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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для самостійного вивчення програмування з інтерактивними завданнями та підказками на основі штучного інтелекту»

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

_____ Микита ШЕРСТНЬОВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Микита ШЕРСТНЬОВ

Керівник: Володимир САДОВЕНКО
к.ф-м.н., доцент

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

Шерстньову Микиті Андрійовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для самостійного вивчення програмування з інтерактивними завданнями та підказками на основі штучного інтелекту»
керівник кваліфікаційної роботи к.ф-м.н., доцент Володимир САДОВЕНКО,
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 методи та засоби самостійного вивчення програмування, науково-технічна література, технічна документація до інструментів для розробки web-застосунків.

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

1. Огляд методів і засобів самостійного вивчення програмування і підвищення якості процесу навчання за допомогою інтеграції в нього моделі штучного інтелекту.
2. Огляд та аналіз існуючих web-застосунків для самостійного вивчення програмування

3. Проєктування web-застосунку для самостійного вивчення програмування з інтерактивними завданнями та підказками на основі штучного інтелекту.
4. Розробка та тестування web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма архітектури.
6. Діаграма послідовності реєстрації та авторизації.
7. Діаграма послідовності роботи із задачами.
8. Діаграма класів.
9. Екранні форми.
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	Огляд існуючих методів і засобів самостійного вивчення програмування та підвищення якості процесу навчання за допомогою інтеграції в нього моделі штучного інтелекту	14.03-18.03.2025	
4	Проектування web-застосунку для самостійного вивчення програмування зі інтерактивними завданнями та підказками на основі штучного інтелекту	19.03-27.03.2025	
5	Програмна реалізація web-застосунку	28.03-20.04.2025	
6	Тестування web-застосунку	21.04-28.04.2025	
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 вищої освіти _____ Микита ШЕРСТНЬОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 1 табл., 12 рис., 22 джерел.

Мета роботи – покращення процесу самостійного навчання програмуванню шляхом інтеграції моделі штучного інтелекту у процес навчання.

Об'єкт дослідження – процеси самостійного вивчення програмування за допомогою web-застосунків.

Предмет дослідження – web-застосунок, що забезпечує інтерактивне навчання програмуванню з використанням технологій штучного інтелекту.

Короткий зміст роботи: В роботі було розглянуто методи та підходи до самостійного навчання використовуючи web-платформи, а також використання моделей штучного інтелекту для покращення цього процесу. Сформульовано вимоги до web-застосунку на основі аналізу предметної області та порівняльного огляду аналогів. Проаналізовано засоби для реалізації web-застосунку для самостійного навчання програмуванню з інтеграцією моделі штучного інтелекту. Спроєктовано та розроблено клієнтську і серверну частини web-застосунку для забезпечення функціональних та нефункціональних вимог. У роботі використано Python та FastAPI для серверної частини, TypeScript та фреймворк React для клієнтської частини, PostgreSQL для збереження даних і модель OpenAI gpt-3.5-turbo для генерації підказок.

КЛЮЧОВІ СЛОВА: ПЛАТФОРМА ДЛЯ САМОСТІЙНОГО НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, WEB-ЗАСТОСУНОК, PYTHON

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 8
ВСТУП 9
1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧ WEB-ЗАСТОСУНКУ ДЛЯ НАВЧАННЯ ПРОГРАМУВАННЮ 11
1.1 Опис предметної області 11
1.2 Аналітичний огляд існуючих рішень 13
1.3 Моделювання вимог до web-застосунку 20
2 ЗАСОБИ РЕАЛІЗАЦІЇ 25
2.1 Мови програмування 25
2.2 Фреймворки 26
2.3 Бази даних 28
2.4 Моделі штучного інтелекту 31
2.5 Середовища розробки 32
3 ПРОЄКТУВАННЯ СИСТЕМИ WEB-ЗАСТОСУНКУ ДЛЯ САМОСТІЙНОГО НАВЧАННЯ ПРОГРАМУВАННЮ 33
3.1 Загальна архітектура системи 33
3.2 Клієнтська частина (Frontend) 35
3.3 Серверна частина (Backend) 37
3.4 Моделі бази даних 40
3.5 Безпека та автентифікація 42
3.6 Інтерфейс користувача 43
4 РЕАЛІЗАЦІЯ СИСТЕМИ ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ 47
4.1 Реалізація клієнтської частини (Frontend) 47
4.2 Реалізація серверної частини (Backend) 50
4.3 Інтеграція моделі штучного інтелекту 52
4.4 Тестування користувацького коду 54
ВИСНОВКИ 57
ПЕРЕЛІК ПОСИЛАНЬ 59
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 62
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ 69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ – штучний інтелект

ПЗ – програмне забезпечення

ORM – object-relational mapping

DRY – don’t repeat yourself

API – application programming interface

HTTP – hypertext transfer protocol

SQL – structured query language

GPT – generative pre-trained

IDE – integrated development environment

DOM – document object model

UML – unified modelling language

CSS – cascading style sheets

ER – entity relationship

JSON – JavaScript object notation

JWT – JSON web token

XSS – cross-site scripting

UX – user experience

UI – user interface

ASGI – asynchronous server gateway interface

RBAC – role-based access control

ВСТУП

Актуальність: програмування – це ключова навичка, яка охоплює майже усі сфери життя у сучасному світі. Попит на вміння та знання у використанні мов програмування для вирішення практичних завдань невпинно зростає. Це створює необхідність у якісних платформах для самостійного вивчення таких технологій, оскільки традиційні освітні підходи часто не відповідають сучасним потребам студентів.

Більшість платформ для самостійного вивчення програмування пропонують лише базовий функціонал та як правило не враховують індивідуальних потреб користувачів, не забезпечують інтерактивності та адаптивності у навчанні. Крім того нерідкою проблемою є недостатня підтримка початківців, які потребують більш детального пояснення.

Впровадження штучного інтелекту у процес навчання створює персоналізований підхід до кожного користувача, забезпечуючи адаптивні підказки та пояснення до завдань, що значно підвищує ефективність навчального процесу [1, с.3].

Розробка web-застосунку для самостійного вивчення програмування з інтерактивними завданнями та підказками на основі ШІ є актуальною, оскільки поєднує зручність використання новітніх технологій з сучасними методами навчання. Такий підхід сприятиме розвитку навичок програмування серед широкої аудиторії незалежно від їх освіти та досвіду.

Об'єктом дослідження є процеси самостійного вивчення програмування за допомогою web-застосунків.

Предметом дослідження є web-застосунок, що забезпечує інтерактивне навчання програмуванню з використанням технологій штучного інтелекту.

Мета роботи - покращення процесу самостійного навчання програмуванню шляхом інтеграції моделі штучного інтелекту у процес навчання.

Методи дослідження: для досягнення поставленої мети спочатку було проаналізовано переваги та недоліки традиційних підходів до навчання, а також яким чином сучасні web-застосунки прагнуть забезпечити кращі альтернативи для того, щоб сформувати початкові функціональні та нефункціональні вимоги до програмного забезпечення. Далі було проведено порівняльний аналіз web-застосунків для самостійного навчання програмуванню, щоб сформувати уявлення про те, що має бути доступно у web-застосунку для того, щоб задовольнити потреби користувачів. Після цього було проведено огляд технологій для розробки програмного забезпечення, які дозволяють реалізувати сформовані вимоги до ПЗ, і було обрано TypeScript та фреймворк React для розробки клієнтської частини web-застосунку, FastAPI для серверної частини, базу даних PostgreSQL для збереження даних та інтеграцію OpenAI моделі штучного інтелекту для генерації адаптивних підказок.

Наукова новизна роботи полягає у використанні та комбінації найсучасніших технологій розробки web-застосунків та інтеграції моделі штучного інтелекту для створення персоналізованого досвіду користувача. Web-застосунок пропонує сучасніший підхід до самостійного навчання.

Практичне значення роботи полягає у можливості використання web-застосунку для самостійного вивчення мов програмування за допомогою інтерактивних задач та адаптивних підказок, які генеруються на основі користувацького вводу моделлю штучного інтелекту для підвищення ефективності навчання та засвоєння матеріалу.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧ WEB-ЗАСТОСУНКУ ДЛЯ НАВЧАННЯ ПРОГРАМУВАННЮ

1.1 Опис предметної області

Програмування відіграє ключову роль у розвитку сучасного суспільства та технологій. У світі стрімкої цифровізації, що охоплює усі сфери життя – від бізнесу до освіти, охорони здоров'я та розваг – навички програмування стають однією з найважливіших компетенцій. Програмування забезпечує інструменти для створення програмного забезпечення, мобільних застосунків, систем штучного інтелекту та багато чого іншого для автоматизації процесів та вирішення проблем.

У сучасному світі програмування є не лише професією, але й засобом самовираження та реалізації інноваційних ідей. До того ж стрімкий розвиток штучного інтелекту, технологій Big Data та інтернету речей ще сильніше збільшує значення цієї навички та необхідність у кваліфікованих спеціалістах.

Зростання попиту на IT-фахівців зумовлює необхідність підготовки нових кадрів, що підкреслює важливість доступу до якісних та ефективних навчальних платформ.

Самостійне вивчення програмування часто стає викликом для початківців через низку труднощів, які виникають у цьому процесі. Основними проблемами є відсутність адаптивного пояснення матеріалу і зворотного зв'язку. Особливо це впливає на засвоєння складного матеріалу такого як алгоритми, структури даних або будь-який просунутий функціонал деякої мови програмування. Інтеграція таких можливостей, зокрема через використання штучного інтелекту, здатна зробити процес навчання доступнішим та ефективнішим.

Чимало онлайн-ресурсів пропонують лише базовий матеріал з загальними рекомендаціями, що змушує початківців та особливо користувачів без технічної освіти, звертатися до інших менш ефективних методів навчання.

Існуючі підходи до самостійного навчання часто не враховують індивідуальні потреби та особливості учнів, що значно знижує ефективність навчання. Традиційні методи також втрачають свою актуальність, оскільки обсяг і складність інформації не дозволяють студентам засвоювати інформацію у достатній мірі.

Інтеграція моделі штучного інтелекту в процес навчання здатна вирішити більшість недоліків сучасних освітніх методів. Наприклад, генеративні моделі здатні змінювати матеріал таким чином, щоб користувач міг його легше та швидше засвоїти. На основі поведінки та визначення сильних і слабких сторін користувача впродовж деякого часу, модель може адаптувати матеріал під кожного окремого користувача [2, с.14].

Додатково, штучний інтелект здатний працювати у інтерактивному форматі, коли модель отримує дані на вхід, та повертає на його основі певну відповідь, а також запам’ятовує інформацію з попередніх діалогів для підвищення точності. У рамках web-застосунку для самостійного навчання це можна використати для підтримки інтерактивного формату роботи [3, с.1856].

Студенти все більше звертаються до більш сучасних платформ для навчання, які пропонують інтерактивні завдання, що дозволяють ефективніше запам’ятовувати матеріал, а також застосовувати його під час виконання практичних завдань. Можливість навчатися будь-де будь-коли, автоматична перевірка результатів, підказки та рекомендації ще сильніше покращують процес навчання. Такі платформи, як Coursera, Udemy, Codeacademy демонструють високу популярність, що також є доказом невпинного зросту попиту на web-застосунки для самостійного навчання.

З огляду на це, потреба у інструментах для самостійного інтерактивного навчання буде зростати, особливо в умовах глобалізації освіти.

1.2 Аналітичний огляд існуючих рішень

Існує безліч онлайн-платформ, які пропонують інструменти для навчання програмуванню. Їх популярність обумовлена зручністю, доступністю і можливістю отримати знання в будь-який час і з будь-якого місця. Розглянемо кілька найбільш відомих платформ, які допомагають користувачам набувати практичних навичок програмування.

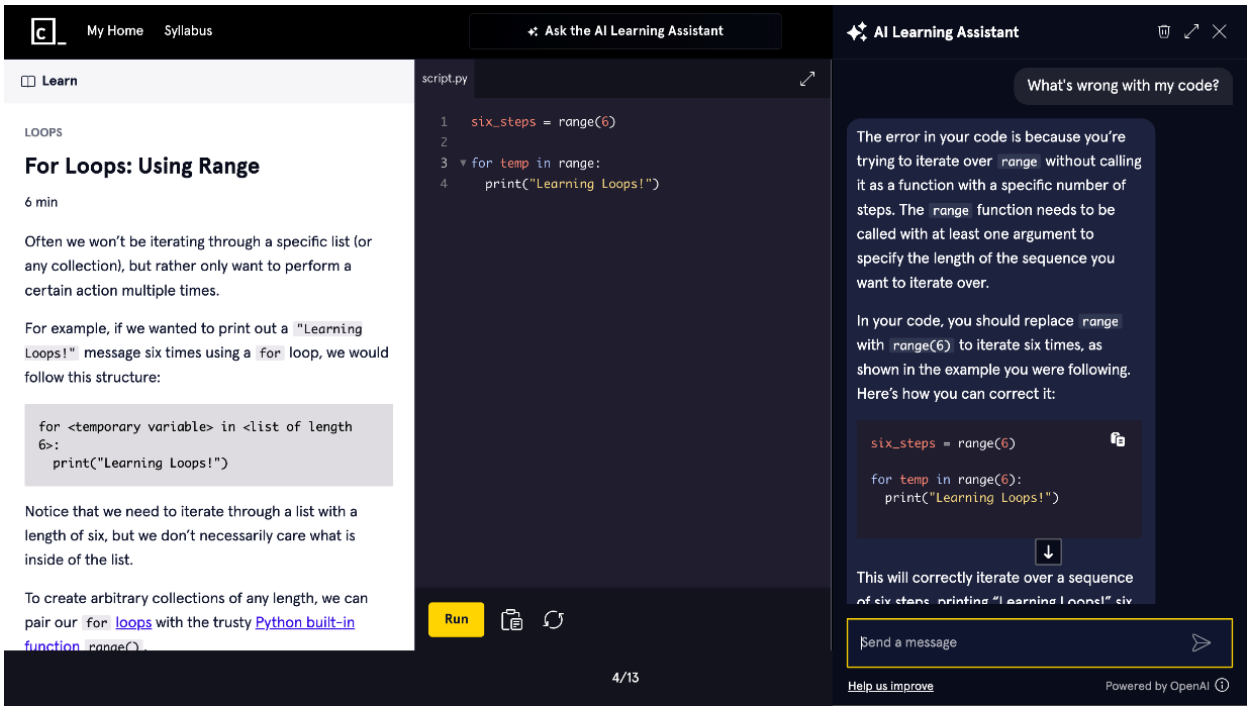


Рис.1.1 Codeacademy

Codeacademy – це популярна платформа з інтерактивними курсами для освоєння навичок програмування для багатьох мов, таких як Java, C++, Python та інші. Користувачі можуть вивчати програмування за допомогою інтерактивних уроків, виконуючи реальні завдання, які відразу перевіряються. Платформа також надає користувачам зворотний зв'язок у реальному часі, що допомагає покращити розуміння матеріалу. Codeacademy підходить як для новачків, так і для більш досвідчених програмістів, оскільки пропонує курси на різних рівнях складності.

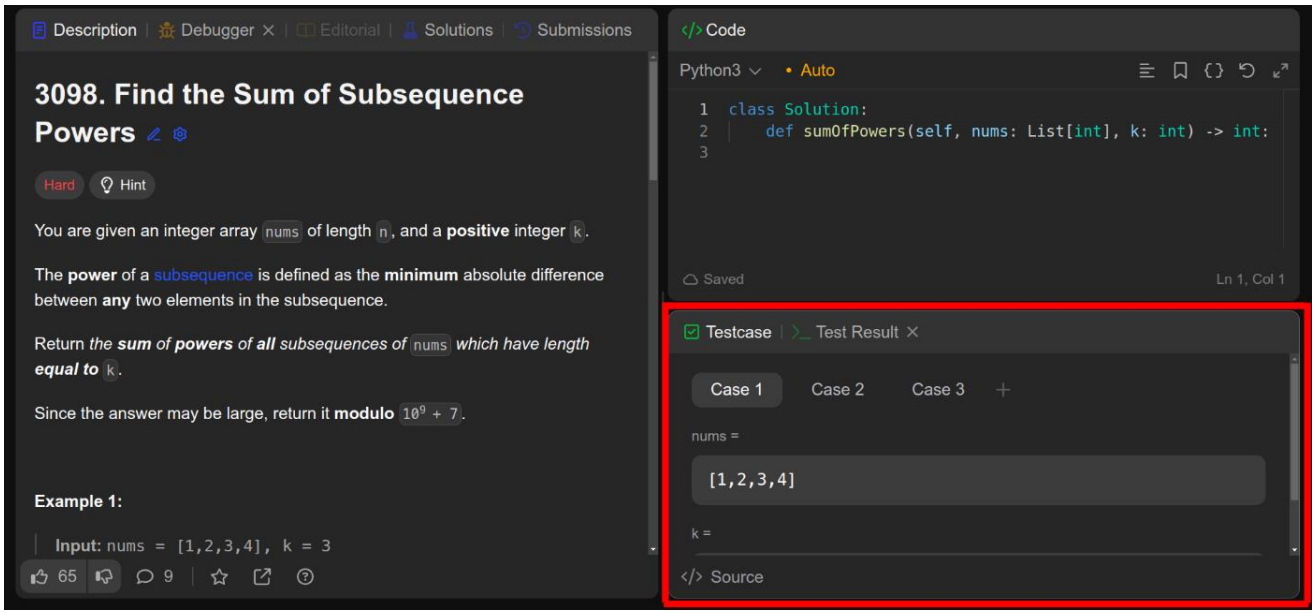


Рис.1.2 LeetCode

LeetCode здобула популярність серед розробників, які готуються до співбесід у провідних технологічних компаніях, таких як Alphabet, Netflix та Meta. Платформа спеціалізується на задачах з алгоритмів і структур даних, які є основою технічних співбесід. LeetCode пропонує велику кількість задач різного рівня складності, починаючи від простих і до дуже складних, що дозволяє користувачам систематично вдосконалювати свої навички. Також на платформі є функціонал для обговорення рішень з іншими учасниками, що дозволяє краще зрозуміти концепції та різні підходи до вирішення завдань, а також порівняння ефективності написаного рішення із результатами інших користувачів. Недолік даної платформи полягає у відсутності послідовності задач хоча й вони поділяються на групи за темою та складністю. На LeetCode відсутні персональні підказки, єдина допомога – це завчасно продумані текстові повідомлення, які поступово пояснюють як вирішувати задачу.

