тематики. Питання формуються на основі матеріалів з теми кави (Coffee), яка може включати опис, класифікацію сортів, методи обсмажування, географію походження тощо. Таким чином, забезпечується змістовна відповідність між теоретичним матеріалом і перевіркою знань.

Кожне питання має набір відповідей (Answer), серед яких можуть бути як правильні, так і неправильні варіанти. Оцінювання здійснюється за допомогою об'єкта Evaluation, у якому зберігається відповідь студента та її оцінка. Це дає змогу системі не лише збирати статистику, а й аналізувати якість засвоєння матеріалу за окремими темами.

Тести (CoffeeTest) — це структуровані перевірочні матеріали, що формуються на основі білетів і питань. Їх може створювати уповноважена особа (QuestionCreator), яка несе відповідальність за якість і релевантність питань.

Кожен елемент моделі чітко пов’язаний з іншим. Наприклад, одне питання не може існувати без теми, білет — без питань, а оцінка — без питання та відповіді. Це забезпечує логічну цілісність та унеможливлює появу ізольованих або «сирітських» записів у базі даних[16].

На рисунку нижче представлена діаграма предметної галузі, яка відображає зв’язки між основними сутностями системи та характер їх відношень (один до одного, один до багатьох, тощо).

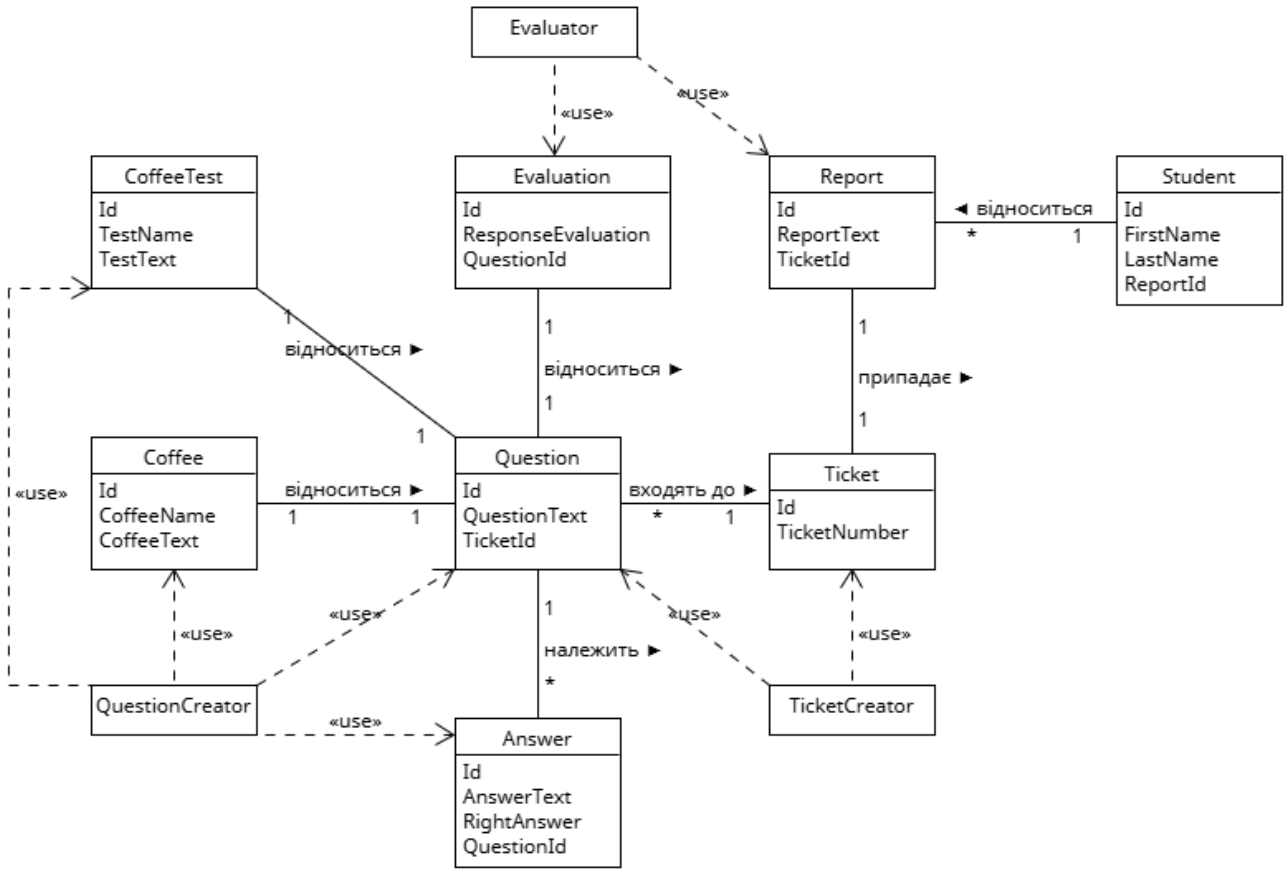


Рис. 1.6 Діаграма предметної галузі у сфері підготовки оцінювачів кави

2 ЗАСОБИ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) - це комплексне програмне забезпечення, яке надає розробникам повний набір інструментів для написання, тестування та налагодження коду. IDE зазвичай включає текстовий редактор, інструменти для автоматичної збірки, дебагер та засоби інтеграції з системами контролю версій. У межах розробки застосунку для підтримки навчання оцінювачів кави було використано два основних середовища: Visual Studio Code та PyCharm.

2.1.1 Visual Studio Code

Visual Studio Code (VS Code) - це потужний текстовий редактор з відкритим вихідним кодом, розроблений компанією Microsoft. Попри те, що VS Code не є повноцінним інтегрованим середовищем розробки у класичному розумінні, завдяки численним плагінам він може виконувати роль повнофункціонального IDE для більшості сучасних мов програмування, зокрема TypeScript, JavaScript, HTML, CSS та інших.

У межах розробки клієнтської частини вебзастосунку платформи для навчання оцінювачів кави Visual Studio Code використовувався для створення React-компонентів, стилізації інтерфейсу, налаштування маршрутизації, обробки API-запитів і керування станом додатку. Розширення, такі як ESLint, Prettier, GitLens, а також вбудований термінал значно полегшили процес написання чистого, читабельного коду[8].

Завдяки можливостям інтеграції з Git, VS Code забезпечив зручне керування репозиторієм проєкту без необхідності перемикатися між інструментами.

Наявність інтелектуального автодоповнення, контекстної документації та відладчика сприяла продуктивній роботі розробника в межах єдиного інтерфейсу.

Таким чином, VS Code став основним інструментом розробки фронтенд-частини системи, забезпечивши швидке створення адаптивного та функціонального користувацького інтерфейсу.

2.1.2 PyCharm

PyCharm - це професійне інтегроване середовище розробки, створене компанією JetBrains спеціально для мови Python. У проєкті застосунок PyCharm використовувався для реалізації серверної логіки вебплатформи, зокрема - розробки API на основі Django REST Framework, налаштування моделей бази даних, маршрутизації запитів та логіки обробки даних[7].

Перевагою PyCharm є глибока інтеграція з Django, що дозволяє автоматично розпізнавати структуру проєкту, налаштовувати середовище розробки, зручно працювати з шаблонами, формами, сигналами, а також здійснювати повнотекстовий пошук по всіх компонентах застосунку. Крім того, середовище підтримує створення та активацію віртуального середовища, управління пакетами через вбудований інтерфейс, а також має потужний інструмент дебагінгу.

Для роботи з базою даних PostgreSQL використовувались вбудовані інструменти перегляду таблиць, виконання SQL-запитів та відлагодження моделей[13]. Також середовище забезпечує повноцінну інтеграцію з Git, що сприяло ефективному контролю версій та спільній роботі над кодом.

Таким чином, PyCharm став ключовим середовищем для розробки бекенд-частини системи, забезпечуючи стабільну роботу API, безпеку даних та зручну архітектуру взаємодії з фронтендом.

2.2 Мови програмування

Мови програмування - це формалізовані системи запису, призначені для створення інструкцій, які виконуються комп’ютером. Вони дозволяють розробникам реалізовувати алгоритми, взаємодіяти з апаратним забезпеченням, опрацьовувати дані, створювати інтерфейси та програмні продукти різного призначення. Сучасні мови програмування підтримують різні парадигми (наприклад, об'єктно-орієнтовану, функціональну, імперативну), що дозволяє ефективно будувати як прості скрипти, так і масштабні корпоративні системи.

У межах даного проєкту були використані дві основні мови програмування - Python для серверної частини та JavaScript для реалізації інтерактивної клієнтської логіки.

2.2.1 Python

Python - це високорівнева, інтерпретована мова програмування, яка набула популярності завдяки своїй простоті, читабельності та універсальності. Вона підтримує кілька стилів програмування, включаючи об'єктно-орієнтований, процедурний та функціональний. Python має велику кількість бібліотек, що робить її ефективним інструментом для розробки вебзастосунків, наукових досліджень, машинного навчання, автоматизації процесів тощо[9].

У цьому проєкті Python був обраний для створення серверної частини (бекенду) за допомогою фреймворку Django. Саме ця мова дозволила ефективно реалізувати обробку даних, логіку взаємодії з базою даних, а також побудову REST API для обміну інформацією з клієнтською частиною.

Python також забезпечив зручну інтеграцію з бібліотеками OpenAI API, що дозволило реалізувати функціонал чат-бота й аналізу речень на основі штучного інтелекту. Завдяки своїй зрозумілій синтаксису та гнучкості, Python значно прискорив процес розробки та тестування функціональності застосунку.

2.2.2 JavaScript

JavaScript - це динамічна мова програмування, яка є стандартом для створення інтерактивних вебінтерфейсів. Вона виконується безпосередньо у браузері користувача, що дозволяє оперативно реагувати на події, керувати DOM-структурою сторінки, взаємодіяти з сервером через API-запити, а також створювати анімації, спливаючі вікна, інтерактивні форми тощо[9-10].

У даному проєкті JavaScript був використаний як основа для створення клієнтської частини за допомогою бібліотеки React. Ця мова забезпечила динамічне оновлення контенту без перезавантаження сторінки, що підвищило зручність взаємодії користувача із системою[12].

Завдяки широкій підтримці браузерами, численним бібліотекам і спільноті, JavaScript став ключовим інструментом у створенні сучасного, адаптивного та функціонального вебінтерфейсу освітньої платформи.

2.3 Веб-фреймворки

Веб-фреймворки - це програмні середовища, які спрощують процес створення вебзастосунків, надаючи готові компоненти, інструменти й архітектурні шаблони. Вони допомагають розробникам швидко реалізувати функціональність для взаємодії з базами даних, обробки запитів, маршрутизації, автентифікації користувачів тощо.

У межах даного проєкту були використані два основні фреймворки: Django та React, а також спеціалізоване розширення для побудови API - Django REST Framework. Такий розподіл дозволив реалізувати чітку архітектуру «клієнт-сервер», що базується на REST-принципах, і забезпечити гнучкий, масштабований і сучасний вебзастосунок.

2.3.1 Django

Django - це високорівневий веб-фреймворк для мови Python, який дозволяє швидко створювати повнофункціональні вебдодатки. Його головна філософія - «DRY» (Don’t Repeat Yourself), яка сприяє повторному використанню коду та зменшенню помилок. Django надає вбудовані засоби для роботи з базами даних, маршрутизації URL, авторизації користувачів, валідації форм та адміністрування вмісту[6].

У цьому проєкті Django використано для реалізації бекенду застосунку - логіки обробки запитів, збереження та обробки даних, автентифікації, а також для створення адміністративної панелі для управління курсами, тестами, користувачами та дегустаційними сесіями.

Django також забезпечує високий рівень безпеки за замовчуванням, захищаючи застосунок від таких типових вразливостей, як SQL-ін’єкції, XSS, CSRF тощо[6].

2.3.2 Django REST Framework

Django REST Framework (DRF) - це потужна бібліотека для Django, яка забезпечує зручні засоби для створення RESTful API. Вона надає класи для серіалізації даних, обробки HTTP-запитів, автентифікації, переглядів і контролерів[7].

DRF був використаний для побудови API, через яке фронтенд-частина React взаємодіє з сервером. Це дало змогу реалізувати архітектуру типу SPA (Single Page Application), де клієнт отримує лише необхідні дані в JSON-форматі, що значно покращує продуктивність та зручність користування.

Ключовими перевагами DRF є автоматичне формування документації, підтримка пагінації, фільтрації, дозволів доступу та гнучка система обробки запитів.

2.3.3 React

React - це бібліотека JavaScript з відкритим кодом, розроблена компанією Meta, призначена для створення користувацьких інтерфейсів. React базується на компонентному підході, де інтерфейс розбивається на незалежні частини - компоненти, які повторно використовуються та легко тестуються[12].

У даному проєкті React використано для створення клієнтської частини застосунку - інтерфейсів користувача для взаємодії з курсами, тестами, аналітикою та чат-ботом. React дозволяє оновлювати лише змінені елементи інтерфейсу, завдяки використанню віртуального DOM, що забезпечує високу продуктивність навіть при складній логіці.

Також React зручно інтегрується з REST API, що дозволило побудувати масштабований і динамічний вебінтерфейс, який швидко реагує на дії користувача без перезавантаження сторінки[12].

2.4 Система управління базами даних

Система управління базами даних (СУБД) - це програмне забезпечення, яке забезпечує створення, організацію, збереження, зміну та отримання даних у структурованому вигляді. СУБД є невід’ємною частиною будь-якого сучасного застосунку, оскільки дозволяє ефективно працювати з великими обсягами інформації, зберігати зв’язки між об’єктами та забезпечувати цілісність і безпеку даних.

У рамках розробки даного застосунку було обрано об’єктно-реляційну СУБД PostgreSQL, яка є однією з найпотужніших і найнадійніших систем з відкритим кодом.

2.4.1 PostgreSQL

PostgreSQL - це сучасна об’єктно-реляційна система управління базами даних, яка забезпечує високу продуктивність, масштабованість і підтримку

складних типів даних. Вона повністю відповідає принципам ACID (атомарність, узгодженість, ізольованість, довговічність), що гарантує надійність збереження даних навіть у разі системних збоїв[13].

Основні особливості PostgreSQL:

1. Підтримка складних запитів: дозволяє реалізовувати гнучкі фільтри, сортування, агрегацію та зв’язки між таблицями.
2. Наслідування, користувацькі типи даних і функції: дозволяють створювати розширені моделі даних.
3. Розширена підтримка JSON: дає змогу зберігати напівструктуровані дані у форматі JSON та працювати з ними за допомогою SQL-запитів.
4. Інтеграція з фреймворками: PostgreSQL повністю сумісна з Django ORM, що дає змогу зручно працювати з базою на рівні Python-класів без написання сирих SQL-запитів.

