НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Онлайн платформа для спілкування в тематичних громадах та автоматичного пошуку наставника на основі хмари

Виконав студент IV курсу, групи ІП-96
(шифр групи)

Бородаєнко Ярослав Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник Храмченко М.С. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст.викл., Вітковська І. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Бородаєнку Ярославу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Хмарна платформа для пошуку менторів з програмування
керівник проєкту Храмченко М.С.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»квітня 2023 р. №_____

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання _____

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: загальні положення, основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог, постановка задачі. _____
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення, аналіз безпеки даних. _____
3) Аналіз якості та тестування програмного забезпечення: опис процесів тестування, підходи до тестування, опис контрольного прикладу. _____
4) Впровадження та супровід програмного забезпечення. _____

5. Перелік графічного матеріалу

1) Схема структурна класів програмного забезпечення _____

2) Схема структурна варіантів використань _____

3) Протитип екранних форм_____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	17.03.2023	
3	Постановка та формалізація задачі	23.03.2023	
4	Розробка інформаційного забезпечення	12.04.2023	
5	Алгоритмізація задачі	15.04.2023	
6	Обґрунтування вибору використаних технічних засобів	27.04.2023	
7	Розробка програмного забезпечення	09.04.2023	
8	Налагодження програми	16.05.2023	
9	Виконання графічних документів	21.05.2023	
10	Оформлення пояснювальної записки	28.05.2023	
11	Подання ДП на попередній захист	02.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Ярослав БОРОДАЄНКО
(підпис) (ініціали, прізвище)

Керівник _____ Микола ХРАМЧЕНКО
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 63 таблиці, 49 рисунків та 33 джерела – загалом 101 сторінка.

Дипломний проєкт присвячений створенню веб застосунку для спілкування у тематичних громадах та автоматичного пошуку наставника у галузі.

Метою розробки є спрощення взаємодії програмістів-початківців між собою та пришвидшення процесу пошуку ментора, за допомогою побудови системи рекомендацій.

Об'єкт дослідження: платформа для спілкування та автоматичного пошуку кваліфікованого наставника у галузі.

Предмет дослідження: соціальна медіа-платформа, як спосіб автоматизації пошуку наставника у галузі.

У розділі аналізу до вимог програмного забезпечення було проаналізовано основні проблеми пошуку наставника, досліджено технічні методи та підходи до вирішення поставленої задачі, а також проведено порівняльний аналіз актуальних реалізацій.

Розділ моделювання та конструювання програмного забезпечення присвячений розгляду та обґрунтуванню головних технічних рішень, зроблених під час реалізації застосунку. Було розглянуто його архітектуру на трьох рівнях.

У розділі аналізу якості та тестування програмного забезпечення було розглянуто стратегію до тестування додатка.

Розділ впровадження та супровід програмного забезпечення присвячений опису процесу розгортання на моніторингу додатка.

КЛЮЧОВІ СЛОВА: НАСТАВНИК, ТЕМАТИЧНІ ГРОМАДИ, РЕКОМЕНДАЦІЙНА СИСТЕМА, ВЕБ-ДОДАТОК, ХМАРА.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 63 tables, 49 figures and 33 sources - a total of 101 pages.

The diploma project is devoted to the creation of a web application for communication in thematic communities and the automatic search of a mentor in the field.

The purpose of the development is to simplify the interaction of novice programmers with each other and speed up the process of finding a mentor by building a system of recommendations.

Research object: a platform for communication and automatic search of a qualified mentor in the industry.

Research subject: social media platform as a way to automate the search for a mentor in the industry.

In the analysis section of software requirements, the main problems of finding a mentor were analyzed, technical methods and approaches to solving the task were investigated, and a comparative analysis of current implementations was conducted.

The software modeling and design section is devoted to the review and justification of the main technical decisions made during the implementation of the application. Its architecture was considered at several levels.

In the section on quality analysis and software testing, the strategy for testing the application was discussed.

The software implementation and maintenance section is devoted to the description of the deployment process for application monitoring.

KEY WORDS: TUTOR, TOPIC COMMUNITIES, RECOMMENDER SYSTEM, WEB APPLICATION, CLOUD.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-9602.045440.01.91	Технічне завдання	14	
3	А4	КПІ.ІП-9602.045440.02.81	Пояснювальна записка	101	
4	А4	КПІ.ІП-9602.045440.03.12	Текст програми	73	
5	А4	КПІ.ІП-9602.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-9602.045440.05.34	Керівництво користувача	9	
7	А3	КПІ.ІП-9602.045440.06.99.ССВ	Схема структурна варіантів	1	
			використань		
10	А3	КПІ.ІП-9602.045440.06.99.ССК	Схема структурна класів	1	
			програмного забезпечення		
11	А3	КПІ.ІП-9602.045440.06.99.КЕ	Креслення вигляду екранних форм	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-9602.045440.00.90		
Розроб.		Бородаєнко Я.С.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Храмченко М.С					
Н.контр.		Вітковська І.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-96	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ХМАРНА ПЛАТФОРМА ДЛЯ ПОШУКУ МЕНТОРІВ З ПРОГРАМУВАННЯ

Технічне завдання

КПІ.ІП-9602.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Ярослав БОРОДАЄНКО

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: хмарна платформа для пошуку менторів з програмування.

Галузь застосування: системи пошуку наставника.

Наведене технічне завдання поширюється на розробку програмного забезпечення “Хмарна платформа для пошуку менторів з програмування” КПІ.ІП-9602.045440.01.91, котра використовується для спілкування у тематичних громадах та автоматичного пошуку наставника в галузі та призначена для студентів та менторів.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки “Хмарна платформа для пошуку менторів з програмування” є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для студентів, які хочу знайти собі наставника та менторів які хочуть знайти собі підопічних.

Метою розробки є побудова платформи для обміну знань між користувачами та автоматизації процесу пошуку ментора.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій: надавати користувачам можливість задавати та відповідати на питання, передивлятися та спілкуватися з іншими користувачами, шукати ментора вручну чи автоматично.

4.1.1 Користувацького інтерфейсу

- наявність сторінки реєстрації користувача;
- наявність сторінки логіну користувача;
- наявність сторінки профілю користувача;
- наявність сторінки списку громад;
- наявність сторінки інформації про громаду;
- наявність сторінки запитання;
- наявність сторінки відповідей на запитання;
- наявність сторінки списку студентів;
- наявність сторінки переписки зі студентом.

Прототипи екранних форм зображені на рисунку 4.1.

Рисунок 4.1 – Прототипи екранних форм.

4.1.2 Для користувача:

– реєстрація та авторизація;
– перегляд профілю іншого користувача;
– пошук громад;
– приєднання до громади;
– обговорення питання.

4.1.3 Для студента:

– перегляд профіля наставник;
– подача заяви ментору;
– спілкування з ментором.

4.1.4 Для наставника

– підтвердження заяви студента;
– перегляд списку студентів;
– спілкування зі студентом.

4.1.5 Додаткові вимоги:

– можливість бачити останні повідомлення іншого користувача у режимі реального часу.

4.2 Вимоги до надійності

Передбачити контроль введення інформації на клієнтській частині застосунку. Забезпечити перевірку даних, що надсилаються на серверну частину застосунку. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 4 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт;

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7;
– об‘єм ОЗП: 8 Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Unix.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: HTTP-запити на сервер з форматом даних типу JSON і методами HTTP GET, HTTP POST, HTTP PUT, HTTP DELETE.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: відповіді на HTTP-запити у вигляді JSON, візуальний інтерфейс веб додатку.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Typescript.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді текстових файлів з розширенням .ts розділений на папки серверної та клієнтської частини.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Інсталювати залежності та згенерувати установчу версію окремо клієнтської та серверної частини застосунку.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;

– технічне завдання;

– текст програми;

– програма та методика тестування;

– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;

– схема структурна компонентів програмного забезпечення;

– схема бази даних;

– архітектура програмного забезпечення;

– схема структурна класів програмного забезпечення;

– діаграма розгортання програмного забезпечення.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.03	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.03	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.04	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.04	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.04	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.04	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.04	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Хмарна платформа для пошуку менторів з програмування

КПІ.ІП-9602.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

LMS	–	Integrated Development Environment, інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
SDK	–	Software development kit
LMS	–	Learning management system, система для керування навчанням
CMS	–	Content management system, система для керуванням контентом
DDD	–	Domain Driven Design, предметно-орієнтоване програмування.
AWS	–	Amazon Web Services, хмарний провайдер.

ВСТУП

ІТ індустрія щоденно змінюється, розвивається та поповнюється новими практиками, технологіями та бібліотеками. Це змушує розробників постійно опановувати та вивчати нові, незнайомі речі. Процес самонавчання, для більшості людей, є достатньо складним, повільним та потребує великого рівня наполегливості. Через це дедалі більше розробників вимушено звертаються до пошуку наставників-професіоналів у яких можна отримати корисну пораду або запитати відгук на фрагмент коду.

Окрім цього, наставництво набуває популярності серед спеціалістів які бажають опанувати іншу сферу задля покращення умов життя. Це є особливо актуальним для індустрії інформаційних технологій України, адже через повномасштабне вторгнення багато людей втратило роботу, а офіційна освіта не має можливості миттєво задовольнити весь попит.

На жаль, традиційні методи пошуку менторів, такі як мережеві заходи або ярмарки праці, займають багато часу та є виснажливими, особливо для тих хто тільки розпочинає свою кар’єру. Також вони обмежені географічно, що значно зменшує кількість та якість кандидатів.

Список спеціалізованих онлайн платформ для пошуку наставників, що вже представлені на ринку, є невеликим з малою кількістю активних користувачів або бракують важливого функціоналу. Через це більшість людей використовують популярні соціальні мережі. Хоча вони також мають певні недоліки, як, наприклад, неможливість врахування інтересів та побажань студента та низький загальний рівень автоматизації процесу знаходження ментора.

Отже, тема дипломної роботи є актуальною, оскільки направлена на пришвидшення та покращення процесу пошуку наставника з програмування, що є життєво важливою деталлю індустрії. Метою розробки є спрощення взаємодії програмістів-початківців між собою та пришвидшення процесу пошуку ментора, за допомогою побудови системи рекомендацій. Програмне

забезпечення, що розроблено, пропонує користувачеві звичний інтерфейс та функціонал соціальних мереж побудований навколо механізму з’єднання студента та ментора. Унікальною відмінністю програмного забезпечення є той факт, що воно базується на різноманітних елементах популярних хмарних провайдерів.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Наставництво – це взаємовідносини, у яких більш досвідчена людина (ментор або наставник) передає зібрані знання, навички та надає всіляку підтримку менш досвідченому учаснику (менті або підопічному). Воно широко використовується в педагогіці та вважається одним з найбільш ефективних заходів освіти [1].

Дослідники розрізняють три основних види наставництва: демократичне, авторитарне та laissez-faire [2]. Демократичний підхід характеризується наданням студенту повної свободи дій, а роль ментора зводиться до перевірки досягнень час від часу. При авторитарній моделі лідер робить більшість важливих рішень щодо підлеглого і жорстко фіксує його цілі. Laissez-faire вирізняється великим ступенем пасивності наставника і, зазвичай, є найменш ефективним видом менторства.

Онлайн менторство або е-менторство – течія, що набула популярності з розвитком Інтернету. На відміну від особистих зустрічей тут практикується використання асинхронних способів комунікації. Причому обидва члени процесу можуть ніколи не зустрітися, що дозволяє їм взаємодіяти незалежно від географії або конфліктів у розкладах [3]. Відомо, що такий перехід не впливає на якість процесу і навіть доповнює можливості традиційного наставництва [4].

Соціальна мережа – група людей у соціальному оточенні що має певні стосунки між собою. Кожен зв'язок може бути міцним або слабким, тимчасовим або тривалим тощо. Крім того, кожен зв'язок може містити в собі одразу декілька функцій як, наприклад, ментор і лектор в університеті. Ці важливі аспекти слугують основою різноманітних соціальних процесів у мережі, як, наприклад, поширення інформації [5].

Соціальні медіа – вид масових медіа-платформ та технологій, доступних в мережі Інтернет, що надають користувачам надійні канали

споживання та розповсюдження контенту. Ключовою особливістю цих медіа є високий рівень взаємодії між користувачами. Так, наприклад, вони можуть ділитися інформацією між собою і, таким чином, отримувати взаємну користь [6].

1.2 Змістовний опис і аналіз предметної області

Для розуміння основних задач і проблем предметної області спочатку розглянемо компоненти процесу менторства. Їх різні дослідники виділяють від трьох до п'яти. Одним прикладом є планування, структурування програми та оцінка [7]. Іншим, більш прийнятим, прикладом є поділ на наступні стадії – пошук та мотивація, соціалізація, обмін інформацією, побудова знань та розвиток [8]. Важливо зазначити що обидві роботи виділяють ранню соціалізацію та знайомство партнерів один з одним як ключовий елемент вдалого наставництва. Також вони визнають важливість платформи яка слугує спільним ресурсом для ментора та його менті та дозволяє їм автоматично покращувати та переоцінювати процес навчання із плином часу.

Розглянемо сучасні способи менторства в контексті даних п’яти стадій, звертаючи особливу увагу на стадію пошуку наставника та обміну інформації.

Першим і найдавнішим є традиційний або «офлайн» спосіб наставництва. Він же є особливо поширеним у навчальних установах. Пошук ментора тут здійснюється відвідуванням спеціалізованих зустрічей та заходів як, наприклад, ярмарки вакансій чи професійної конференції. Обмін інформацією відбувається періодично у форматі живого спілкування, часто у відведених приміщеннях, аудиторіях, при якому студент демонструє набуті знання у вигляді здачі та захисту завдань. Хоча цей метод є найпростішим він має значний недолік – обмеженість тією самою географічною локацією, що значно зменшує кількість опцій, особливо у невеликих містах.

Слід зазначити, що ведуться спроби подолання наявних проблем традиційного методу у сфері освіти за допомогою онлайн платформ типів

LMS чи CMS. Але такі системи, зазвичай, виявляються занадто орієнтовані на вчителів чи окремі курси, а не на студентів [9].

Другим підходом є використання спеціалізованих платформ. Часто до нього звертаються великі компанії, які хочуть таким чином налагодити процес передачі знань всередині команди. Пошук наставника може відбуватися як автоматично (призначення більш досвідченого члена команди новачку) так і самостійно менті на основі власних побажань. Формат комунікації може бути офлайн, онлайн чи гібридним. Даний метод надає великий ступінь гнучкості у побудові процесу з'єднання користувачів але, зазвичай, є занадто спеціалізованим і страждає від нестачі активних користувачів.

Третім методом є використання популярних платформ соціальних медіа як Twitter чи Reddit. Головною його перевагою є надзвичайно велика кількість активних користувачів і їх залученість. Так, наприклад, близько 23% американських дорослих використовують Twitter щоденно [10]. Але це і є певним недоліком, адже знайти відповідального ментора серед такої великої аудиторії може бути важко, адже сама платформа не надає жодних механізмів пошуку окрім пошуку на ім'я. Тому користувачі зазвичай групуються у невеликі громади присвячені одному напрямку, наприклад програмуванню на мові Java. Комунікація, таким чином, відбувається повністю онлайн за допомогою відеозв'язку або, частіше, звичайних приватних чатів.

Останнім способом, який набуває особливої популярності в останні роки, є використання платформ онлайн відео трансляцій [11]. Пошук ментора, при цьому, лежить повністю на користувачеві. Але, на відміну від усіх попередніх пропозицій, даний спосіб дає підопічному можливість подивитися на навички й манеру спілкування потенційного ментора до початку спілкування з ним. Водночас спілкування відбувається у режимі реального часу, де усі користувачі що присутні на відео трансляції можуть поставити запитання у загальному чаті. Це ідеально підходить для

проведення онлайн лекцій чи семінарів для відносно невеликої аудиторії (до 100 осіб) і все більше використовується у закладах офіційної освіти [12].

Аналізуючи вище наведені механізми можна зробити висновок що ідеальна платформа для пошуку менторів має бути максимально привабливою для користувачів та містити достатню кількість специфічного доменного контексту задля автоматизації процесу пошуку відповідного ментора студенту.

Отже, дана робота буде основана на способах 2 та 3. Кінцевим результатом стане соціальна медіа-платформа для програмістів з розширеним профілем користувача який включає таку специфічну інформацію як інтереси, хобі та технічні навички. Отримані дані дозволять розробити ефективну рекомендаційну систему яка стане основою автоматичного алгоритму підбору якісного наставника.

Слід зазначити, що для залучення більшої кількості користувачів, програмне забезпечення також може містити модуль, що дозволяє проводити онлайн відео та аудіо трансляції. Однак даний функціонал є досить складним у реалізації та не являється реквізитом прототипу який буде продемонстровано. Тому в даній роботі наведений інструментарій буде опущено.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації платформи для пошуку ментора на основі хмари. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

1.3.1.1 Механізми для взаємодії користувачів онлайн

1.3.1.1.1 Архітектура текстового онлайн чату

Перші програмні застосунки для широкої аудиторії, що дозволяли користувачам спілкуватися один з одним, незалежно від географічної локації, стали з’являтися ще у 2000-х роках. Відомим прикладом є програма Skype, що побачила світ вперше у 2003 році [13].