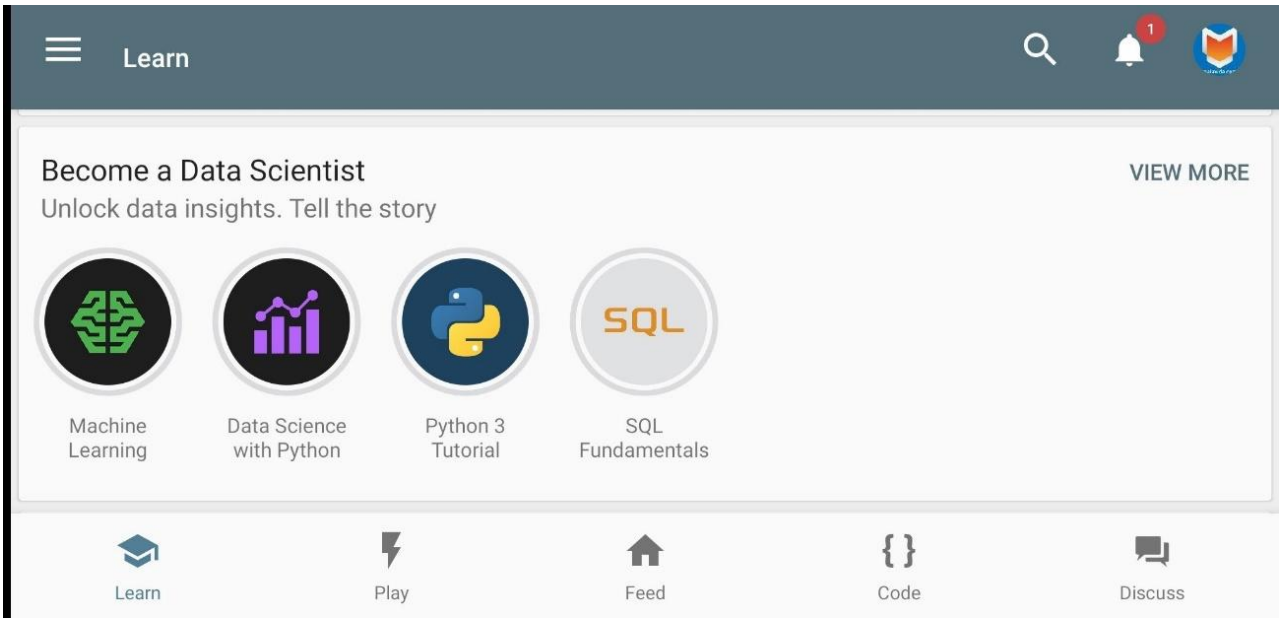


Рис.1.3 SoloLearn

SoloLearn – це платформа, орієнтована на навчання програмуванню за допомогою міні-уроків та інтерактивних завдань. Вона надає курси з багатьох мов програмування, таких як Python, JavaScript, C++, Swift та інші. Особливістю SoloLearn є доступність курсів для новачків і можливість змагатися з іншими користувачами в режимі реального часу, що робить навчання більш інтерактивним і мотиваційним. Матеріал на SoloLearn подається у вигляді короткий теоретичних пояснень та інтерактивних вправ для гарного засвоєння матеріалу. Платформа також має мобільний застосунок, що дозволяє вчитися в будь-який час, навіть без доступу до комп'ютера. Особливість цієї платформи полягає у тому, що вона розраховано на новачків та використовує елементи гейміфікації у процесі навчання. На платформі також є спільнота, де користувачі можуть обговорювати матеріал, задачі та рішення.

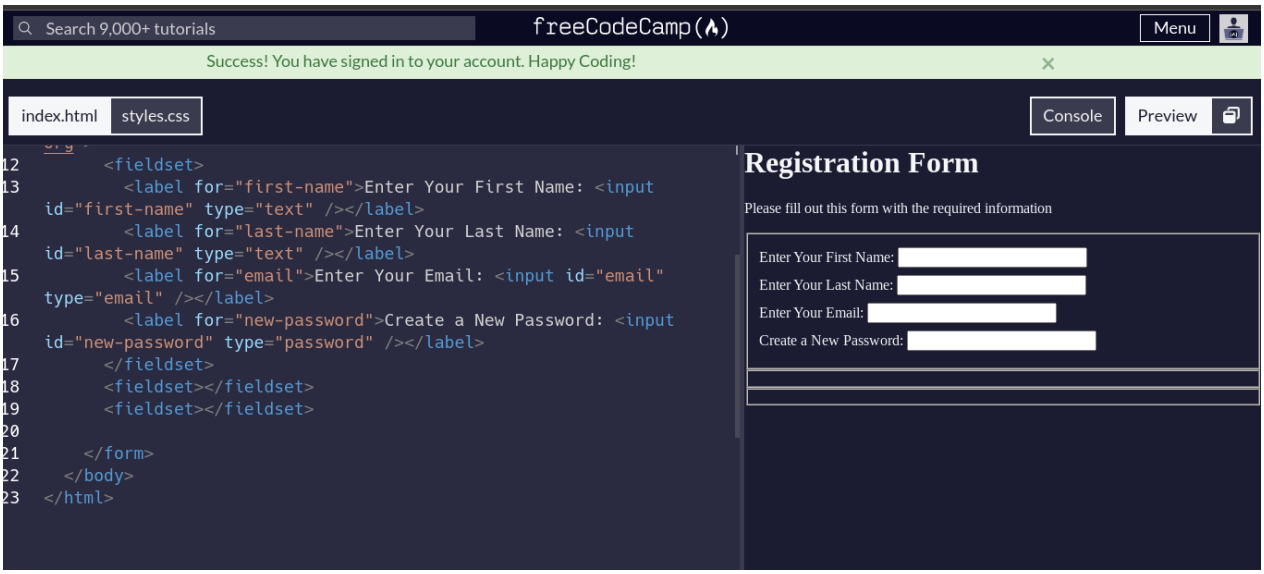


Рис.1.4 freeCodeCamp

freeCodeCamp є безкоштовною платформою для вивчення web-розробки, програмування та роботи з різними технологіями. Вона пропонує велику кількість практичних завдань та проєктів, що дозволяють студентам отримати реальний досвід у розробці застосунків і web-сайтів у умовах, які максимально наближені до реального світу. Крім того, freeCodeCamp пропонує сертифікацію за різними напрямками, такими як web-розробка, бази даних, алгоритми та інші. Оскільки весь контент на платформі безкоштовний, вона стала популярною серед тих, хто хоче отримати практичні навички без витрат на навчання. freeCodeCamp також має свою спільноту, яка через велику популярність платформи дозволяє користувачам швидко знаходити відповіді на будь-які питання. Це дозволяє новачкам легко починати вивчати нові технології та долати складні перешкоди.

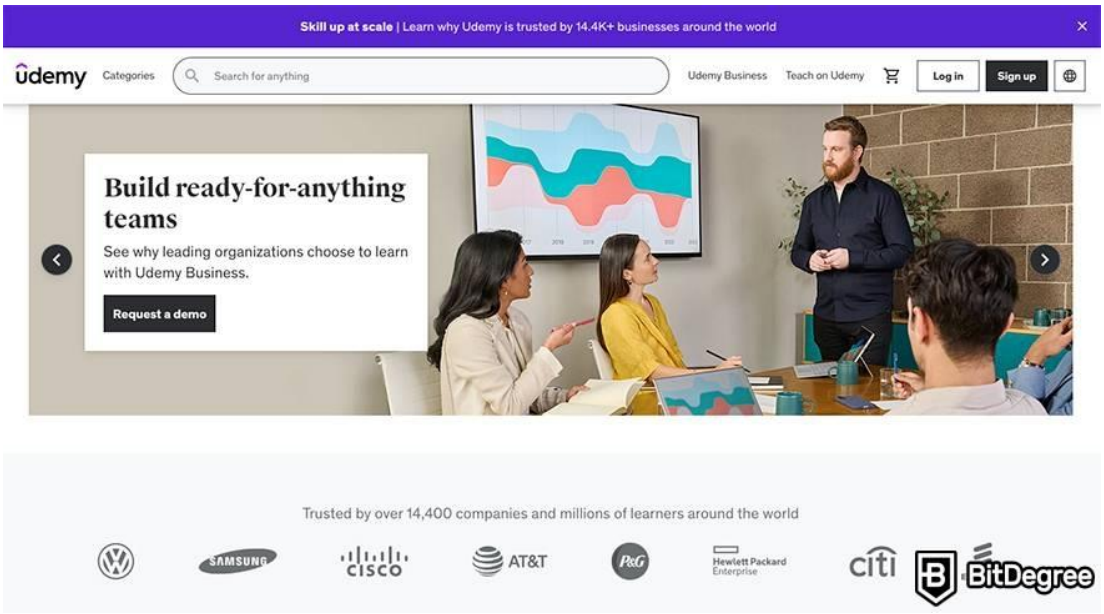


Рис.1.5 Udemy

Udemy — одна з найбільших платформ для онлайн-курсів, яка пропонує навчальні матеріали з програмування, web-розробки, аналізу даних, машинного навчання та багатьох інших тем. На платформі представлені курси від професіоналів і викладачів з усього світу. Користувачі можуть вибирати курси на основі своїх інтересів і рівня знань, а також отримувати сертифікати після завершення навчання. Udemy відома своєю великою бібліотекою курсів, що дозволяє задовольнити потреби різних категорій студентів.

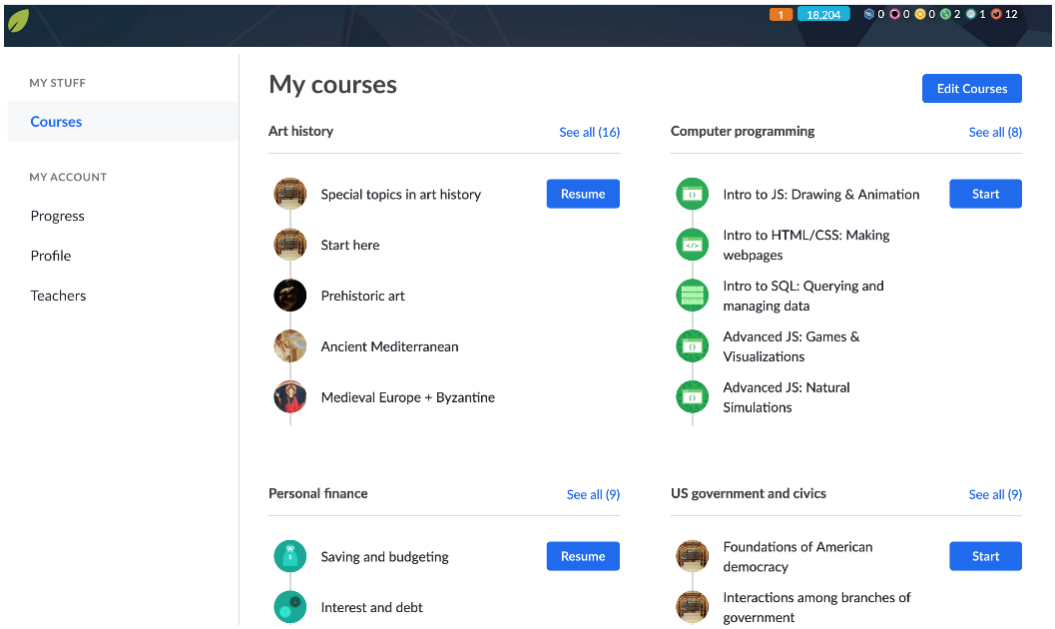


Рис.1.6 Khan Academy

Khan Academy пропонує безкоштовний доступ до навчальних матеріалів з різних предметів, включаючи програмування. Особливістю цієї платформи є простота пояснення складних концепцій, що робить її доступною навіть для молодших учнів або тих, хто тільки починає освоювати програмування. Платформа надає інтерактивні уроки з основ програмування, а також візуалізацію алгоритмів і структур даних. Система має вбудовану базову адаптацію у вигляді повторного пояснення тем за необхідності, проте це не дає сильного ефекту на засвоєння матеріалу.

Окремі сучасні платформи, наприклад, CodeGym або JavaRush активно використовують гейміфіковані методи навчання, тобто процес супроводжується ігровими механіками: система балів, рейтинг, візуалізація прогресу, персонажі, квести та інше. Такі підходи стимулюють залученість студентів і ефективність їх навчання, а також позитивно впливають на загальний користувацький досвід. Хоча з іншого боку це орієнтовано в першу чергу на дитячу аудиторію та ефективність таких механік може падати із підвищенням віку користувачів [4].

Аналіз публікацій платформ, що пропонують інтерактивні інструменти для вивчення програмуванню, підтверджують, що необхідність у таких застосунках невпинно зростає. Наведені метрики та статистики показують, що кількість користувачів таких платформ активно зростає [5, с.281-284].

Всі ці платформи мають спільну мету – допомогти користувачам освоїти програмування завдяки інтерактивним завданням. Кожна з платформ має різні переваги та матеріали, що пропонуються на них. Вони орієнтовані на різні категорії користувачів і пропонують різноманітні підходи до навчання. Проте ці платформи не вирішують усі потреби сучасних студентів. Не дивлячись на продуманий матеріал та велику кількість практичних завдань, високий поріг входу та відсутність адаптивності знижує ефективність навіть цих передових навчальних платформ.

Зведені результати порівняльного аналізу web-платформ для навчання наведено у таблиці 1.1.

Таблиця 1.1

Зведені результати порівняльного аналізу web-платформ для навчання

Продукт/ Функціо-нальність	Code Academy	LeetCode	SoloLearn	freeCodeCamp	Udemy	Khan Academy	Dream Learn
Інтер-активний код у браузері	+	+	+	+	-	+	+
Структу-рованість	+	-	-	+	+	+	+
Алгорит-мічні задачі	-	+	-	-	-	-	+
Адаптивні підказки/ матеріал	-	-	-	-	-	-	+
Автома-тична перевірка коду	+	+	-	+	-	-	+
Відстеже-ння прогресу	+	+	+	+	+	+	+

1.3 Моделювання вимог до web-застосунку

Web-застосунок для самостійного вивчення програмування має на меті забезпечити користувачам інтерактивне навчання з різноманітними завданнями, підказками та можливістю відстеження їхнього прогресу. Це дозволяє створити персоналізований навчальний досвід, що підтримує мотивацію та ефективність навчання.

Одним із ключових аспектів застосунку є наявність інтерактивних завдань, які користувачі можуть виконувати безпосередньо на платформі. Завдання повинні бути різноманітними та охоплювати основні аспекти програмування, такі як синтаксис, алгоритми, структури даних, розв'язання задач мовами програмування (наприклад, Python чи JavaScript). Кожне завдання повинно включати пояснення теоретичних аспектів, а також практичне завдання.

Завдання повинні бути поділені за рівнями складності, щоб користувач міг поступово проходити від простих до складних. Це дозволить створити ефективну траєкторію навчання і підтримувати поступовий розвиток навичок.

Користувач повинен мати змогу надсилати написаний код на перевірку, яка має автоматично перевіряти правильність рішення та повідомляти про результат користувача.

Інтерактивність навчального процесу також включає систему підказок, яка допомагає користувачам, коли вони стикаються з труднощами у виконанні завдань. Підказки виглядають як короткі текстові повідомлення, які дають пояснення деяких концептів або натякають на наявність помилок у коді. Вони допомагають зрозуміти матеріал, а також як його використовувати на практиці, якщо теоретичного матеріалу було недостатньо.

Важливою особливістю є адаптивність підказок, оскільки вони позитивно впливають на результати та залучення студентів. Спочатку система може давати загальні поради, але, якщо користувач не в змозі вирішити завдання, система має пропонувати більш детальні підказки, аж до повного рішення або фрагментів коду [6, с.13].

Для забезпечення відстеження досягнень користувачів в процесі навчання, система повинна включати механізм збереження прогресу. Кожен користувач повинен мати свій персональний обліковий запис, де зберігається історія виконаних завдань які вони успішно виконали, щоб бачити свій прогрес.

Прогрес користувача повинен зберігатися навіть після виходу з системи, що дозволяє користувачеві повернутися до навчання в будь-який час без втрати результатів. Для збереження прогресу буде використовуватися окрема таблиця в базі даних, де кожен користувач матиме свої записи з історією виконаних завдань.

Функціональні вимоги до web-застосунку включають інтерактивні завдання для навчання програмуванню з теоретичним поясненням, структурованість матеріалу, автоматичну перевірку рішень, систему підказок, що допомагає користувачам долати труднощі, а також можливість збереження прогресу, що дозволяє користувачеві відстежувати свої досягнення та ефективно проходити навчальний курс. Всі ці елементи взаємодіють між собою, створюючи зручне та персоналізоване середовище для самостійного вивчення програмування.

Нефункціональні вимоги до web-застосунку для самостійного вивчення програмування охоплюють важливі аспекти та виконують значущу роль у розробці надійного програмного забезпечення, які не стосуються безпосередньо виконуваних функцій, але значно впливають на ефективність роботи системи, безпеку, досвід користувача та можливість масштабування. Для цього застосунку три основні нефункціональні вимоги включають швидкість роботи, безпеку та масштабованість, кожна з яких має свій вплив на загальну якість системи [7].

Швидкість роботи є критично важливою для забезпечення комфортного користувацького досвіду, тому система повинна надавати відповідь на користувацькі дії не більше ніж за 300 мс у 95% випадків. Користувачі очікують, що система буде працювати швидко та реагувати без затримок на їхні запити. Одним із основних способів досягнення цієї мети є оптимізація запитів до бази

даних, адже часто саме це стає вузьким місцем у продуктивності. За допомогою індексації та вибору оптимальних запитів можна забезпечити швидке завантаження необхідних даних. Крім того, велике значення має мінімізація часу завантаження сторінок – менше 1 секунди, що може бути досягнуто через кешування ресурсів і зменшення кількості HTTP запитів до сервера. Усе це допоможе створити зручний і безперебійний досвід користувача.

Масштабованість є важливою вимогою для підтримки належної роботи web-застосунку при збільшенні кількості користувачів або зростанні навантаження на систему. Система повинна бути спроєктована таким чином, щоб мати можливість адаптуватися до змін, не втрачаючи продуктивності та масштабуватись горизонтально без втрати продуктивності при збільшенні кількості одночасних запитів на сервер до 1000. Для цього передбачено використання розподіленої архітектури серверів, що дозволяє рівномірно розподіляти навантаження та зберігати високу доступність сервісу.