У даному проєкті PostgreSQL використовується для збереження інформації про курси, уроки, користувачів, тестові сесії, дегустаційні оцінки та інші сутності. Надійність, продуктивність і розширюваність цієї СУБД зробили її оптимальним вибором для реалізації серверної частини освітнього вебзастосунку.

2.5 Система контролю версій

Система контролю версій - це невід’ємний інструмент у процесі командної або індивідуальної розробки програмного забезпечення. Вона дозволяє зберігати історію змін у проєкті, відслідковувати модифікації в коді, повертатися до попередніх версій, а також ефективно організовувати співпрацю між кількома розробниками. У межах розробки даного застосунку було використано систему контролю версій Git та платформу для зберігання репозиторіїв GitHub.

2.5.1 Git

Git - це розподілена система контролю версій, яка дозволяє створювати локальні репозиторії, працювати з ними незалежно від мережі, а згодом синхронізувати зміни з віддаленими сховищами. Кожна зміна зберігається як окремий коміт із власним унікальним ідентифікатором, що дозволяє точно відстежувати внесені модифікації, хто їх вніс і коли.

У межах проєкту Git використовувався для:

1. фіксації етапів розробки;
2. створення гілок для реалізації окремих модулів (курси, тести, аналітика тощо);
3. злиття гілок із попередньою перевіркою змін (pull request);
4. відстеження історії змін та відкату до попередніх версій у разі потреби.

Git значно підвищив ефективність розробки та надійність роботи з кодовою базою.

2.5.2 GitHub

GitHub - це хмарна платформа для зберігання Git-репозиторіїв, яка надає зручний вебінтерфейс для перегляду коду, комітів, гілок, а також підтримує механізми спільної роботи над проєктами. Завдяки GitHub було забезпечено централізоване сховище проєкту з доступом до історії розробки, можливістю створення pull request, перегляду змін, коментування коду та автоматичного розгортання (CI/CD, за потреби).

У даному проєкті GitHub використовувався як головна платформа для зберігання та обміну кодом, створення резервних копій, контролю версій, а також ведення документації у вигляді README-файлів. Це забезпечило структуровану та прозору організацію всієї роботи над застосунком.

2.6 Інструменти дизайну інтерфейсу

Інструменти дизайну інтерфейсу користувача (UI) відіграють важливу роль у процесі розробки вебзастосунків, оскільки саме вони забезпечують створення зручного, зрозумілого та візуально привабливого середовища для користувача. Якісний дизайн підвищує ефективність взаємодії з системою, покращує користувацький досвід і сприяє легшому засвоєнню інформації.

У межах розробки освітнього застосунку для оцінювачів кави було використано сучасний вебсервіс Figma - один з найпопулярніших інструментів для створення UI/UX-дизайну.

2.6.1 Figma

Figma - це вебзастосунок для створення макетів інтерфейсів, прототипів та дизайн-систем, який працює прямо в браузері. На відміну від традиційних програм, Figma не потребує встановлення на комп’ютер і дозволяє кільком користувачам одночасно працювати над дизайном в реальному часі.

У цьому проєкті Figma використовувалася для розробки основних макетів інтерфейсу користувача: головної сторінки, панелі користувача, перегляду курсу, симуляції дегустації, тестування тощо. Завдяки зручному інтерфейсу та можливості створювати інтерактивні прототипи, розробники мали змогу візуально спланувати логіку переходів між сторінками й оцінити загальну структуру майбутнього застосунку ще до етапу програмної реалізації.

Figma також підтримує створення дизайн-систем, що дозволяє повторно використовувати стилі, компоненти та кольорову палітру в усіх частинах інтерфейсу, забезпечуючи візуальну цілісність та послідовність.

Використання Figma дозволило суттєво скоротити час на узгодження дизайну та забезпечити ефективну комунікацію між розробниками, дизайнерами та іншими учасниками проєкту.

3 ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Проєктування архітектури застосунку

Під час розробки програмного забезпечення особливу увагу було приділено проєктуванню архітектури системи, яка повинна бути не лише ефективною та надійною, а й гнучкою для подальшого масштабування та розширення функціоналу. Було обрано класичну трирівневу архітектуру з чітким поділом на клієнтський рівень, серверну частину та рівень зберігання даних. Такий підхід дозволяє досягти високої модульності та забезпечити простоту супроводу і модернізації системи в майбутньому.

У межах клієнтського рівня працює інтерфейс користувача, створений за допомогою JavaScript-бібліотеки React. Цей інтерфейс завантажується безпосередньо у веб-браузері користувача і є точкою входу до всієї системи. React-застосунок відповідає за відображення сторінок, навігацію, обробку дій користувача та відправку запитів до серверу через HTTP. Таким чином, уся взаємодія користувача з платформою, починаючи з реєстрації, проходження курсів, тестування та роботи з аналітикою, відбувається через зручний та динамічний веб-інтерфейс.

Серверна частина реалізована з використанням фреймворку Django у зв’язці з Django REST Framework[7]. Саме ця частина системи обробляє всі запити, що надходять з клієнта: перевіряє автентичність користувачів, обробляє запити до бази даних, реалізує бізнес-логіку та відповідає REST API[8]. Сервер виступає посередником між React-інтерфейсом і базою даних, забезпечуючи безпечну та узгоджену взаємодію між ними. Через API сервер надає дані про курси, уроки, тести, результати користувача, а також приймає введені відповіді, коментарі або інші дії користувача.

Зберігання всієї інформації, що використовується у платформі CoffeeLearn, здійснюється за допомогою системи управління базами даних PostgreSQL. Це сучасна реляційна СУБД, яка забезпечує високу швидкість обробки запитів, надійність зберігання, підтримку транзакцій та широкі можливості для розширення. Через ORM (Object-Relational Mapping), що надається Django, усі операції із записом, оновленням та отриманням інформації з бази даних здійснюються з використанням об'єктно-орієнтованого підходу, що спрощує розробку і підвищує безпеку.

Для наочного відображення взаємодії між основними компонентами системи було побудовано діаграму розгортання, що відображає логіку зв'язку між клієнтом, сервером і базою даних.

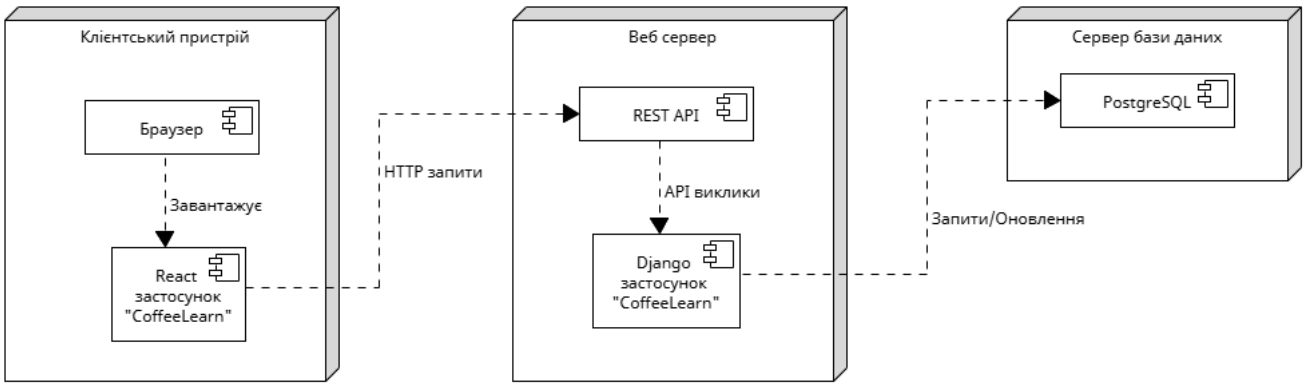


Рис. 3.1 Діаграма розгортання клієнт-серверної моделі

3.1.1 Діаграма діяльності користувача

У рамках проєктування архітектури системи важливим етапом є моделювання логіки взаємодії користувача із застосунком. Для цього було побудовано діаграму діяльності, яка відображає послідовність дій користувача під час проходження навчального курсу на платформі CoffeeLearn.

Ця діаграма описує основний сценарій використання системи:

1. На початку відбувається перевірка, чи записаний користувач на курс. Якщо ні — йому пропонується зареєструватися на обраний курс.

2. Після запису користувач переходить до ознайомлення з навчальним матеріалом (вивчення уроків).

3. По завершенню кожного уроку користувач може перейти до наступного або, у випадку завершення всієї програми курсу, перейти до проходження підсумкового тестування.

4. Після проходження тесту користувач отримує результат, який фіксується в системі.

Ця діаграма дозволяє чітко уявити процес навчання та логіку переходів між етапами. Вона є важливою частиною загального архітектурного проєктування, оскільки демонструє поведінку системи з точки зору кінцевого користувача.

Рис. 3.2 Діаграма діяльності користувача під час проходження курсу

3.1.2 Діаграма варіантів використання

Для візуалізації функціональних можливостей інформаційної системи та взаємодії користувачів із нею була створена діаграма варіантів використання (Use Case Diagram). Цей тип діаграми дозволяє наочно представити основні сценарії використання системи, ролі користувачів та відповідні функціональні можливості.

Діаграма охоплює три основні ролі (акторів):

1. Користувач — кінцевий споживач навчального контенту, який реєструється на платформі, проходить навчальні курси, взаємодіє з чат-ботом, бере участь у дегустаційних сесіях та отримує аналітичну інформацію щодо результатів.
2. Адміністратор — відповідальний за управління платформою, який має змогу створювати, редагувати та видаляти курси, уроки, тести, керувати користувачами, а також контролювати проведення дегустаційних сесій і аналізувати результати.
3. OpenAI API — зовнішній сервіс, інтегрований у платформу для забезпечення інтелектуальної підтримки користувачів через чат-бота. Цей актор автоматично надає відповіді на запити, пов’язані з тематикою кави або функціоналом системи, але не відповідає на тестові завдання.

Користувач має змогу:

1. Зареєструватися у системі та записатися на курс.
2. Ознайомитися з навчальними матеріалами у вигляді уроків.
3. Пройти курс та виконати тестові завдання, які перевіряються автоматично.
4. Взаємодіяти з чат-ботом, який надає допомогу щодо кавової тематики або навігації платформою.
5. Участь у дегустаційних сесіях, у межах яких він оцінює зразки кави.
6. Отримати аналітичну інформацію щодо власних результатів.

Адміністратор виконує такі дії:

1. Керує курсами, що включають створення та редагування уроків і тестів.

2. Здійснює управління користувачами.

3. Контролює проведення дегустаційних сесій.

4. Має доступ до аналітики, що стосується успішності користувачів, як у навчанні, так і в практичній частині (дегустаціях).

Діаграма варіантів використання є важливим етапом проєктування, оскільки вона дозволяє структурувати функціональні вимоги та забезпечити відповідність системи очікуванням користувачів і адміністратора.

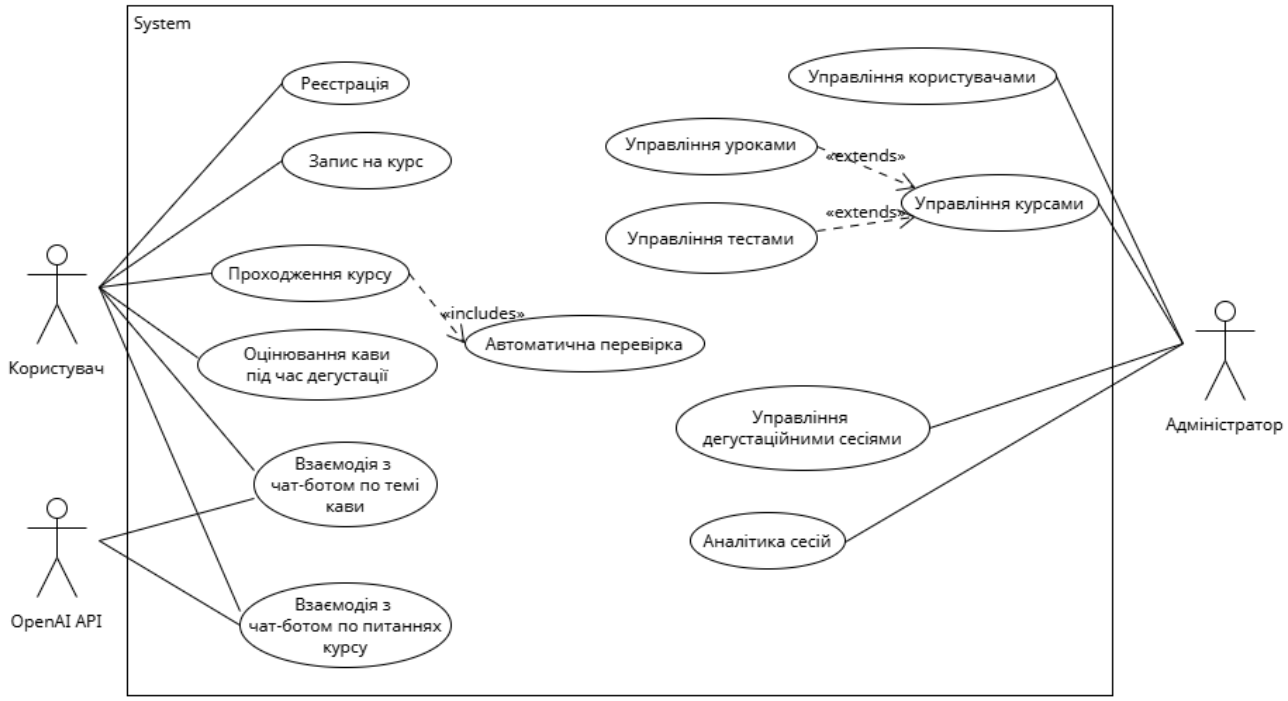


Рис. 3.3 Діаграма варіантів використання системи

3.2 Архітектура клієнтської частини

Клієнтська частина застосунку CoffeeLearn побудована на основі бібліотеки React, що дозволяє створювати динамічний односторінковий інтерфейс. Архітектура застосунку реалізована за компонентним підходом: кожен елемент інтерфейсу винесено в окремий ізольований компонент, що спрощує супровід, тестування та повторне використання коду.

Центральним елементом клієнтської частини є компонент App.tsx, який підключає глобальні компоненти, налаштовує маршрутизацію та ініціалізує застосунок. До глобальних належать такі елементи як Header.tsx - шапка інтерфейсу, та ChatWidget.tsx - віджет для взаємодії з помічником. Ці компоненти є спільними для всіх сторінок і відображаються незалежно від контенту.