Хоча колись ця технологія здавалась проривом, зараз вона є набагато доступнішою через появу новітнього протоколу WebSocket. Популярний веб-браузер з відкритим кодом Chromium, яким користуються понад 70% користувачів мережі Інтернет [14], додав підтримку цього протоколу вперше у 2009 році [15].

Для того, щоб зрозуміти що таке протокол WebSocket та як він працює звернемося до офіційної документації. «Протокол WebSocket забезпечує двосторонній зв'язок між клієнтом, що виконує потенційно небезпечний код, та контрольованого сервера який погодився на комунікацію з даним клієнтом. Модель безпеки, що використовується для цього, є типовою моделлю безпеки веб-браузерів. Ця технологія забезпечує механізм для програми, яка потребує двостороннього зв’язку з сервером, який не покладається на відкриття кількох HTTP-з’єднань за допомогою XMLHttpRequest або тривале опитування (long polling)» [16, с1].

Даний протокол складається з початкового рукостискання, за яким ідуть повідомлення, розділені на пакети. Все це відбувається поверх протоколу TCP.

Отже, механізм текстової комунікації між користувачами це не що інше, як обмін повідомленнями за допомогою WebSocket.

1.3.1.1.2 Архітектура онлайн відео чату

Згодом платформи для миттєвого обміну повідомленнями розширялися і набули можливості додатково створювати відео чати й конференції. Згадана раніше платформа Skype додала цю функціональність у 2006 році [17].

На відміну від обміну невеликими повідомленнями, передача аудіо та відео контенту у режимі реального часу одним сервером потребує технічних компромісів та оптимізації.

Виділяють три основні топології систем відео комунікації: multiport control unit (MCU), selective forwarding unit (SFU), peer to peer (P2P). Вони подані на рисунку 1.1.

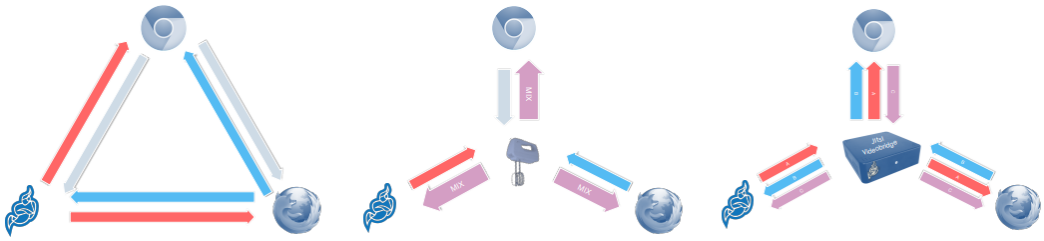

Рисунок 1.1 – Архітектури P2P, MCU, SFU відповідно [18]

MCU топологія вимагає від клієнтів відправляти єдиний потік інформації на головний сервер, який об'єднує дані потоки у єдиний і надсилає його у зворотному напрямку. Таким чином, все навантаження припадає на сервер. При цьому до пристрою клієнта особливих технічних вимог не виставляється, через що даний підхід часто використовується при великій кількості мобільних клієнтів. Прикладом продукту, що використовує даний підхід є Zoom Meetings [19].

В основі SFU підходу лежить центральний сервер який отримує відео та аудіо потоки від клієнтів. На відміну від MCU сервер не займається їх обробкою, а просто транслює всім іншим клієнтам. Головною перевагою цього алгоритму над MCU є його низька затримка і помірні вимоги на головний сервер. Однак при збільшенні кількості одночасних клієнтів у чаті значно зростає кількість даних що віддається з сервера. Це призводить до

значних проблем у масштабуванні, адже мережеві інтерфейси сервера мають обмежену пропускну здатність.

Третьою топологією є P2P. Це повністю децентралізована система де кожен клієнт транслює потоки даних іншим. Як і будь-яка розподілена система вона є надзвичайно стійкою, адже в ній немає єдиної точки відмови, і масштабованою. Проте вона є складнішою у реалізації та системи що її використовують бракують деякої важливої функціональності. Наприклад запису відеоконференцій з метою їх повторного перегляду.

Для того, щоб максимально абстрагувати розробників від налаштування кожного елементу системи окремо компанія Google випустила у 2011 році [20] проект під назвою WebRTC з відкритим кодом призначений полегшити організацію голосового та відеозв'язку у режимі реального часу. Він включений до World Wide Web Consortium (W3C) та підтримується основними веб-браузерами – Chrome, Bind, Opera, Firefox.

Встановлення WebRTC підключення відбувається у дві стадії. Першою стадією є обмін початковими даними про пристрій користувача і встановлення сесії. Причому WebRTC не відповідає за цей крок і не описує як само передача має бути реалізована. Популярним рішенням є використання протоколу WebSocket згаданого вище. Другою стадією є встановлення P2P зв'язку між користувачами. Це є відповідальністю протоколу і зазвичай використовує окремий STUN сервер. Він потрібен для того, щоб обійти всі перетворення мережевих адрес (Network Address Translation – NAT) на шляху між учасниками чату.

Візуально архітектура WebRTC зображена на рисунку 1.2.

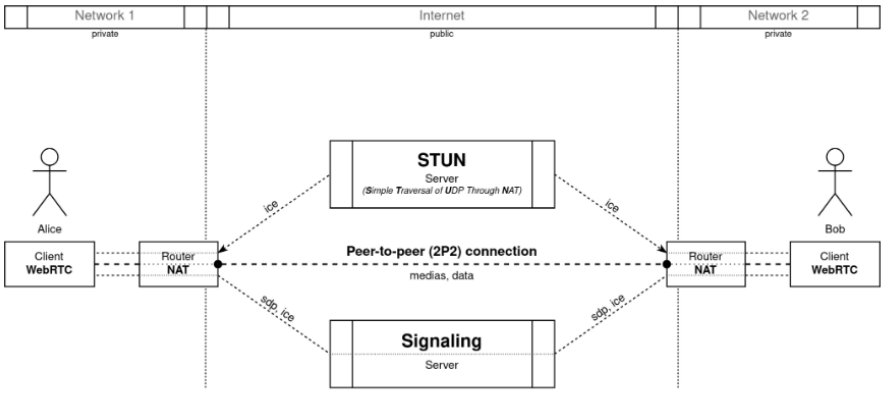

Рисунок 1.2 – Архітектура WebRTC [21]

1.3.1.1.3 Порівняння та висновки

Хоча відео чат є дуже прогресивним і ефективним способом комунікації висока складність реалізації та налаштування не дозволяє включити його у мінімально життєздатний продукт (minimum viable product, MVP). Натомість його аналог – текстовий чат – інтегрується дуже легко і слугує надійним каналом зв’язку ментора та студента.

1.3.1.2 Системи рекомендацій

Рекомендаційна система це система фільтрації інформації, яка розв’язує проблему перевантаження інформацією. Це досягається агрегацією та сортуванням об’єктів згідно з рейтингом їх придатності конкретному користувачеві.

Існує два основних підходи до побудови подібних систем – колаборативна фільтрація (Collaborative filtering) та фільтрація основана на вмісті (Content-Based Filtering) [22].

1.3.1.2.1 Фільтрація основана на вмісті

Такий підхід використовує всю наявну інформацію про об’єкти рекомендації для знаходження їх спільних характеристик. Тобто для кожного об’єкту, наприклад фільму чи ментора, система може швидко знайти подібні елементи та відсортувати їх за схожістю. Як тільки користувач переглядає

випадковий елемент з колекції алгоритм одразу може щось йому запропонувати.

Традиційно для поставленої задачі використовуються графові бази даних [23] адже вони надають можливість комбінувати різні елементи за схожими характеристиками та здатні виконувати комплексні алгоритми на графах швидше за бази даних інших типів. Приклад моделювання даних таким способом поданий на рисунку 1.3.

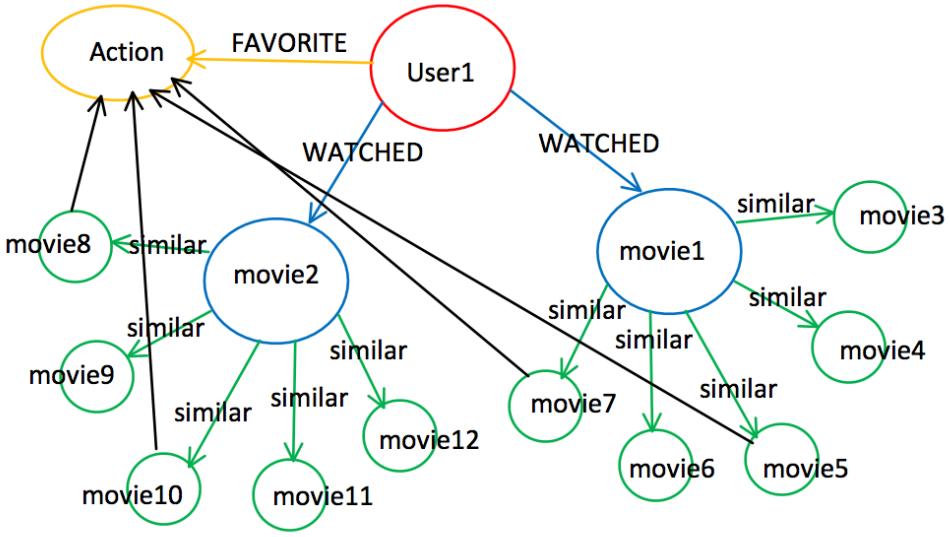

Рисунок 1.3 – Структура рекомендаційної системи у вигляді графа [24]

Недоліками такої системи є велика кількість інформації що необхідно містити в системі та необхідність спеціалізованої нестандартної бази даних.

1.3.1.2.2 Колаборативна фільтрація

Інший метод базується на зборі та аналізі попередніх дій користувача з метою покращення алгоритму підбору у наступній ітерації. Така система намагається класифікувати та розділити аудиторію на невеликі кластери та пропонувати рекомендації кожному користувачеві спираючись на уподобання кластера. Тобто, на противагу фільтрації по змісту, тут аналізується сам користувач, а не предмет рекомендації. Вважається, що регулярної обробки попередніх дій вистачає для виявлення схожих за поведінкою користувачів. Даний алгоритм можна побачити на рисунку 1.4.

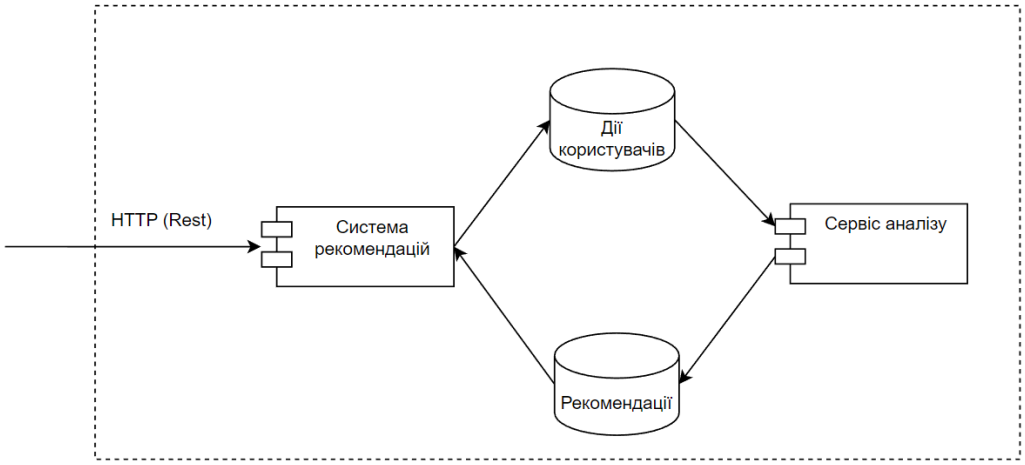

Рисунок 1.4 – Архітектура рекомендаційної системи побудованої за допомогою методу колаборативної фільтрації

Для кластеризації зазвичай використовують алгоритм пошуку найближчих сусідів у графі (KNN, K-Nearest Neighbor algorithm) [25] або нейронні мережі, причому саме вони видають кращі результати [26].

Головним недоліком цього підходу є проблема холодного старту (cold start) – нестачі користувацьких дій при першому запуску системи. Тобто потрібен певний час поки накопичиться мінімальний набір даних що дозволять натренувати першу версію нейронної мережі.

1.3.1.2.3 Порівняння та висновки

В цій роботі буде розглянуто рекомендаційна система на основі колаборативної фільтрації, адже вона не потребує спеціалізованого сховища даних, а популярні бібліотеки, що будуть розглянути далі, дозволяють значно спростити поставлену задачу.

1.3.1.3 Місця та способи розгортання застосунку

Взявши до уваги попередні розділи, очевидно, що програмне забезпечення, що розглядається, складається з великої кількості інфраструктурних елементів. Саме тому потребується ретельне завчасне мислення про підхід до розгортання застосунку. В подальших розділах буде

розглянуто два місця розгортання додатка та два підходи до оновлення робочого артефакту застосунку.

1.3.1.3.1 Місця розгортання програмного забезпечення

1.3.1.3.1.1 Власний сервер (on premise)

Традиційним способом розгортання комплексного веб додатка, що використовувався ще на початку розквіту мережі Інтернет, є його розгортання на власному сервері. Це надає власнику додатка певні переваги:

– повна свобода дій і дуже велика гнучкість в налаштуванні системи;

– можливість використання вже наявних серверів компанії.

Але такий підхід породжує цілий ряд питань на які розробник має знайти відповідь:

– як слідкувати за оновленнями різних елементів системи, наприклад операційними системами серверів;

– як безпечно налаштувати доступ розробників до серверу на якому запущено код або базу даних;

– як ефективно налаштувати мережу яка буде з'єднувати додаток із зовнішнім світом;

– як зібрати метрики про сервер та відобразити їх розробнику;

– як масштабувати систему завчасно при збільшенні кількості користувачів;

– як скопіювати систему в інші регіони світу.

Кожне з цих питань потребує великого ступеня кваліфікації від розробника і зменшує стабільність, надійність і простоту системи в цілому.

1.3.1.3.1.2 Хмара (Infrastructure as a Service, IaaS)

Для вирішення вище згаданих недоліків все частіше і частіше розробники та власники проектів роблять вибір на користь хмари.

Як зазначається на веб-сторінці одного з найбільших постачальників хмарних послуг: «Хмарні обчислення – це можливість використовувати IT-ресурсу на вимогу через Інтернет з оплатою за використання. Замість того, щоб купувати, володіти та підтримувати фізичні центри обробки даних і сервери, ви можете отримати доступ до технологічних послуг, таких як обчислювальна потужність, сховище та бази даних, за потреби від хмарного постачальника» [27].

Даний спосіб надає наступні переваги:

– можливість зменшувати або збільшувати розмір та кількість серверів за лічені хвилини;

– збільшення швидкості експериментів, адже розгортання нових елементів інфраструктури вимагає лише натискання однієї кнопки;

– присутність центрів обробки даних по всьому світу;

– автоматичні оновлення операційних систем;

– повна відсутня необхідність налаштовувати мережі.

Проте він має і певні недоліки:

– розробник має опановувати новий набір технологій та підходів;

– вартість може бути вищою ніж при розгортанні на власний сервер;

– команда все ще має контролювати кількість серверів та налаштовувати правила їх масштабування.

1.3.1.3.1.3 Без сервера (Platform as a Service, PaaS)

Саме останній недолік попереднього пункту, а саме необхідність вираховувати кількість серверів що використовуються, стимулює розвиток «без серверного» (serverless) підходу. Він відрізняється від інших тим, що сам автоматично збільшує або зменшує кількість комп’ютерів, що використовуються, із плином часу. Це дозволяє значно зменшити витрати на інфраструктуру. Детальніше механізм роботи зображено на рисунку 1.5.