Усі дані між користувачем та клієнтом, а також між клієнтом та сервером мають бути зашифровані та використовувати протокол HTTPS. Крім того конфіденційні дані клієнтів, наприклад, пароль до облікового запису повинні зберігатися у зашифрованому вигляді. Також для уникнення витоку інформації з бази даних, система повинна бути захищена від поширених атак, таких як SQL-ін’єкції, XSS та CSRF.

Таким чином, нефункціональні вимоги, такі як швидкість роботи, безпека та масштабованість, створюють основи для стабільної і ефективної роботи web-застосунку, забезпечуючи зручний і безпечний досвід користувачів, а також дозволяючи системі адаптуватися до змінних умов і зростаючого попиту.

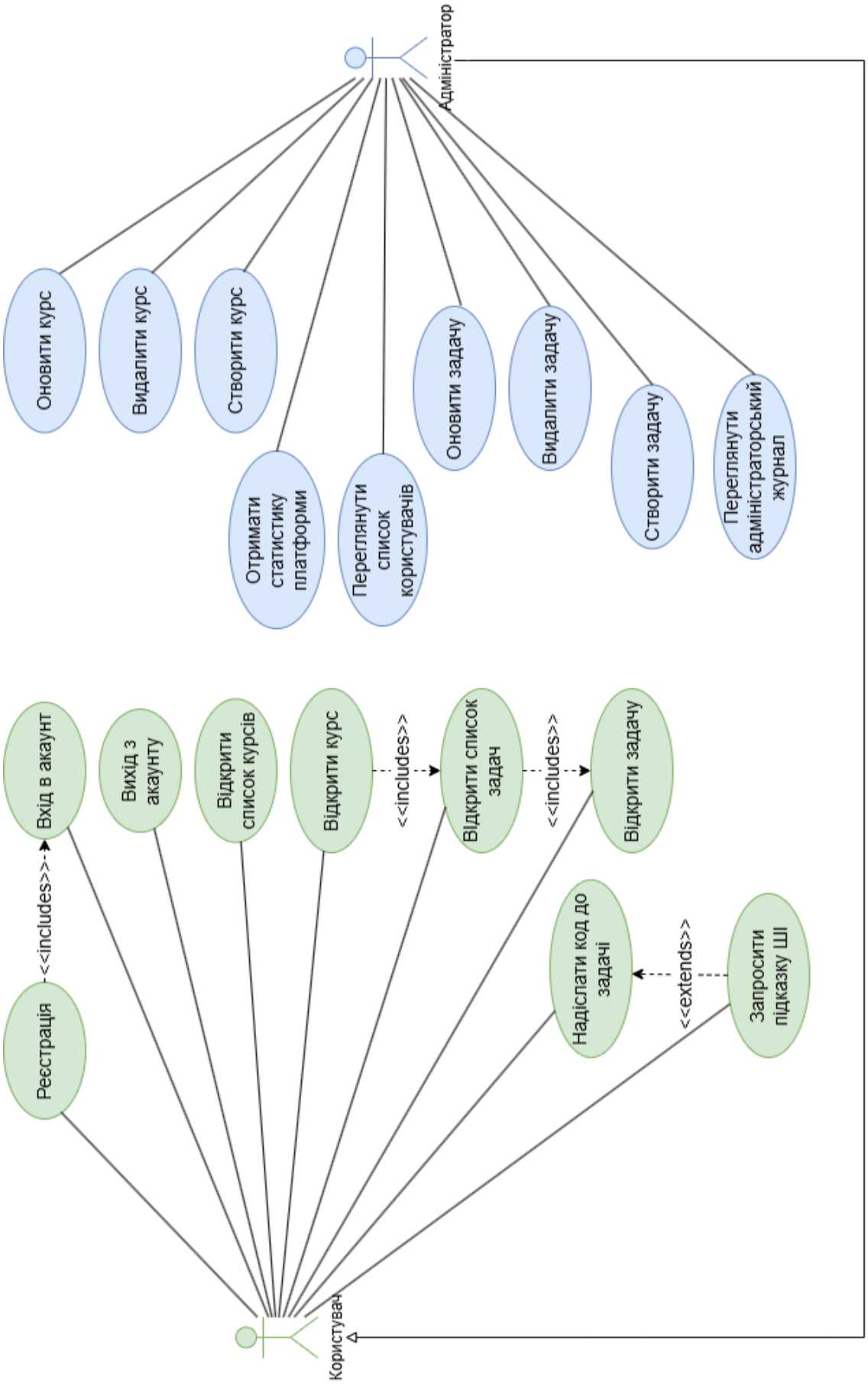


Рис.1.7 Діаграма варіантів використання

Діаграми варіантів використання є важливими інструментами в процесі проектування web-застосунку, оскільки вони дозволяють чітко визначити взаємодію користувачів із системою та виявити необхідні функції для ефективного функціонування застосунку. Ці діаграми є візуальним відображенням того, як користувачі виконують свої дії в системі та як система відповідає на ці дії. В цій діаграмі також є доповнення до зв’язків у вигляді надписів includes та excludes, які позначають, що дія є частиною іншої базової дії або є опціональною після іншої базової дії.

Для web-застосунку, спрямованого на самостійне вивчення програмування, важливо розробити чітке розуміння основних типів користувачів (акторів) та їх взаємодії із застосунком. У цьому випадку основними акторами є користувачі, які проходять навчання, а також сама система, яка надає підказки, оцінює прогрес і пропонує інтерактивні завдання.

На платформі користувачі зможуть реєструватися, а після реєстрації входити в обліковий запис або виходити з нього. Користувачі повинні мати можливість відкривати список курсів, самі курси, списки задач, а також самі задачі. Для виконання задач користувачі можуть надсилати свої результати для перевірки на сервер і отримувати підказки від моделі штучного інтелекту.

В застосунку також є адміністратори, які можуть керувати контентом на платформі, тобто створювати курси, задачі, а також переглядати список користувачів. При тому адміністратори здатні виконувати усі дії, що й інші користувачі.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Мови програмування

Python є однією з найпопулярніших мов програмування для розробки web-застосунків завдяки своїй простоті, читабельності коду та потужній екосистемі бібліотек та фреймворків. Він особливо підходить для швидкої розробки застосунків [8].

Python є інтерпретованою мовою програмування, тобто не потрібно встановлювати окремо компілятор, а також здійснювати за допомогою нього компіляцію проєкту. Необхідно лише передати інтерпретатору скрипт Python, який самостійно обробляє всі інструкції. Це дозволяє набагато швидше розробляти та тестувати застосунок.

Python активно використовується у розробці web-застосунків, які охоплюють як легкі мікросервіси, так і великі складні платформи, а для створення бекенду використовуються популярні фреймворки, серед яких Flask, Django і FastAPI.

Крім того Python має зручні бібліотеки для легкої інтеграції моделей штучного інтелекту у застосунок, такі як openai або transformers. Також ця мова програмування забезпечує широкі можливості у машинному навчанні та обробці даних, що є критично важливим у застосунку з генерацією підказок.

Було розглянуто альтернативні мови, наприклад, Node.js та Go, проте переваги Python у контексті web-застосунку для самостійного навчання роблять його ідеальним вибором.

Для розробки фронтенду web-застосунку для самостійного вивчення програмування використано TypeScript. TypeScript – це надбудова над JavaScript з визначеною сильною типізацією.

Основною перевагою TypeScript є визначені типи, що дозволяє виконати статичні перевірки й уникнути багатьох помилок ще перед самим запуском коду.

Завдяки цьому швидкість розробки збільшується в декілька разів, а також кількість помилок та багів значно зменшується [9, с.2].

Фактично TypeScript має той самий синтаксис і оточення як і JavaScript. Більш того код компілюється у стандартний JavaScript код, вилучаючи типи, звісно за умови, що у коді відсутні синтаксичні помилки. Під час компіляції всі анотації TypeScript прибираються та на виході отримується чистий JavaScript код готовий для виконання у браузері.

2.2 Фреймворки

Flask, мікрофреймворк для Python, створений для забезпечення мінімальних інструментів для web-розробки. Його ключовою особливістю є простота та гнучкість. Flask не нав'язує розробникам чіткої структури проекту, що дає їм повний контроль над організацією коду, маршрутизацією, базами даних та іншими елементами. Через те, що Flask надає тільки базові інструменти для роботи з запитами протоколу HTTP, шаблонами та маршрутизацією, розробники мають можливість вибирати додаткові бібліотеки для функціональності, яка їм потрібна. Це робить Flask гарним вибором для дрібних або середніх проектів. Фреймворк має велику кількість сторонніх розширень для додаткової функціональності: Flask-SQLAlchemy(ORM), Flask-WTF(web-форми), Flask-Login(автентифікація) та інші [10, с.3].

Django — це потужний і повноцінний фреймворк, що надає все необхідне для швидкої розробки web-застосунку. Однією з головних переваг Django є те, що він надає широкий набір інструментів "з коробки". Це включає в себе все: підтримка баз даних через ORM (Object-Relational Mapping), механізми роботи з користувацькими сесіями, аутентифікацією, маршрутизацією, обробкою форм та навіть створення адміністративної панелі для керування контентом web-застосунка. Завдяки цьому розробники можуть зосередитися на розв'язанні конкретних завдань, а не витрачати час на створення базових функціональностей з нуля, як у Flask.

Цей фреймворк підтримує принцип DRY (Don’t Repeat Yourself), що означає, що код не повторюється без необхідності. Це дозволяє уникати дублювання логіки та значно спрощує процес обслуговування і розширення програми.

Вибір між Flask та Django для розробки web-застосунку залежить від ряду факторів, включаючи масштаб проекту, потреби у гнучкості, швидкості розробки, а також досвід команди розробників. Правильний вибір фреймворку може значно вплинути на ефективність роботи та успіх проекту.

FastAPI – це сучасний, ефективний і швидкий фреймворк для будування API мовою Python на основі базових типів мови. Фреймворк набрав популярності через свою швидкість та простоту. FastAPI використовує нативні методи Python для виконання асинхронних операцій та сучасний функціонал мови, що робить його ідеальним для побудування розширюваних API. Фреймворк побудований на основі Starlette для обробки HTTP-запитів і маршрутизації, та використовує Pydantic для автоматичної валідації даних, а також генерує автоматичну документацію Swagger.

На відміну від Flask та Django, FastAPI дозволяє нативно працювати асинхронно за допомогою async/await, що особливо корисно для високонавантажених API та підходить для мікросервісної архітектури.

Підсумовуючи, Django підходить для розробки складних монолітних систем(e-commerce, фінансові платформи). Для простіших проектів з меншою кількістю компонентів або мікросервісних архітектур варто обрати Flask або FastAPI. Зокрема, для розробки web-платформи призначеної для самостійного вивчення програмування ідеальним вибором буде саме FastAPI завдяки своїй швидкості та асинхронності

Для розробки клієнтської частини основні фреймворки – це React, Vue і Angular. React.js є одним із найбільш популярних фреймворків для розробки інтерфейсів користувача. Його головною перевагою є компонентний підхід до розробки, що дозволяє створювати багаторазові елементи інтерфейсу. Також його сильна сторона – це віртуальний DOM, який оптимізує рендерінг сторінок

та підвищує загальну швидкодію застосунку. Цей фреймворк має велику спільноту розробників та дуже активно розвивається, тому екосистема React - це відмінний вибір для розробки фронтенд частини web-застосунку [11, с.1713].

Vue.js є легшим та гнучкішим фреймворком для розробки інтерфейсів та SPA. Це новіший та більш прогресивний фреймворк, який поєднує сильні сторони React та Angular пропонуючи простоту та адаптивність. Vue також використовує компонентний підхід, але до того ж пропонує реактивне зв’язування даних, що спрощує оновлення інтерфейсів на основі змін даних у застосунку. Перевага цього фреймворку – це низький поріг входу, що дозволяє використовувати потужний функціонал із самого старту і швидко розробляти застосунок.

Angular, хоча й є менш популярним у порівнянні з React та Vue, є більш комплексним фреймворком, який включає в собі функціонал для роботи з маршрутизацією, управлінням стану застосунку, HTTP сервісами та іншим. На відміну від Vue та React, які використовують JavaScript за замовчуванням, Angular повністю побудований на TypeScript, що додатково включає усі переваги типізації. Цей фреймворк є ідеальним вибором для великих web-застосунків, оскільки забезпечує зручні методи для побудови проєктів із великою та складною структурою [12, с.936].

Для розробки web-застосунку для самостійного навчання програмуванню було обрано React, тому що він надає баланс між швидкістю розробки, складністю, продуктивністю та функціоналом.

2.3 Бази даних

Вибір бази даних для зберігання даних в web-застосунку залежить від вимог до масштабованості, продуктивності та складності структури даних. Для web-застосунку для навчання програмуванню, де потрібно зберігати завдання, відповіді користувачів, їхню статистику та прогрес, варто з самого початку

розглянути вибір між реляційними(SQL) та нереляційними(NoSQL) базами даних.

SQL-бази даних є традиційними та використовують структуровану схему для зберігання даних у вигляді таблиць з чітко визначеними зв'язками між ними. Вони є оптимальними для застосунків, де дані мають чітку структуру, наприклад, інформація про користувачів, курси, завдання, прогрес користувачів тощо. Основними перевагами SQL є надійність, можливість виконання складних запитів з використанням JOIN, а також здатність до масштабування за допомогою реплікації та розподілу навантаження.

З іншого боку, NoSQL бази даних більше підходять для застосунків, які працюють із великою кількістю нерелевантних або структурованих даних, з можливістю гнучкого управління схемами та масштабуванням на великі обсяги даних. NoSQL-системи, такі як MongoDB, Redis або Cassandra, забезпечують високу швидкість запису та зчитування, що може бути корисно, якщо застосунок планує обробляти великий обсяг запитів чи великі обсяги даних у реальному часі.

Для web-застосунку для самостійного навчання програмування основними факторами вибору є тип даних та обсяг запитів. Оскільки застосунок потребує зберігання структурованої інформації (курси, користувачі, завдання, прогрес), SQL є логічним вибором завдяки своїй підтримці складних зв'язків та запитів. Наприклад, таблиця завдань пов'язана з таблицею курсів і прогресом, що в свою чергу пов’язаний з користувачами.

Вибір SQL-системи, такої як MySQL, PostgreSQL або SQLite, є доцільним для цього типу застосунка, оскільки вони пропонують високу надійність, підтримку транзакцій, а також можливість використовувати складні SQL-запити для отримання статистики або аналізу прогресу користувачів. Крім того, реляційні бази даних мають добре розвинену екосистему та документацію, що забезпечує зручність розробки та інтеграції з іншими компонентами web-застосунку.

MySQL є потужною реляційною базою даних, яка підтримує складні запити, транзакції та має великий набір функцій для роботи з великими обсягами даних. Це ідеальний вибір для складних проектів, які потребують високої надійності, масштабованості та можливості обробляти великі обсяги даних. MySQL також надає можливості для роботи з різними типами даних, що дає змогу організувати зберігання завдань, відповідей, статистики та іншої інформації максимально ефективно.

PostgreSQL – це об’єктно реляційна база даних, сильними сторонами якої є швидкість та розширюваність. Так само як і MySQL може обробляти складні запити, транзакції і ефективно працювати з великими наборами даних. Найголовніша різниця між ними полягає у тому, що PostgreSQL є популярнішою, розробляється швидше, звідси останні версії пропонують більше сучасних методів для ефективного будування баз даних та їх використання. Крім того, PostgreSQL має низьке споживання ресурсів у режимі очікування. PostgreSQL також підтримує складні типи (масиви, JSONB), що актуально для складних структур даних.

SQLite, з іншого боку, є вбудованою базою даних, яка не потребує окремого серверного процесу. Це ідеальне рішення для невеликих або середніх проектів, де не потрібно обробляти величезні обсяги даних або працювати з високими навантаженнями. SQLite зручний для простих застосунків, де важлива швидкість розробки і відсутність складної інфраструктури. Однак для більш масштабних і складних проектів, SQLite може бути недостатньо потужним.

Для web-застосунку з інтерактивними завданнями та підказками на основі ШІ, PostgreSQL буде оптимальним вибором завдяки своїй масштабованості, здатності ефективно працювати з великими наборами даних та підтримці сучасних функцій для складної аналітики та зберігання структурованих даних.

2.4 Моделі штучного інтелекту

Інтеграція штучного інтелекту у web-застосунки стала популярним та важливим напрямком для багатьох сучасних систем, особливо коли мова йде про створення інтерактивних платформ з адаптивним контентом. Найбільш популярний метод – це використання готових API, які пропонують простий користувацький інтерфейс для комунікації з моделлю штучного інтелекту, наприклад, OpenAI, Deepseek, Gemini та ін. Звісно для максимізації ефективності моделі варто її будувати самостійно з нуля, проте це дуже довгий та дорогий процес.

OpenAI пропонує декілька GPT(generative pre-trained) моделей, які об'єднують у собі високу швидкість та новітній функціонал. До того ж це один з найперших продуктів, який з'явився на ринку штучного інтелекту. Ці моделі спеціалізуються на обробці тексту, тобто отримання інструкцій та генерація відповіді у текстовому вигляді, що робить його найзручнішим для інтеграції у web-застосунок [13, с.7].

DeepSeek – це відносно нова модель, яка фактично є заміною моделей від OpenAI та має відкритий код. Вона використовує лише деяку частину параметрів під час розрахунку, на відміну від GPT моделей, які використовують усі параметри, що потребує більше ресурсів. Як висновок: DeepSeek є дешевшим варіантом, проте може надавати менш точні відповіді та працювати повільніше.

Gemini від Google також дуже схожий на дві попередні моделі, проте має деякі проблеми з оптимізацією та не підтримує такого ж широкого функціоналу для роботи з текстом як GPT моделі. Не дивлячись на переваги Gemini у обробці інформації у різних форматах, він поступається своєю точністю та швидкістю іншим моделям.

У випадку web-застосунку для самостійного навчання програмуванню було обрано OpenAI, а саме модель gpt-3.5-turbo через її надійність, потужні можливості, швидкість та популярність.

2.5 Середовища розробки

Середовище розробки є критичним компонентом, який повинен бути обраний ще перед самим створенням програмного забезпечення, оскільки це впливає на швидкість всього процесу. Ідеальне середовище розробки для проєкту повинне підтримувати роботу з обраними технологіями(Python і TypeScript), мати обширний вбудований функціонал або ж якісні розширення для написання, запуску та тестування коду. Найпопулярніші та найсучасніші рішення для цього є Visual Studio Code та JetBrains Fleet.