Сторінки застосунку (Home.tsx, Login.tsx, Dashboard.tsx тощо) відповідають за відображення основних розділів платформи та доступ до навчального контенту, профілю користувача, а також аналітики. Вони організовані як окремі маршрути за допомогою React Router, що дозволяє змінювати контент без перезавантаження сторінки.

Функціональні компоненти реалізують ключові можливості платформи: CourseList.tsx і CourseDetail.tsx забезпечують доступ до навчальних матеріалів, TestDetail.tsx відповідає за проходження тестів, а CuppingForm.tsx разом з іншими компонентами дегустації реалізують збір і обробку оцінок. Ці компоненти, як правило, підключаються до сторінок або один до одного, залежно від сценарію.

Для організації бізнес-логіки та роботи з даними використовуються службові модулі: api.ts містить запити до бекенду, theme.ts - глобальні стилі, а global.d.ts - типи для підтримки TypeScript. Це забезпечує гнучкість, масштабованість і уніфіковану взаємодію з серверною частиною.

На діаграмі нижче представлено логічну структуру клієнтської частини: компоненти згруповано за призначенням, а стрілками показано включення та залежності між ними.

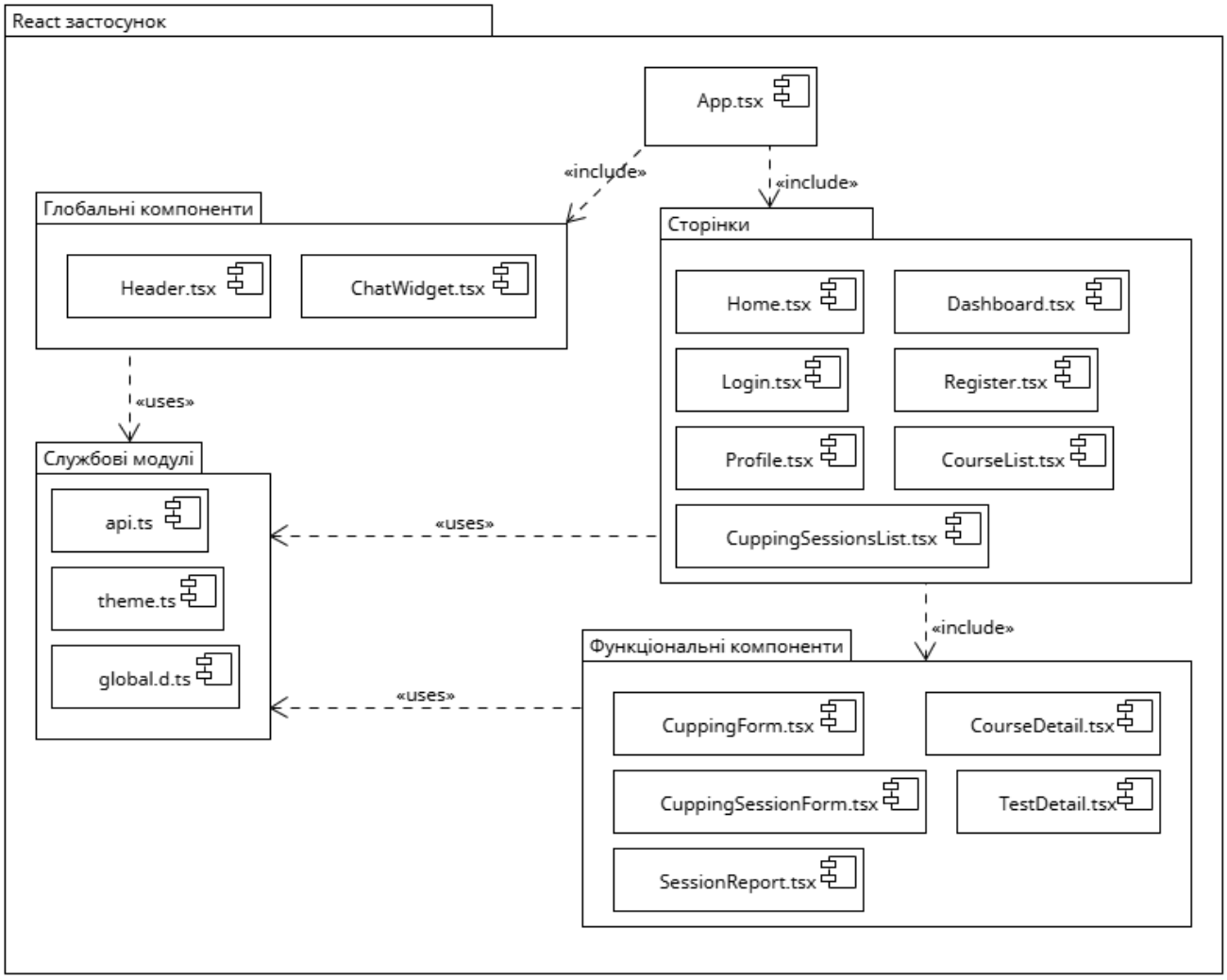


Рис. 3.4 Діаграма компонентів React застосунку

Архітектура забезпечує модульність, зручне розширення і високу підтримуваність, що є важливим для подальшого розвитку платформи.

3.3 Архітектура серверної частини

Серверна частина застосунку CoffeeLearn побудована на основі вебфреймворку Django, який вирізняється чіткою модульною структурою та підтримкою принципів MVC-подібної архітектури (Model–View–Template). Django дозволяє ефективно розділяти відповідальність між різними частинами застосунку, що суттєво полегшує розробку, тестування та масштабування системи.

На верхньому рівні знаходиться ядро проєкту - базова конфігурація та налаштування, які охоплюють увесь застосунок. Сюди входять файли settings.py, urls.py, wsgi.py, які відповідають за налаштування середовища, маршрутизацію запитів і запуск застосунку на сервері. Цей рівень є спільним для всіх внутрішніх модулів і визначає загальні правила роботи всієї системи.

Нижче розташовані окремі функціональні модулі (додатки), кожен з яких виконує певну логічну роль у межах платформи. У випадку CoffeeLearn це модулі theory, practice, cupping, кожен з яких містить власні представлення, моделі, тести, серіалізатори та URL-конфігурації.

Кожен модуль побудований за принципом тришарової архітектури:

1. Presentation Layer (Шар представлення): відповідає за обробку HTTP-запитів користувача та формування відповідей. Файл views.py містить представлення, які реалізують поведінку застосунку у відповідь на запити до REST API. В цьому шарі визначається, які дії виконуються при взаємодії з клієнтською частиною.

2. Business Logic Layer (Шар бізнес-логіки): реалізує логіку роботи модуля, включаючи структуру бази даних (models.py), тести для перевірки правильності роботи (tests.py) та локальну маршрутизацію (urls.py). Саме тут визначаються сутності, їх зв’язки та поведінка в межах платформи.

3. Data Access Layer (Шар доступу до даних): відповідає за перетворення моделей у формат, зручний для передавання через API. У файлі serializers.py за допомогою Django REST Framework реалізується логіка серіалізації - перетворення об’єктів моделі у JSON та назад.

Така структура дозволяє ізолювати логіку окремих частин застосунку, зменшити кількість залежностей між модулями та забезпечити чисту архітектуру з можливістю повторного використання компонентів.

Нижче наведено діаграму архітектури серверної частини, що візуалізує взаємозв’язок між основними модулями проєкту.

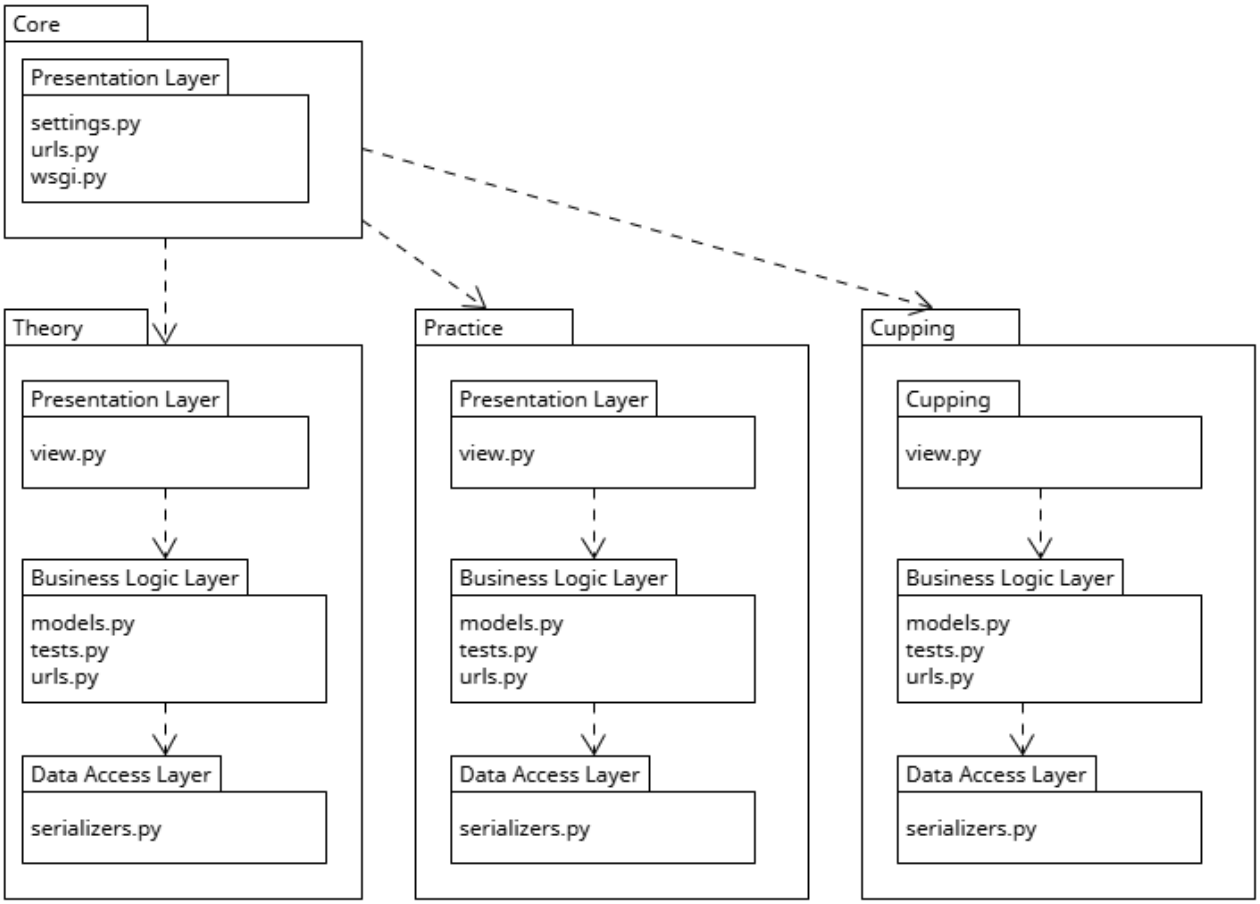


Рис. 3.5 Діаграма пакетів архітектури серверної частини застосунку

Цей підхід дозволяє легко орієнтуватися в проєкті, додавати нові модулі, а також швидко тестувати окремі компоненти без ризику вплинути на роботу всього застосунку.

На рисунку зображено діаграму класів серверної частини, що демонструє структуру моделей Django. Вона ілюструє взаємозв’язки між курсами, уроками, тестами, сесіями тестування, а також елементами дегустації кави (cupping)[17].

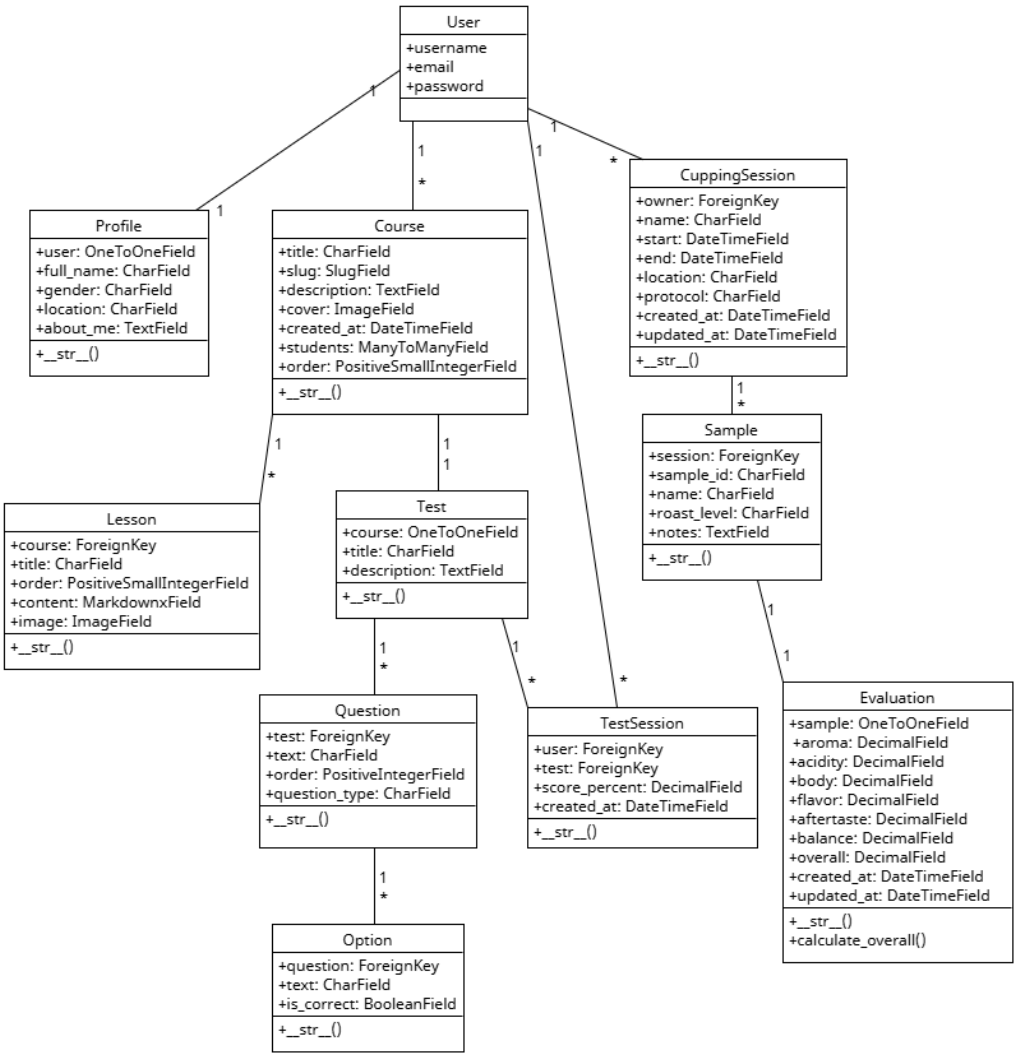


Рис. 3.6 Діаграма класів серверної частини застосунку

Клас Course пов’язаний з класами Lesson, Test, TestSession, що дозволяє реалізувати повноцінну навчальну систему. Паралельно, класи CuppingSession, Sample та Evaluation відповідають за дегустаційну логіку.