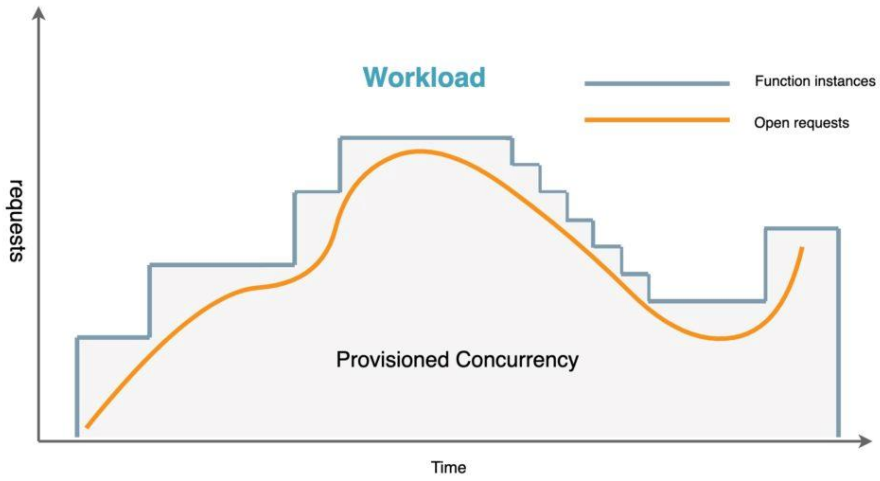

Рисунок 1.5 – Автоматичне масштабування кількості активних екземплярів сервера під час зміни навантаження на систему [28]

1.3.1.3.1.4 Порівняння та висновки

Попередні два пункти добре підсумовує рисунок 1.6.

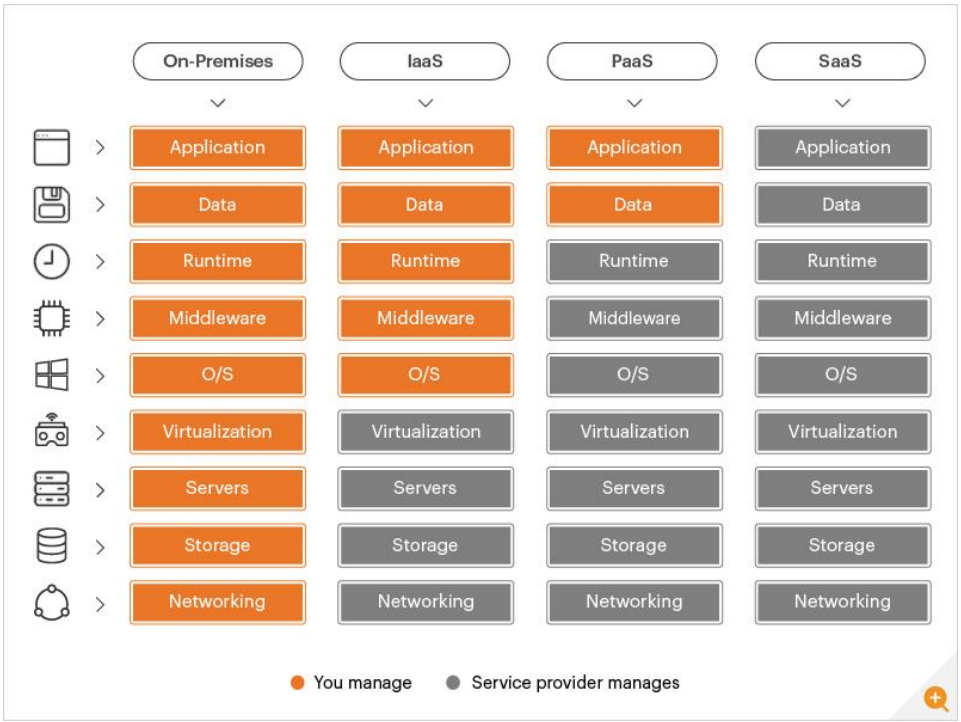

Рисунок 1.6 – Порівняння on-premise та IaaS [29]

Отже, розгортання у хмарі значно спрощує налаштування системи, адже хмарні постачальники беруть на себе задачу забезпечення і налаштування сервера, мережі, сховища та операційної системи. Однак використання такого способу потребує від команди розробки певної

кваліфікації. В даній роботі обрано розгортання у хмарі, а саме комбінація IaaS та PaaS підходів.

1.3.1.3.2 Процес розгортання програмного забезпечення

1.3.1.3.2.1 Вручну

Першим та найпростішим способом розгортання програмного забезпечення на сервері є його періодична збірка та заміна відповідного файлу або файлів членом команди розробки. Алгоритм оновлення при цьому, зазвичай, задокументовано. Хоча цей підхід є простим і швидким при невеликому розмірі додатка, він є дуже ненадійним і не використовується у середніх та великих проєктах.

1.3.1.3.2.2 Автоматичний

Перші спроби автоматизувати процес розгортання відбувалися ще у 2000-х роках. Наприклад, сервер автоматизації Jenkins вперше побачив світ у 2008 році [30]. Саме він поширив дві основні концепції – безперервна інтеграція (continuous integration, CI) та безперервне доставлення або розгортання (continuous delivery, CD).

Безперервна інтеграція належить до процесу розробки та полягає у якомога частішій синхронізації коду над якими працює розробник та коду, що знаходиться у головній гілці гіт репозиторію. Цей процес називають інтеграцією.

Безперервне доставлення є наступним кроком після безперервної інтеграції й відповідає за автоматичну збірку, перевірку і розгортання програмного забезпечення при кожному новому коміті у головну гілку гіт репозиторію.

Ці дві концепції містять у собі безліч переваг:

– зменшення ризику помилки, адже весь процес автоматизовано;

– збільшення швидкості оновлення, адже деякі кроки можуть відбуватися паралельно і не потребують людського нагляду;

– процес стає масштабованим.

Наразі існує велика кількість різних серверів автоматизації. Більшість з них об’єднують можливості безперервної інтеграції та доставлення.

1.3.1.3.2.3 Порівняння та висновки

Отже, автоматизація процесу розгортання є життєво важливим кроком у побудові будь-якого програмного забезпечення. Тому в даній дипломній роботі обрано само автоматичне розгортання додатка.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Аналогічно до попереднього пункту розділимо задачу на три – побудова ефективної онлайн медіа-платформи, розробка алгоритму рекомендації та вибір місця розгортання програмного забезпечення.

Для побудови онлайн медіа-платформи, тобто веб застосунку, влучно буде використати фреймворк NestJS для побудови серверної частини та бібліотеку React – для клієнтської частини додатка. Вони написані однаковою мовою програмування – Typescript – що дозволяє повторно використовувати частини коду.

Typescript – це мова програмування з відкритим кодом, що додає статичну типізацію в іншу відому мову програмування – Javascript. Ключовою перевагою даної мови програмування є її надзвичайна розповсюдженість і, як наслідок, велика кількість різноманітних допоміжних бібліотек. Вона є єдиною мовою програмування що вбудована в усі веб-браузери. Це дозволяє отримати доступ до будь-якого функціонала веб-браузера, наприклад арі протоколу WebSocket.

NestJS – популярний фреймворк для побудови серверних застосунків на мові Typescript за допомогою платформи Node.JS. Відрізняється від інших великою кількістю вбудованого функціонала та можливістю генерувати

шаблонний код викликом команди із командної стрічки, що дозволяє швидко розгортати нові застосунки. Має вбудовані модулі для взаємодії з протоколом WebSocket, авторизації, підключення до більшості баз даних тощо.

Node.JS – платформа для побудови серверних частин застосунків мовою Javascript/Typescript. Має асинхронний ввід/вивід (input/output) та можливість відкладено виконувати код. На відміну від інших платформ, як, наприклад, JVM, вона не створює окремий потік на кожний запит, а використовує повторно вже створені, що дозволяє легко масштабуватися [31].

React – найбільш вживана клієнтська бібліотека що дозволяє будувати адаптивні клієнтські застосунки [32]. Активно розроблюється і підтримується компанією Meta. Має бібліотеки для швидкої побудови широковживаних різноманітних веб елементів: форм, полів вводу, кнопок, графіків тощо.

Для розробки алгоритму рекомендацій буде використано мову програмування Python та бібліотеку TensorFlow.

Python – це інтерпретована мова програмування що активно використовується у напрямку машинного навчання та штучного інтелекту та має багато допоміжних бібліотек таких як TensorFlow, Keras, Pandas. Це є головною причиною її вибору.

TensorFlow – це бібліотека з відкритим кодом для побудови та запуску нейронних мереж. В даній роботі використовується для побудови нейронної мережі системи рекомендацій.

Обидві мови програмування підтримують та заохочують використання менеджерів пакетів для роботи зі сторонніми бібліотеками. Для Javascript це npm, а для Python – pip. Існує одразу декілька переваг у їх використання: економія часу розробника, автоматична перевірка на оновлення використаних бібліотек та перевірка безпечності їх використання.

Для розгортання обрано хмарного постачальника Amazon Web Services, адже він надає понад 200 різних сервісів, багато з яких масштабуються без

втручання розробника. Також це найстаріший хмарний постачальник та має безшовну інтеграцію між сервісами.

Існує декілька інтегрованих середовищ розробки, що підходять для мов Typecript та Python. З одного боку є складні, проте багатофункціональні WebStorm, PyCharm від компанії Intellij, а з іншого боку – модульна Visual Studio Code від компанії Microsoft. В даній роботі обрано модульний текстовий редактор Visual Studio Code через його підтримку одразу обох мов програмування та значній кількості розширень для бібліотеки React.

1.3.3 Аналіз відомих програмних продуктів

Почнемо огляд відомих продуктів з аналізу спеціалізованих систем управління навчання, адже вони цілком можуть використовуватися як середовище спілкування студента і його ментора. Дані застосунки часто використовуються в академічній сфері через їх багатофункціональність. Відомим прикладом є система Moodle. Її інтерфейс зображено на рисунку 1.7 та 1.8.

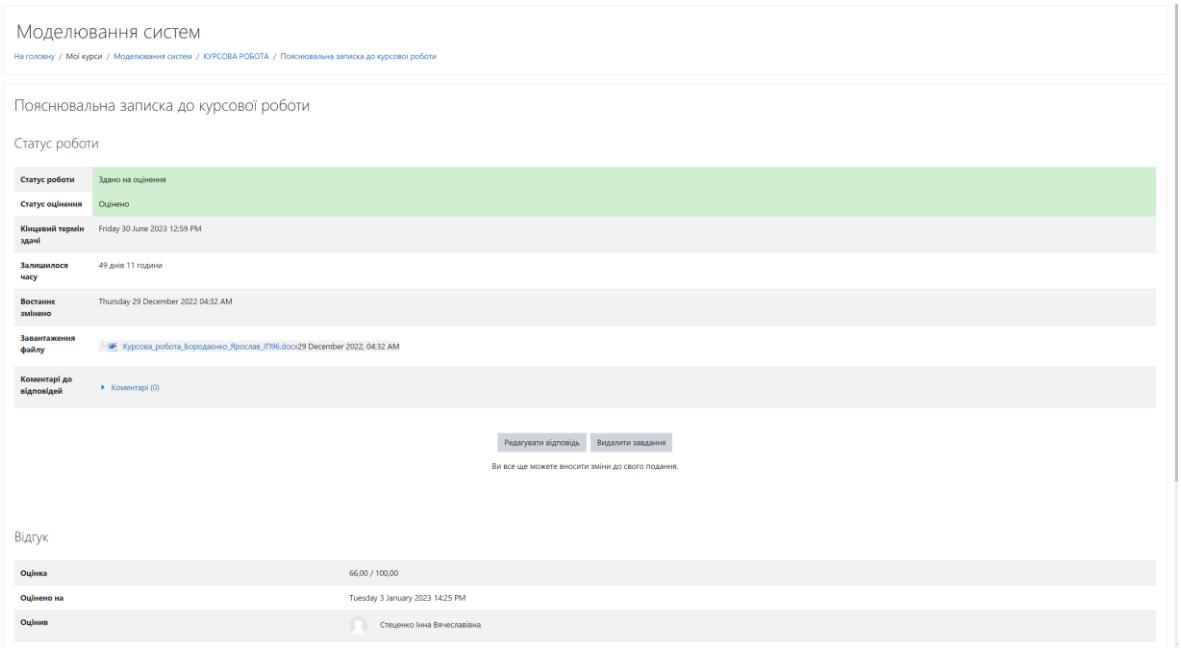

Рисунок 1.7 – Інтерфейс контролю прогресу підопічного системи Moodle

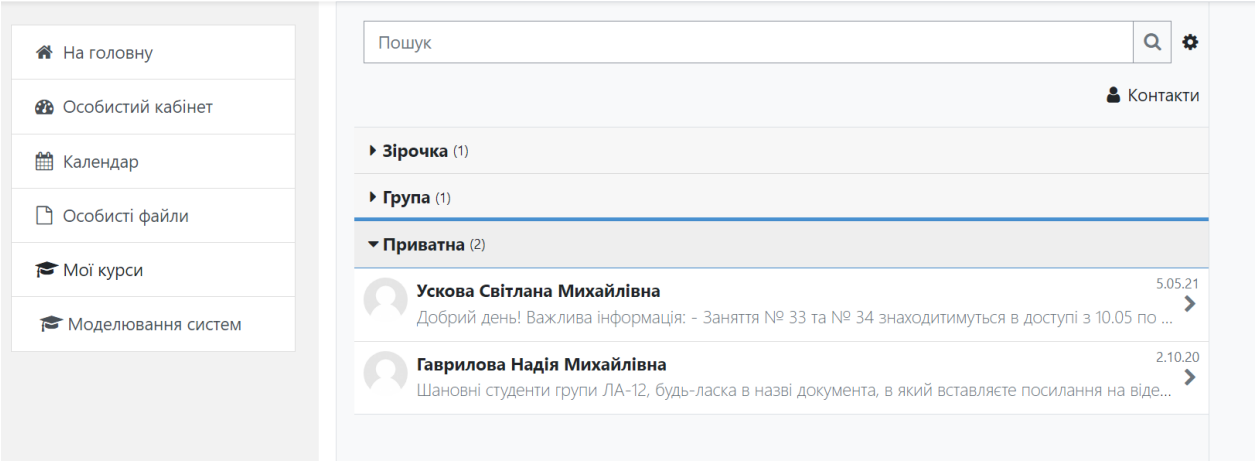

Рисунок 1.8 – Інтерфейс спілкування з ментором системи Moodle

Дана система має наступні переваги:

– відкритий код;

– протестована тисячами університетів по всьому світі.

Також вона має наступні недоліки:

– немає функціонала автоматичного пошуку наставника;

– містить багато модулів які не належать до проблеми що розглядається, як, наприклад, модуль журналу оцінок;

– більше налаштована на офіційний спосіб освіти;

– відсутня можливість шукати однодумців, об'єднуватись у групи та задавати один одному.

Через нестачу відчуття взаємодії з іншими користувачами системи звернемо увагу на популярну у наш час течію – соціальні медіа-платформи. Їх головною перевагою є легкість взаємодії з іншими користувачами що збільшує ймовірність встановлення тісного контакту між ними [33]. Для прикладу візьмемо популярну платформу новин і дискусій Reddit. Її інтерфейс зображено на рисунку 1.9 та 1.10.