Visual Studio Code – це один з найпопулярніших редакторів коду, проте через величезну підтримку від розробників та ентузіастів з усього світу цей застосунок об’єднує в собі багато розширень, які надають підтримку до усіх сучасних технологій та перетворює звичайний редактор на повноцінне IDE.

Навіть без встановлення розширень VS Code має нативну підтримку багатьох мов програмування, що для більшості потреб є достатнім. Головна його перевага – це те, що фундаментально він залишається звичайним редактором, що робить його дуже швидким для роботи з проєктами будь-якого розміру.

Компанія JetBrains концентрується на створенні застосунків для розробників. Саме тому їх рішення є дуже популярними, оскільки вони пропонують найновіший функціонал та підтримують усі необхідні технології для роботи з певними оточеннями.

Fleet – це один з продуктів компанії JetBrains. Fleet підтримує більшість сучасних мов програмування, включаючи Python та TypeScript, а також пропонує вбудовану допомогу на основі штучного інтелекту. На відміну від інших продуктів від JetBrains, наприклад, PyCharm, Fleet побудований із мінімалістичним інтерфейсом і розділеною архітектурою, що робить його швидшим ніж його попередники. Проте зараз цей застосунок знаходиться ще на стадії публічного тестування, тому він не є настільки ж надійним як Visual Studio Code.

Середовище Visual Studio Code було обрано для розробки web-застосунку через його надійність, швидкість та широкий функціонал.

3 ПРОЄКТУВАННЯ СИСТЕМИ WEB-ЗАСТОСУНКУ ДЛЯ САМОСТІЙНОГО НАВЧАННЯ ПРОГРАМУВАННЮ

3.1 Загальна архітектура системи

При розробці web-застосунку для самостійного навчання програмуванню необхідно обрати оптимальну архітектуру, яка забезпечить масштабованість, надійність та можливість подальшого розвитку.

Серед можливих варіантів архітектури розглядаються такі підходи:

Монолітна архітектура — усі компоненти web-застосунку (бекенд, фронтенд, база даних) працюють у єдиному середовищі. Підходить для невеликих проєктів, але має проблеми зі масштабованістю.

Клієнт-серверна архітектура — класичний підхід, де клієнтська частина (фронтенд) взаємодіє з серверною частиною (бекендом) через API. Забезпечує гнучкість і розділення логіки.

Мікросервісна архітектура — кожен компонент системи реалізується у вигляді окремого сервісу. Це дозволяє легко масштабувати систему, але ускладнює управління.

З огляду на вимоги до системи (інтерактивність, швидка взаємодія користувача з бекендом, масштабованість) було обрано клієнт-серверну архітектуру.

Клієнт-серверна архітектура є основою для багатьох сучасних web-застосунків, включаючи ті, що побудовані з використанням технологій, таких як TypeScript і React для фронтенду та Python з FastAPI для бекенду. Цей підхід забезпечує чітке розділення логіки між клієнтською і серверною частинами, що дозволяє ефективно керувати великими і складними проектами [14, с.401].

В цій архітектурі фронтенд виконує роль посередника між користувачем та сервером, а його обов’язки включають: обробку подій, візуалізацію даних та відправку запитів на сервер. Бекенд, в свою чергу, забезпечую обробку бізнес-логіки, роботу з базою даних та виконання складних обчислень. Кожен

компонент зосереджений на своїх функціях, що спрощує підтримку та масштабування системи.

Іншою важливою перевагою є легка інтеграція API. Клієнт-серверна архітектура дозволяє ефективно налаштовувати взаємодію між компонентами через стандартизовані інтерфейси, такі як RESTful API або GraphQL. Це дає змогу фронтенду та бекенду працювати незалежно один від одного, що полегшує оновлення, зміну технологій або додавання нових функцій без необхідності значних змін у всій системі. API служить посередником, що дозволяє безшовно обмінюватися даними між клієнтським інтерфейсом та серверною логікою [15, с.807].

Обмін даними між клієнтом та сервером буде відбуватися через протокол REST. Це рішення дозволяє використовувати прості HTTP запити та описувати дані через JSON. Цей протокол є одним з найпоширеніших через свою простоту, високу швидкість та легкість в реалізації. Це рішення ідеально підходить для проєктів різних масштабів з мікросервісною або клінт-серверною архітектурою [16].

Оскільки клієнт та сервер фактично незалежні один від одного та спілкуються через визначений інтерфейс, кожен компонент може вільно використовувати будь-які технології, які підходять найкраще. Це також надає гнучкість для ефективної розробки та роботи системи.

Окрім того, клієнт-серверна архітектура забезпечує можливість подальшого розширення проекту. В разі необхідності масштабувати систему або додавати нові функціональні можливості, можна перейти на мікросервісну архітектуру.

Загалом, клієнт-серверна архітектура є високоефективним та гнучким підходом, що дозволяє розробникам зосереджуватися на окремих аспектах системи, забезпечуючи високий рівень масштабованості, інтеграції та підтримки.

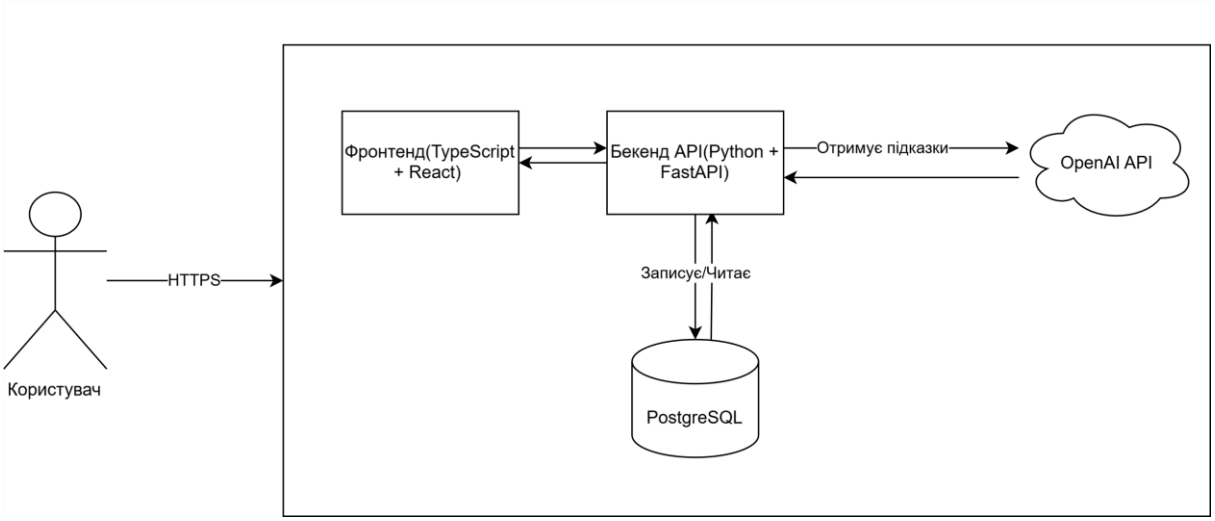


Рис.3.1 Діаграма архітектури

Технологічний стек

— Фронтенд: TypeScript (React)

— Бекенд: Python (FastAPI)

— База даних: PostgreSQL

— Протокол взаємодії: HTTPS

Таким чином, клієнт-серверний підхід забезпечує оптимальну гнучкість і простоту в реалізації, що робить його найкращим вибором для цього проєкту.

3.2 Клієнтська частина (Frontend)

Для реалізації клієнтської частини обрано React.js, оскільки цей фреймворк дозволяє створювати інтерактивні компоненти і зручно працювати з динамічними даними. React також забезпечує ефективне оновлення інтерфейсу завдяки віртуальному DOM та підтримує модульний підхід у розробці, що спрощує подальше розширення функціоналу системи.

Для реалізації комунікації між клієнтом та API через HTTP на сервері буде використана бібліотека Axios, яка є обгорткою над fetch та підтримує перехоплення запитів, обробку помилок, тайм-аути тощо. Ця бібліотека дозволяє асинхронно отримувати дані з серверу.

Стилізація має бути виконана за допомогою Tailwind CSS, яка дозволить використовувати допоміжні класи для уніфікованого підходу до оформлення.

Процеси взаємодії користувачів із web-застосунком можуть бути доволі складними, оскільки потрібно продумати, які саме дії буде робити користувач та як на них реагуватимуть компоненти, а також спілкування користувача з клієнтом, і клієнту з сервером. Для цього можна використати діаграму послідовності. Такі діаграми відображають процеси і сутності, а також повідомлення, якими обмінюються актори для відтворення певної функціональності.

На рисунку 3.2 показано діаграму послідовності, яка описує процес реєстрації, входу та виходу в систему зі сторони користувача. Клієнтська частина перевіряє дані введені користувачем та після цього надсилає серверу і опрацьовує відповідь, наприклад, повідомлення про помилку або ж успішну реєстрацію з токеном. Використання токенів для автентифікації дозволяє безпечно працювати з сесіями користувачів.

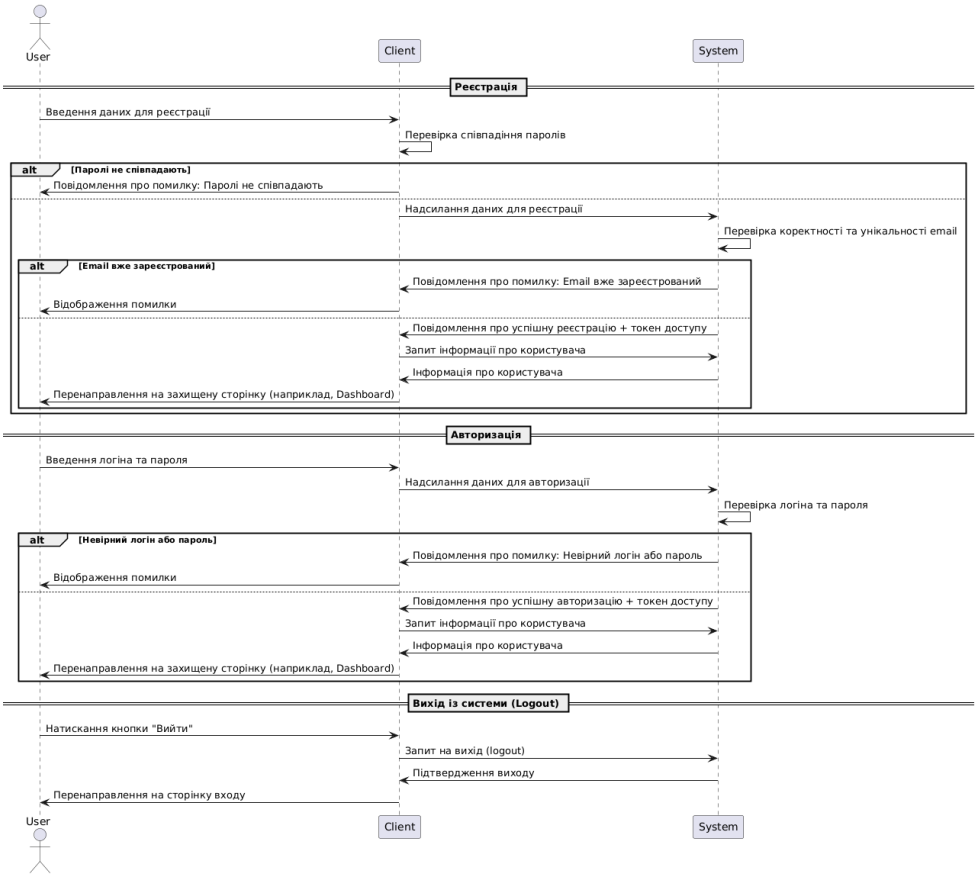


Рис.3.2 Діаграма послідовності реєстрації та авторизації

Діаграма ілюстрована на рисунку 3.3 відображає як користувач вибирає задачі, надсилає код та опціонально отримує підказку від штучного інтелекту. Після надсилання коду користувач отримує негайну відповідь щодо коректності рішення, некоректності або ж помилки під час перевірки. Якщо користувач не може виконати завдання, він може запросити підказку від штучного інтелекту, де клієнт відправить запит серверу, який у свою чергу комунікує з моделлю штучного інтелекту.

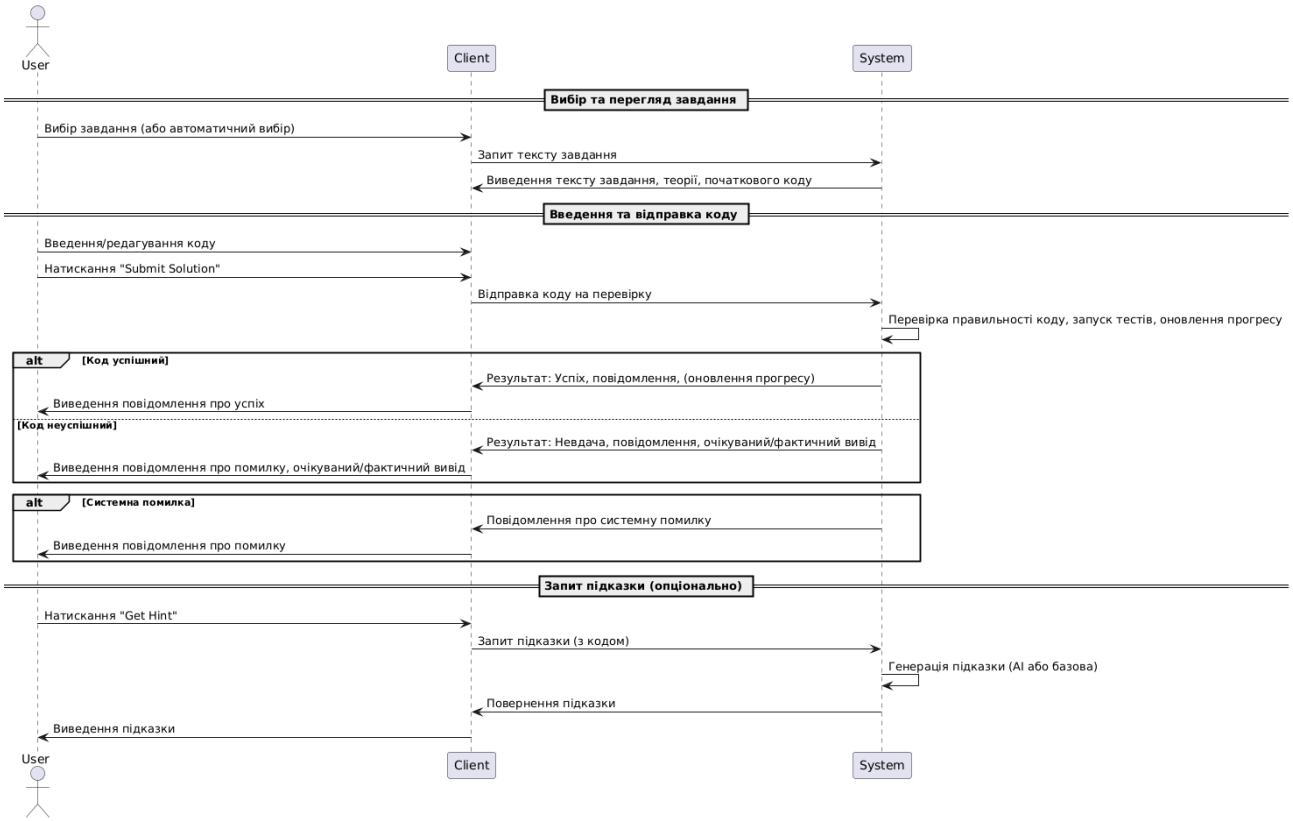


Рис.3.3 Діаграма послідовності роботи із задачами

3.3 Серверна частина (Backend)

Основою серверної частини є FastAPI, сучасний фреймворк на мові Python, який забезпечує високу швидкодію завдяки асинхронній обробці запитів і підтримці OpenAPI для автоматичної генерації документації. FastAPI дозволяє швидко розробляти API, необхідний для взаємодії клієнта з сервером, а також

підтримує інтеграцію з ORM-бібліотекою SQLAlchemy, яка використовується для роботи з базою даних.

Основна взаємодія в системі здійснюється через API (створене на базі FastAPI). Сервер, на основі отриманих даних з HTTP запитів створених клієнтом, виконує необхідну логіку і повертає результат (наприклад, підказки до завдання або підтвердження правильності відповіді).

FastAPI забезпечує з'єднання з PostgreSQL через ORM SQLAlchemy, що дозволяє серверу зберігати та отримувати дані з бази даних. Кожен користувач має свій обліковий запис, що містить інформацію про виконані завдання і прогреси по курсам. Кожне завдання зберігається в окремій таблиці в базі даних разом та прив'язане до курсу.

Коли користувач проходить завдання або зберігає свій прогрес, сервер здійснює запити до бази даних для оновлення інформації. Це дозволяє відстежувати прогрес користувача в реальному часі і надавати персоналізовані рекомендації чи підказки.

Для забезпечення динамічної підтримки користувача в процесі навчання вбудовуються модулі штучного інтелекту. Сервер може відправляти дані на аналіз штучним інтелектом, щоб отримати підказки або поради щодо виконання завдань.

Наприклад, коли користувач виконує програмування певної задачі, сервер може аналізувати написаний код за допомогою моделей ШІ, а потім надсилати відповідні рекомендації або підказки, щоб допомогти йому виправити помилки. Ця взаємодія дозволяє підвищити точність та якість навчання користувача.

Взаємодія між усіма цими компонентами забезпечує ефективну і гнучку архітектуру web-застосунку. Всі запити від користувача, включаючи запити до бази даних та взаємодію з AI, проходять через сервер, що гарантує централізовану обробку даних та зручну інтеграцію між усіма компонентами.

Для ефективного оперування сутностями всередині застосунку такими як користувачі або ж задачі потрібно створити класи, які будуть абстракціями над цими сутностями. Ці класи будуть зберігати в собі усі необхідні поля пов'язані з

ними. UML-діаграма класів описує структуру класів у системі та їх взаємодію. Вона показує, які класи існують у програмі, які їхні атрибути та методи, а також зв'язки між ними. На рисунку 3.4 можна побачити, як об'єкти взаємодіють і як організовані атрибути.