3.4 Архітектура бази даних

Архітектура бази даних проєкту побудована на реляційній моделі, яка реалізована за допомогою системи управління базами даних PostgreSQL. Реляційний підхід забезпечує надійне, структуроване та логічно організоване зберігання даних, що особливо важливо при обробці великої кількості пов’язаних

сутностей, таких як користувачі, курси, уроки, результати тестів і дегустаційні сесії.

База даних складається з набору таблиць, де кожна таблиця відображає певну сутність у системі. Наприклад, таблиця auth_user зберігає облікові дані користувачів, theory_course містить інформацію про навчальні курси, а cupping_evaluation - про оцінки якості зразків кави.

Кожна таблиця має:

- Колонки (поля) - визначають тип даних, що зберігаються (текст, число, дата тощо). У Django це поля моделей, які описуються у вигляді атрибутів класу.

– Рядки (записи) - представляють окремі екземпляри сутностей. Наприклад, один запис у таблиці theory_lesson - це окремий урок одного з курсів.

Зв’язки між таблицями реалізовано за допомогою зовнішніх ключів, що дозволяє встановлювати відношення між сутностями. Django використовує ORM (Object-Relational Mapping), що дає змогу описувати ці зв’язки через класи моделей, забезпечуючи прозору й безпечну роботу з базою.

У системі реалізовані типові реляційні зв’язки:

- Один до одного (One-to-One): наприклад, users_profile пов’язаний з auth_user, і кожен користувач має лише один профіль.

- Один до багатьох (One-to-Many): кожен курс (theory_course) може містити багато уроків (theory_lesson), але кожен урок належить лише одному курсу.

- Багато до багатьох (Many-to-Many): реалізовано, наприклад, через таблицю theory_course_students, яка зберігає факт запису користувача на курс.

Для наочного відображення структури бази даних побудовано ER-діаграму, яка демонструє таблиці, типи даних, первинні та зовнішні ключі, а також зв’язки між основними сутностями системи.

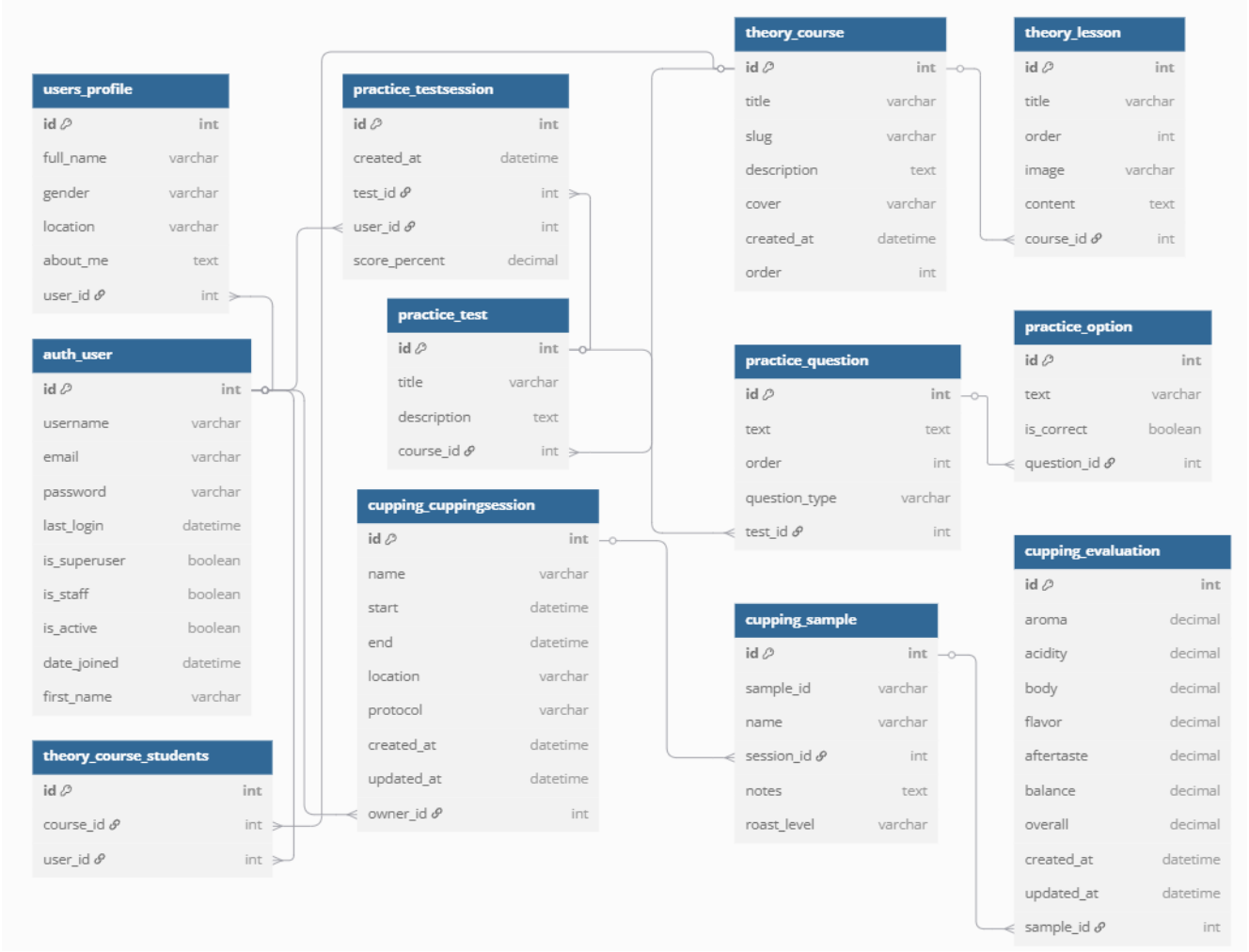


Рис. 3.7 ER-діаграма структури бази даних системи

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Реалізація серверної частини

Серверна частина системи CoffeeLearn реалізована з використанням фреймворку Django - високорівневого фреймворку для розробки вебзастосунків мовою Python. Основні причини вибору Django полягають у його надійності, швидкій розробці, зручній структурі та інтеграції з системами автентифікації, адміністрування та REST API через Django REST Framework (DRF)[15].

На першому етапі було створено базовий Django-проєкт командою:

- django-admin startproject coffeelearning

Після цього було створено окремі додатки відповідно до логічних блоків системи:

- python manage.py startapp users
- python manage.py startapp theory
- python manage.py startapp practice
- python manage.py startapp cupping

Кожен додаток відповідає за чітко визначений функціональний сегмент:

1. users - обробка реєстрації, входу в систему, профілів;
2. theory - управління навчальними курсами та уроками;
3. practice - тестування, запитання, варіанти, сесії;
4. cupping - проведення дегустацій, оцінювання кавових зразків.

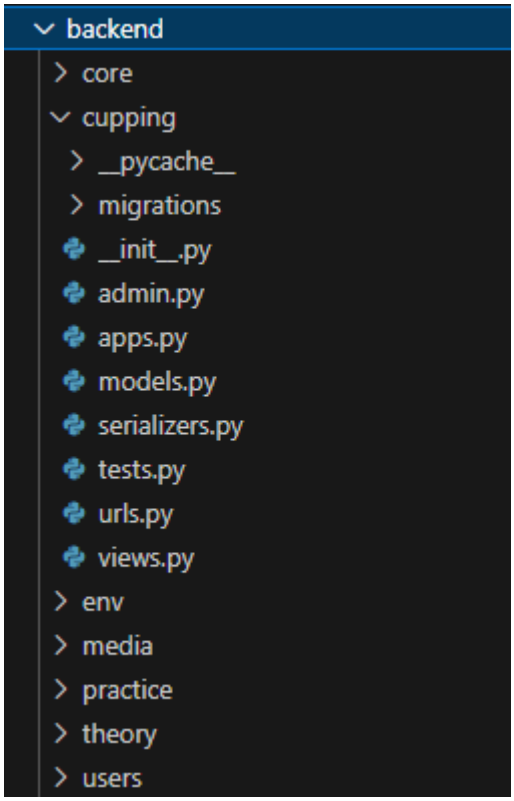


Рис. 4.1 Структура серверної частини проєкту

Після створення додатків було здійснено їх підключення до основного проєкту через змінну INSTALLED_APPS у файлі settings.py. Окрім стандартних Django-додатків, також підключено сторонні бібліотеки:

```
INSTALLED_APPS = [
…
    'rest_framework',
    'corsheaders',
    'markdownx',
    'users',
    'theory',
    'practice',
    'cupping',
]
```

1. rest_framework - для створення REST API;
2. corsheaders - для обробки CORS-запитів із клієнтської частини;

3. markdownx - для зберігання та рендерингу текстів уроків у Markdown-форматі.

Для кожного додатку було реалізовано відповідні моделі, що відображають структуру даних:

1. Course, Lesson - структура навчального контенту;
2. Test, Question, Option, TestSession - компоненти тестування;
3. CuppingSession, Sample, Evaluation - проведення дегустацій;
4. Profile - розширення стандартної моделі користувача.

У моделях були чітко визначені:

1. типи полів (CharField, TextField, DecimalField тощо);
2. обмеження (choices, unique, blank);
3. методи __str__() для зручного відображення в адмінці;
4. сортування за допомогою Meta.ordering

Після створення моделей були згенеровані та застосовані міграції:

- python manage.py makemigrations
- python manage.py migrate

Було реалізовано повноцінне API для взаємодії з клієнтом за допомогою Django REST Framework.

1. Кожна модель має окремий серіалізатор (serializers.py), що відповідає за перетворення об’єктів у JSON та навпаки.

2. У представленнях (views.py) використано GenericAPIView, APIView та ModelViewSet, щоб реалізувати всі необхідні дії: перегляд списку, перегляд детальної інформації, створення, редагування та видалення.

Приклад серіалізатора курсу:

```
class CourseListSerializer(serializers.ModelSerializer):
    class Meta:
        model = Course
        fields = ['id', 'title', 'slug', 'cover', 'description']
```

Для більш складної логіки, наприклад перевірки правильності відповідей у тестах або створення кількох зразків у дегустації, було реалізовано спеціальні методи validate(), create(), get_result() тощо.

Для автентифікації API було інтегровано SimpleJWT, що дозволяє видавати JWT-токени при вході та працювати з авторизованими запитами:

- pip install djangorestframework-simplejwt

Конфігурація токенів додана до settings.py, а ендпоінти доступні через:

1. /api/token/ - отримання токена;
2. /api/token/refresh/ - оновлення токена.

Django має вбудовану потужну адміністративну панель, яку було використано для внутрішнього керування платформою:

1. зареєстровано всі сутності у admin.py;
2. налаштовано поля відображення (list_display, search_fields, readonly_fields);
3. реалізовано вкладені форми для Question > Option, CuppingSession > Sample.

```
class OptionInline(admin.TabularInline):
    model = Option
    extra = 1


@admin.register(Question)
class QuestionAdmin(admin.ModelAdmin):
    inlines = [OptionInline]
```

Це дозволило адміністраторам легко переглядати та редагувати вміст усіх курсів, уроків, тестів, зразків, оцінок тощо без створення додаткових інтерфейсів.