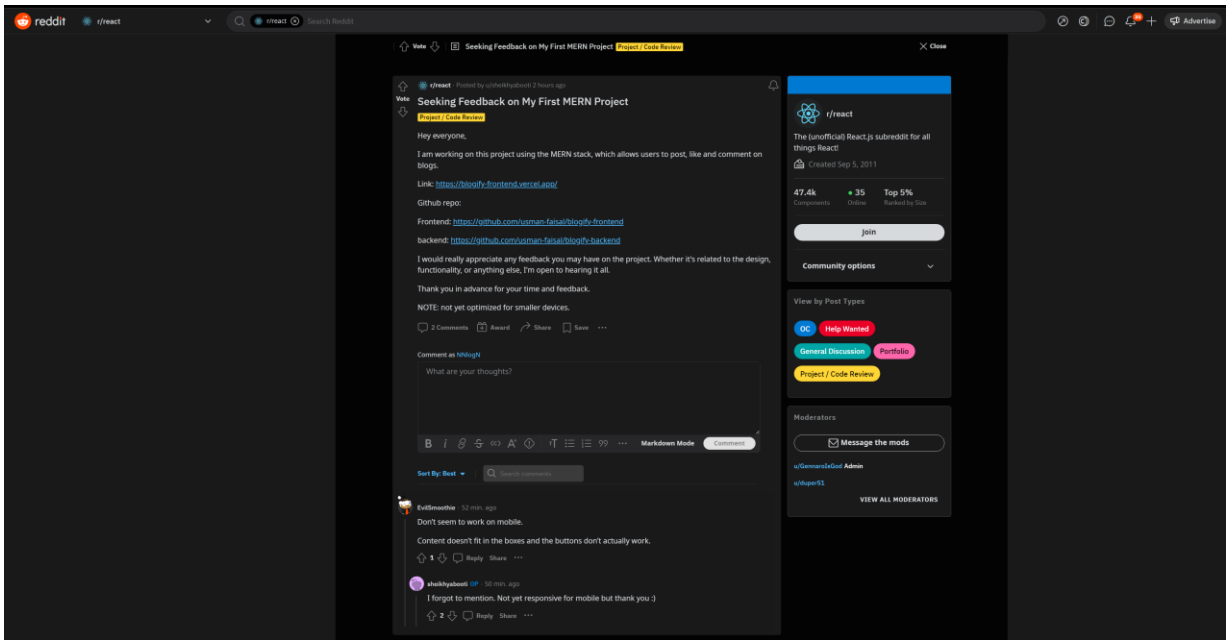

Рисунок 1.9 – Інтерфейс обговорення конкретної проблеми платформи Reddit

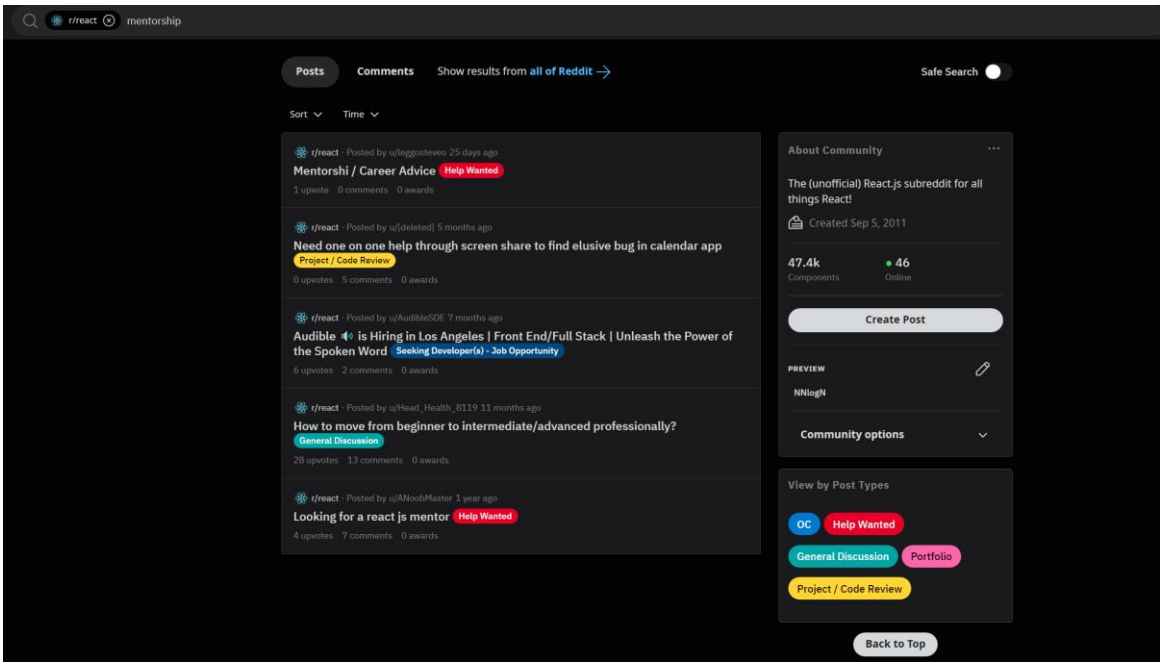

Рисунок 1.10 – Інтерфейс пошуку ментора у конкретному напрямі розробки платформи Reddit

Дана система має наступні переваги:

– дозволяє ставити питання та отримувати на них відповіді від усієї громади;

– дозволяє створювати онлайн чати з одним або багатьма членами;

– має відкритий api, що дозволяє розширювати платформу певним чином, наприклад створювати оголошення з певною періодичністю.

Також вона має наступні недоліки:

– код закритий, хоча сама платформа безплатна;

– не збирає важливої інформації про користувача як, наприклад, його уподобання, хобі та навички;

– ніяк не автоматизує процес підбору ментора.

Перейдемо до застосунків, які орієнтовані на процес менторства.

Першим таким застосунком стане чат-бот ItKpiMentorBot на платформі Telegram. Візуально він поданий на рисунку 1.11.

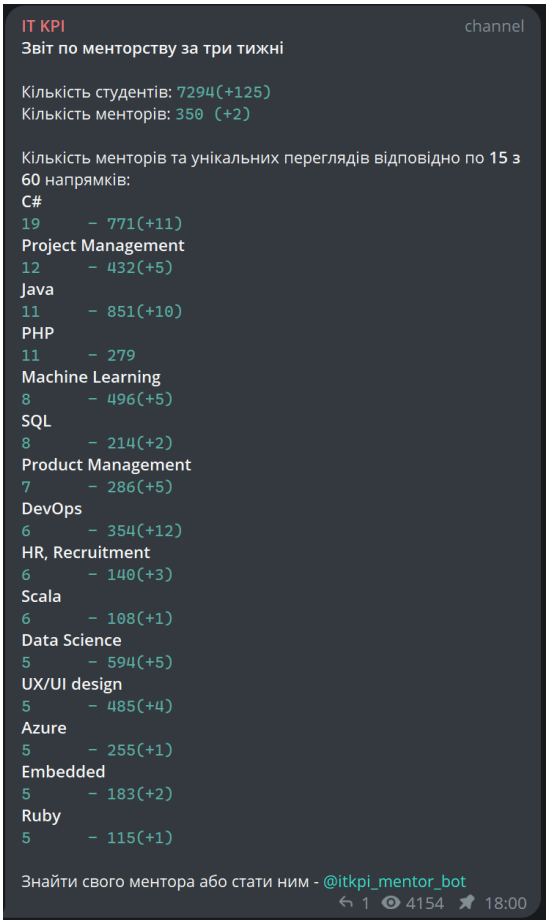

Рисунок 1.11 – Вигляд звіту по менторству за три тижні користування чат-ботом

Даний підхід має певні переваги:

– користувачам не потрібно вчити нові системи, адже вони вже користуються платформою Telegram;

– telegram надає можливості спілкування як у текстовому, так і у відео форматі;

– присутня можливість автоматичного пошуку ментора за обраними напрямками;

– присутня можливість давати оцінку ментору і змінити його якщо на це є потреба.

Але він має і певні недоліки:

– telegram є платформою з повністю закритим кодом, хоча код самого чат-боту відкритий і доступний у системі контролю версій GitHub;

– обмежена можливість ставити й обговорювати питання, адже Telegram не містить звичної системи публікацій і коментарів до неї.

Останньою платформою що розглянеться є codementor.io. Її можна побачити на рисунку 1.12.

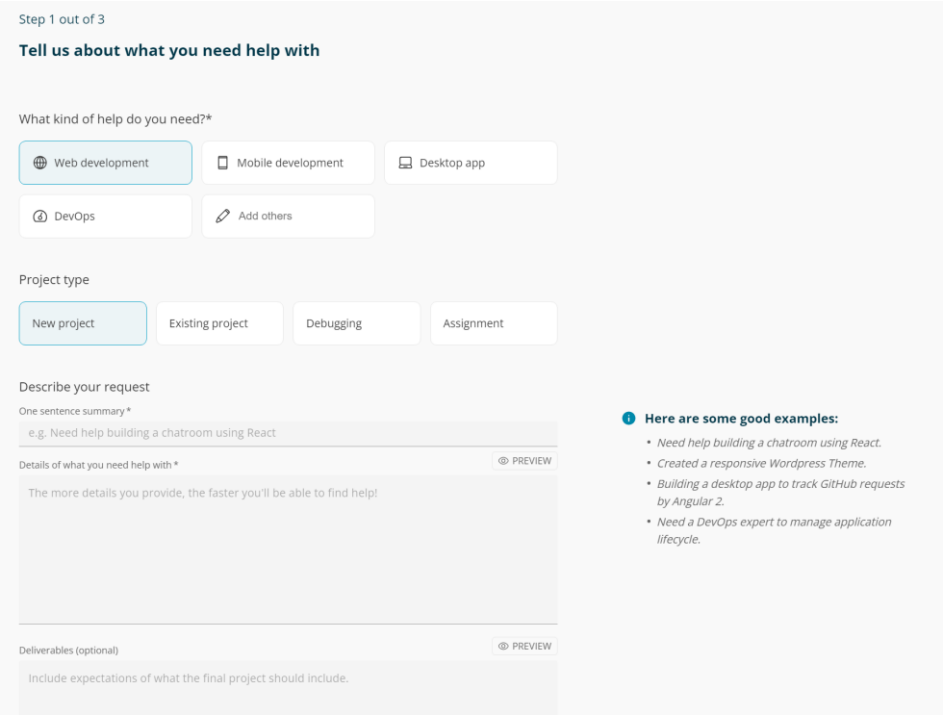

Рисунок 1.12 – Інтерфейс запиту на пошук ментора платформи codementor.io

Дана платформа працює за принципом запит – відповідь. Тобто студент має створити запит на пошук ментора, а наставник – підопічного. Через деякий час система вибере найкращу пару і повідомить про це учасників.

Оцінимо її переваги:

– платформа збирає достатньо інформації про користувача, щоб провести усвідомлений якісний пошук ментора;

– має можливість вказати яка саме поміч потребується від наставника. Наприклад, можна знайти напарника на цікавий проект чи навіть нового робітника в компанію.

Однак вона має і певний ряд недоліків:

– закритий код;

– розробляється маловідомою невеликою компанією;

– має відносно невелику базу активних користувачів;

– не така захоплива як соціальні мережі;

– можливості вільно спілкуватися з іншими користувачами немає.

Для порівняння курсової роботи з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	Дипломний проєкт(назва)	Moodle	Reddit	ItKpiMentorBot	codementors
Дозволяє шукати однодумців і об'єднуватися у спеціалізовані громади	+	-	+	+	-
Дозволяє ставити питання та обговорювати їх з іншими користувачами	+	-	+	+	-
Дозволяє спілкуватися з іншими користувачами у режимі реального часу	+ (тільки текст)	+ (тільки текст)	+ (тільки текст)	+ (текст і відео)	+ (текст і відео)

Продовження таблиці 1.1

Дозволяє задавати користувачеві специфічну доменну інформацію: мови програмування що володіє тощо	+	-	-	+	+
Дозволяє шукати ментора самостійно	+	+	+	+	+
Шукає ментора автоматично	+	-	-	+	+
Надає можливість створювати завдання студентові та оцінювати результати	-	+	-	-	-
Відкритий код	+	+	-	+/-	-
Весь необхідний функціонал безкоштовний	+	+	+	+	+

Отже, з таблиці 1.1 можна зробити висновок що запропонована розробка поєднує в собі найбільші переваги аналогів та зменшує їх недоліки, навіть попри те що вона в деяких незначних моментах уступає конкурентам.

1.4 Аналіз вимог до програмного забезпечення

Дослідивши та проаналізувавши вимоги до програмного забезпечення, а також вивчивши ринок схожих онлайн платформ, було сформовано основні вимоги до функціонала платформи які будуть взяті до уваги в даній дипломній роботі.

Через те, що системою мають можливість користуватися користувачі з різних куточків світу то вона має бути доступною у мережі Інтернет. Як і будь-яка система розрахована на велику кількість користувачів, вона буде містити обов'язкову реєстрацію та подальшу авторизацію користувача. Причому користувач може створити обліковий запис або як студент або як ментор.

Деяка частина функціонала платформи, а саме пошук громад та взаємодія з ними, доступна всім користувачам, незалежно від їх ролі. Студентам також буде надана можливість переглядати профілі менторів та подавати їм заяви на менторство. Наставники, своєю чергою, можуть затверджувати подані їм заяви. Після цього вони зможуть почати обмін текстовими повідомленнями.

Повний список доступного функціоналу системи можна побачити у вигляді діаграми варіантів використання поданої на рисунку 1.13 та у таблицях 1.2 – 1.22.

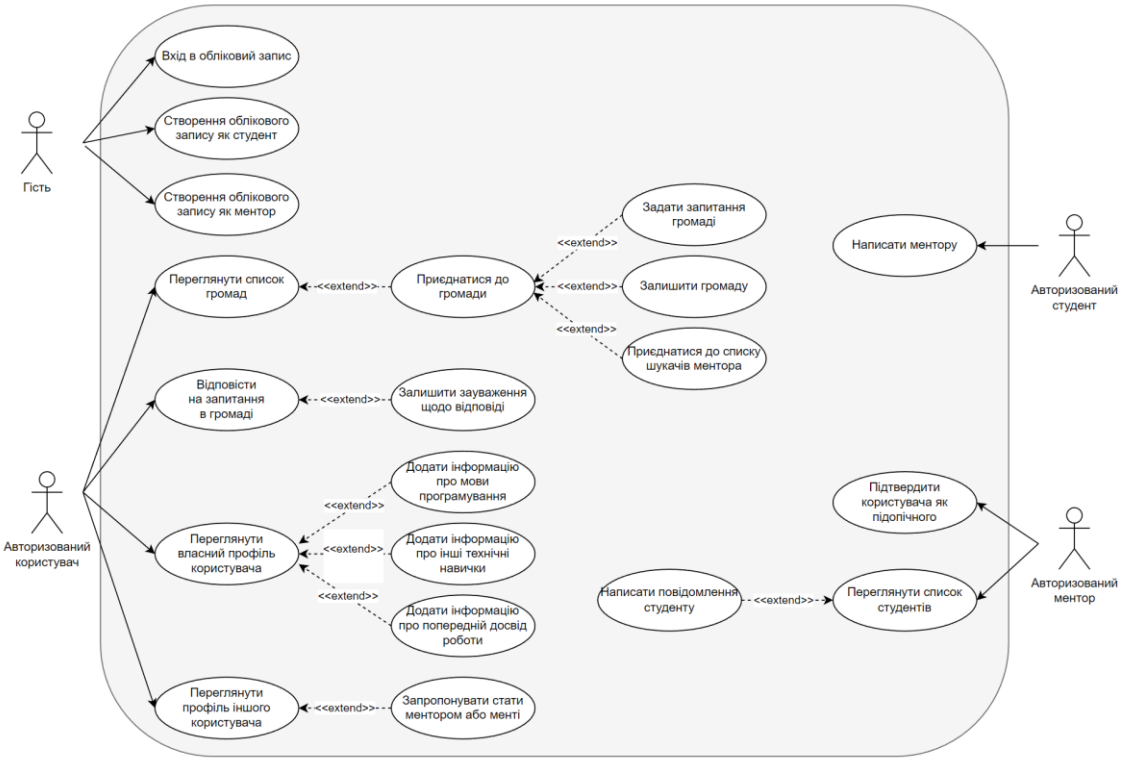

Рисунок 1.13 – Діаграма варіантів використання

Таблиця 1.2 - Варіант використання UC-1

Use case name	Створення облікового запису як студент
Use case ID	UC-1
Goals	Реєстрація нового студента в системі
Actors	Гість (незареєстрований користувач)
Trigger	Студент бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації та натискає кнопку «Зареєструватися як студент». В поля форми для реєстрації вводяться відповідні дані: пошта користувача, пароль в системі, та його повтор для підтвердження. Після заповнення даних користувач натискає кнопку реєстрації. Після цього з’являється текстове повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу.
Extension	У випадку введення не коректних даних, користувач отримує текстове повідомлення з текстом помилки та відповідні поля форми виділяються червоним кольором.
Post-Condition	Створення профілю студента, перехід на сторінку входу

Таблиця 1.3 - Варіант використання UC-2

Use case name	Створення облікового запису як ментор
Use case ID	UC-2
Goals	Реєстрація нового ментора в системі
Actors	Гість (незареєстрований користувач)
Trigger	Ментор бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації та натискає кнопку «Зареєструватися як наставник». В поля форми для реєстрації вводяться відповідні дані: пошта користувача, пароль та його повтор для підтвердження. Після заповнення даних користувач натискає кнопку реєстрації. Після цього з’являється текстове повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу.

Продовження таблиці 1.3

Extension	У випадку введення не коректних даних, користувач отримує текстове повідомлення з текстом помилки та відповідні поля форми виділяються червоним кольором.
Post-Condition	Створення профілю ментора, перехід на сторінку входу

Таблиця 1.4 - Варіант використання UC-3

Use case name	Вхід в обліковий запис
Use case ID	UC-3
Goals	Вхід користувача в його обліковий запис
Actors	Гість (неавторизований користувач)
Trigger	Користувач зареєструвався і бажає авторизуватися
Pre-conditions	Реєстрація завершена
Flow of Events	Користувач переходить на сторінку авторизації. В поля форми вводяться відповідні дані: пошта користувача та пароль в системі. Після заповнення даних користувача натискає кнопку авторизації. Після цього користувач перенаправляється на сторінку його профілю.
Extension	У випадку введення не коректних даних, користувач отримує текстове повідомлення з текстом помилки.
Post-Condition	Вхід користувача, перехід на сторінку профілю

Таблиця 1.5 - Варіант використання UC-4

Use case name	Переглянути список громад
Use case ID	UC-4
Goals	Відобразити користувачу список доступних громад
Actors	Авторизований користувач
Trigger	Користувач бажає передивитися список тематичних громад
Pre-conditions	-

Продовження таблиці 1.5

Flow of Events	Користувач переходить на сторінку списку громад. Кожний елемент списку має наступні дані: назву, логотип та кількість користувачів. Також кожна громада має кнопку «Приєднатися», якщо користувач не входить у неї, або «Покинути», якщо він є її членом.
Extension	У випадку відсутності доступних громад користувачу буде відображено наступний текст: «Список доступних громад порожній. Спробуйте пізніше».
Post-Condition	-

Таблиця 1.6 - Варіант використання UC-5

Use case name	Приєднатися до громади
Use case ID	UC-5
Goals	Приєднати користувача до громади
Actors	Авторизований користувач
Trigger	Користувач бажає приєднатися до громади
Pre-conditions	Користувач знаходиться на сторінці списку громад
Flow of Events	Користувач натискає на кнопку «Приєднатися», що знаходиться справа від назви громади.
Extension	-
Post-Condition	Користувач приєднується до громади

Таблиця 1.7 - Варіант використання UC-6

Use case name	Залишити громаду
Use case ID	UC-6
Goals	Видалити користувача з громади
Actors	Авторизований користувач
Trigger	Користувач бажає залишити громаду
Pre-conditions	Користувач знаходиться на сторінці громад, до якої від приєднаний
Flow of Events	Користувач натискає на кнопку «Залишити», що знаходиться справа від назви громади.