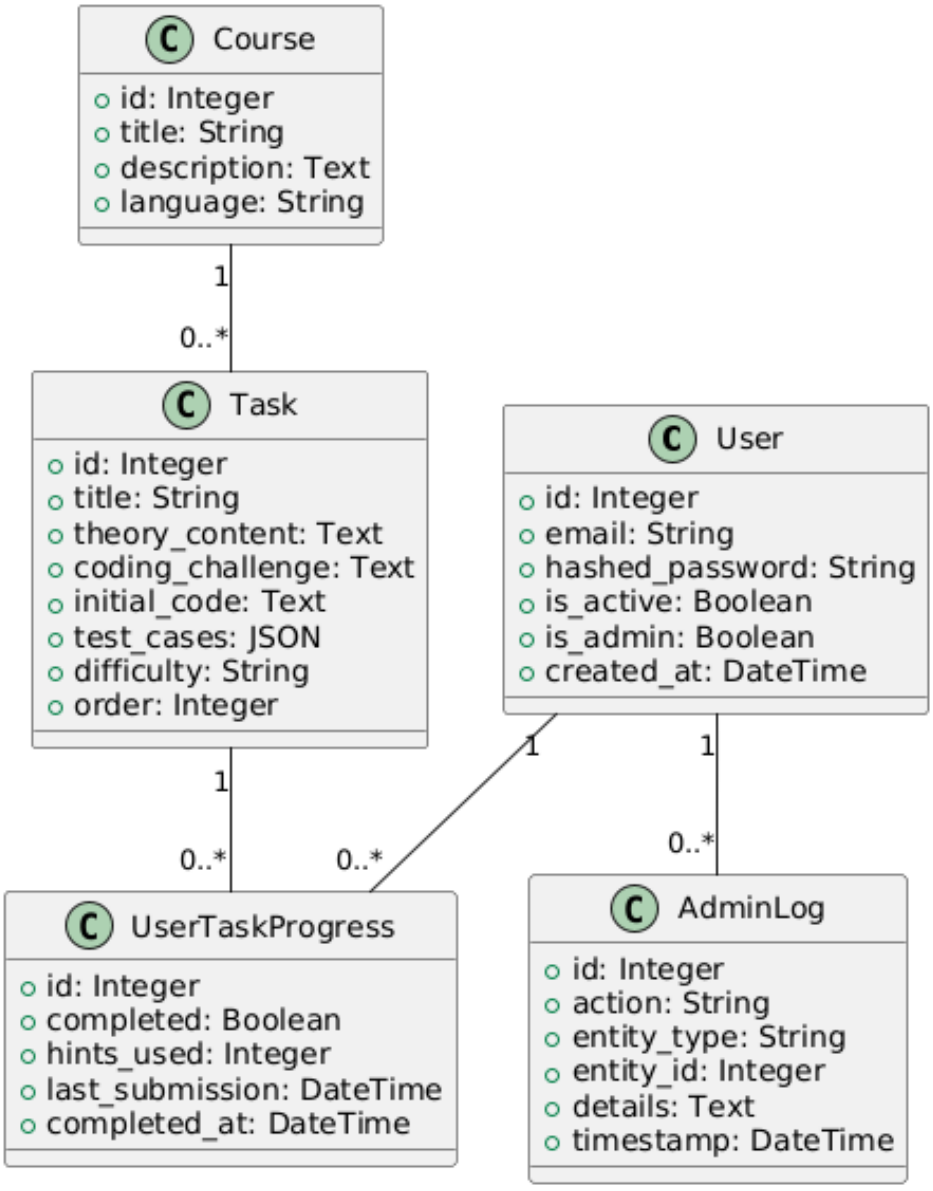


Рис.3.4 UML-діаграма класів

Клас Course – це абстракція над сутністю курсів. В нього є заголовок, короткий опис, а також вказана мова програмування, яка вивчається на цьому курсі.

Клас курс напряму пов’язаний з сутністю Task. Ця сутність має в собі посилання на відповідний курс, а також заголовок завдання, теоретичний матеріал, суть задачі, тести, а також складність.

Сутність User зберігає в собі дані користувача, а саме – захешований пароль, електронну пошту, дату створення, чи активний акаунт і чи користувач є адміністратором. Якщо користувач є адміністратором, то для нього буде доступна особлива панель управління платформою, в якій він може керувати завданнями, курсами та переглядати список користувачів. Будь-яка дія у адміністраторській платформі створить запис у вигляді сутності AdminLog для аудиту.

Прогрес користувача зберігається в окремій сутності UserTaskProgress, яка посилається на певного користувача і на певну задачу. Завдяки такій гнучкості можна ефективно відслідковувати прогрес та відображати його на платформі.

З діаграми класів видно, що фактично усі сутності тісно пов’язані один з одним. До того ж між ними наявні такі зв’язки як один-до-багатьох і багато-до-багатьох.

3.4 Моделі бази даних

Структура бази даних у web-застосунку включає кілька основних сутностей: користувач (User), навчальні завдання (Task), курс (Course), прогрес користувача (UserTaskProgress), а також логування дій адміністраторів (AdminLog). Перераховані сутності дозволяють ефективно зберігати навчальний матеріал, користувачів та їх прогрес, і бачити дії адміністраторів (рисунок 3.5).

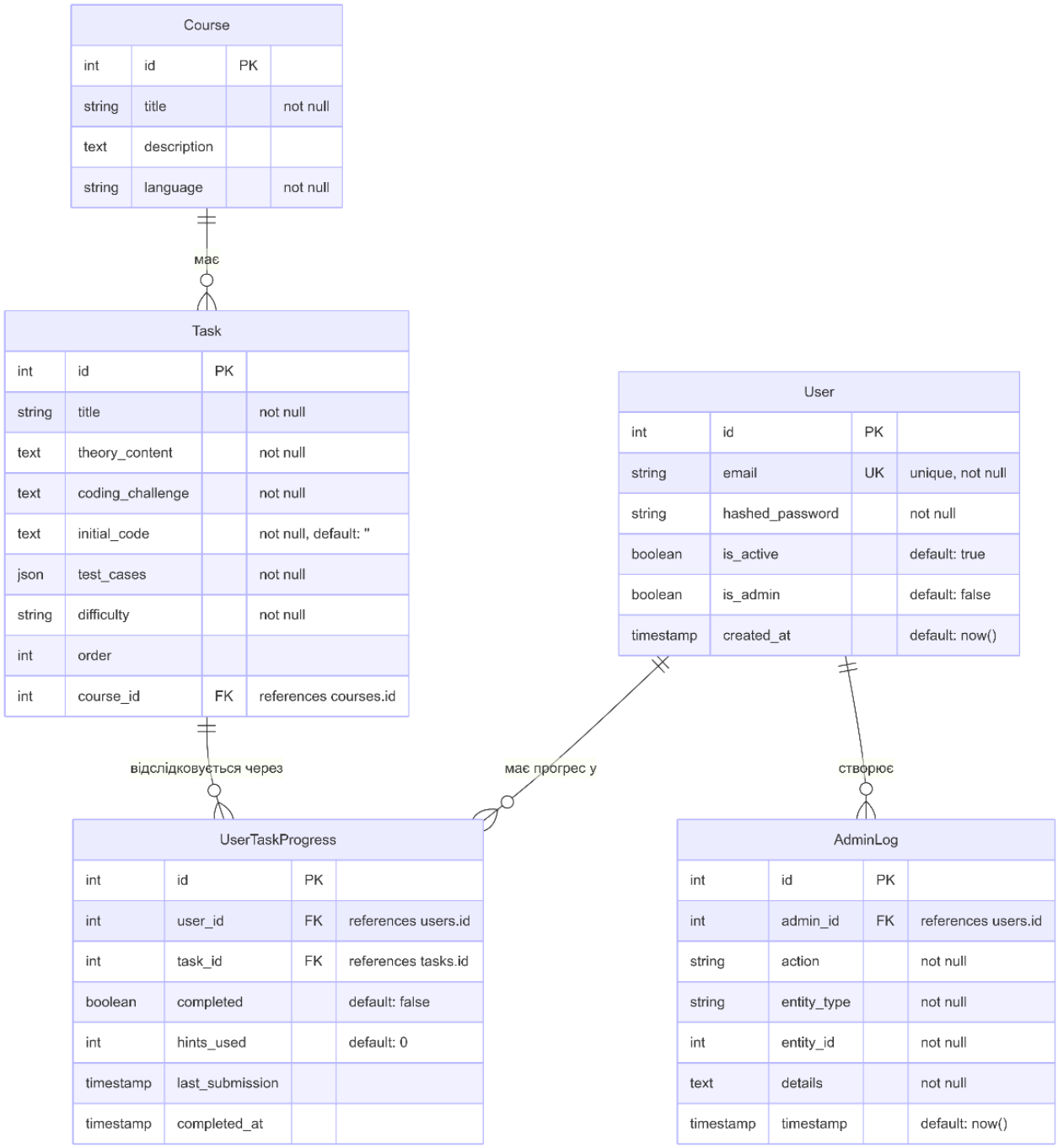


Рис.3.5 ER-діаграма структури бази даних

Модель збереження даних є важливою складовою архітектури системи для web-застосунку, оскільки вона визначає структуру бази даних і способи взаємодії з нею. У випадку web-застосунку для самостійного вивчення програмування важливо мати чітке уявлення про те, як зберігати та організовувати дані щодо користувачів, прогресу навчання, завдань та результатів.

Кожен користувач буде мати окремий профіль у системі, який містить основну інформацію, таку як ім’я, електронну пошту, пароль (захищений хешуванням) та інші налаштування, пов’язані з його акаунтом. Для цього створюється таблиця Users, де кожен користувач буде унікальним через електронну пошту. Крім того, якщо користувач також є адміністратором, то його дії будуть записані і будуть створюватися записи у таблицю AdminLog.

Прогрес кожного користувача буде зберігатися в окремій таблиці UserTaskProgress, яка міститиме інформацію про виконані завдання, час та кількість використаних підказок. Зокрема, це включає відслідковування, які завдання користувач уже виконав, які завдання ще потрібно пройти, та загальний результат проходження курсу. Прогрес буде прив’язаний до певного користувача та певної задачі.

Завдання є ключовими елементами навчання, і для їх збереження створюється таблиця Tasks, яка містить теоретичний матеріал, текст завдання, рівень складності, вказівки та початковий код. Кожне завдання має унікальний ідентифікатор, який дозволяє прив’язати його до конкретного курсу та відображати в інтерфейсі користувача.

Таблиця Courses буде зберігати курси – сутність, яка об’єднує в собі 1 або більше задачі, які безпосередньо належать до нього. Користувач зможе вибирати певні курси, бачити задачі, а також переглядати загальний прогрес по курсам.

3.5 Безпека та автентифікація

Безпека є ще одним важливим аспектом, адже застосунок працює з персональними даними користувачів, що потребують належного захисту. Для забезпечення безпеки будуть застосовуватися різноманітні механізми захисту, зокрема шифрування даних, таких як паролі та інша чутлива інформація. Для цього буде використовуватися алгоритм bcrypt, який є одним з найбільш ефективних для збереження паролів у безпечному вигляді. Крім того, впроваджено систему аутентифікації та авторизації, що забезпечує доступ тільки

до особистої інформації користувача за умови підтвердження його особи. Для цього буде використовуватися JWT-токени.

JWT-токени будуть зберігатися за допомогою httpOnly кукі, які пропонують додатковий захист проти XSS атак та обмежують доступ до змісту кукі користувацькому коду. До того ж через встановлення додаткових налаштувань таких як SameSite, Secure та Path можна значно зміцнити шар захисту користувацьких даних та уникнути викрадення цих даних зловмисниками [17].

Доступ контролю на основі ролей(RBAC) також є обов'язковою частиною застосунку для того, щоб автоматично розрізняти звичайних користувачів та адміністраторів. Шляхи на клієнтській частині та серверній захищені на основі ролей, щоб лише авторизовані користувачі мали доступ до них.

Захист від неавторизованих команд до серверу має бути захищений за допомогою механізмів CSRF, які дають можливість згенерувати токен та прикріпити його до користувацьких сесій. Клієнт повинен прикріплювати токен до необхідних запитів, а сервер буде їх перевіряти для уникнення поширених web-атак.

Для забезпечення максимального рівня захисту потрібно також встановити додаткові захисні HTTP хедери, які дозволять уникнути MIME підслуховування, увімкнути вбудовану фільтрацію XSS у браузері та захистити ресурси від недовірених доменів.

3.6 Інтерфейс користувача

Основні принципи UX/UI для навчальних платформ є ключовими для забезпечення зручного та ефективного досвіду користувача під час навчання. Вони спрямовані на створення інтерфейсу, який відповідає потребам користувачів та оптимізує процес взаємодії . Інтерфейс має бути інтуїтивно зрозумілим, простим у використанні та привабливим для того, щоб не лише

забезпечити безперешкодне виконання навчальних завдань, але й створити комфортне середовище для користувачів з різним рівнем досвіду.

Для навчальних платформ важливим є забезпечення простоти та ефективності, що дозволяє користувачам зосереджуватись на навчанні без зайвих зволікань. Один із основних принципів UX/UI для таких платформ — це надання чіткої навігації, яка дозволяє швидко знаходити потрібну інформацію або перейти до необхідних розділів. Користувач повинен відразу розуміти, як працює система та де знайти матеріали для навчання, завдання чи прогрес [18, с.324].

Інтерактивність також відіграє важливу роль. Користувачі мають бути активними учасниками процесу навчання, тому важливо передбачити елементи, які стимулюють їх до взаємодії з платформою. Це можуть бути завдання з покроковими підказками, тестування або вправи, які надають миттєвий зворотний зв’язок. Важливо, щоб кожна дія користувача мала чітке пояснення і результат, що підтримує мотивацію навчатися.

З точки зору візуального дизайну, важливо використовувати мінімалістичний стиль, який не відволікає увагу від основного контенту. Вибір кольорів, шрифтів та елементів інтерфейсу має сприяти комфортному читанню та простоті навігації. Колірна схема повинна бути збалансованою, з використанням контрастних кольорів для виділення важливих елементів, таких як кнопки, індикатори прогресу чи повідомлення.

Нарешті, зворотний зв'язок та підтримка користувачів є важливими для кожної навчальної платформи. Система має надавати користувачеві чіткі повідомлення про його прогрес, досягнення та результати. Крім того, інтеграція підказок та допомоги у реальному часі дозволяє користувачам не відчувати себе загубленими під час навчання [19].

Таким чином, принципи UX/UI для навчальних платформ спрямовані на забезпечення максимально ефективного та комфортного досвіду для користувачів, дозволяючи їм зосередитись на процесі навчання і досягненні результатів, не відволікаючись на технічні або навігаційні труднощі.

Дизайн сторінок web-застосунку для самостійного вивчення програмування має бути орієнтований на забезпечення зручності та ефективності у використанні, водночас не відволікаючи користувачів від навчального процесу. Кожна сторінка повинна виконувати конкретну роль у навчальному процесі та бути інтуїтивно зрозумілою.

Головна сторінка — це перше місце, з яким користувач стикається після входу в систему, тому вона повинна бути максимально простою та інформативною. Головне завдання цієї сторінки — орієнтувати користувача на те, що він може робити далі: обирати курс, починати нове навчання або переглядати поточний прогрес. Для цього важливо наявність чіткої навігації та видимих кнопок для доступу до основних функцій. Візуально головна сторінка повинна бути зручно організована, з акцентом на найбільш важливі елементи — такі як список доступних курсів, кнопки для реєстрації та авторизації. Важливо, щоб дизайн був легким та не перенавантажував користувача зайвою інформацією, а також забезпечував адаптивність для різних пристроїв.

Сторінка редактора коду є центральним елементом для виконання завдань. Тут користувач повинен мати можливість писати, редагувати та тестувати код. Дизайн цієї сторінки має бути спрямований на забезпечення зручності для програмування. Вона повинна містити великий текстовий редактор, де зручно писати код, а також можливість взаємодіяти з підказками та повідомленнями про помилки. Важливо забезпечити підтримку кольорового виділення синтаксису для різних мов програмування, що допоможе користувачам легше орієнтуватися в коді. Додатково, панелі з інструментами, такими як кнопки для запуску коду, тестування або очищення, повинні бути розташовані в такій формі, щоб користувачі могли зручно їх використовувати, не відволікаючись від основної роботи.

Усі ці сторінки мають бути організовані таким чином, щоб не лише надавати всю необхідну інформацію, а й робити це у зрозумілій, приємній для користувача формі.

Важливою частиною дизайну є забезпечення гармонійної взаємодії між текстом, графікою та елементами інтерфейсу, щоб кожен з елементів сторінки допомагав досягненню основної мети — ефективного навчання.

Діаграми взаємодії користувача з системою відображають послідовність кроків, які користувач здійснює під час взаємодії із застосунком, а також реакцію системи на ці дії. Вони допомагають розуміти, як система реагує на певні запити та як забезпечується безперервність процесу, що важливо для проектування ефективного інтерфейсу та взаємодії.

В рамках web-застосунку для самостійного вивчення програмування можна побудувати кілька діаграм взаємодії, які відображатимуть основні дії користувача, такі як реєстрація, вибір курсу, проходження завдань та перегляд результатів.

4 РЕАЛІЗАЦІЯ СИСТЕМИ ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

4.1 Реалізація клієнтської частини (Frontend)

Першим кроком у розробці фронтенду за допомогою React є його коректна ініціалізація, щоб створилися уся необхідна структура файлів. Для цього було використано утиліту Create React App (CRA). Це рекомендований метод, оскільки він пропонує стандартизований підхід до створення якісного оточення без попередніх конфігурацій та абстрагує складні налаштування таких інструментів як Webpack і Babel. Крім того CRA пропонує зручні функції для розробки, наприклад, автоматичне оновлення застосунку, коли змінився код, тобто без перезапуску застосунку можна побачити ефекти змін.

Створення конфігурацій для роботи з TypeScript також робиться через CRA через передавання аргументу шаблону. Це створює файли для того, щоб TypeScript файли включалися під час збірки проєкту.

Webpack виконує роль пакувальника (bundler) для застосунку, він збирає вихідні файли разом з усіма залежностями в повноцінний оптимізований застосунок, транскомпілює TypeScript файли, обробляє файли стилів Tailwind CSS через PostCSS, а також включає підстановку модулів при зміні у вихідних файлах [20, с.176-177].

Усі вихідні файли зберігаються у папці src, яка поділена на інші директорії та файли. Головним файлом виступає index.tsx, який фактично є вхідною точкою до застосунку. Він ініціалізує React проєкт та монтує кореневий компонент в DOM, а також встановлює контекст відображення.

React побудований на основі компонентів, і головним компонентом є App, який відповідає за загальне компонування (layout) застосунку та маршрутизацію і навігацію. Глобальні стилі описуються у файлі index.css, які будують фундаментальний вигляд для усіх сторінок.