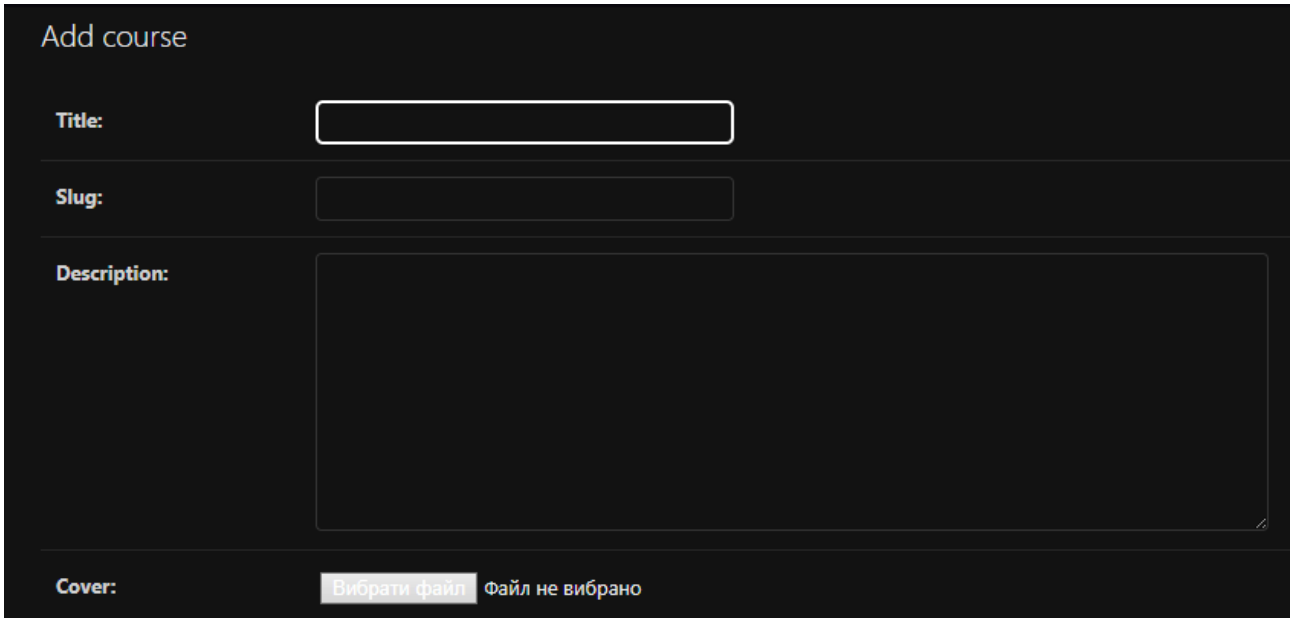


Рис. 4.2 Створення нового курсу за допомогою адміністративної панелі Django

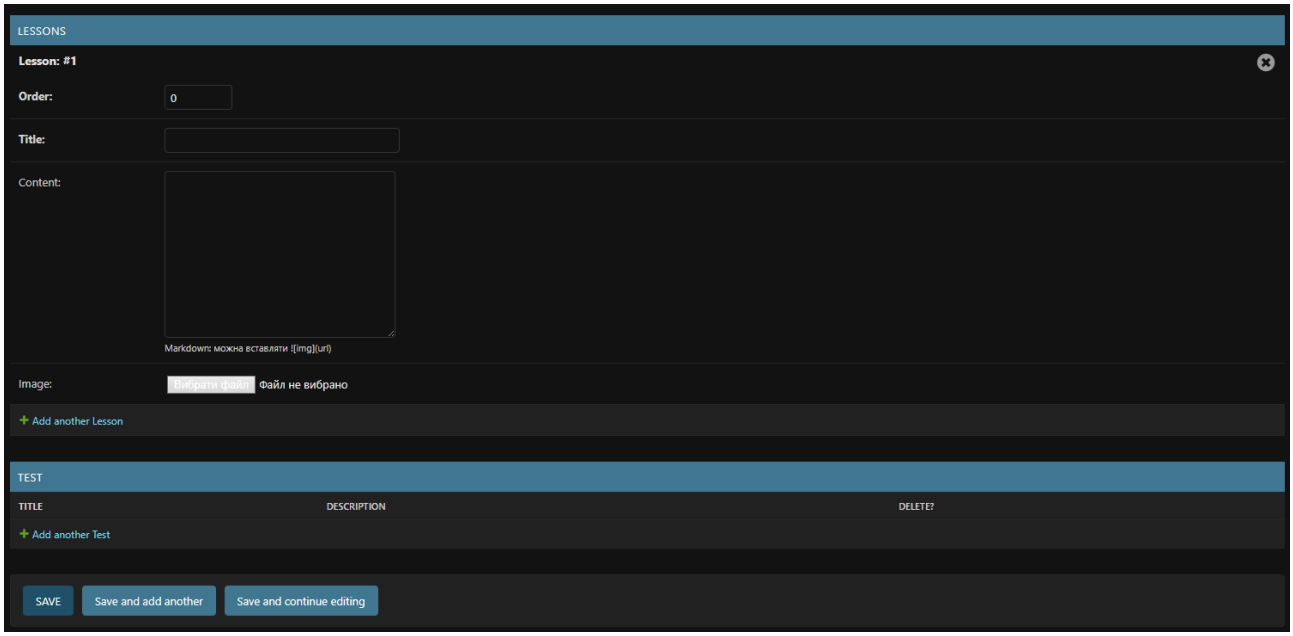


Рис. 4.3 Редагування уроків та додавання тестів через адміністративну панель Django

4.2 Реалізація клієнтської частини

Клієнтська частина системи CoffeeLearn була реалізована з використанням сучасної JavaScript-бібліотеки React, що дозволяє створювати швидкі, динамічні та

зручні у використанні інтерфейси користувача. Застосунок функціонує як SPA (Single Page Application), тобто завантажується один раз, після чого динамічно оновлює контент без повного перезавантаження сторінки, що позитивно впливає на швидкість та зручність взаємодії.

Проєкт було створено за допомогою утиліти Create React App:

- npx create-react-app coffeelearn-frontend --template typescript

Це дозволило одразу отримати середовище з підтримкою TypeScript, що забезпечує статичну типізацію та зменшує кількість помилок при розробці.

Клієнтський застосунок має чітку структуру директорій:

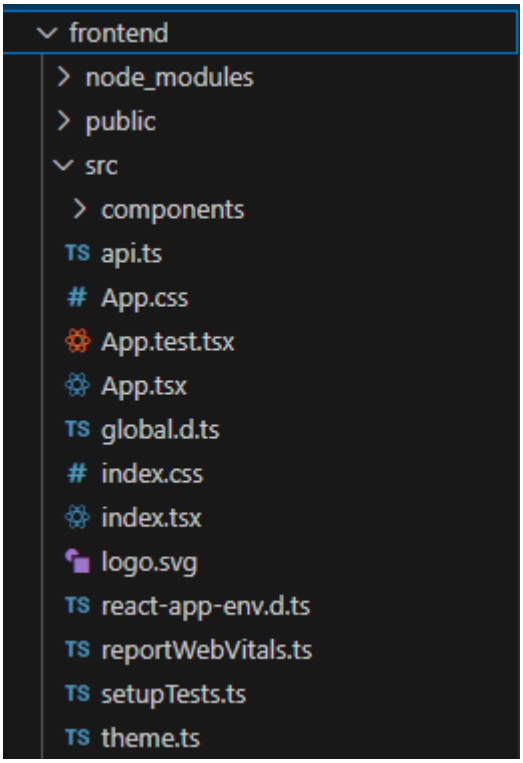


Рис. 4.4 Структура клієнтської частини проєкту

Кожна сторінка - це окремий компонент, який завантажує та візуалізує потрібні дані, а компоненти оформлення забезпечують єдиний стиль інтерфейсу.

Для взаємодії з серверною частиною було створено окремий файл api.ts, де зібрані всі функції, що здійснюють HTTP-запити до Django REST API за

допомогою fetch або axios. Для авторизованих запитів передаються JWT-токени у заголовках:

```
const token = localStorage.getItem("access");
const headers = {
  "Content-Type": "application/json",
  Authorization: `Bearer ${token}`,
};
```

Зокрема, реалізовані функції:

1. отримання списку курсів;
2. отримання деталей курсу з уроками;
3. реєстрація та вхід користувача;
4. проходження тестів;
5. створення дегустаційної сесії.

Для організації навігації в застосунку використовується React Router. У файлі App.tsx визначено всі основні маршрути:

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/dashboard" element={<Dashboard />} />
  <Route path="/courses/:slug" element={<CourseDetail />} />
  <Route path="/tests/:slug" element={<TestDetail />} />
  <Route path="/cupping" element={<CuppingList />} />
</Routes>
```

Це дозволяє швидко перемикатись між сторінками без перезавантаження браузера.

Головним орієнтиром у дизайні було створення інтуїтивного, простого та чистого інтерфейсу, орієнтованого на користувача. Кожна сторінка має заголовок, контентну частину, а також елементи управління (кнопки, форми, слайдери тощо).

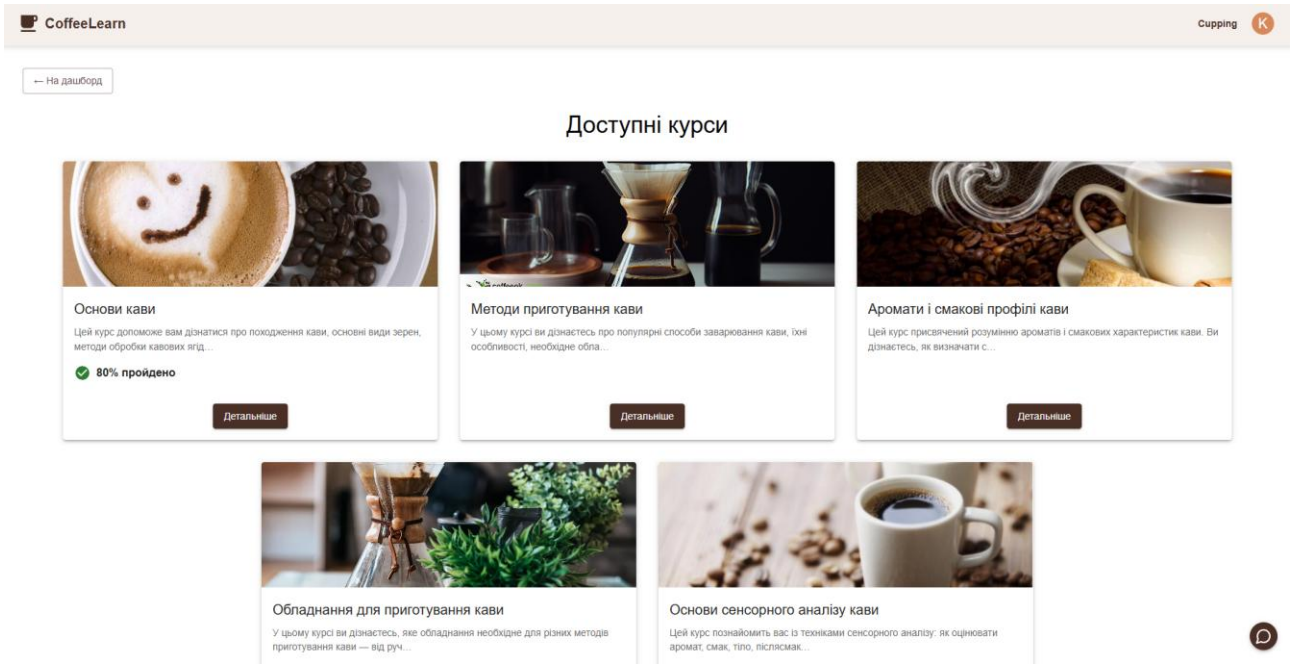


Рис. 4.5 Сторінка зі всіма доступними курсами

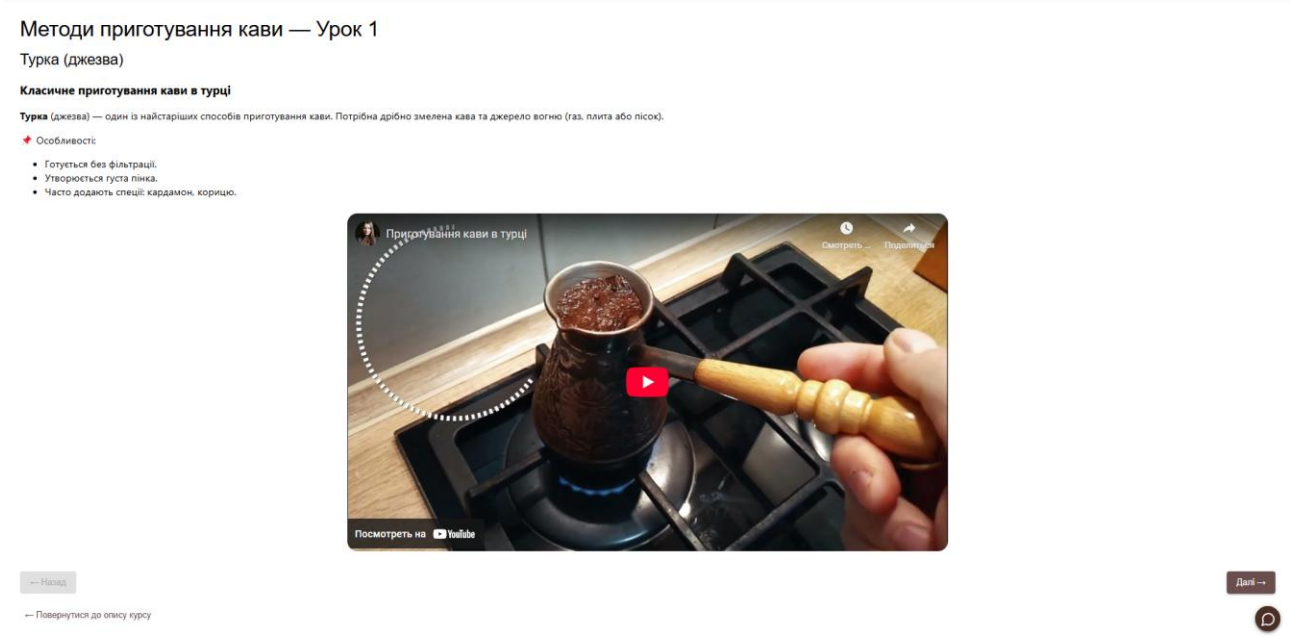


Рис. 4.6 Сторінка уроку в курсі

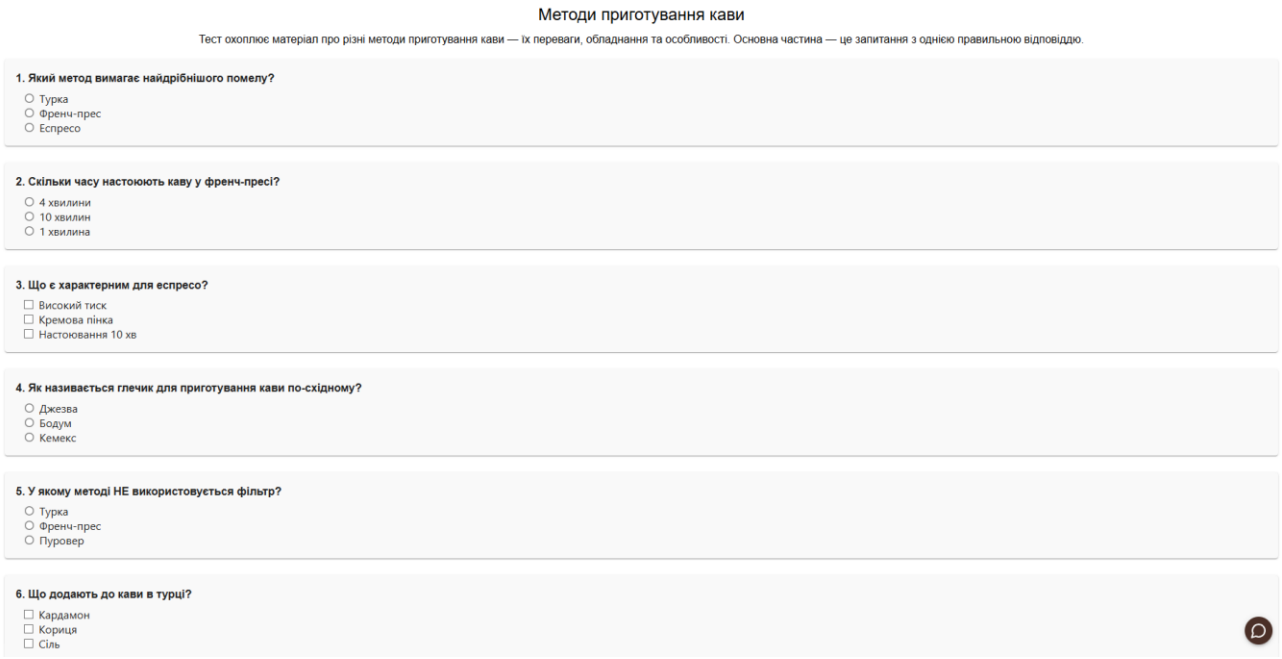


Рис. 4.7 Інтерфейс проходження тесту з питаннями та варіантами відповідей

Після успішної реєстрації або входу користувача токен зберігається у localStorage та використовується для авторизованих запитів. Також реалізовано перевірку автентифікації при завантаженні застосунку:

```
useEffect(() => {
  const token = localStorage.getItem("access");
  if (token) {
    setIsAuth(true);
  }
}, []);
```

У разі неавторизованого доступу - користувача перенаправляє на сторінку входу.

Інтерфейс модуля дегустацій дозволяє створювати сесію, вносити оцінки за параметрами (арома, кислотність, тіло тощо), автоматично підраховувати середній бал, та відображати результати.

Рис. 4.8 Форма дегустації з введенням оцінок

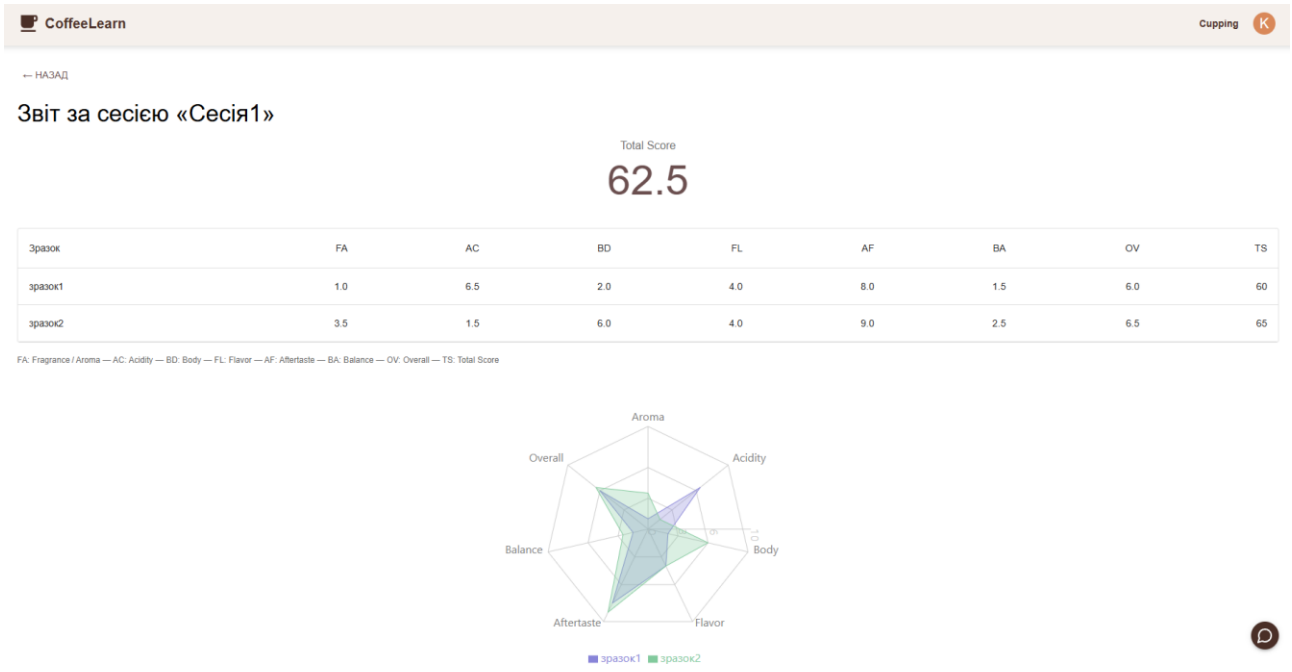


Рис. 4.9 Згенерований звіт за дегустаційною сесією

4.3 Інтеграція та взаємодія компонентів

Після реалізації окремих функціональних частин застосунку CoffeeLearn було здійснено їх повноцінну інтеграцію в єдину систему. Компоненти клієнтської

частини (React), серверної частини (Django) та бази даних (PostgreSQL) взаємодіють між собою через REST API, забезпечуючи безперервну й стабільну роботу системи.

Уся взаємодія між фронтендом і бекендом побудована на HTTP-запитах до REST API. React-застосунок надсилає запити до визначених ендпоінтів, отримує у відповідь JSON-дані та відображає їх у вигляді інтерактивних інтерфейсів. Наприклад, при завантаженні сторінки курсу надсилається запит на /api/courses/<slug>/, у відповіді надходить повна інформація про курс і список його уроків.

Реєстрація, вхід, проходження тестів, створення дегустацій, оновлення профілю - усе це реалізовано через відповідні API-запити. На стороні сервера ці запити обробляються класами APIView або ModelViewSet, які взаємодіють з ORM та повертають відповідь у форматі JSON.

Після успішної авторизації користувач отримує JWT-токен, який зберігається у localStorage браузера та додається до заголовків кожного запиту. Це дозволяє реалізувати захищений доступ до особистих даних, історії навчання, результатів тестування та дегустацій. Серверна частина перевіряє автентичність кожного запиту через permissions.IsAuthenticated, а також виконує додаткову перевірку - чи має користувач доступ до відповідного ресурсу.

Клієнтська частина працює з усіма основними модулями платформи:

1. модуль курсів (Course, Lesson) - дозволяє переглядати й вивчати контент;
2. модуль тестів (Test, Question, Option, TestSession) - забезпечує інтерактивне тестування та перевірку знань;
3. модуль дегустацій (CuppingSession, Sample, Evaluation) - дозволяє оцінювати зразки кави за визначеними критеріями.

Кожна дія користувача викликає відповідну логіку на сервері, яка, у свою чергу, змінює стан у базі даних, після чого результати передаються назад і відображаються на екрані. Наприклад, при створенні нової дегустації сервер

автоматично створює задану кількість зразків і повертає користувачу ідентифікатори для подальшої оцінки.

Кожен елемент інтерфейсу працює з актуальними даними. Наприклад, при проходженні тесту система перевіряє, чи користувач уже проходив цей тест. Якщо так - замість форми проходження відображається результат. Аналогічно, у курсах видно, чи користувач уже записаний, і який його прогрес.

Завдяки інтеграції всієї логіки через єдиний API застосунок легко розширюється. Додавання нового модуля або функціоналу (наприклад, рейтингу курсів, коментарів або аналітики) потребує лише додавання нового API-ендпоінта і його обробки у клієнті.

4.4 Тестування застосунку

Тестування є важливою частиною життєвого циклу програмного забезпечення, що дозволяє перевірити коректність реалізації функціональності, виявити помилки та переконатися у стабільності системи. У межах даного проєкту для платформи CoffeeLearn було реалізовано модульне та інтеграційне тестування основних компонентів серверної частини, зокрема модулів theory, practice та cupping.

Для реалізації тестів використовувались вбудовані можливості Django (TestCase) та Django REST Framework (APITestCase), що дозволяє перевіряти як роботу моделей, так і логіку обробки REST-запитів[14].

Тестування модуля theory охоплює такі сценарії:

1. створення курсу та уроку;
2. отримання списку доступних курсів;
3. отримання детальної інформації про курс;
4. реєстрація користувача на курс;
5. перевірка правильного формування дашборду з даними про прогрес користувача.

Приклад тесту для перевірки API отримання курсу:

```
def test_get_course_list(self):
    response = self.client.get("/courses/")
    self.assertEqual(response.status_code, 200)
    self.assertIn("title", response.data[0])
```

Модуль practice відповідає за проходження тестів у кінці кожного курсу. Було протестовано:

отримання тесту користувачем;

надсилання відповідей та обчислення результату;

створення сесії тестування (TestSession);

заборона повторного проходження тесту.

Фрагмент тесту перевірки правильного проходження:

```
def test_submit_correct_answer(self):
    response = self.client.post(f"/tests/{self.course.slug}/", {
        "answers": [{
            "question": self.question.id,
            "selected_options": [self.correct_option.id]
        }]
    }, format="json")
    self.assertEqual(response.status_code, 201)
    self.assertEqual(response.data["score_percent"], 100.0)
```

Модуль cupping реалізує створення дегустаційних сесій, додавання зразків і оцінювання. Було протестовано:

1. створення сесії з вказаною кількістю зразків;
2. автоматичне додавання зразків до сесії;
3. додавання оцінки (Evaluation) до зразка;
4. перевірка обмежень доступу для неавторизованих користувачів;
5. перегляд інформації про сесію.

Приклад тесту створення сесії:

```
def test_create_cupping_session_with_samples(self):
    data = {
        "name": "Сесія №1",
        "start": timezone.now().isoformat(),
        "location": "Київ",
        "protocol": "Arabica",
        "number_of_samples": 3
    }
    response = self.client.post("/cupping/sessions/", data, format="json")
    self.assertEqual(response.status_code, 201)
    self.assertEqual(Sample.objects.count(), 3)
```

Для запуску всіх тестів використовувалась команда:

- python manage.py test

У результаті виконання тестів було підтверджено коректність реалізації логіки, стабільність роботи REST API, дотримання прав доступу та правильність збереження даних у базі.

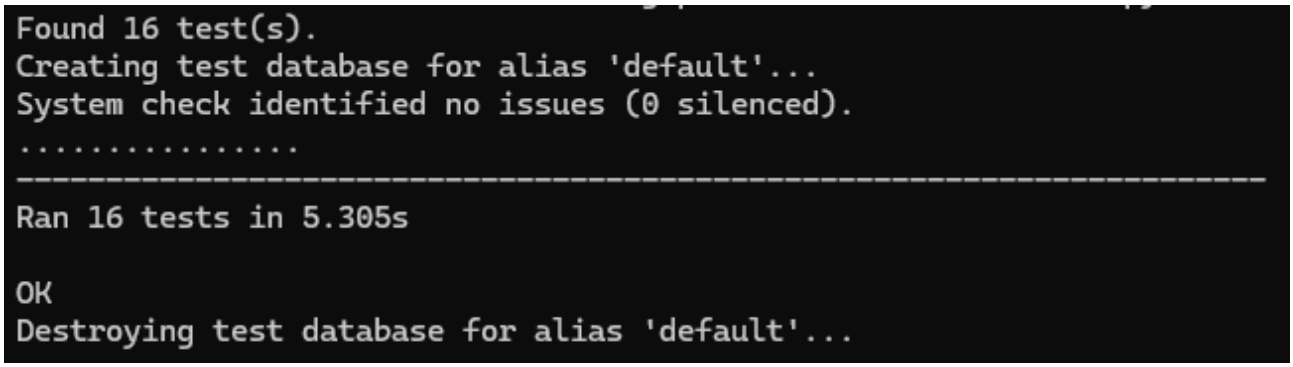


Рис. 4.10 Успішне проходження усіх тестів

ВИСНОВКИ

У процесі виконання дипломної роботи було створено повноцінний веб-застосунок, покликаний підтримувати навчання та оцінювання професійних дегустаторів кави. Під час реалізації проєкту було виконано аналіз предметної області, обрано оптимальні інструменти розробки, спроєктовано архітектуру системи, реалізовано функціональні компоненти та проведено тестування.

- Проведено аналіз предметної галузі навчання оцінювачів кави, що дозволило краще зрозуміти специфіку підготовки фахівців, принципи оцінювання якості кави, стандарти Q-Grader і вимоги до навчального процесу. На основі цього сформульовано функціональні вимоги до майбутнього застосунку.

- Проведено огляд технологій, фреймворків та мов програмування, які можуть бути використані для реалізації проєкту. Було обґрунтовано вибір Django та Django REST Framework для серверної частини, React - для клієнтської частини, а PostgreSQL - для зберігання даних.

- Розроблено архітектуру застосунку за принципами багаторівневої організації з чітким поділом на клієнтську, серверну та базову частини. Передбачено окремі модулі для теорії, практики та дегустацій, що відповідають ключовим аспектам навчання оцінювачів кави.

- Реалізовано основну функціональність: перегляд курсів і уроків, проходження тестування, ведення дегустаційних сесій, збереження результатів оцінювання зразків. Усі компоненти взаємодіють між собою через REST API.

- Проведено повноцінне модульне та API-тестування всіх основних модулів застосунку (theory, practice, cupping), що підтвердило коректну роботу реалізованої логіки. Усі тести виконано успішно, що свідчить про стабільність і готовність системи до використання.

- Загалом, усі поставлені задачі було виконано. Результатом роботи став сучасний веб-застосунок, який може бути ефективно використаний у процесі навчання та сертифікації професійних дегустаторів кави.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Козачок К.Ю., Гаманюк І.М. Розробка моделі предметної галузі застосунку для підтримки навчання професійних оцінювачів кави: Матеріали XII Всеукраїнської науково-практичної конференції молодих учених "ІТ-2025". 15 травня 2025 року, КУБГ, м. Київ. С.140-142.
2. Козачок К.Ю., Гаманюк І.М. Розробка функціональності застосунку для підтримки навчання професійних оцінювачів кави: Матеріали XII Всеукраїнської науково-практичної конференції молодих учених "ІТ-2025". 15 травня 2025 року, КУБГ, м. Київ. С.142-143.

ПЕРЕЛІК ПОСИЛАНЬ

1. Evolving the SCA Cupping Protocol and Form. *Specialty Coffee Association*. URL: https://sca.coffee/sca-news/read/evolving-the-sca-cupping-protocol-and-form-an-overview-of-the-pilot-testing-process (date of access: 01.05.2025).

2. Sensory analysis of the flavor profile. *Nature*. URL: https://www.nature.com/articles/s41598-024-69867-6 (date of access: 01.05.2025).

3. Wang X., Lim L.-T. Effects of grind size, temperature, and brewing ratio on immersion cold brewed and french press hot brewed coffees. *Applied food research*. 2023. P. 100334. URL: https://doi.org/10.1016/j.afres.2023.100334 (date of access: 01.05.2025).

4. Kim Y., An J., Lee J. The espresso protocol as a tool for sensory quality evaluation. *Food Research International*. 2025. P. 115670. URL: https://doi.org/10.1016/j.foodres.2025.115670 (date of access: 01.05.2025).

5. Industrial food quality and consumer choice: Artificial intelligence-based tools in the chemistry of sensory notes in comfort foods (coffee, cocoa and tea) / E. Bagnulo et al. *Trends in Food Science & Technology*. 2024. P. 104415. URL: https://doi.org/10.1016/j.tifs.2024.104415 (date of access: 01.05.2025).

6. Django documentation | Django documentation. *Django Project*. URL: https://docs.djangoproject.com/en/5.2/ (date of access: 11.05.2025).

7. Django web development framework: powering the modern web / S. Chen et al. *American journal of trade and policy*. 2020. Vol. 7, no. 3. P. 99–106. URL: https://doi.org/10.18034/ajtp.v7i3.675 (date of access: 13.05.2025).

8. OpenAPI Specification v3.1.0. *OpenAPI Initiative Publications*. URL: https://spec.openapis.org/oas/v3.1.0 (date of access: 05.05.2025).

9. JavaScript Guide - JavaScript | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (date of access: 08.05.2025).

10. Python 3.13 documentation. *Python documentation*. URL: https://docs.python.org/3/ (date of access: 09.05.2025).

11. Python for web development / U. Patkar et al. *International journal of computer science and mobile computing*. 2022. Vol. 11, no. 4. P. 36–48. URL: https://doi.org/10.47760/ijcsmc.2022.v11i04.006 (date of access: 13.05.2025).

12. React documentation. *React Documentation*. URL: https://react.dev/learn (date of access: 10.05.2025).

13. PostgreSQL: Documentation. *PostgreSQL*. URL: https://www.postgresql.org/docs/ (date of access: 10.05.2025).

14. Testing in Django | Django documentation. *Django Project*. URL: https://docs.djangoproject.com/en/5.2/topics/testing/ (date of access: 20.05.2025).

15. The Django admin site | Django documentation. *Django Project*. URL: https://docs.djangoproject.com/en/5.2/ref/contrib/admin/ (date of access: 18.05.2025).

16. Models | Django documentation. *Django Project*. URL: https://docs.djangoproject.com/en/5.2/topics/db/models/ (date of access: 18.05.2025).

17. GeeksforGeeks. Class Diagram | Unified Modeling Language. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/ (date of access: 14.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО -КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмних засобів для підтримки навчання професійних оцінювачів кави

Виконав студент 4 курсу
групи ПД-42
Козачок Костянтин Юрійович
Керівник роботи
Старший викладач кафедри ІПЗ Гаманюк Ігор Михайлович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: удосконалення процесу навчання та оцінювання професійних дегустаторів кави на основі розробленого ПЗ, що сприяє підвищенню ефективності підготовки фахівців у цій сфері.

Об'єкт дослідження: процес навчання та професійної підготовки оцінювачів кави.

Предмет дослідження: програмні засоби, що забезпечують підтримку навчального процесу та моделювання практичного оцінювання кави з використанням цифрових технологій .

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Ознайомитися з професійними стандартами та методиками навчання оцінювачів кави (Q-Grader, SCA тощо).
2. Провести аналіз існуючих програмних рішень для підтримки навчання у сфері дегустації кави.
3. Провести огляд та аналіз засобів реалізації застосунку для підтримки навчання професійних оцінювачів кави.
4. Проектування архітектури застосунку для підтримки навчання професійних оцінювачів кави.
5. Розробка застосунку для підтримки навчання професійних оцінювачів кави.
6. Виконати тестування розробленого функціоналу .

3

АНАЛІЗ АНАЛОГІВ

Критерії	Barista Hustle	Tastify	CoffeeMind Academy	CoffeeLearn
Теоретичний модуль	+	-	+	+
Практичні тести з оцінкою знань	+	-	-	+
Практичні дегустаційні сесії	-	+	-	+
Зберігання історії дегустацій користувача	-	+	-	+
Докладна аналітика результатів дегустацій	-	+	-	+
Адаптивність під різні пристрої	+	+	+	+
AI-асистент	-	-	-	+
Українська мова інтерфейсу	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Реєстрація та авторизація користувачів.
2. Ведення особистого кабінету з відображенням прогресу.
3. Надання теоретичних матеріалів з оцінки якості кави.
4. Проведення інтерактивного тестування для перевірки знань.
5. Симуляція процесу дегустації кави з можливістю введення оцінок.
6. Автоматична перевірка результатів навчання та тестів.
7. Надання зворотного зв’язку щодо результатів.
8. Можливість адміністрування контенту та управління користувачами.
9. Відображення статистики та аналітики результатів користувача.
10. Чат-бот асистент для допомоги у вивченні матеріалу про каву.

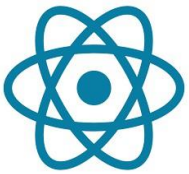
Нефункціональні вимоги:

1. Використання авторизації та шифрування даних.
2. Завантаження сторінки менш ніж за 2 секунди.
3. Адаптивність інтерфейсу до мобільних пристроїв, планшетів і десктопів.
4. Підтримка сучасних браузерів (Chrome, Firefox, Safari, Edge).

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

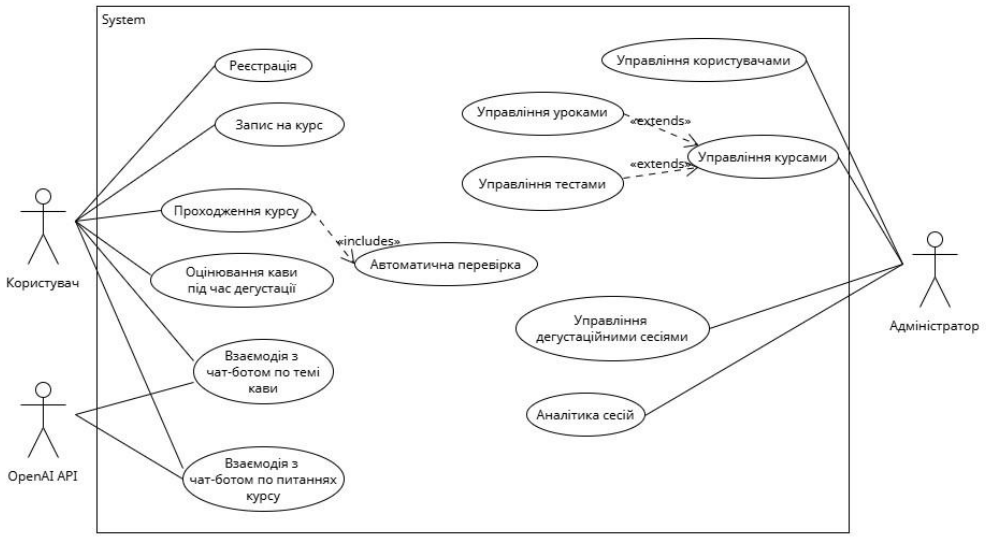
Visual Studio Code



React

6

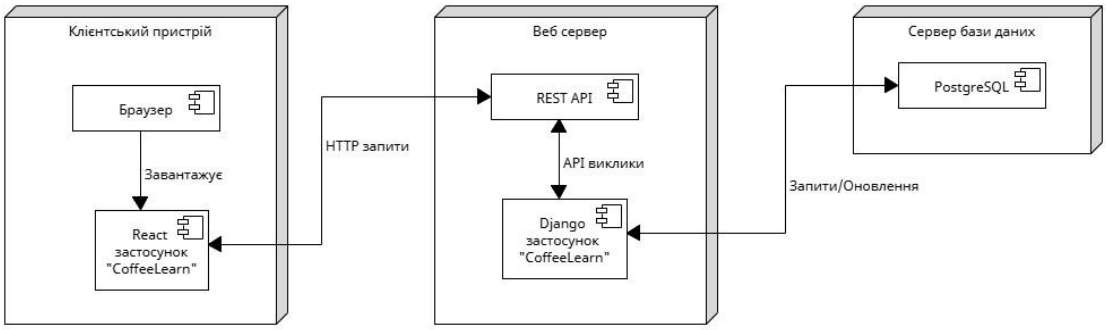
МОДЕЛЮВАННЯ ФУНКЦІОНАЛЬНОСТІ ЗАСТОСУНКУ



Діаграма варіантів використання

7

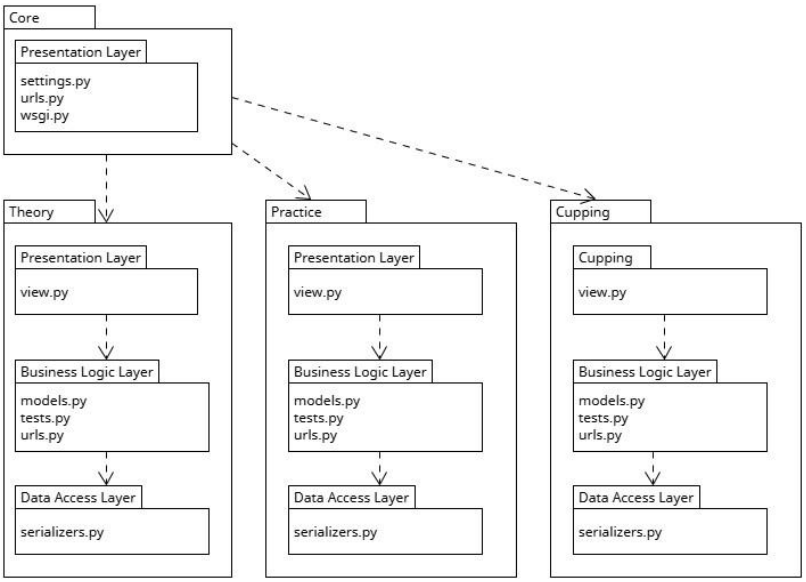
МОДЕЛЮВАННЯ РОЗГОРТАННЯ ЗАСТОСУНКУ



Діаграма розгортання

8

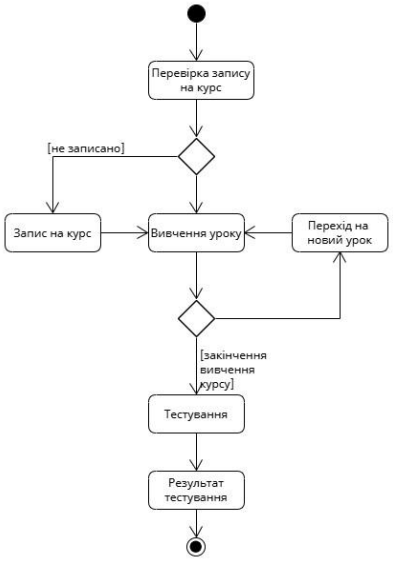
МОДЕЛЮВАННЯ АРХІТЕКТУРИ



Діаграма пакетів

9

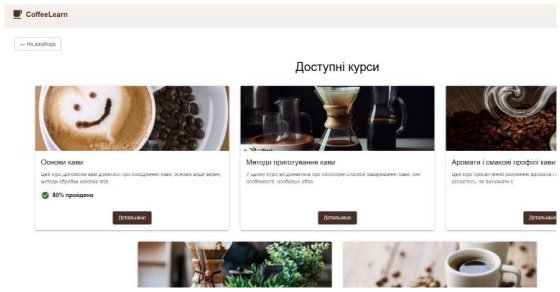
МОДЕЛЮВАННЯ ДІЯЛЬНОСТІ



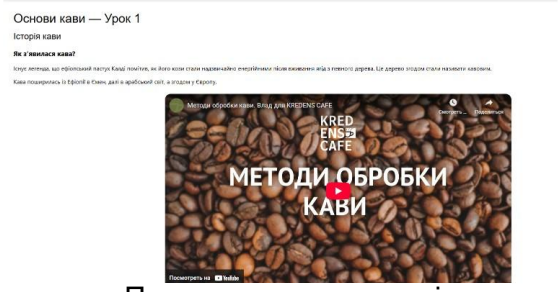
Діаграма діяльності

10

ЕКРАННІ ФОРМИ



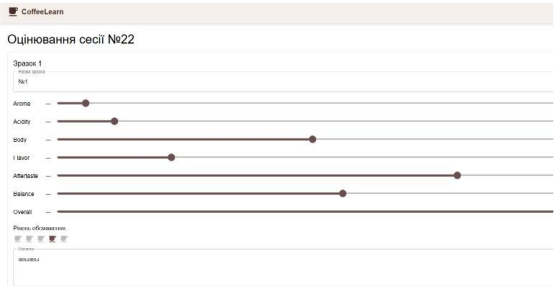
Сторінка доступних курсів



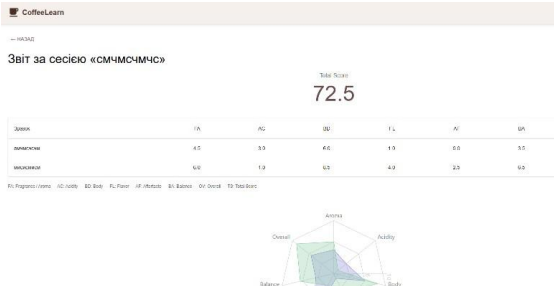
Перегляд уроку в курсі

11

ЕКРАННІ ФОРМИ



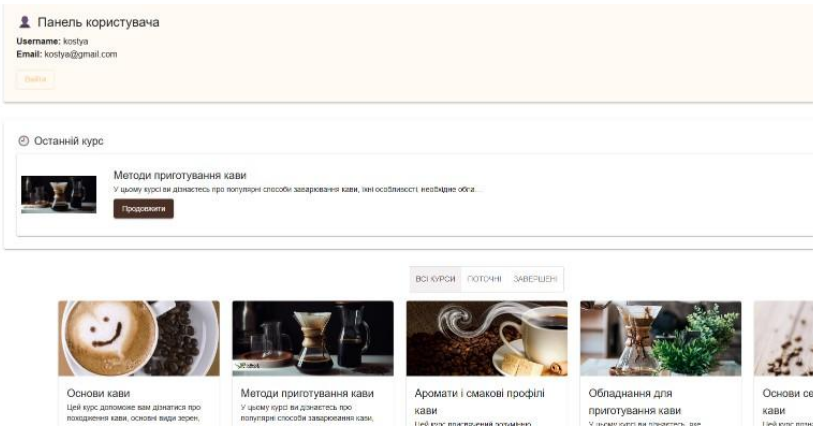
Оцінювання дегустаційної сесії



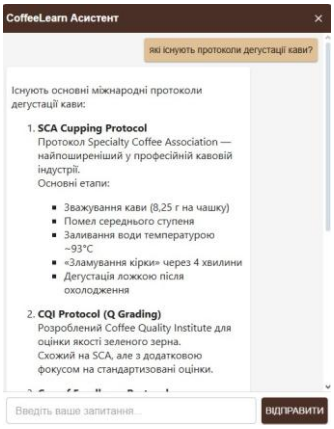
Звіт за дегустаційною сесією

12

ЕКРАННІ ФОРМИ



Панель користувача



Вбудований AI-ассистент

13

ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Козачок К.Ю., Гаманюк І.М. Розробка моделі предметної галузі застосунку для підтримки навчання професійних оцінювачів кави: Матеріали XII Всеукраїнської науково-практичної конференції молодих учених "ІТ-2025". 15 травня 2025 року, КУБГ, м. Київ. С.140-142.
2. Козачок К.Ю., Гаманюк І.М. Розробка функціональності застосунку для підтримки навчання професійних оцінювачів кави: Матеріали XII Всеукраїнської науково-практичної конференції молодих учених "ІТ-2025". 15 травня 2025 року, КУБГ, м. Київ. С.142-143.

15

ВИСНОВКИ

1. Проведено аналіз професійних стандартів і методик навчання оцінювачів кави (зокрема, Q-Grader, SCA) для визначення ключових вимог до застосунку.
2. Досліджено й проаналізовано існуючі програмні рішення, що використовуються у сфері дегустації кави, виявлено їх функціональні обмеження та переваги.
3. Виконано огляд та аналіз програмних засобів, обрано стек технологій для реалізації системи: VS Code, Python, Django, React, PostgreSQL, Git.
4. Спроектовано архітектуру застосунку з використанням UML-діаграм: варіантів використання, діаграми діяльності, компонентів та розгортання.
5. Реалізовано повнофункціональний застосунок для підтримки навчання професійних оцінювачів кави з теоретичним, практичним та аналітичним модулями.
6. Проведено тестування функціоналу, що підтвердило працездатність і відповідність системи заданим вимогам.

16

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Серверна частина
models.py