Продовження таблиці 1.7

Extension	-
Post-Condition	Користувач залишає громаду, але може повторно до неї приєднатися.

Таблиця 1.8 - Варіант використання UC-7

Use case name	Приєднатися до списку шукачів ментора
Use case ID	UC-7
Goals	Додати користувача до списку шукачів ментора
Actors	Авторизований користувач
Trigger	Користувач бажає знайти ментора
Pre-conditions	Користувач знаходиться на сторінці громади
Flow of Events	Користувач натискає на кнопку «Я бажаю знайти ментора», що знаходиться справа від назви громади.
Extension	-
Post-Condition	Користувач може переглянути список вільних менторів.

Таблиця 1.9 - Варіант використання UC-8

Use case name	Поставити запитання громаді
Use case ID	UC-8
Goals	Створити запитання до громади
Actors	Авторизований користувач
Trigger	Користувач бажає поставити запитання громаді
Pre-conditions	Користувач знаходиться на сторінці громади
Flow of Events	Користувач натискає на кнопку «Поставити запитання», що знаходиться справа від назви громади. У формі, що з’являється, він набирає текст свого запитання і натискає кнопку «Зберегти та відправити».
Extension	-
Post-Condition	Користувач бачить своє запитання у списку запитань

Таблиця 1.10 - Варіант використання UC-9

Use case name	Відповісти на запитання в громаді
Use case ID	UC-9
Goals	Надати відповідь на запитання у громади
Actors	Авторизований користувач
Trigger	Користувач бажає відповісти на поставлене запитання
Pre-conditions	Користувач знаходиться на сторінці запитання
Flow of Events	Користувач натискає на кнопку «Відповісти», що знаходиться під низом від тексту запитання. У текстовому полі, що з’являється, він набирає текст відповіді й натискає кнопку «Зберегти та відправити».
Extension	-
Post-Condition	Користувач бачить своє свою відповідь.

Таблиця 1.11 - Варіант використання UC-10

Use case name	Залишити зауваження щодо відповіді
Use case ID	UC-10
Goals	Залишити зауваження щодо раніше наданої відповіді
Actors	Авторизований користувач
Trigger	Користувач бажає надати правки щодо раніше наданої відповіді
Pre-conditions	Користувач знаходиться на сторінці запитання
Flow of Events	Користувач натискає на кнопку «Відповісти», що знаходиться під низом від тексту відповіді. У текстовому полі, що з’являється, він набирає текст відповіді й натискає кнопку «Зберегти та відправити».
Extension	-
Post-Condition	Користувач бачить своє свої правки.

Таблиця 1.12 - Варіант використання UC-11

Use case name	Переглянути власний профіль користувача
Use case ID	UC-11

Продовження таблиці 1.12

Goals	Переглянути власний профіль користувача
Actors	Авторизований користувач
Trigger	Користувач бажає переглянути власний профіль
Pre-conditions	-
Flow of Events	Користувач натискає на своє ім’я, що знаходиться справа згори. Після цього користувач перенаправляється на сторінку його профілю, де він може побачити своє ім’я список мов програмування якими він володіє, список інших навичок та попередній досвід роботи.
Extension	-
Post-Condition	Користувач бачить сторінку власного профілю

Таблиця 1.13 - Варіант використання UC-12

Use case name	Додати інформацію про мови програмування
Use case ID	UC-12
Goals	Оновити список мов програмування на сторінці профілю
Actors	Авторизований користувач
Trigger	Користувач бажає оновити список мов програмування на сторінці профілю
Pre-conditions	Користувач знаходиться на сторінці власного профілю
Flow of Events	Користувач натискає на кнопку «Додати», що знаходиться справа від списку мов програмування. У текстовому полі, що з’являється, він набирає текст і натискає кнопку «Зберегти».
Extension	-
Post-Condition	Користувач бачить оновлений список.

Таблиця 1.14 - Варіант використання UC-13

Use case name	Додати інформацію про інші технічні навички
Use case ID	UC-13
Goals	Оновити список інших технічних навичок на сторінці профілю
Actors	Авторизований користувач

Продовження таблиці 1.14

Trigger	Користувач бажає оновити список інших технічних навичок на сторінці профілю
Pre-conditions	Користувач знаходиться на сторінці власного профілю
Flow of Events	Користувач натискає на кнопку «Додати», що знаходиться справа від списку технічних навичок. У текстовому полі, що з’являється, він набирає текст і натискає кнопку «Зберегти».
Extension	-
Post-Condition	Користувач бачить оновлений список.

Таблиця 1.15 - Варіант використання UC-14

Use case name	Додати інформацію про попередній досвід роботи
Use case ID	UC-14
Goals	Оновити попередній досвід роботи на сторінці профілю
Actors	Авторизований користувач
Trigger	Користувач бажає оновити попередній досвід роботи на сторінці профілю
Pre-conditions	Користувач знаходиться на сторінці власного профілю
Flow of Events	Користувач натискає на кнопку «Додати», що знаходиться справа від інформації про попередні місця роботи. У текстовому полі, що з’являється, він набирає текст і натискає кнопку «Зберегти».
Extension	-
Post-Condition	Користувач бачить оновлений список.

Таблиця 1.16 - Варіант використання UC-15

Use case name	Переглянути профіль іншого користувача
Use case ID	UC-15
Goals	Відобразити користувачу профіль іншого користувача
Actors	Авторизований користувач
Trigger	Користувач бажає профіль іншого користувача
Pre-conditions	-

Продовження таблиці 1.16

Flow of Events	Користувач натискає на кнопку ім’я користувача який його цікавить. Після цього він перенаправляється на сторінку профілю даного користувача.
Extension	-
Post-Condition	Користувач бачить профіль іншої людини.

Таблиця 1.17 - Варіант використання UC-16

Use case name	Запропонувати стати ментором
Use case ID	UC-16
Goals	Запропонувати ментору взяти нового студента
Actors	Авторизований студент
Trigger	Користувач бажає стати студентом у ментора
Pre-conditions	Користувач знаходиться на сторінці профілю ментора
Flow of Events	Користувач натискає на кнопку «Приєднатися», що знаходиться справа від ім’я ментора.
Extension	-
Post-Condition	Ментор дізнається про нового користувача який хоче стати його студентом.

Таблиця 1.18 - Варіант використання UC-17

Use case name	Написати ментору
Use case ID	UC-17
Goals	Надіслати текстове повідомлення від студента до ментора
Actors	Авторизований студент
Trigger	Студент бажає написати його ментору
Pre-conditions	Користувач знаходиться на сторінці профілю ментора
Flow of Events	Користувач натискає на кнопку «Надіслати повідомлення», що знаходиться справа від ім’я ментора. У текстовому полі, що з’являється він набирає текст повідомлення і натискає «Відправити».
Extension	-
Post-Condition	Ментор дізнається про нове повідомлення від його студента.

Таблиця 1.19 - Варіант використання UC-18

Use case name	Підтвердити користувача як підопічного
Use case ID	UC-18
Goals	Підтвердити користувача як підопічного
Actors	Авторизований ментор
Trigger	Користувач подав заяву ментору
Pre-conditions	Користувач подав заяву ментору
Flow of Events	Ментор натискає на кнопку «Підтвердити», що знаходиться справа від ім’я користувача.
Extension	-
Post-Condition	Користувач стає студентом ментора.

Таблиця 1.20 - Варіант використання UC-19

Use case name	Переглянути список студентів
Use case ID	UC-19
Goals	Відобразити ментору список його студентів
Actors	Авторизований ментор
Trigger	Ментор бажає переглянути список його студентів
Pre-conditions	-
Flow of Events	Ментор натискає на кнопку «Студенти», що знаходиться в лівому верхньому куті. Після цього він перенаправляється на сторінку зі списком його студентів.
Extension	-
Post-Condition	-

Таблиця 1.21 - Варіант використання UC-20

Use case name	Написати повідомлення студенту
Use case ID	UC-20
Goals	Надіслати текстове повідомлення від ментора до студента
Actors	Авторизований ментор
Trigger	Ментор бажає написати його студенту
Pre-conditions	Ментор знаходиться на сторінці його студентів

Продовження таблиці 1.21

Flow of Events	Ментор натискає на кнопку «Надіслати повідомлення», що знаходиться правіше від імені студента. У текстовому полі він вводить своє повідомлення і натискає «Відправити».
Extension	-
Post-Condition	Студент дізнається про нове повідомлення від його ментора.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на два модулі – соціальна медіа-платформа та алгоритм рекомендації ментора студенту. Розглянемо функціональні вимоги до кожного модуля окремо.

Соціальна медіа-платформа має наступні вимоги, що представлені у загальному вигляді у таблиці 1.22 та детально у таблицях 1.23 - 1.39.

Таблиця 1.22 – Загальна модель вимог

Номер	Вимога	Код
1	Реєстрація користувача	FR-1
2	Вхід користувача	FR-2
2.1	Перегляд профілю	FR-3
2.2	Редагування профілю	FR-4
3	Відправка запиту менту	FR-5
3.1	Підтвердження запиту ментором	FR-6
3.2	Перегляд списку студентів	FR-7
4	Відправка повідомлення ментору	FR-8
4.1	Отримання повідомлення від ментора	FR-9
4.2	Відправка повідомлення студенту	FR-10
4.3	Отримання повідомлення від студента	FR-11
5	Перегляд списку громад	FR-12
5.1	Приєднання до громади	FR-13
5.2	Покидання громади	FR-14
5.3	Поставити запитання громаді	FR-15

Продовження таблиці 1.22

6	Надання відповідь на запитання	FR-16
7	Надати правки до відповіді	FR-17
8	Додавання до списку шукачів студента	FR-18
9	Вибір найкращого ментора	FR-19

Таблиця 1.23 – Функціональна вимога FR-1

Назва	Реєстрація користувача
Опис	Користувач може зареєструватися у системі за допомогою пошти та паролю.

Таблиця 1.24 – Функціональна вимога FR-2

Назва	Вхід користувача
Опис	Користувач може увійти в систему за допомогою пошти та паролю.

Таблиця 1.25 – Функціональна вимога FR-3

Назва	Перегляд профілю
Опис	Користувач може переглянути профіль довільного користувача.

Таблиця 1.26 – Функціональна вимога FR-4

Назва	Редагування профілю
Опис	Користувач може відредагувати власний профіль.

Таблиця 1.27 – Функціональна вимога FR-5

Назва	Відправка запиту менту
Опис	Користувач може надіслати запит ментору.

Таблиця 1.28 – Функціональна вимога FR-6

Назва	Підтвердження запиту ментором
Опис	Ментор може підтвердити запит і стати наставником студента.

Таблиця 1.29 – Функціональна вимога FR-7

Назва	Перегляд списку студентів
Опис	Ментор може переглянути список своїх студентів.

Таблиця 1.30 – Функціональна вимога FR-8

Назва	Відправка повідомлення ментору
Опис	Студент може відправити повідомлення ментору.

Таблиця 1.31 – Функціональна вимога FR-9

Назва	Отримання повідомлення від ментора
Опис	Студент може переглянути повідомлення від ментора.

Таблиця 1.32 – Функціональна вимога FR-10

Назва	Відправка повідомлення студенту
Опис	Ментор може відправити повідомлення студенту.

Таблиця 1.33 – Функціональна вимога FR-11

Назва	Отримання повідомлення від студента
Опис	Ментор може переглянути повідомлення від студента.

Таблиця 1.34 – Функціональна вимога FR-12

Назва	Перегляд списку громад
Опис	Користувач може переглянути список тематичних громад

Таблиця 1.35 – Функціональна вимога FR-13

Назва	Приєднання до громади
Опис	Користувач може приєднатися до тематичних громад

Таблиця 1.36 – Функціональна вимога FR-14

Назва	Покидання громади
Опис	Користувач може покинути громаду

Таблиця 1.37 – Функціональна вимога FR-15

Назва	Поставити запитання громаді
Опис	Користувач може поставити запитання громаді

Таблиця 1.38 – Функціональна вимога FR-16

Назва	Надання відповідь на запитання
Опис	Користувач може відповісти на поставлене запитання

Таблиця 1.39 – Функціональна вимога FR-17

Назва	Надати правки до відповіді
Опис	Користувач може надати правки до відповіді

Алгоритм автоматичного підбору ментору має вимоги що представлені у таблицях 1.40 – 1.41.

Таблиця 1.40 – Функціональна вимога FR-18

Назва	Додавання до списку шукачів студента
Опис	Ментор може додати себе до списку людей, що знаходяться у пошуку студентів

Таблиця 1.41 – Функціональна вимога FR-19

Назва	Вибір найкращого ментора
Опис	Студент може передивитися список менторів та вибрати кращого самостійно або попросити рекомендацію у системи.

Матрицю трасування вимог можна побачити на рисунку 1.14. Модель функціональних вимог у загальному вигляді представлена на рисунку 1.15.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19
UC-1	*																		
UC-2	*																		
UC-3		*																	
UC-4												*							
UC-5													*						
UC-6														*					
UC-7																		*	*
UC-8															*				
UC-9																*			
UC-10																	*		
UC-11			*																
UC-12			*																
UC-13				*															
UC-14				*															
UC-15				*															
UC-16					*														
UC-17								*	*										
UC-18						*													
UC-19							*												
UC-20										*	*								

Рисунок 1.14 – Матриця трасування вимог

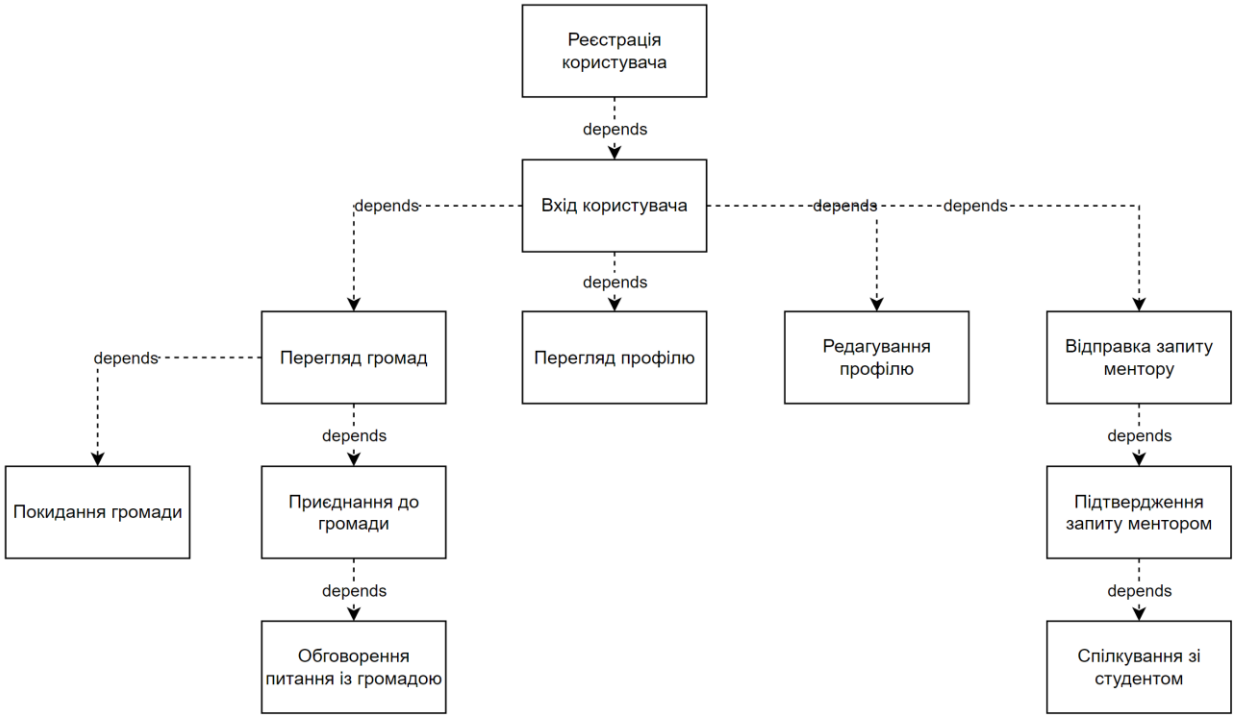

Рисунок 1.15 – Модель функціональних вимог у загальному вигляді

1.4.2 Розроблення нефункціональних вимог

Виділимо наступні нефункціональні вимоги:

– система повинна мати веб-інтерфейс англійською мовою;

– інтерфейс системи має бути схожим на всіх сторінках;

– система повинна підтримувати спілкування у режимі реального часу;

– система повинна автоматично розгортатися у хмарі;

– система повинна бути масштабованою;

– система повинна мати відкритий код.