Решта елементів клієнту як і основні в React також побудовані на основі модульних компонентів. Папка components зберігає у собі такі елементи

інтерфейсу як Navbar та TaskView, які можуть бути перевикористані у різних частинах застосунку, що робить код читабельним та більш організованим. Директорія pages зберігає компоненти сторінок, які відповідають певним маршрутам у web-застосунку, наприклад, Home або Dashboard. Таке розділення компонентів дозволяє будувати складні сторінки використовуючи прості компоненти.

Файли для керування станом застосунку зберігаються у окремій директорії contexts, яка має контекстні React провайдери, що контролюють глобальний стан застосунку. Це дозволяє мати чітке бачення того як дані проходять через різні частини застосунку.

Бізнес-логіка зберігається у файлах в директорії services, яка фактично відділяє шар сервісів від шару інтерфейсу.

Конфігурація клієнта для спілкування з API налаштовується у окремій папці api. В ній вказується уся інформація: куди клієнт має надсилати запити, у якому форматі, з яким токеном, а також правила обробки помилок запитів. Це також відділяє межі відповідальності між іншими компонентами.

Для реалізації зв'язку між фронтендом і бекендом використовуються стандартні HTTP методи:

1. GET — використовується для отримання даних з сервера, наприклад, для завантаження курсів, отримання результатів виконання завдань або історії навчання користувача.
2. POST — використовується для надсилання даних на сервер, наприклад, для реєстрації користувача, виконання завдання або збереження прогресу.
3. PUT/PATCH — використовуються для оновлення існуючих даних на сервері, наприклад, для оновлення результатів виконання завдань.
4. DELETE — використовується для видалення даних, наприклад, для видалення облікового запису користувача або пов’язаних із ним даних про прогрес.

Для виконання практичних завдань у web-застосунок вбудований редактор коду Monaco Editor. Цей редактор надає користувачу відчуття роботи в реальній

IDE завдяки підсвічуванню синтаксису та перевірки написаного. Одна з його переваг – це можливість гнучкого налаштування та підтримка багатьох мов програмування, таких як Java, JavaScript, Python, C++ та багато інших.

Крім того редактор коду Monaco дуже швидкий та оптимізований, що є необхідною потребою, оскільки кожна інтеграція та елемент створюють додаткове навантаження на сторінку.

Уніфікована структура даних всередині застосунку досягається через використання статичної типізації TypeScript. Опис того, як дані повинні виглядати та передаватися між компонентами знаходиться у папці types. Такий метод додає надійність та спрощує підтримку коду, що дозволяє ефективно знаходити та вирішувати проблеми під час розробки.

Така модульна структура надає можливість зручно додавати новий функціонал у застосунок, зберігаючи при цьому читабельність коду. Чітке розділення між відповідальностями різних компонентів також спрощує тестування системи і підтримує можливості для масштабування. Використання глобальних провайдерів забезпечує централізоване управління глобальним станом застосунку і підтримує уніфіковану комунікацію з API.

Детальна архітектура компонентів клієнтської частини відображена на рисунку 4.1.

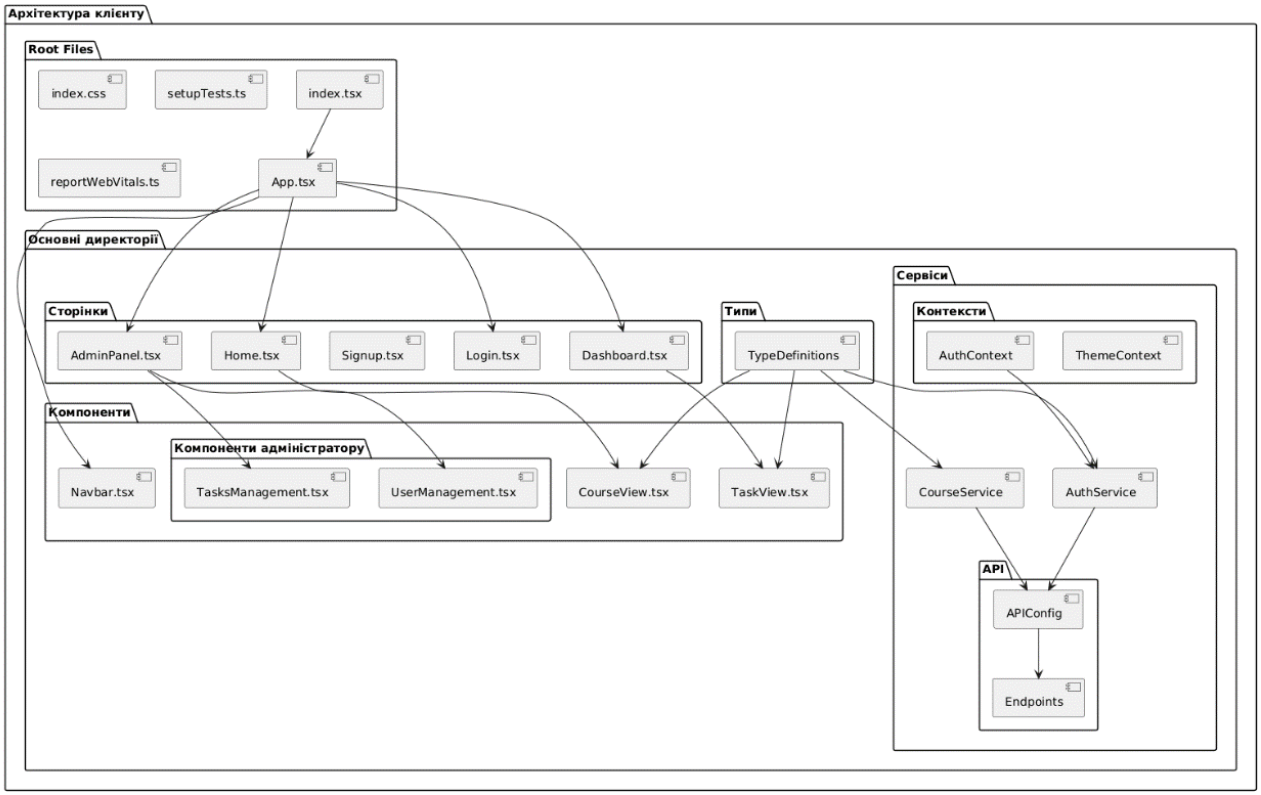


Рис.4.1 Діаграма архітектури клієнта

4.2 Реалізація серверної частини (Backend)

Структура серверної частини web-застосунку для самостійного вивчення програмування є важливим елементом, який забезпечує ефективну взаємодію між клієнтом та сервером. Вона включає фреймворк, архітектуру API та логіку обробки запитів, які разом забезпечують коректну роботу всієї системи.

Для серверної частини використовується FastAPI та застосовується архітектура REST API, що дозволяє створити стандартизовані кінцеві точки (endpoints) для взаємодії між клієнтом та сервером. REST API використовує стандартні HTTP-методи, такі як GET, POST, PUT, DELETE, щоб клієнтський застосунок надсилав запити до сервера для отримання або зміни даних.

Логіка обробки запитів на сервері є важливим етапом у функціонуванні web-застосунку, оскільки вона забезпечує правильну взаємодію між користувачем і сервером. Спершу сервер здійснює автентифікацію та авторизацію користувача. Це критичний етап, на якому перевіряється, чи має

користувач доступ до певних ресурсів. Для цього використовуються JWT-токени та механізм контролю доступу на основі ролей (RBAC), і на їх основі надається доступ до захищених кінцевих точок

FastAPI підтримує потужний функціонал, який реалізує впровадження залежностей. За допомогою методу Depends можна ефективно розбити роботу автентифікації та роботи з базою даних від бізнес-логіки, оскільки цей метод буде автоматично включати певний функціонал у обробку шляхів. Це значно спрощує структуру застосунку, а також підвищує гнучкість.

Наступним етапом є обробка даних. Сервер отримує запит від клієнта, обробляє його відповідно до логіки програми і виконує необхідні дії. Це може включати збереження або отримання даних з бази даних. Наприклад, при виконанні завдання користувачем сервер може обчислити результат чи зберегти прогрес, який потім буде доступний для аналізу.

Головною точкою входу у застосунок слугує файл main, який запускає uvicorn ASGI сервер, підключає маршрутизатори та встановлює налаштування безпеки такі як секретний ключ, кукі сесій, допустимі джерела та інші.

Налаштування зв’язку з базою даних описано у файлі database, який створює підключення до бази даних, створює сесію і дає можливість працювати із таблицями. Решта компонентів застосунку отримують доступ до бази даних саме через методи описані у цьому файлі

Маршрутизатори до застосунку описані у папці routers, вони відкривають кінцеві точки, до яких клієнт зможе надсилати запити. Окрім обробки самих HTTP запитів, вони також реалізують бізнес-логіку, перевіряють запит на коректність та опрацьовують помилки. Основні шляхи API включають:

— /admin – операції адміністраторів

— /auth – операції пов’язані з автентифікацією та реєстрацією

— /courses – операції пов’язані із керуванням курсами

— /hints – операції пов’язані з обробкою підказок

— /tasks – операції пов’язані із керуванням задачами

Кожен із шляхів включає свої кінцеві точки, які можуть потребувати окремі параметри для виконання різноманітних запитів в залежності від їх типу.

Ці кінцеві точки повертають дані, які FastAPI автоматично серіалізує у формат JSON, що дозволяє зручно обробляти їх на стороні клієнта. Кінцеві точки повинні отримувати чітко визначені структури даних, для того щоб підтримувати надійність виконання запитів або обробки помилок. Такі схеми описані у папці schemas за допомогою Pydantic, що дозволяє до того ж перевіряти дані на коректність, а також трансформувати їх з об'єктів Python у JSON.

Схеми даних створені на основі моделей бази даних, які описані у папці models. Вони реалізовані за допомогою типів SQLAlchemy та дозволяють використовувати усі сильні сторони цього ORM фреймворку та розділяти шар бази даних від решти логіки застосунку. Схеми тісно пов'язані із моделями та конвертуються одна до одної за допомогою Pydantic.

Крім того Pydantic грає важливу роль у перевірці коректності даних ще на початковій стадії обробки запиту. Ця бібліотека перевіряє, щоб значення у запиті чітко відповідали вказаним структурам даних та обмеженням. Через це лише запити із коректними даними попадають у частину застосунку, яка обробляє бізнес-логіку. Це значно спрощує підтримку та розробку застосунку, а також дозволяє уникнути багатьох помилок під час роботи.

Деяка логіка всередині застосунку повторюється, тому вона описана в окремих файлах у папці utils для ефективного перевикористання коду. Така логіка включає функції автентифікації, генерації підказок, перевірки коду користувача, перевірки безпеки та інші загальні утиліти, що використовуються у декількох модулях.

4.3 Інтеграція моделі штучного інтелекту

Використання моделі штучного інтелекту, зокрема моделей GPT від OpenAI для генерації підказок є важливою частиною інтерактивних навчальних платформ, оскільки допомагає користувачам швидше опановувати матеріал і

вирішувати завдання. Це дозволяє створити адаптивну систему, яка надає індивідуальні підказки на основі контексту виконуваного завдання.

З OpenAI API можна отримати доступ до різноманітних моделей штучного інтелекту, наприклад, gpt-3.5-turbo. Такі моделі навчені на величезних обсягах даних та при правильних налаштуваннях, їх використання здатне генерувати ідеальні підказки під кожного користувача. API отримує запит та генерує відповідь моделі штучного інтелекту, при тому коректність відповіді дуже залежить від правильно сформованого запиту. У іншому випадку модель може повернути абсолютно не підходящу відповідь.

Модель штучного інтелекту інтегровано у серверну частину та налаштовується за допомогою окремої бібліотеки OpenAI. Застосунок очікує наявність змінної середовища OPENAI_API_KEY, в якій буде знаходитися ключ для зв'язку з OpenAI API. Використовуючи ключ створюється клієнт для зв'язку з API.

Запит до моделі штучного інтелекту чітко структурований та включає у себе системне повідомлення, яке встановлює роль моделі, щоб вона поводила себе як викладач, який намагається допомогти користувачу засвоїти матеріал та зрозуміти як виконується задача через персональні підказки. Це дозволяє обмежити поведінку моделі штучного інтелекту та отримувати точніші відповіді.

Генерація самої підказки здійснюється через метод створеного попередньо клієнту chat.completions.create(). До запиту на підказку прикріпляється сама задача, очікуваний результат та код користувача, це дуже важлива інформація, оскільки саме від неї залежить точність відповіді моделі, і вказується сама модель – gpt-3.5-turbo.

Оскільки користувацький код може потенційно включати директиви, що можуть зламати поведінку моделі штучного інтелекту, запит також чітко вказує, де починається користувацький код. Це забезпечує надійну роботу моделі, а також, що її відповідь буде згенерована коректно.

Крім того для запиту встановлюється значення температури, яке відповідає за те, наскільки відповідь моделі може бути довільною. Це значення має бути у

межах від 0 до 2, у застосунку встановлено значення температури 0.7 для збалансованої відповіді. Проте для максимальної точності відповідей моделі потрібно провести окреме дослідження для встановлення ідеальних значень температури. Додатково встановлюється значення токенів, тобто максимальний розмір відповіді від моделі [21].

Приклад відповіді моделі: “You're on the right track with the '+' sign, but remember that you need to add numbers, not just display them as strings. Think about how you can perform the addition operation to get the correct sum. Keep going!”. Модель врахувала усі попередження та налаштування і згенерувала слушну відповідь, в якій не надала конкретного рішення, але натякнула користувачу, що він повинен зробити для виконання задачі.

Для тестування функціональності під час розробки було створено окремий метод для емуляції підказки моделі штучного інтелекту та його автоматичне підключення, якщо не встановлено змінну оточення IMITATE_AI_HINTS значенням False. Цей метод має завчасно описані підказки, які використовуються за замовчуванням, та у випадку, якщо за якоїсь причини OpenAI API буде недоступний, система зможе автоматично почати використовувати завчасно згенеровані підказки, щоб уникнути випадків, коли користувачі отримуватимуть помилки від серверу.

4.4 Тестування користувацького коду

Коли користувач виконує задачу та надсилає розроблений код на сервер, він перевіряється за допомогою методу контейнеризації Docker. Цей підхід дозволяє запускати код у ізольованому середовищі, з чітко виділеною кількістю ресурсів та мінімальними доступами як всередині контейнеру так і до системи серверу [22, с.189].

Оскільки користувач може надіслати будь-який код, який забажає, система повинна бути достатньо захищена, щоб надісланий результат не міг взламати

застосунок або ж отримати персональні дані. Саме тому використання Docker контейнерів для перевірки користувацького коду є ідеальним рішенням.

Docker – це платформа, яка дозволяє запускати застосунок у портативному середовищі. У контейнер завантажуються необхідні бібліотеки, залежності, конфігураційні файли, а також самі файли застосунку.

На відміну від віртуальних машин, в яких гіпервізор емулює повноцінну операційну систему, Docker контейнери розділяють ядро операційної системи машини, на якій і запускаються. Це призводить до сильного пришвидшення запуску застосунку, ефективнішого використання ресурсів і звісно рівня захисту системи від застосунку.

Docker працює через процес, який запущений на задньому фоні у системі – dockerd. Цей процес відповідальний за керування контейнерами та напряму виконує системні запити, щоб реалізувати цей функціонал.

Для роботи з Docker використовується командна утиліта або ж REST інтерфейс. Docker відкриває REST API, що дозволяє стороннім програмам використовувати його функціонал та керувати контейнерами.

Усі операції пов’язані з Docker виконуються через спеціальний клас DockerService з бібліотеки Docker. При створенні цього класу вказуються ліміти по ресурсам та скільки контейнер може працювати. У застосунку використовуються стандартні публічні образи, що спрощує процес додавання підтримки нової мови до застосунку. Потрібно лише вибрати правильний образ та коректно налаштувати команди, що повинні запуститися після створення контейнеру, оскільки користувацький код має бути інтерпретований або скомпільований, і результат виконання буде врахований застосунком.

Наразі підтримуються дві мови – Python та JavaScript, оскільки ці мови не потрібно окремо компілювати. Список підтримуваних мов зберігається в окремій змінній, і перед запуском контейнеру система перевіряє чи працює сервіс Docker і чи є потрібна мова у списку. Якщо мова є в списку, то перевіряється чи встановлений образ вказаний у налаштуваннях, якщо ні, то система спробує його встановити. В іншому випадку застосунок повідомить про помилку.

Після успішних перевірок система запускає образ разом з потрібними командами і кодом користувача. Коли процес перевірки завершується, застосунок видаляє створений контейнер із системи та опрацьовує результат, щоб сформувати коректну відповідь клієнту.

Клієнт може отримати 3 різні результати: успішна перевірка та повідомлення про виконане завдання, неуспішна перевірка і повідомлення з фактичним та очікуваним результатами або ж помилка про неробочий код. Для досягнення такого результату застосунок аналізує результат інтерпретатору або ж компілятору та готує відповідь користувачу.

Така система дозволяє безпечно запускати будь-який користувацький код на сервері, оскільки код не матиме ніякого доступу до ресурсів окрім власного ізольованого оточення.

ВИСНОВКИ

У ході виконання дипломної роботи:

1. Проведено аналітичний огляд предметної області на основі якого встановлено актуальність застосування інтеграції штучного інтелекту у процес навчання. Проведений порівняльний аналіз web-застосунків, аналогічних запропонованому автором, показав їх сильні та слабкі сторони. На основі аналізу можливостей web-застосунків для спроектованого застосунку розроблені функціональні та нефункціональні вимоги.

2. Виконано порівняння різноманітних сучасних засобів для реалізації web-застосунку на основі попередньо визначених функціональних і нефункціональних вимог. В якості інструментальних засобів для розробки клієнтської частини обрано TypeScript та фреймворк React, а для серверної частини - Python та FastAPI. Обрано PostgreSQL базу даних для збереження даних застосунку. А також уся розробка виконувалася у середовищі Visual Studio Code.

3. Розглянуто найпопулярніші моделі штучного інтелекту та обрано модель від OpenAI gpt-3.5-turbo.

4. Спроєктовано клієнтську та серверну частини web-застосунку, реалізовано зв’язок між ними, створені моделі бази даних і сутності. Додатково розглянуто тему безпеки та автентифікації користувачів.

5. Розроблено web-застосунок для самостійного вивчення програмування з інтерактивними завданнями та підказками на основі інтеграції моделі штучного інтелекту у процес навчання, що привело до його значного покращення.