```
from django.db import models
from django.conf import settings
from markdownx.models import MarkdownxField

User = settings.AUTH_USER_MODEL

class Course(models.Model):
    title       = models.CharField(max_length=255)
    slug        = models.SlugField(unique=True)
    description = models.TextField()
    cover       =
models.ImageField(upload_to='course_covers/')
    created_at  =
models.DateTimeField(auto_now_add=True)
    students    = models.ManyToManyField(User,
related_name='enrolled_courses', blank=True)
    order       =
models.PositiveSmallIntegerField(default=0)

    class Meta:
        ordering = ['order']
    def __str__(self):
        return self.title

class Lesson(models.Model):
    course     = models.ForeignKey(Course,
related_name='lessons', on_delete=models.CASCADE)
    title      = models.CharField(max_length=255)
    order      =
models.PositiveSmallIntegerField(default=0)
    content    = MarkdownxField(blank=True,
help_text="Markdown: можна вставляти ![img](url)")
    image      =
models.ImageField(upload_to='lesson_images/',
blank=True, null=True)

    class Meta:
        ordering = ['order']

    def __str__(self):
        return f"{self.course.title} — {self.order}.
{self.title}"
from django.db import models
from django.conf import settings
from theory.models import Course

User = settings.AUTH_USER_MODEL

class Test(models.Model):
    course = models.OneToOneField(Course,
related_name='test', on_delete=models.CASCADE)
    title = models.CharField(max_length=255)
    description = models.TextField(blank=True)

    def __str__(self):
        return f"{self.title} ({self.course.title})"

class Question(models.Model):
    SINGLE = 'single'
    MULTIPLE = 'multiple'
    QUESTION_TYPES = [
        (SINGLE, 'Один правильний варіант'),
        (MULTIPLE, 'Декілька правильних варіантів'),
    ]

    test = models.ForeignKey(Test,
related_name='questions',
on_delete=models.CASCADE)
    text = models.CharField(max_length=500)
    order = models.PositiveIntegerField(default=0)
    question_type = models.CharField(max_length=10,
choices=QUESTION_TYPES, default=SINGLE)

    class Meta:
        ordering = ['order']

    def __str__(self):
        return f"Q{self.order} – {self.text[:30]}"

class Option(models.Model):
    question = models.ForeignKey(Question,
related_name='options', on_delete=models.CASCADE)
    text = models.CharField(max_length=300)
    is_correct = models.BooleanField(default=False)

    def __str__(self):
        return self.text

class TestSession(models.Model):
    user = models.ForeignKey(User,
on_delete=models.CASCADE)
    test = models.ForeignKey(Test,
related_name='sessions', on_delete=models.CASCADE)
    score_percent = models.DecimalField(max_digits=5,
decimal_places=2)
    created_at =
models.DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = ('user', 'test')
        ordering = ['-created_at']

    def __str__(self):
        return f"{self.user} → {self.test}:
{self.score_percent}%"
from django.db import models
from django.conf import settings

class CuppingSession(models.Model):
```

```
    owner       = models.ForeignKey(
                  settings.AUTH_USER_MODEL,
                  on_delete=models.CASCADE,
                  related_name='cupping_sessions'
                )
    name        = models.CharField("Назва сесії",
max_length=200)
    start       = models.DateTimeField("Початок")
    end         = models.DateTimeField("Кінець",
null=True, blank=True)
    location    = models.CharField("Локація",
max_length=200, blank=True)
    protocol    = models.CharField(
                  "Протокол",
                  max_length=100,
                  choices=[
                    ('Arabica','Арабіка'),
                    ('Robusta','Робуста'),
                    ('Cup of excellence','Кубок
досконалості'),
                    ('SCA CVA Affective','SCA CVA
Афективний'),
                    ('SCA CVA Descriptive','SCA CVA
Дескриптивний'),
                    ('SCA CVA Combined','SCA CVA
Комбінований'),
                  ],
                  default='Arabica'
                )
    created_at  =
models.DateTimeField(auto_now_add=True)
    updated_at  = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f"{self.name}
({self.start.strftime('%d.%m.%Y %H:%M')})"

class Sample(models.Model):
    session     = models.ForeignKey(
                  CuppingSession,
                  on_delete=models.CASCADE,
                  related_name='samples'
                )
    sample_id   = models.CharField("ID зразка",
max_length=50)
    name        = models.CharField("Назва зразка",
max_length=200, blank=True)
    roast_level = models.CharField("Рівень
обсмаження", max_length=50, blank=True)
    notes       = models.TextField("Нотатки", blank=True)
serializer.py

from rest_framework import serializers
from .models import Test, Question, Option, TestSession

class OptionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Option
        fields = ('id', 'text')

class QuestionSerializer(serializers.ModelSerializer):
    def __str__(self):
        return f"{self.sample_id} – {self.name or '«без
назви»'}"

class Evaluation(models.Model):
    sample      = models.OneToOneField(
                  Sample,
                  on_delete=models.CASCADE,
                  related_name='evaluation'
                )
    aroma       = models.DecimalField("Арома",
max_digits=4, decimal_places=2, default=0)
    acidity     = models.DecimalField("Кислинка",
max_digits=4, decimal_places=2, default=0)
    body        = models.DecimalField("М'якість",
max_digits=4, decimal_places=2, default=0)
    flavor      = models.DecimalField("Смак",
max_digits=4, decimal_places=2, default=0)
    aftertaste  = models.DecimalField("Післясмак",
max_digits=4, decimal_places=2, default=0)
    balance     = models.DecimalField("Баланс",
max_digits=4, decimal_places=2, default=0)
    overall     = models.DecimalField(
                  "Загальна оцінка",
                  max_digits=4,
                  decimal_places=2,
                  default=0,
                  help_text="Якщо залишити порожнім —
обчислюється середнім"
                )
    created_at  =
models.DateTimeField(auto_now_add=True)
    updated_at  = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f"Evaluation for {self.sample.sample_id}"

    def calculate_overall(self):
        vals = [
            float(self.aroma),
            float(self.acidity),
            float(self.body),
            float(self.flavor),
            float(self.aftertaste),
            float(self.balance),
        ]
        return sum(vals) / len(vals) if vals else 0


    options = OptionSerializer(many=True,
read_only=True)
    class Meta:
        model = Question
        fields = ('id', 'text', 'order', 'question_type', 'options')

class TestDetailSerializer(serializers.ModelSerializer):
    questions = QuestionSerializer(many=True,
read_only=True)
    result = serializers.SerializerMethodField()

    class Meta:
```

```
        model = Test
        fields = ('id', 'title', 'description', 'questions', 'result')

    def get_result(self, obj):
        user = self.context['request'].user
        session = obj.sessions.filter(user=user).first()
        if session:
            return {
                "score_percent": session.score_percent,
                "created_at": session.created_at
            }
        return None

class SubmitTestSerializer(serializers.Serializer):
    answers = serializers.ListField(
        child=serializers.DictField(), write_only=True
    )

    def validate_answers(self, value):
        test = self.context['test']
        question_ids = set(test.questions.values_list('id',
flat=True))
        for ans in value:
            if ans.get('question') not in question_ids:
                raise serializers.ValidationError(f"Питання
{ans.get('question')} не в цьому тесті.")
        return value

    def validate(self, attrs):
        user = self.context['request'].user
        test = self.context['test']
view.py

from rest_framework import generics, permissions,
status
from rest_framework.response import Response
from rest_framework.views import APIView
from .models import Course
from .serializers import CourseListSerializer,
CourseDetailSerializer
from practice.models import TestSession

class CourseListView(generics.ListAPIView):
    queryset = Course.objects.all()
    serializer_class = CourseListSerializer
    permission_classes = (permissions.AllowAny,)

class CourseDetailView(generics.RetrieveAPIView):
    queryset = Course.objects.all()
    lookup_field = 'slug'
    serializer_class = CourseDetailSerializer
    permission_classes = (permissions.AllowAny,)

class EnrollCourseView(generics.GenericAPIView):
    queryset = Course.objects.all()
    lookup_field = 'slug'
    permission_classes = (permissions.IsAuthenticated,)

    def post(self, request, slug):
        course = self.get_object()
        course.students.add(request.user)
        if TestSession.objects.filter(user=user,
test=test).exists():
            raise serializers.ValidationError("Тест уже
пройдено.")
        return attrs

    def create(self, validated_data):
        user = self.context['request'].user
        test = self.context['test']
        total = test.questions.count()
        correct = 0
        answers_map = {
            a['question']: set(a.get('selected_options', [])) for
a in validated_data['answers']
        }

        for question in test.questions.all():
            correct_ids =
set(question.options.filter(is_correct=True).values_list('i
d', flat=True))
            selected_ids = answers_map.get(question.id,
set())
            if correct_ids == selected_ids:
                correct += 1

        score = round((correct / total) * 100, 2) if total else
0
        return TestSession.objects.create(user=user,
test=test, score_percent=score)


        return Response({'detail': 'Enrolled successfully'},
status=status.HTTP_200_OK)

class DashboardCourseListView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request):
        user = request.user
        all_courses = Course.objects.all()
        test_sessions = TestSession.objects.filter(user=user)

        latest_session = test_sessions.order_by('-
created_at').first()
        last_course_id = latest_session.test.course_id if
latest_session else None

        data = []
        for course in all_courses:
            session =
test_sessions.filter(test__course=course).first()
            score = session.score_percent if session else
None

            data.append({
                "id": course.id,
                "title": course.title,
                "slug": course.slug,
                "cover": course.cover.url if course.cover else '',
                "description": course.description,
                "test_result": score,
```

```
            "is_enrolled":
course.students.filter(pk=user.pk).exists(),
            "is_completed": score is not None and score
>= 60,
            "is_started": score is not None,
```

Клієнтська частина

App.tsx

```
import React, { useState, useEffect } from 'react';
import {
  BrowserRouter,
  Routes,
  Route,
  Navigate,
  useNavigate,
} from 'react-router-dom';
import { ThemeProvider, Dialog, CircularProgress,
Box } from '@mui/material';
import theme from './theme';

import Header from './components/Header';
import Home from './components/Home';
import Register from './components/Register';
import Login from './components/Login';
import Dashboard from './components/Dashboard';
import Profile from './components/Profile';
import CourseList from './components/CourseList';
import CourseDetail from './components/CourseDetail';
import ChatWidget from './components/ChatWidget';

import CuppingSessionsList from
'./components/CuppingSessionsList';
import CuppingSessionForm from
'./components/CuppingSessionForm';
import CuppingForm from './components/CuppingForm';
import SessionReport from
'./components/SessionReport';

const AppContent: React.FC = () => {
  const [isAuth, setIsAuth]         = useState(false);
  const [username, setUsername]     = useState('');
  const [showLogin, setShowLogin]   = useState(false);
  const [showRegister, setShowRegister] =
useState(false);
  const [loading, setLoading]       = useState(true);
  const navigate = useNavigate();

  useEffect(() => {
    const access = localStorage.getItem('access_token');
    const user   = localStorage.getItem('username');
    if (access && user) {
      setUsername(user);
      setIsAuth(true);
    }
    setLoading(false);
  }, []);

  const handleLoginSuccess = (
    user: string,
    access: string,
    refresh: string
  ) => {
```

```
            "is_last": course.id == last_course_id
          })

        return Response(data)
```

```
    localStorage.setItem('access_token', access);
    localStorage.setItem('refresh_token', refresh);
    localStorage.setItem('username', user);
    setUsername(user);
    setIsAuth(true);
    setShowLogin(false);
    navigate('/dashboard');
  };

  const handleLogout = () => {
    localStorage.removeItem('access_token');
    localStorage.removeItem('refresh_token');
    localStorage.removeItem('username');
    setIsAuth(false);
    setUsername('');
    navigate('/');
  };

  if (loading) {
    return (
      <Box
        sx={{
          display:    'flex',
          justifyContent: 'center',
          alignItems: 'center',
          height:     '100vh',
        }}
      >
        <CircularProgress />
      </Box>
    );
  }

  return (
    <>
      <Header
        isAuthenticated={isAuth}
        username={username}
        onLoginClick={() => setShowLogin(true)}
        onRegisterClick={() => setShowRegister(true)}
        onNavigateDashboard={() =>
navigate('/dashboard')}
        onNavigateProfile={()  => navigate('/profile')}
        onNavigateCupping={() =>
navigate('/cupping/sessions')}
        onLogout={handleLogout}
      />

      <Dialog open={showLogin}   onClose={() =>
setShowLogin(false)}>
        <Login onLoginSuccess={handleLoginSuccess} />
      </Dialog>
      <Dialog open={showRegister} onClose={() =>
setShowRegister(false)}>
        <Register
onRegisterSuccess={handleLoginSuccess} />
      </Dialog>
```

```
<Routes>
  <Route path="/" element={<Home />} />

  <Route
    path="/dashboard"
    element={
      isAuth
        ? <Dashboard onLogout={handleLogout} />
        : <Navigate to="/" replace />
    }
  />
  <Route
    path="/profile"
    element={
      isAuth
        ? <Profile />
        : <Navigate to="/" replace />
    }
  />

  <Route
    path="/theory/courses"
    element={
      isAuth
        ? <CourseList />
        : <Navigate to="/" replace />
    }
  />
  <Route
    path="/theory/courses/:slug"
    element={
      isAuth
        ? <CourseDetail />
        : <Navigate to="/" replace />
    }
  />

  <Route
    path="/cupping/sessions"
    element={
      isAuth
        ? <CuppingSessionsList />
        : <Navigate to="/" replace />
    }
  />
  <Route
    path="/cupping/sessions/new"
    element={
      isAuth
        ? <CuppingSessionForm />
        : <Navigate to="/" replace />
    }
  />
  <Route
    path="/cupping/sessions/:id/edit"
    element={
      isAuth
        ? <CuppingSessionForm />
        : <Navigate to="/" replace />
    }
  />
  <Route
    path="/cupping/sessions/:id"
    element={
      isAuth
        ? <CuppingForm />
        : <Navigate to="/" replace />
    }
  />
  <Route
    path="/cupping/sessions/:id/report"
    element={
      isAuth
        ? <SessionReport />
        : <Navigate to="/" replace />
    }
  />

  <Route path="*" element={<Navigate to="/" replace />} />
</Routes>

<ChatWidget />
</>
);
};

const App: React.FC = () => (
<ThemeProvider theme={theme}>
<BrowserRouter>
<AppContent />
</BrowserRouter>
</ThemeProvider>
);

export default App;
```