1.5 Постановка задачі

Дане програмне забезпечення призначено для того, щоб спростити процес пошуку ментора у сфері інформаційних технологій, внаслідок автоматизації вибору відповідного наставника.

Метою є розробка масштабованої онлайн медіа-платформи для програмістів яка має заохочувати їх користуватися нею та автоматично єднати ментора та, відповідно до його навичок та інтересів, студента.

Для досягнення поставленої мети необхідно розв’язати наступні задачі:

– додавання нових користувачів до системи;

– створення тематичних громад;

– обмін досвідом всередині громад за допомогою запитань та відповідей;

– отримання рекомендації щодо кращого ментора;

– консультація з ментором в режимі реального часу за допомогою текстових повідомлень;

– автоматичне розгортання системи у хмару.

Висновки до розділу

В цьому розділі були описані базові положення та загальний опис предметної області для її кращого розуміння.

Далі було розглянуто її основні проблеми та алгоритми їх вирішення. Було порівняно основні способи комунікації між користувачами мережі Інтернет у режимі реального часу, як у форматі текстового чату, так і відео конференції. Також було описано два основних підходи до побудови рекомендаційних користувацьких систем. Вибір фінального рішення проводився після порівняння переваг і недоліків запропонованих підходів.

Також було розглянуто та порівняно відомі програмні розв’язки поставленої проблеми на основі цілого ряду критеріїв. Було враховано їх мінуси та описано плюси.

Останнім кроком став опис основних сценаріїв використання даного програмного забезпечення та сформульовано ряд функціональних вимог. Окрім цього, було визначено набір нефункціональних вимог які ставлять на меті зробити систему більш надійною та зручною у користуванні.

На основі виконаної роботи було сформульовано постановку задачі.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Перед початком реалізації програмного забезпечення, необхідно детально описати всі бізнес-процеси, що відбувається. Такими процесами є:

а) пошук та приєднання до громади;

б) обговорення запитання з громадою;

в) пошук ментора вручну;

г) пошук ментора за допомогою рекомендацій;

д) комунікація студента з ментором.

Представлені процеси будуть в вигляді діаграм діяльності, що представлені на рисунку 2.1 – 2.5 відповідно.

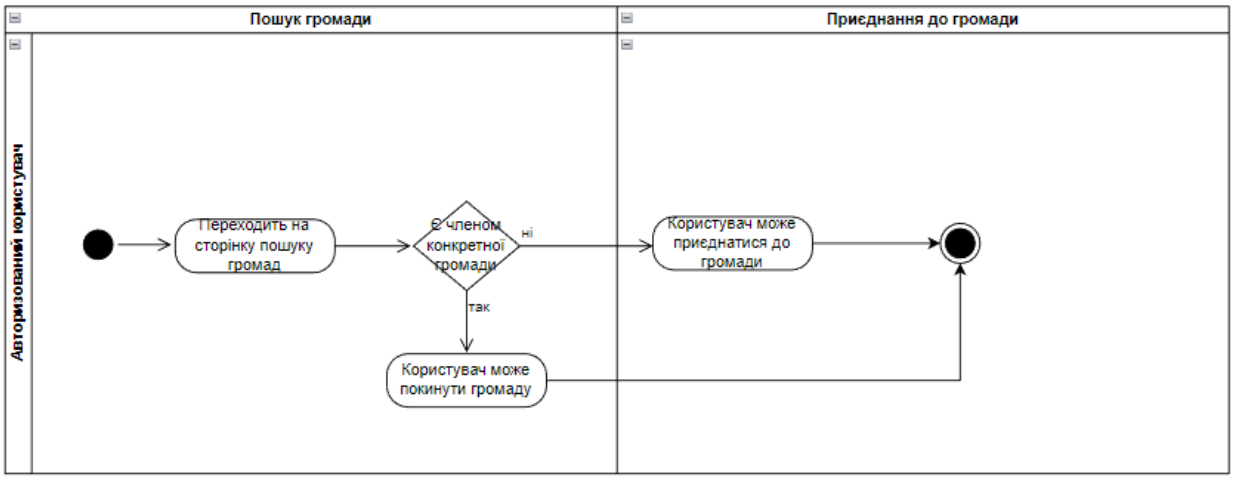

Рисунок 2.1 – Діаграма діяльності процесу пошуку та приєднання до громади

Послідовний опис процесу пошуку та приєднання до громади користувачем:

а) користувач заходить на сторінку зі списком громад;

б) користувач натискає на кнопку «Приєднатися», що знаходиться праворуч від імені громади;

в) користувач перенаправляється на сторінку громади.

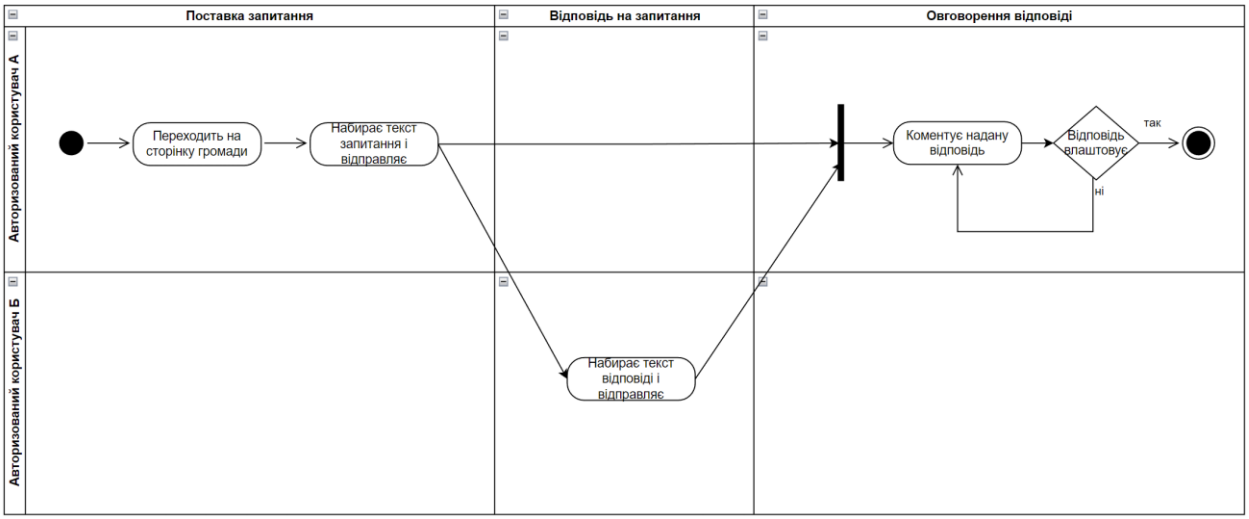

Рисунок 2.2 – Діаграма діяльності процесу обговорення запитання з громадою

Послідовний опис процесу обговорення питання з громадою:

а) користувач заходить на сторінку громади;

б) користувач натискає на кнопку «Поставити запитання», що знаходиться праворуч від імені громади;

в) користувач набирає текст запитання і натискає «Відправити»;

г) інший користувач бачить запитання і натискає на кнопку «Відповісти». Після вводу тексту відповіді він натискає «Відправити»;

д) користувач бачить відповідь і може прокоментувати її.

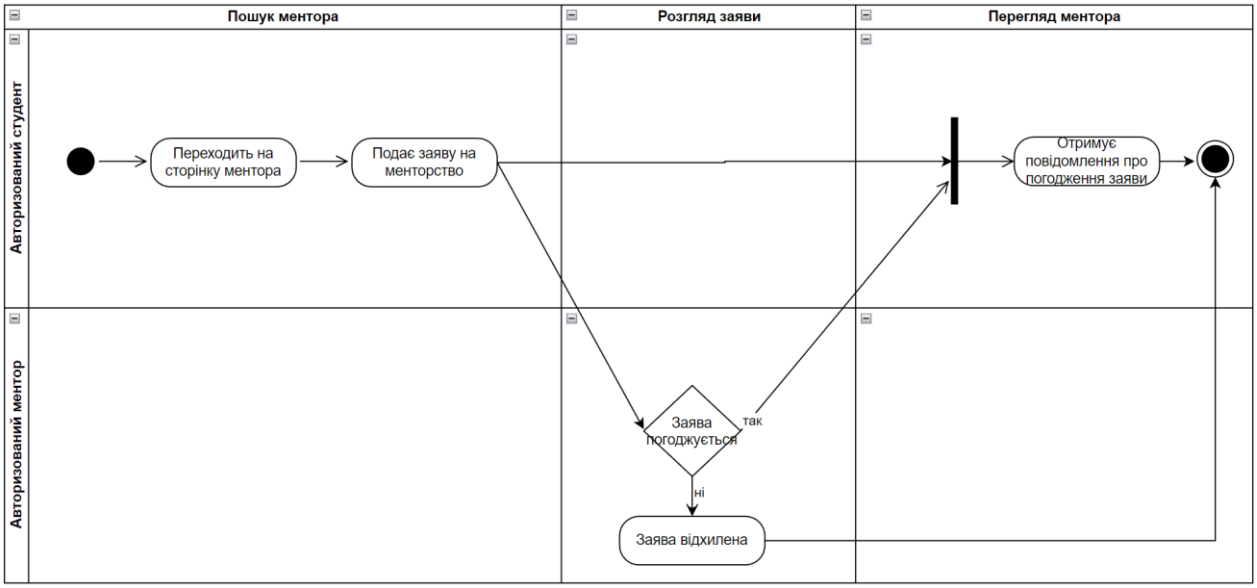

Рисунок 2.3 – Діаграма діяльності процесу пошуку ментора вручну

Послідовний опис процесу пошуку ментора вручну:

а) користувач заходить на сторінку громади;

б) користувач натискає на кнопку «Поставити запитання», що знаходиться праворуч від імені громади;

в) користувач набирає текст запитання і натискає «Відправити»;

г) інший користувач бачить запитання і натискає на кнопку «Відповісти». Після вводу тексту відповіді він натискає «Відправити»;

д) користувач бачить відповідь і може прокоментувати її.

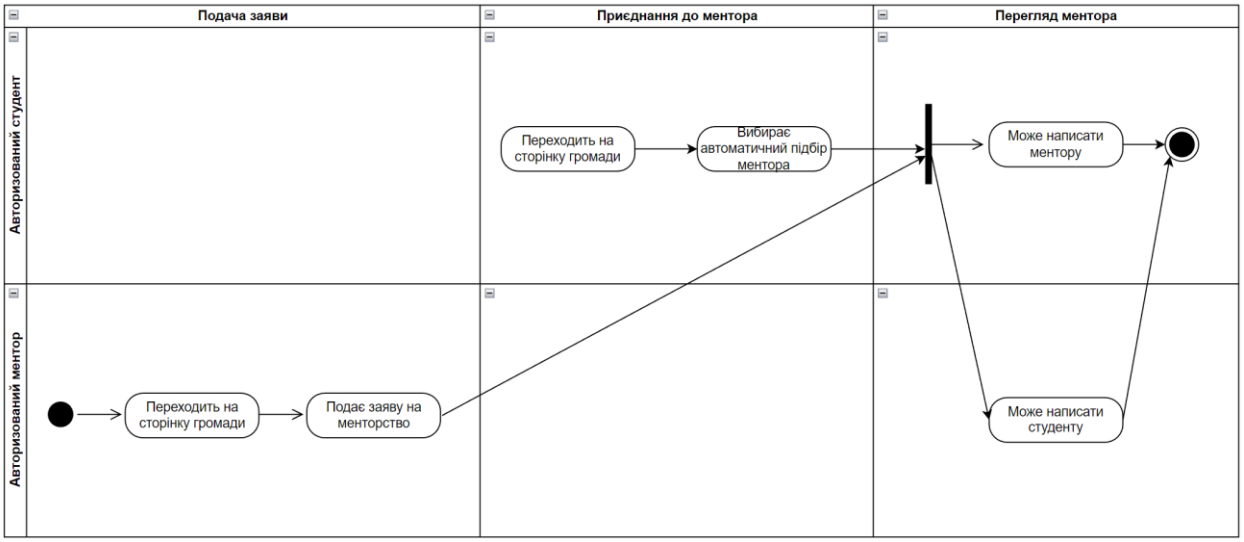

Рисунок 2.4 – Діаграма діяльності процесу пошуку ментора за допомогою рекомендації

Послідовний опис процесу пошуку наставника за допомогою рекомендацій:

а) ментор заходить на сторінку громади;

б) ментор натискає на кнопку «Стати ментором», що знаходиться праворуч від імені громади;

в) студент заходить на сторінку громади;

г) студент натискає на кнопку «Запропонувати ментора»;

д) студент перенаправляється на сторінку профілю ментора.

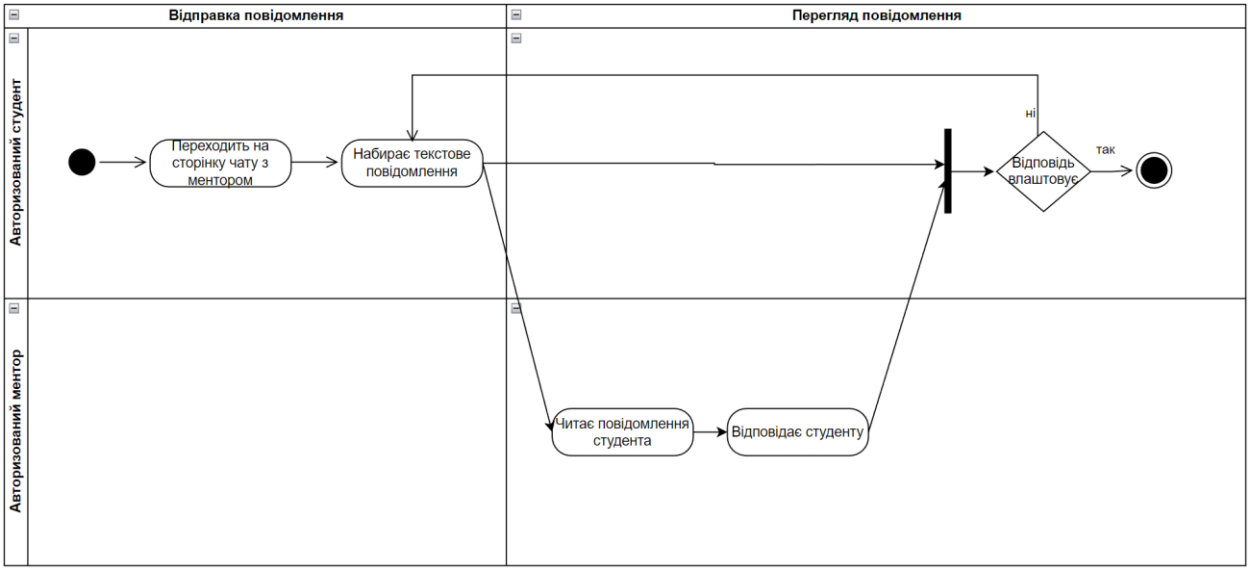

Рисунок 2.5 – Діаграма діяльності процесу комунікації з ментором

Послідовний опис процесу комунікації з ментором:

а) студент заходить на сторінку чату з ментором;

б) студент набирає текстове повідомлення та натискає на кнопку «Надіслати»;

в) ментор отримає сповіщення про нове повідомлення від студента;

г) ментор набирає текстове повідомлення та натискає на кнопку «Надіслати»;

д) студент отримає сповіщення про нове повідомлення від ментора.

2.2 Архітектура програмного забезпечення

Через високу технічну складність запропонованого рішення його архітектура буде розглянута у три кроки. Першим етапом стане загальний огляд сервісів, їх функцій та взаємодії один з одним. Далі буде описано інфраструктурні елементи хмарного постачальника AWS що дозволяються розгорнути необхідні сервіси. Останнім кроком стане детальний огляд кожного сервісу окремо на прикладному рівні.

2.2.1 Високорівнева архітектура веб-застосування

З архітектурної точно зору додаток можна розділити на 3 частини: клієнтську частину, сервіс що відповідає за рекомендації та основний сервіс, що відповідає за управління користувачами та громадами. Діаграма такої архітектури зображена на рисунку 2.6.