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

1. Шерстньов М.А., Садовенко В.С. Порівняльний аналіз платформ для самостійного вивчення програмування. VI Всеукраїнська науково-технічна

конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.40-42.

2. Шерстньов М.А., Садовенко В.С. Перспективи використання штучного інтелекту для покращення процесу навчання. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.2025, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.366-368.

3. Шерстньов М.А. Порівняльний аналіз методів автентифікації у веб-застосунках. XII Всеукраїнська науково-практична конференція молодих учених «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ – 2025», 15.05.2025., Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез. К.: КСУ імені Бориса Грінченка, 2025. С. 343-345.

ПЕРЕЛІК ПОСИЛАНЬ

1. Gligorea I., Cioca M., Oancea R., Gorski A.-T., Gorski H., Tudorache P. Adaptive learning using artificial intelligence in e-learning: A literature review. *Education Sciences*. 2023. Vol. 13, no. 12. P. 1216.

2. Lasha L., Maya G., Lela M. Role of AI chatbots in education: systematic literature review. *International Journal of Educational Technology in Higher Education*. 2023. Т. 20, вип. 1. С. 1–17. URL: https://doi.org/10.1186/s41239-023-00426-1

3. Cachola K., Vu K.-Ph.L. Investigating the Use of Chatbots as an Educational Tool. *Human Factors in Design, Engineering, and Computing*. 2024. Vol. 159, no. 159. P.1849-1859.

4. Савельєв Є.М. Впровадження гейміфікації у веб-застосунок: вплив на користувацький досвід. *IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті»*, 15 травня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій.

5. Alaqsam A., Ghabban F., Ameerbakhsh O., Alfadli I., Fayez A. Current trends in online programming languages learning tools: a systematic literature review. *Journal of Software Engineering and Applications*. 2021. Vol. 14, no. 7. P. 277–297.

6. du Plooy E., Casteleijn D., Franzsen D. Personalized adaptive learning in higher education: a scoping review of key characteristics and impact on academic performance and engagement. *Heliyon*. 2024. Vol. 10, no. 21.

7. Non-Functional Requirements: How to Ensure Performance, Availability, and Security. 29.10.2024. *EPAM Ukraine*. URL: https://careers.epam.ua/blog/non-functional-requirements-how-to-ensure-performance-availability-and-security (дата звернення: 01.05.2025).

8. The Evolution and Impact of Python in Modern Programming. 27.05.2024. *Data Science Society*. URL: https://www.datasciencesociety.net/the-evolution-and-impact-of-python-in-modern-programming/ (дата звернення: 01.05.2025).

9. George J. The Impact of TypeScript on Developer Productivity: Does Static Typing Really Help? *Federal University Oye Ekiti*. 2024.

10. Braganca W.O., Kho I.E. Comparative Analysis of Python Microframeworks: Flask, Dash, and CherryPy. *Swiss German University, Faculty of Engineering and Information Technology*. 2023. URL: http://dx.doi.org/10.13140/RG.2.2.17101.82402

11. Komperla V., Pratiba D., Ghuli P., Pattar R. React: A detailed survey. *Indonesian Journal of Electrical Engineering and Computer Science*. 2022. Vol. 26, no. 3. P. 1710–1717. URL: https://doi.org/10.11591/ijeecs.v26.i3.pp1710-1717

12. Sultan M. Angular and the trending frameworks of mobile and web-based platform technologies: A comparative analysis. *Proc. Future Technologies Conference*. 2017. P. 928–936.

13. Comparative Analysis Based on DeepSeek, ChatGPT, and Google Gemini: Features, Techniques, Performance, Future Prospects / A. Rahman та ін. URL: https://doi.org/10.48550/arXiv.2503.04783.

14. Papazoglou M.P., Van Den Heuvel W.-J. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*. 2007. Vol. 16. P. 389-415.

15. Pautasso C., Zimmermann O., Leymann F. Restful web services vs. "big" web services: making the right architectural decision. *Proceedings of the 17th International Conference on World Wide Web*. 2008. P. 805–814.

16. Гулеватий О.О. Вибір оптимального протоколу API для веб-сервісів. *Всеукраїнська науково-практична конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*, 24 квітня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій.

17. Using HTTPOnly Cookies to Strengthen User Authentication. 23.01.2024. *Notarize*. URL: https://www.notarize.com/blog/using-httponly-cookies-to-strengthen-user-authentication (дата звернення: 03.05.2025).

18. Miya T.K., Govender I. UX/UI design of online learning platforms and their impact on learning: A review. *International Journal of Research in Business & Social Science*. 2022. Vol. 11.

19. Wienand, M., Wulfert, T., & Hoang, H. (2024). Design principles for e-learning platforms featuring higher-education students' enterprise systems end-user training. *Discover Education*, 3(82). URL: https://doi.org/10.1007/s44217-024-00165-z.

20. Zammetti F. Modern Full-Stack Development: Using TypeScript, React, Node.js, Webpack, Python, Django, and Docker. Apress, 2022. 430 p.

21. Gandolfi A. GPT-4 in education: Evaluating aptness, reliability, and loss of coherence in solving calculus problems and grading submissions. *International Journal of Artificial Intelligence in Education*. 2024. P. 1–31.

22. Kothapalli M. Securing Microservices Architecture: Best Practices and Challenges. *Journal of Scientific and Engineering Research*. 2021. Vol. 8, no. 10. P. 187-192.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для самостійного вивчення програмування з інтерактивними завданнями та підказками на основі штучного інтелекту

Виконав студент 4 курсу

групи ПД-42
Шерстньов Микита Андрійович
Керівник роботи

к.ф-м.н., доц, професор кафедри ІПЗ Садовенко Володимир Сергійович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - покращення процесу самостійного навчання програмуванню шляхом інтеграції моделі штучного інтелекту у процес навчання.
- **Об’єкт дослідження -** процеси самостійного вивчення програмування за допомогою web -застосунків.
- **Предмет дослідження -** web-застосунок, що забезпечує інтерактивне навчання програмуванню з використанням технологій штучного інтелекту.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести огляд методів і засобів самостійного вивчення програмування і підвищення якості процесу навчання за допомогою інтеграції в нього штучного інтелекту.
2. Виконати порівняльний аналіз аналогічних web-застосунків для самостійного вивчення програмування.
3. Сформувати вимоги до функціоналу web-застосунку.
4. Дослідити та обрати засоби реалізації для розробки web-застосунку.
5. Спроєктувати web-застосунок для самостійного навчання програмуванню з інтерактивними завданнями та підказками на основі штучного інтелекту.
6. Розробити серверну та клієнтську частини на основі обраних засобів реалізації
7. Інтегрувати модель штучного інтелекту для генерації підказок.
8. Протестувати розроблений web-застосунок.

3

АНАЛІЗ АНАЛОГІВ

Продукт/Функціональність	CodeAcademy	LeetCode	SoloLearn	freeCodeCamp	Udemy	Khan Academy	DreamLearn
Інтерактивний код у браузері	+	+	+	+	-	+	+
Структурованість	+	-	-	+	+	+	+
Алгоритмічні задачі	-	+	-	-	-	-	+
Адаптивні підказки/матеріал	-	-	-	-	-	-	+
Автоматична перевірка коду	+	+	-	+	-	-	+
Відстеження прогресу	+	+	+	+	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Виконання інтерактивних завдань з програмування
2. Супроводження кожного завдання теоретичним поясненням
3. Поділ завдань на рівні складності
4. Надсилання коду на перевірку
5. Система динамічних персональних підказок на основі штучного інтелекту
6. Збереження прогресу користувача

Нефункціональні вимоги:

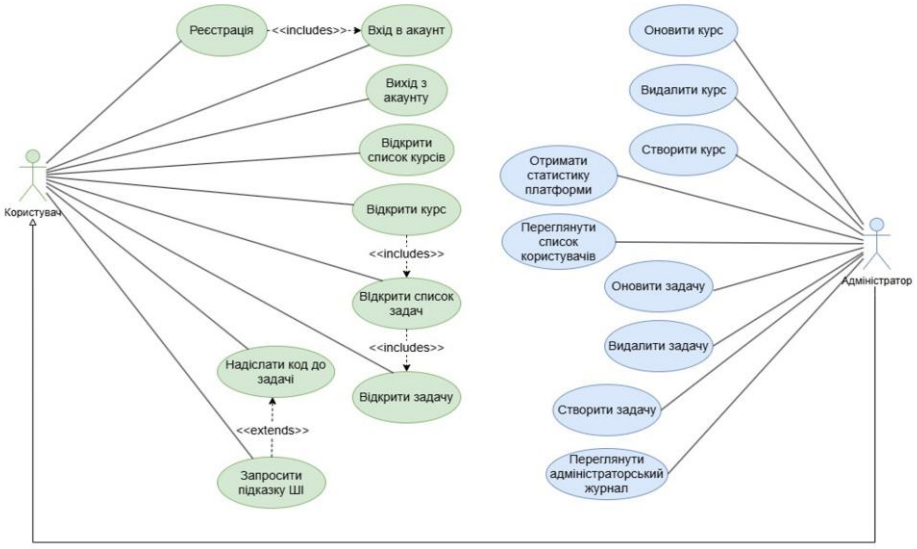
1. Час відповіді на дії користувачів не повинен перевищувати 300 мс у 95% випадків.
2. Повна завантаженість сторінки має відбуватись менше ніж за 1 секунду
3. Оптимізація запитів до бази даних(індексація).
4. Використання кешування сторінок
5. Горизонтальне масштабування
6. Передача усіх даних через захищенне з'єднання(HTTPS)

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

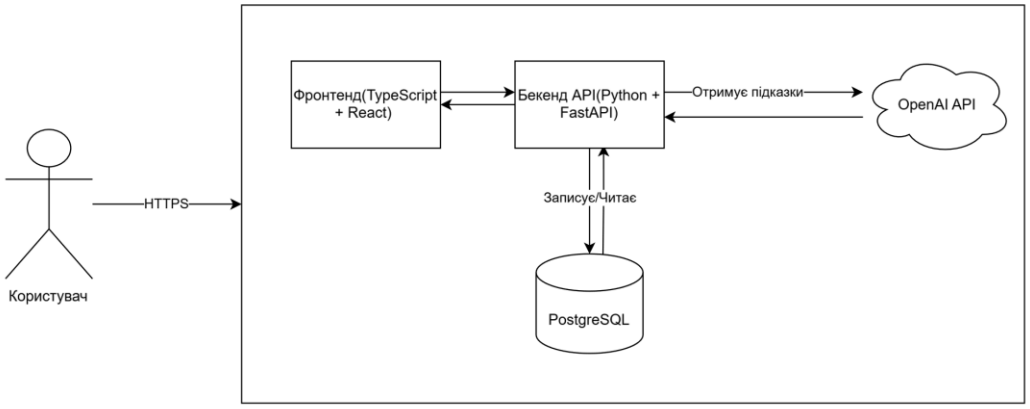
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



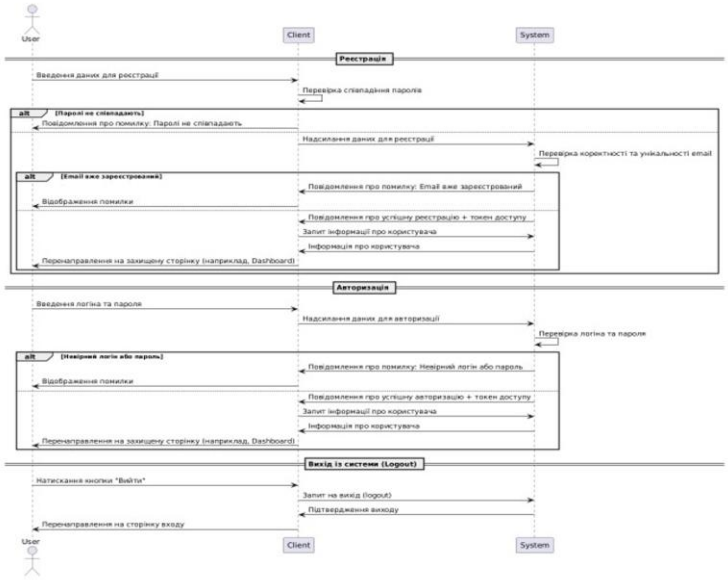
7

ДІАГРАМА АРХІТЕКТУРИ



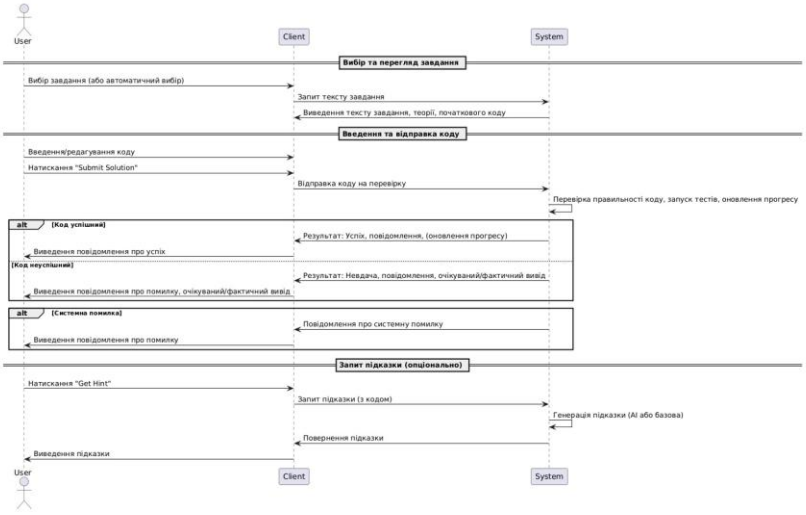
8

ДІАГРАМА ПОСЛІДОВНОСТІ РЕЄСТРАЦІЇ ТА АВТОРИЗАЦІЇ



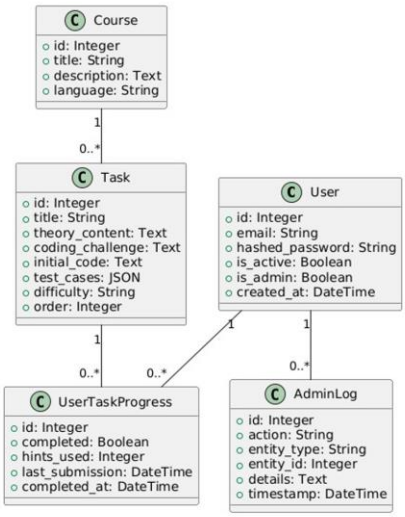
9

ДІАГРАМА ПОСЛІДОВНОСТІ РОБОТИ ІЗ ЗАДАЧАМИ



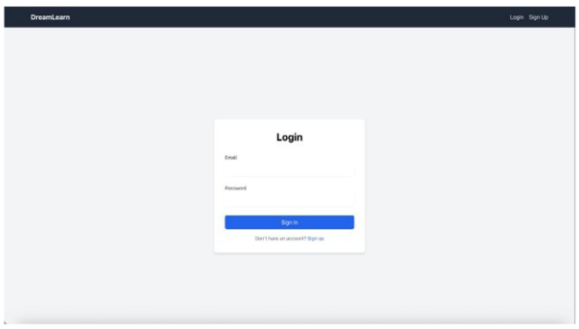
10

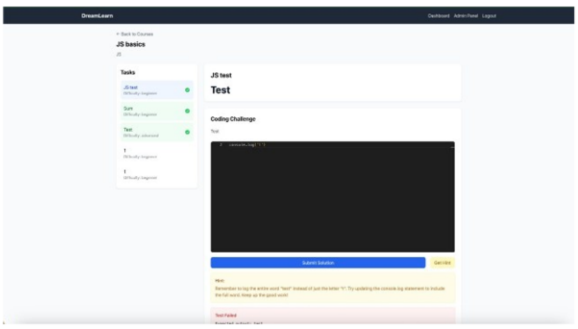
ДІАГРАМА КЛАСІВ



11

ЕКРАННІ ФОРМИ WEB-ЗАСТОСУНКУ





12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Шерстньов М.А., Садовенко В.С. Порівняльний аналіз платформ для самостійного вивчення програмування. VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». 24 квітня 2025 року, ДУІКТ, м. Київ .К: ДУІКТ, 2025 С. 4042.
2. Шерстньов М.А., Садовенко В.С. Перспективи використання штучного інтелекту для покращення процесу навчання. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». 24 квітня 2025 року, ДУІКТ, м. Київ .К: ДУІКТ, 2025 С. 366368.
3. Шерстньов М.А. Порівняльний аналіз методів автентифікації у веб-застосунках. Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології - 2025»,, 15 травня 2025 року, КСУ імені Бориса Грінченка, м. Київ. К.: КСУ імені Бориса Грінченка, 2025. С. 343-345.

14

ВИСНОВКИ

1. Проведено огляд підходів до самостійного вивчення програмування та методів вдосконалення процесу навчання. Визначено необхідність інтеграції в цей процес моделі штучного інтелекту.
2. Виконано порівняльний аналіз аналогічнихweb-застосунків для самостійного вивчення програмування.
3. Визначено функціональні і нефункціональні вимоги.
4. Обрано засоби реалізації для розробкиweb-застосунку.
5. Спроєктовано web-застосунок для самостійного вивчення програмування з інтерактивними завданнями та підказками на основі штучного інтелекту.
6. Розроблено серверну та клієнтську частини використовуючи обрані засоби реалізації.
7. Інтегровано модель штучного інтелектуgpt-3.5-turbo від OpenAI для генерації підказок.