Розділ серверного додатка на 2 частини має сенс, адже вони мають зовсім різні вимоги до часу відповіді та надійності. Сервер рекомендацій відповідає за запуск нейронних мереж і потребує більш потужних серверів. Також покладена на нього функціональність не є критичною, а отже допускаються простої. Основний сервіс, в свою чергу, відповідає за критично важливі процеси системи – керування користувачами, громадами, чатами. Це означає, що він має бути якомога більш надійним.

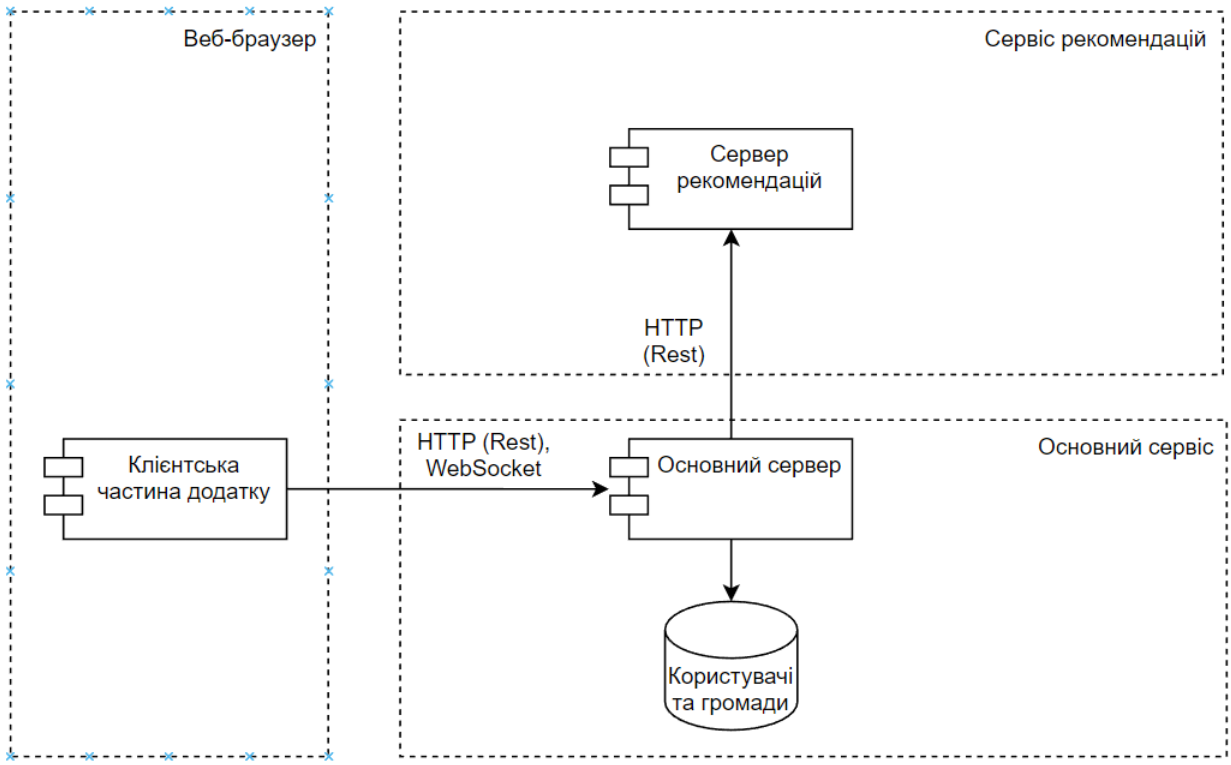

Рисунок 2.6 – Діаграма високорівневої архітектури веб-застосування

2.2.2 Інфраструктурна архітектура веб-застосування

Перенесемо запропоновану архітектуру на сервіси, що пропонує хмарний провайдер Amazon Web Services. Діаграма архітектури подана на рисунку 2.7.

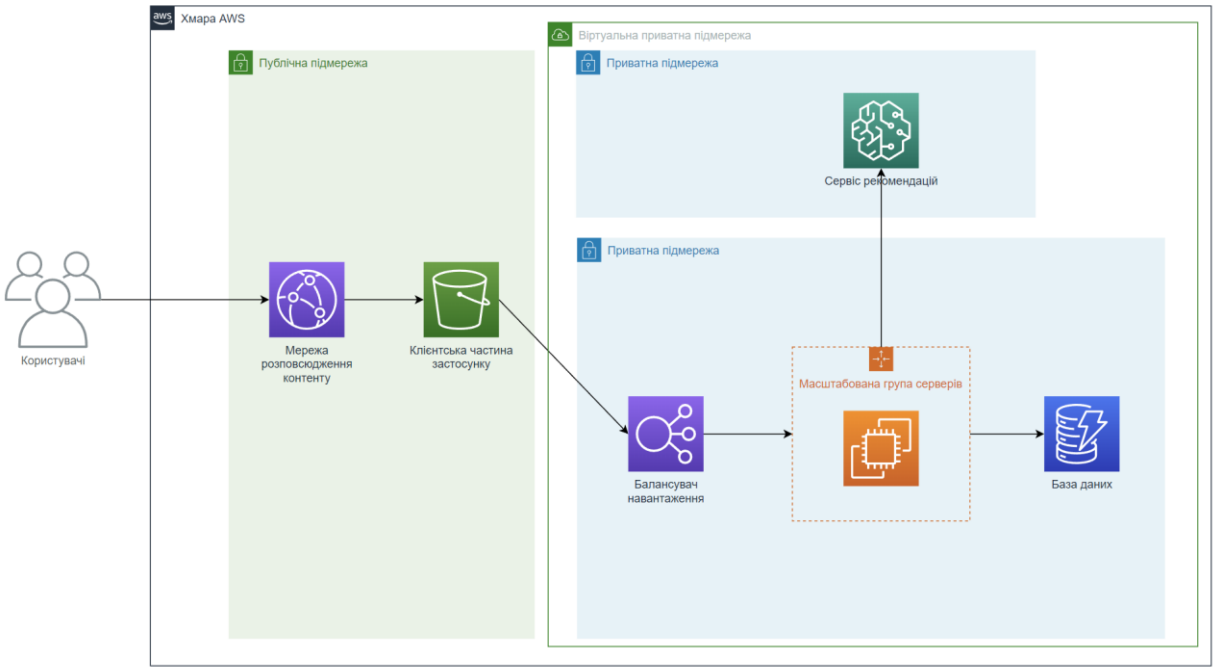

Рисунок 2.7 – Діаграма інфраструктури веб-застосування

Розглянемо детальніше сервіси, що тут використовуються.

AWS Cloudfront – це мережа розповсюдження статичного контенту (content delivery network, CDN), що змінюється нечасто. Даний сервіс кешує контент, що часто запитується користувачами, і дозволяє помітно зменшити вартість розгортання застосунку та зменшити час обробки запиту користувача. Також він підтримує клонування даних між різними регіонами, тобто розгорнутий у Європі додаток одразу копіюється у регіон Сполучена Америка тощо.

AWS S3 (Simple Storage Service) – це сервіс що дозволяє зберігати та завантажувати файли різного розміру. Він добре підходить до зберігання статичних файлів, зображень тощо. Він є обов'язковим для роботи з AWS Cloudfront.

AWS EC2 (Elastic Compute Cloud) – це сервіс, що дозволяє розгортати віртуальні машини. Є основним хмарним сервісом на якому базується багато інших сервісів. Він дозволяє налаштовувати багато елементів віртуальних машин, а саме їх операційну систему, кількість ядер процесора, розмір диска та його тип.

AWS Autoscaling Group – надбудова над AWS EC2, що дозволяє об'єднувати віртуальні машини у групу фіксованого розміру. Хмара, при цьому, несе відповідальність за збереження фіксованого розміру групи та при втраті однієї машини з набору вона одразу запускає іншу. Також гарантується що всі машини з групи будуть рівномірно розподілені по фізичних пристроях у центрі обробки даних що зменшує ризик одночасного відказу групи через технічну несправність.

AWS ELB (Elastic Load Balancer) – сервіс, що розподіляє запити користувачів рівномірно щодо авто-масштабованої групи. Працює на рівнях OSI 7 HTTP та OSI 4 TCP. Він буде слугувати api gateway нашої системи, тобто вхідною точкою запитів на сервер.

AWS DynamoDB – NoSql база даних типу ключ - значення, що має лінійну криву масштабування. Особливістю даної бази даних є той факт що вона є «serverless», тобто від розробника вимагається лише вказати очікувану кількість читань та записів в секунду. Також вона надає певну кількість запитів (burst capacity) що можуть бути оброблені в часи пікового навантаження. Це покращує доступність додатка, що її використовує.

AWS Sagemaker – комплексний сервіс, що має на менті розв'язати основні проблеми пов'язані з тренуванням нейронних мереж. Він дозволяє тренувати нейронні мережі та зберігати різні їх версії. Також він підтримує запуск та обмін будь-яких програм на мові Python у вигляді блокнотів розробників (jupyter notebooks). Очевидно, що окрім тренування, даний сервіс дозволяє розгортати та запускати моделі як у режимі реального часу, так і асинхронно.

2.2.3 Низькорівнева архітектура веб-застосування

2.2.3.1 Низькорівнева архітектура основного сервісу

Для того, щоб перенести функціональні вимоги у код без змін та значних відхилень було використано спеціальний підхід – предметно-орієнтоване проєктування (надалі – DDD від англ. domain driven design).

Його основна мета – створити та пошити спільну мову (ubiquitous language) [34] між командною розробки та командою, що відповідає за опис задачі. Ця мова схожа на інші мови формального опису вимог та має наступні елементи – ядро агрегату (aggregate root), сутність (entity), об’єкт зі значенням (value object), доменна подія (domain event), політика (policy) та команда (command). Її простота і широта дозволяє значно збільшити взаєморозуміння всередині команди, адже всі використовують однакові слова для опису елементів предметної області.

Сутність – певний об’єкт системи, що має ідентифікатор і може змінюватися з часом. Прикладом такого об’єкта в нашій системі є коментар.

Об’єкт зі значенням – незмінний об’єкт, на який посилаються сутності та який не має ідентифікатору. За необхідності оновити даний об’єкт він створюється другий раз з новими значеннями атрибутів. Може бути використаний декількома сутностями. Прикладом такого об’єкта є адреса користувача.

Ядро агрегату – набір сутностей та об’єктів зі значеннями, які можна розглядати як єдине ціле для надання можливості виконувати операції над ними атомарно. Прикладом ядра агрегату є користувач.

Доменна подія – подія, яка виникає внаслідок важливої зміни всередині системи. Вона необхідна, в більшості випадків, для того, щоб повідомити інші рівні про зміни. Прикладом події може слугувати відправка електронного листа.

Політика – інваріант, який підтримує система. Прикладом політики може слугувати вимога унікальності пошти користувача.

Команда – певна зміна системи. Зазвичай команди належать до варіанту використання як один до одного. Прикладом команди є запит на створення ментора.

Єдину мову, розроблену для досліджуваного домену, можна побачити на рисунку 2.8.

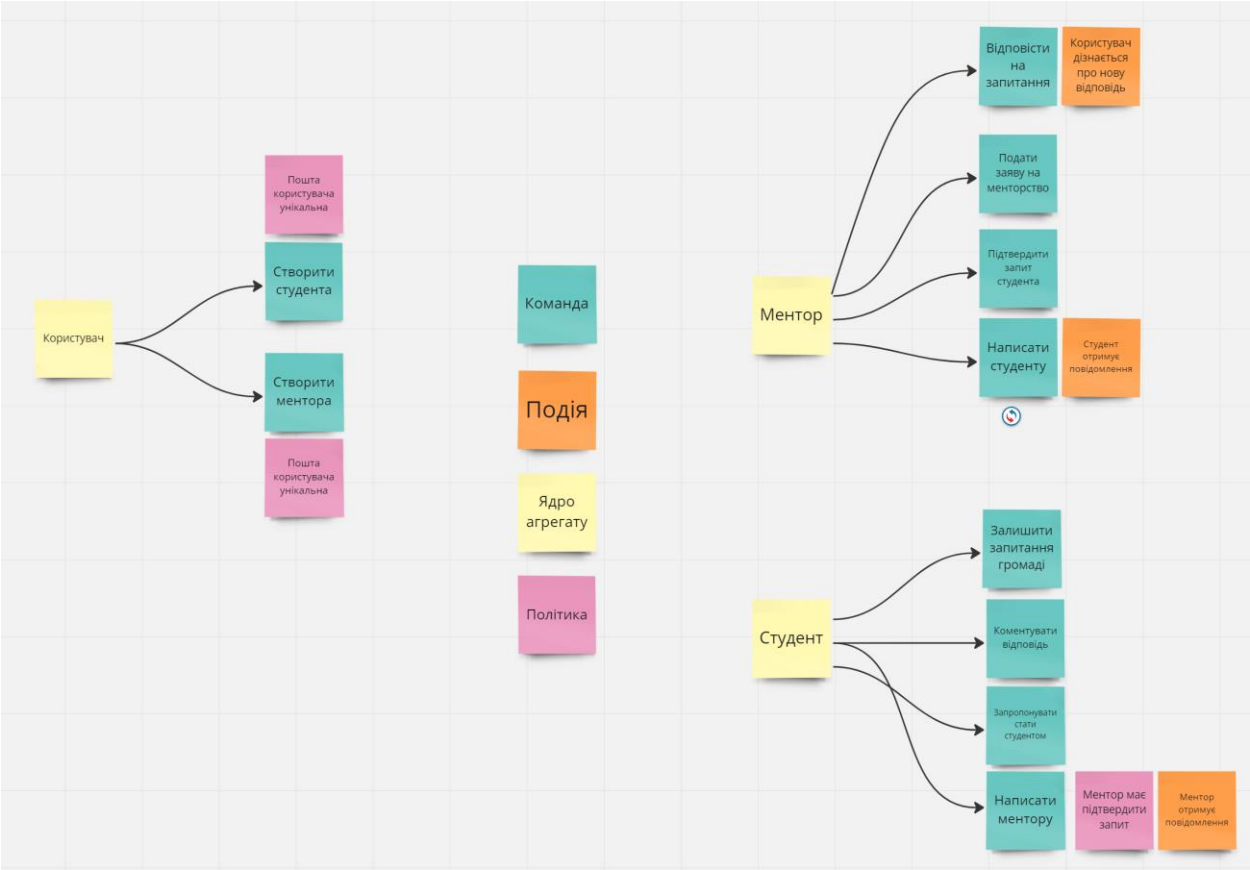

Рисунок 2.8 – Діаграма спільної мови у DDD

Також в роботі було використано класичну трирівневу архітектуру, що складається з домену (domain), додатку (application) та інфраструктури (infrastructure). Графічно дана архітектура зображена на рисунку 2.9. Розглянемо кожен рівень детальніше.

Головним є рівень домену. Він містить елементи описані за допомогою DDD, а саме сутності, об’єкти зі значенням, ядра агрегатів, події домену та політики. Важливо зауважити, що даний рівень не має залежності на інші рівні. Це зроблено для того, щоб домен не забруднювався класами з інших рівнів, наприклад класами для роботи з базою даних. Якщо необхідно викликати клас з іншого рівня використовується інверсія залежності – в рівні

домену створюється необхідний інтерфейс, який потім реалізовується в іншому рівні.

Рівень додатка містить команди та запити до бази даних.

Рівень інфраструктури містить у собі всі елементи пов’язані з зовнішнім світом. Тут зберігається логіка по роботі з базою даних, веб запитами тощо.