15

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider, useAuth } from './contexts/AuthContext';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import Login from './pages/Login';
import Signup from './pages/Signup';
import Dashboard from './pages/Dashboard';
import AdminPanel from './pages/AdminPanel';
import CoursePage from './pages/CoursePage';

const ProtectedRoute: React.FC<{ element: React.ReactElement }> = ({ element }) => {
  const { isAuthenticated, isLoading } = useAuth();

  if (isLoading) {
    return <div>Loading...</div>;
  }

    return isAuthenticated ? element : <Navigate to="/login" />;
};

const AdminRoute: React.FC<{ element: React.ReactElement }> = ({ element }) => {
  const { isAuthenticated, isLoading, user } = useAuth();

  if (isLoading) {
    return <div>Loading...</div>;
  }

  if (!isAuthenticated) {
    return <Navigate to="/login" />;
  }

  if (!user?.is_admin) {
    return <Navigate to="/" />;
  }

  return element;
};

const AppRoutes: React.FC = () => {
  return (
    <div className="min-h-screen bg-gray-50">
      <Navbar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/signup" element={<Signup />} />
                    <Route path="/dashboard" element={<ProtectedRoute element={<Dashboard />} />} />
          <Route path="/admin" element={<AdminRoute element={<AdminPanel />} />} />
                <Route path="/courses/:courseId" element={<ProtectedRoute element={<CoursePage />} />} />
      </Routes>
    </div>
  );
};

const App: React.FC = () => {
  return (
    <Router>
      <AuthProvider>
        <AppRoutes />
      </AuthProvider>
    </Router>
  );
};

export default App;
```

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider, useAuth } from './contexts/AuthContext';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import Login from './pages/Login';
import Signup from './pages/Signup';
import Dashboard from './pages/Dashboard';
import AdminPanel from './pages/AdminPanel';
import CoursePage from './pages/CoursePage';

const ProtectedRoute: React.FC<{ element: React.ReactElement }> = ({ element }) => {
 const { isAuthenticated, isLoading } = useAuth();

 if (isLoading) {
  return <div>Loading...</div>;
 }

  return isAuthenticated ? element : <Navigate to="/login" />;
};

const AdminRoute: React.FC<{ element: React.ReactElement }> = ({ element }) => {
 const { isAuthenticated, isLoading, user } = useAuth();

 if (isLoading) {
  return <div>Loading...</div>;
 }

 if (!isAuthenticated) {
  return <Navigate to="/login" />;
 }

 if (!user?.is_admin) {
  return <Navigate to="/" />;
 }

 return element;
};

const AppRoutes: React.FC = () => {
 return (
  <div className="min-h-screen bg-gray-50">
   <Navbar />
   <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/login" element={<Login />} />
    <Route path="/signup" element={<Signup />} />
                    <Route path="/dashboard" element={<ProtectedRoute element={<Dashboard />} />} />
     <Route path="/admin" element={<AdminRoute element={<AdminPanel />} />} />
                <Route path="/courses/:courseId" element={<ProtectedRoute element={<CoursePage />} />} />
   </Routes>
  </div>
 );
};

const App: React.FC = () => {
 return (
  <Router>
   <AuthProvider>
    <AppRoutes />
   </AuthProvider>
  </Router>
 );
};

import React, { useState, useEffect } from 'react';
import Editor from '@monaco-editor/react';
import ReactMarkdown from 'react-markdown';
import { Task } from '../types/task';
import { Course } from '../types/course';
import api from '../api/config';

interface TaskViewProps {
 task: Task;
```

```
  onTaskCompleted?: () => void;
}

interface SubmissionResult {
  success: boolean;
  output: string;
  expected_output: string;
  message: string;
}

interface HintResponse {
  hint: string;
  hints_remaining: number;
}

const TaskView: React.FC<TaskViewProps> = ({ task, onTaskCompleted }) => {
  const [code, setCode] = useState(task.initial_code || '# Write your Python code here\n');
      const [isSubmitting, setIsSubmitting] = useState(false);
    const [isRequestingHint, setIsRequestingHint] = useState(false);
  const [result, setResult] = useState<SubmissionResult | null>(null);
    const [hint, setHint] = useState<HintResponse | null>(null);
  const [error, setError] = useState<string | null>(null);
    const [course, setCourse] = useState<Course | null>(null);
    const [isLoadingCourse, setIsLoadingCourse] = useState(true);

  useEffect(() => {
    const fetchCourse = async () => {
      setIsLoadingCourse(true);
      try {
                          const response = await api.get(`/courses/${task.course_id}`);
        setCourse(response.data);
      } catch (error) {
        console.error('Error fetching course:', error);
      } finally {
        setIsLoadingCourse(false);
      }
    };

    fetchCourse();
  }, [task.course_id]);

  useEffect(() => {
    setCode(task.initial_code || '# Write your Python code here\n');
    setResult(null);
    setHint(null);
    setError(null);
  }, [task.id, task.initial_code]);

  const handleSubmit = async () => {
    try {
      setIsSubmitting(true);
      setResult(null);
      setError(null);

                          const response = await api.post(`/tasks/${task.id}/submit`, {
        code: code
      });

      setResult(response.data);

      if (response.data.success && onTaskCompleted) {
        setTimeout(() => {
          onTaskCompleted();
        }, 2000);
      }
    } catch (error) {
      console.error('Error submitting code:', error);
      setResult({
        success: false,
        output: '',
        expected_output: '',
        message: 'An error occurred while submitting your code.'
```

```
    });
   } finally {
    setIsSubmitting(false);
   }
  };

  const handleRequestHint = async () => {
   try {
    setIsRequestingHint(true);
    setError(null);

    const response = await api.post(`/hints/${task.id}`, {
     code: code
    });

    setHint(response.data);
   } catch (error: any) {
    console.error('Error requesting hint:', error);
      setError(error.response?.data?.detail || 'An error occurred while requesting a hint.');
   } finally {
    setIsRequestingHint(false);
   }
  };

  return (
   <div className="space-y-6">
    {/* Theory Content */}
        <div className="bg-white p-6 rounded-lg shadow">
          <h2 className="text-2xl font-bold mb-4">{task.title}</h2>
      <div className="prose max-w-none">
       <ReactMarkdown>{task.theory_content}</ReactMarkdown>
      </div>
    </div>

    {/* Coding Challenge */}
        <div className="bg-white p-6 rounded-lg shadow">
      <h3 className="text-xl font-bold mb-4">Coding Challenge</h3>
      <div className="prose max-w-none mb-6">
       <ReactMarkdown>{task.coding_challenge}</ReactMarkdown>
      </div>

      {/* Code Editor */}
       <div className="h-[400px] border rounded-lg overflow-hidden mb-4">
        {isLoadingCourse ? (
         <div className="h-full flex items-center justify-center">
          <div className="text-gray-500">Loading editor...</div>
         </div>
        ) : (
         <Editor
          height="100%"
          defaultLanguage={course?.language || 'python'}
          theme="vs-dark"
          value={code}
          onChange={(value) => setCode(value || '')}
          options={{
           minimap: { enabled: false },
           fontSize: 14,
           lineNumbers: 'on',
           automaticLayout: true,
           quickSuggestions: false,
           suggestOnTriggerCharacters: false,
           wordBasedSuggestions: 'off',
           parameterHints: { enabled: false },
           suggest: { showWords: false }
          }}
         />
        )}
       </div>

      {/* Action Buttons */}
      <div className="flex gap-4 mb-4">
       <button
        onClick={handleSubmit}
```

```
          disabled={isSubmitting}
          className={`flex-1 py-2 px-4 rounded-lg text-white font-medium ${
            isSubmitting
              ? 'bg-blue-400 cursor-not-allowed'
              : 'bg-blue-600 hover:bg-blue-700'
          }`}
        >
              {isSubmitting ? 'Submitting...' : 'Submit Solution'}
        </button>

        <button
          onClick={handleRequestHint}
          disabled={isRequestingHint}
          className={`py-2 px-4 rounded-lg font-medium ${
            isRequestingHint
                ? 'bg-gray-200 text-gray-500 cursor-not-allowed'
                : 'bg-yellow-100 text-yellow-700 hover:bg-yellow-200'
          }`}
        >
          {isRequestingHint ? 'Getting Hint...' : 'Get Hint'}
        </button>
      </div>

      {/* Error Message */}
      {error && (
        <div className="mb-4 p-4 bg-red-50 border border-red-200 rounded-lg text-red-700">
          {error}
        </div>
      )}

      {/* Hint Display */}
      {hint && hint.hint && (
        <div className="mb-4 p-4 bg-yellow-50 border border-yellow-200 rounded-lg">
          <div className="font-medium text-yellow-800 mb-1">Hint:</div>
                    <div     className="text-yellow-700">{hint.hint}</div>
        </div>
      )}

      {/* Results */}
      {result && (
        <div className={`mt-4 p-4 rounded-lg ${
          result.success ? 'bg-green-50 border border-green-200' : 'bg-red-50 border border-red-200'
        }`}>
          <div className={`font-medium mb-2 ${
            result.success ? 'text-green-700' : 'text-red-700'
          }`}>
            {result.success ? (
              <div className="flex items-center">
                    <svg  className="w-5 h-5 mr-2" fill="currentColor" viewBox="0 0 20 20">
                  <path fillRule="evenodd" d="M10 18a8 8 0 100-16 8 8 0 000 16zm3.707-9.293a1 1 0 00-1.414-1.414L9 10.586 7.707 9.293a1 1 0 00-1.414 1.414l2 2a1 1 0 001.414 0l4-4z" clipRule="evenodd" />
                </svg>
                    Congratulations! All tests passed successfully!
              </div>
            ) : 'Test Failed'}
          </div>
          <div className="text-sm">
                    <pre  className="whitespace-pre-wrap">{result.message}</pre>
                {!result.success && result.output !== result.expected_output && (
            <div className="mt-2">
              <div className="font-medium">Expected Output:</div>
              <pre className="bg-white p-2 rounded mt-1 text-sm overflow-x-auto">
                {result.expected_output}
              </pre>
              <div className="font-medium mt-2">Your Output:</div>
```

```
            <pre className="bg-white p-2 rounded mt-
1 text-sm overflow-x-auto">
              {result.output}
            </pre>
          </div>
        )}
      </div>
    </div>
  )}
 </div>
 </div>
 );
};

export default TaskView;

import React from 'react';
import { Link } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

const Navbar: React.FC = () => {
 const { isAuthenticated, logout, user } = useAuth();

 return (
  <nav className="bg-gray-800 text-white p-4">
   <div className="container mx-auto flex justify-
between items-center">
            <Link to="/" className="text-xl font-
bold">DreamLearn</Link>
    <div className="space-x-4">
     {isAuthenticated ? (
      <>
       <Link to="/dashboard" className="hover:text-
gray-300">Dashboard</Link>
        {user?.is_admin && (
          <Link to="/admin" className="hover:text-
gray-300">Admin Panel</Link>
        )}
        <button
         onClick={logout}
         className="hover:text-gray-300"
        >
          Logout
        </button>
      </>
     ) : (
      <>
       <Link to="/login" className="hover:text-gray-
300">Login</Link>
          <Link to="/signup" className="hover:text-
gray-300">Sign Up</Link>
      </>
     )}
    </div>
   </div>
  </nav>
 );
};

export default Navbar;

export default App;

from fastapi import FastAPI, Depends, status
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
from fastapi.middleware.trustedhost import
TrustedHostMiddleware
from starlette.middleware.sessions import
SessionMiddleware
from sqlalchemy.orm import Session
from database import get_db, verify_db_connection
from routers import auth_router, tasks_router,
admin_router, hints_router, courses_router
import secrets

app = FastAPI(
    title="AI Programming Learning Platform
DreamLearn",
   description="Backend API for the AI-powered
programming learning platform",
  version="0.1.0"
)
```

```
SECRET_KEY = secrets.token_urlsafe(32)

@app.middleware("http")
async def add_security_headers(request, call_next):
    response = await call_next(request)
        response.headers["X-Content-Type-Options"] = "nosniff"
    response.headers["X-Frame-Options"] = "DENY"
         response.headers["X-XSS-Protection"] = "1; mode=block"
        response.headers["Strict-Transport-Security"] = "max-age=31536000; includeSubDomains"
         response.headers["Content-Security-Policy"] = "default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-src 'self' 'unsafe-inline';"
    return response

app.add_middleware(
    SessionMiddleware,
    secret_key=SECRET_KEY,
    session_cookie="session",
    max_age=3600,  # 1 hour
    same_site="lax",
    https_only=True
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.add_middleware(
    TrustedHostMiddleware,
    allowed_hosts=["localhost", "localhost:3000"]
)

app.include_router(auth_router)
app.include_router(tasks_router)
app.include_router(admin_router)
app.include_router(hints_router)
app.include_router(courses_router)

@app.get("/health")
async def health_check(db: Session = Depends(get_db)):
    """Health check endpoint to verify API and database are running"""
    db_status = verify_db_connection()
    response = {
        "status": "ok",
        "database": "ok" if db_status else "error"
    }

    if not db_status:
        return JSONResponse(
            status_code=status.HTTP_503_SERVICE_UNAVAILABLE,
            content=response
        )

    return response

if __name__ == "__main__":
    import uvicorn
     uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True)

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
from sqlalchemy.exc import SQLAlchemyError
import os
import logging
from sqlalchemy import text

DATABASE_URL = os.getenv(
    "DATABASE_URL",
     "postgresql://postgres:postgres@localhost:5432/ai_learning_platform"
)
```

```
engine = create_engine(DATABASE_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

def get_db():
    """Get database session"""
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

def verify_db_connection():
    """Verify database connection is working"""
    try:
        db = SessionLocal()
        try:
            db.execute(text("SELECT 1"))
            return True
        except SQLAlchemyError as e:
                logging.error(f"Database connection failed: {str(e)}")
            return False
        finally:
            db.close()
    except Exception as e:
         logging.error(f"Failed to create database session: {str(e)}")
        return False

from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from sqlalchemy import func
from typing import List
from database import get_db
from models import User, Task, UserTaskProgress, Course
from schemas.task import Task as TaskSchema, TaskWithProgress, CodeSubmission, CodeSubmissionResponse, TaskCreate
from utils.auth import get_current_user, get_current_admin_user
from utils.code_runner import evaluate_code
import docker
import logging

logger = logging.getLogger(__name__)

router = APIRouter(prefix="/tasks", tags=["tasks"])

@router.get("/", response_model=List[TaskWithProgress])
async def get_tasks(
    course_id: int = None,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
):
    """Get all tasks with user progress, optionally filtered by course"""
    query = db.query(Task)
    if course_id:
        query = query.filter(Task.course_id == course_id)
    tasks = query.order_by(Task.order).all()

    progress_dict = {
                        p.task_id: p for p in db.query(UserTaskProgress).filter(
            UserTaskProgress.user_id == current_user.id
        ).all()
    }

    result = []
    for task in tasks:
        progress = progress_dict.get(task.id)
                                            task_dict = TaskSchema.model_validate(task).model_dump()
        task_dict["progress"] = {
            "completed": progress.completed if progress else False,
```

```
            "hints_used": progress.hints_used if progress else
0,
            "last_submission": progress.last_submission if
progress else None,
            "completed_at": progress.completed_at if
progress else None
        }
        result.append(task_dict)

    return result

@router.get("/{task_id}",
response_model=TaskWithProgress)
async def get_task(
    task_id: int,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
):
    """Get a specific task with user progress"""
    task = db.query(Task).filter(Task.id == task_id).first()
    if not task:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Task not found"
        )

    progress = db.query(UserTaskProgress).filter(
        UserTaskProgress.user_id == current_user.id,
        UserTaskProgress.task_id == task_id
    ).first()

                                    task_dict =
TaskSchema.model_validate(task).model_dump()
    task_dict["progress"] = {
        "completed": progress.completed if progress else
False,
        "hints_used": progress.hints_used if progress else
0,
            "last_submission": progress.last_submission if
progress else None,
        "completed_at": progress.completed_at if progress
else None
    }

    return task_dict

@router.post("/", response_model=TaskSchema,
status_code=status.HTTP_201_CREATED)
async def create_task(
    task: TaskCreate,
    db: Session = Depends(get_db),
                    current_user: User =
Depends(get_current_admin_user)
):
    """Create a new task (admin only)"""
        course = db.query(Course).filter(Course.id ==
task.course_id).first()
    if not course:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Course not found"
        )

    db_task = Task(**task.model_dump())
    db.add(db_task)
    db.commit()
    db.refresh(db_task)
    return db_task

@router.put("/{task_id}",
response_model=TaskSchema)
async def update_task(
    task_id: int,
    task: TaskCreate,
    db: Session = Depends(get_db),
                    current_user: User =
Depends(get_current_admin_user)
):
    """Update a task (admin only)"""
        db_task = db.query(Task).filter(Task.id ==
task_id).first()
    if not db_task:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
```

```
            detail="Task not found"
        )

    if task.course_id != db_task.course_id:
            course = db.query(Course).filter(Course.id ==
task.course_id).first()
        if not course:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUN
D,
                detail="Course not found"
            )

    for key, value in task.model_dump().items():
        setattr(db_task, key, value)

    db.commit()
    db.refresh(db_task)
    return db_task

@router.delete("/{task_id}",
status_code=status.HTTP_204_NO_CONTENT)
async def delete_task(
    task_id: int,
    db: Session = Depends(get_db),
                        current_user:          User          =
Depends(get_current_admin_user)
):
    """Delete a task (admin only)"""
        db_task = db.query(Task).filter(Task.id ==
task_id).first()
    if not db_task:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Task not found"
        )

    db.delete(db_task)
    db.commit()
    return None

@router.post("/{task_id}/submit",
response_model=CodeSubmissionResponse)
async def submit_code(
    task_id: int,
    submission: CodeSubmission,
    db: Session = Depends(get_db),
    current_user: User = Depends(get_current_user)
):
    """Submit code for a task"""
    try:
            task = db.query(Task).filter(Task.id ==
task_id).first()
        if not task:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUN
D,
                detail="Task not found"
            )

            course = db.query(Course).filter(Course.id ==
task.course_id).first()
        if not course:
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUN
D,
                detail="Course not found"
            )

        progress = db.query(UserTaskProgress).filter(
            UserTaskProgress.user_id == current_user.id,
            UserTaskProgress.task_id == task_id
        ).first()

        if not progress:
            progress = UserTaskProgress(
                user_id=current_user.id,
                task_id=task_id
            )
            db.add(progress)

        try:
            evaluation_result = await evaluate_code(
```

```
            code=submission.code,
            test_cases=task.test_cases,
            language=course.language
        )

        current_time = func.now()
        progress.last_submission = current_time
        if evaluation_result["success"]:
            progress.completed = True
            progress.completed_at = current_time

        db.commit()

        return evaluation_result

    except docker.errors.DockerException as e:
        error_message = str(e)
        if "Cannot connect to the Docker daemon" in error_message:
            error_message = "Docker service is not running. Please contact administrator."
        elif "No such image" in error_message:
            error_message = f"Required Docker image for {course.language} is not available. Please contact administrator."

        return {
            "success": False,
            "message": "System error: " + error_message,
            "test_results": [],
            "hints": ["The system encountered an error. Please try again later or contact support."]
        }

    except Exception as e:
        logger.error(f"Error in code submission: {str(e)}")
        return {
            "success": False,
            "message": "An unexpected error occurred",
            "test_results": [],
            "hints": ["The system encountered an error. Please try again later."]
        }
```