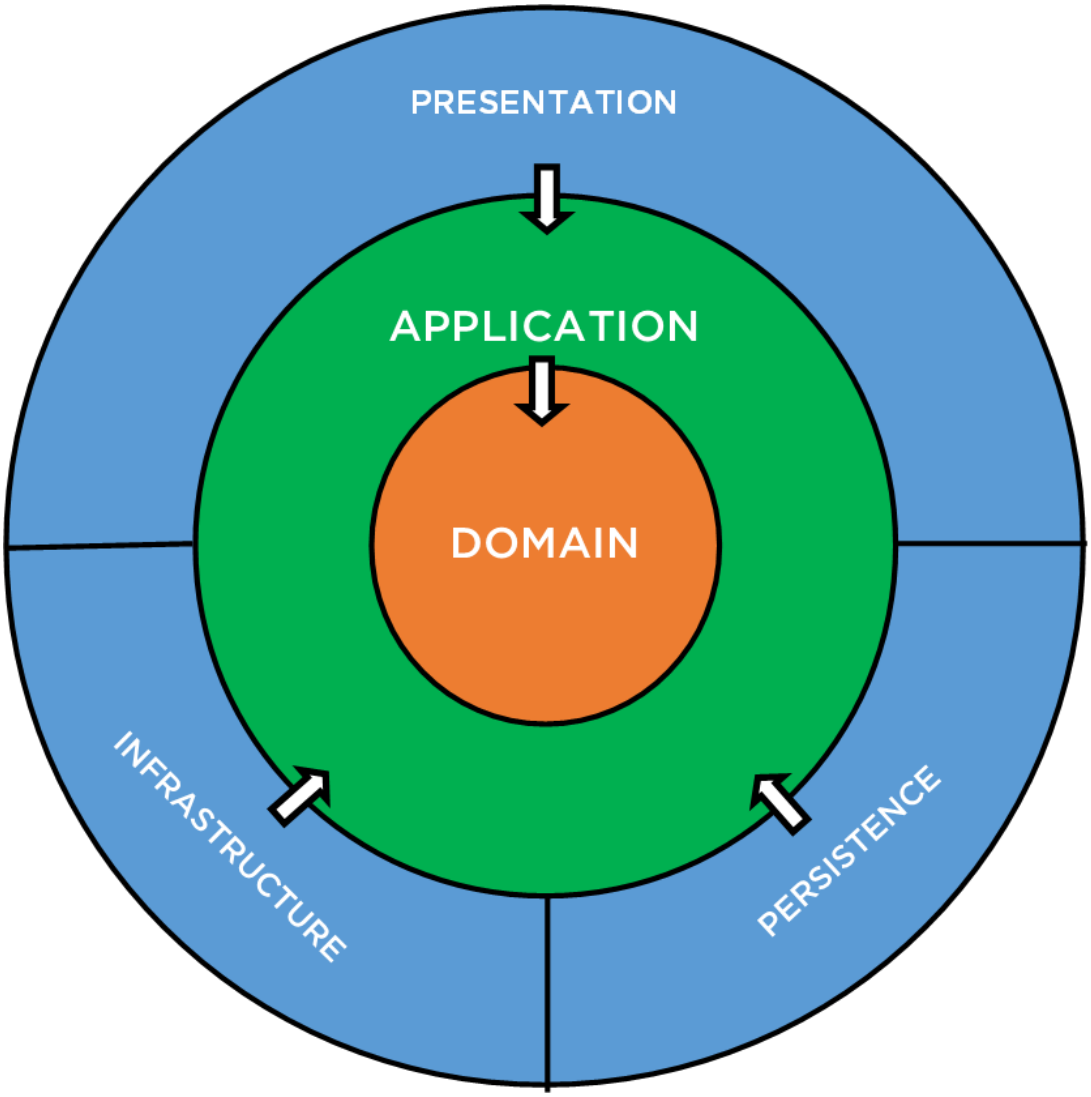

Рисунок 2.9 – Трирівнева архітектура [35]

2.2.3.2 Низькорівнева архітектура сервісу рекомендацій

Сервіс рекомендацій являє собою натреновану та розгорнуту глибинну модель за допомогою шаблону для створення моделей рекомендацій AWS Sagemaker JumpStart Customized Recommender System [36].

Архітектура моделі складається з 11 рівнів і представлена на рисунку 2.10. В якості вхідних даних вона приймає унітарний код студента та ментора, для яких треба визначити ступінь сумісності.

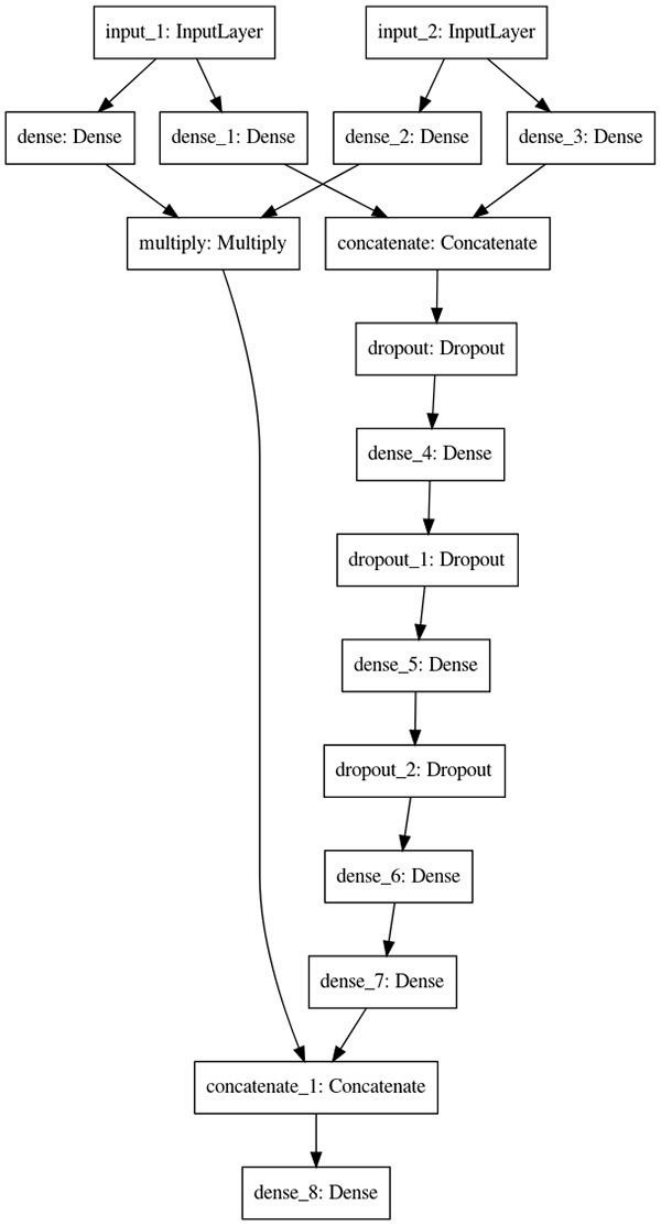

Рисунок 2.10 – Архітектура моделі

Натренована модель була на невеликому наборі даних, згенерованому самостійно. Даний дата сет містить наступні колонки:

– ідентифікатор студента – число;
– ідентифікатор наставника – число;
– якість рекомендації – число від 0 до 5.

2.3 Конструювання програмного забезпечення

2.3.1 Опис рівня домену

Почнемо з деталізації класів рівня домену. Загальна діаграма класів подана на рисунку 2.11. Індивідуально класи описані в таблиці 2.1 – 2.10.

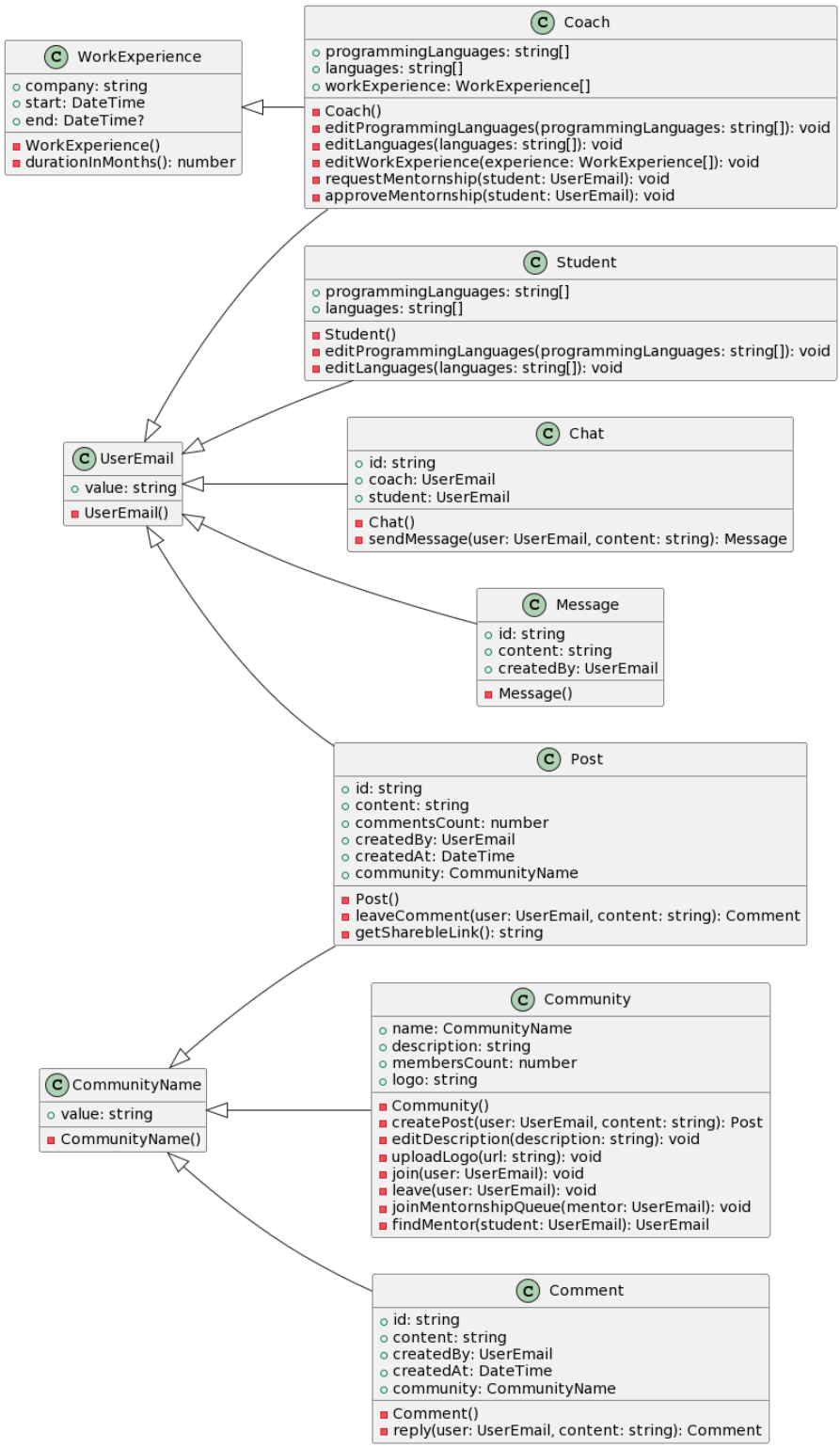

Рисунок 2.11 – Діаграма класів

Таблиця 2.1 – Опис класу WorkExperience

Назва поля	Тип поля	Опис
company	Поле	Назва компанії, до якої відноситься даний проміжок досвіду роботи
start	Поле	Дата початку роботи в компанії
end	Поле. Може бути порожнім	Дата завершення роботи в компанії
WorkExperience()	Конструктор	Конструктор
durationInMonths()	Метод, повертає число	Метод, що повертає час роботи у місяцях. Якщо немає дати завершення роботи, то обирається поточний час

Таблиця 2.2 – Опис класу UserEmail

Назва поля	Тип поля	Опис
value	Поле	Значення пошти користувача
UserEmail()	Конструктор	Конструктор

Таблиця 2.3 – Опис класу CommunityName

Назва поля	Тип поля	Опис
value	Поле	Значення назви громади
UserEmail()	Конструктор	Конструктор

Таблиця 2.4 – Опис класу Coach

Назва поля	Тип поля	Опис
programmingLanguages	Поле	Список мов програмування, якими володіє ментор
languages	Поле	Список мов, якими володіє ментор
workExperience	Поле	Список попередніх місць роботи ментора
Coach()	Конструктор	Конструктор
editProgrammingLanguages()	Метод. Приймає список текстових значень. Нічого не повертає	Оновлює масив мов програмування, якими володіє ментор
editLanguages()	Метод. Приймає список текстових значень. Нічого не повертає	Оновлює масив мов, якими володіє ментор
editWorkExperience()	Метод. Приймає список об’єктів типу WorkExperience. Нічого не повертає	Оновлює список попередніх місць роботи ментора
requestMentorship()	Метод. Приймає об’єкт типу UserEmail. Нічого не повертає	Отримує запит від студента на менторство
approveMentorship()	Метод. Приймає об’єкт типу UserEmail. Нічого не повертає	Приймає користувача як свого студента

Таблиця 2.5 – Опис класу Student

Назва поля	Тип поля	Опис
programmingLanguages	Поле	Список мов програмування, якими володіє студент
languages	Поле	Список мов, якими володіє студент
Student()	Конструктор	Конструктор
editProgrammingLanguages()	Метод. Приймає список текстових значень. Нічого не повертає	Оновлює масив мов програмування, якими володіє студент
editLanguages()	Метод. Приймає список текстових значень. Нічого не повертає	Оновлює масив мов, якими володіє студент

Таблиця 2.6 – Опис класу Chat

Назва поля	Тип поля	Опис
id	Поле	Унікальний ідентифікатор чату
coach	Поле	Пошта ментора, що бере участь в чаті
student	Поле	Пошта студента, що бере участь в чаті
Chat()	Конструктор	Конструктор

Продовження таблиці 2.6

sendMessage()	Метод. Приймає текстове значення та об’єкт типу UserEmail. Повертає об’єкт типу Message	Надсилає повідомлення від користувача у чат

Таблиця 2.7 – Опис класу Message

Назва поля	Тип поля	Опис
id	Поле	Унікальний ідентифікатор повідомлення
content	Поле	Текстове значення повідомлення
createdBy	Поле	Пошта користувача, що надіслав повідомлення
Message()	Конструктор	Конструктор

Таблиця 2.8 – Опис класу Post

Назва поля	Тип поля	Опис
id	Поле	Унікальний ідентифікатор запитання
content	Поле	Текстове значення запитання
commentsCount	Поле	Кількість відповідей до запитання

Продовження таблиці 2.8

createdBy	Поле	Значення пошти користувача, що поставив запитання
createdAt	Поле	Дата і час коли було поставлене запитання
community	Поле	Назва громади, у якій поставлене запитання
Post()	Конструктор	Конструктор
leaveComment()	Метод. Приймає текстове значення та об’єкт типу UserEmail. Повертає об’єкт типу Message	Створює нову відповідь на запитання
getSharebleLink()	Метод. Нічого не приймає. Повертає текстове значення	Повертає посилання на запитання

Таблиця 2.9 – Опис класу Community

Назва поля	Тип поля	Опис
name	Поле	Назва громади
description	Поле	Опис громади
membersCount	Поле	Кількість учасників громади
logo	Поле	Посилання на логотип громади
Community()	Конструктор	Конструктор

Продовження таблиці 2.9

createPost()	Метод. Приймає об’єкт типу UserEmail та текстове значення. Повертає об’єкт типу Post	Створює нове запитання до громади
editDescription()	Метод. Приймає текстове значення. Нічого не повертає	Оновлює опис громади
uploadLogo()	Метод. Приймає текстове значення. Нічого не повертає	Оновлює логотип громади
join()	Метод. Приймає об’єкт типу UserEmail. Нічого не повертає	Додає користувача до громади
leave()	Метод. Приймає об’єкт типу UserEmail. Нічого не повертає	Видаляє користувача з громади
joinMentorshipQueue	Метод. Приймає об’єкт типу UserEmail. Нічого не повертає	Додає ментора до черги очікування
findMentor	Метод. Приймає об’єкт типу UserEmail. Повертає об’єкт типу UserEmail	Знаходить найкращого ментора студентові

Таблиця 2.10 – Опис класу Comment

id	Поле	Ідентифікатор коментаря
content	Поле	Текстове значення коментаря
createdBy	Поле	Пошта користувача, що залишив коментар
createdAt	Поле	Час і дата в яку було створено коментар
community	Поле	Назва громади, в якій було залишено запитання
Comment()	Конструктор	Конструктор
reply()	Метод. Приймає текстове значення та об’єкт типу UserEmail. Повертає об’єкт типу Comment	Створює новий коментар до відповіді

2.3.2 Опис рівня додатка

Рівень додатка являє собою спрощену реалізацію патерну розділення на команди й запити (Command and Query Responsibility Segregation, CQRS) [37]. Його основна ідея – це розділення інтерфейсу роботи з базою даних на два: один для читання, інший для запису. В оригінальній реалізації цього шаблону база даних також розділюється для того, щоб масштабувати читання окремо від запису, але в даній роботі це розглянуто не буде через високу складність реалізації. Це може допомогти у ситуаціях коли розподіл запитів не 50/50, а, наприклад, 10/90. Кожен окремий запит до сховища даних, в свою

чергу, зберігається в окремому класі. Це спрощує пошук та оновлення окремих даних.

Описувати окрему команду (command) не є ефективним, адже більшість команд виконують наступний алгоритм:

а) знайти ядро агрегату за певним ідентифікатором;

б) викликати один чи декілька методів агрегату;

в) зберегти агрегат до бази даних.

Тут проявляється друга важлива перевага DDD – повна відсутність складних методів додавання чи оновлення об’єктів в базі даних. Кожен раз, коли вичитується ядро агрегату воно повертається з усіма даними для роботи. При збереженні агрегату виконується перезапис кожного його поля у базі даних. Важливим є той факт, що запис має відбуватися всередині однієї транзакції, особливо якщо він оперує деякими сутностями всередині ядра агрегату.

Класи читання (query) не містять ніякого функціонала, окрім роботи з базою даних, і розглядатися не будуть.

Другим важливим патерном даного рівня є шаблон ін’єкція залежностей (dependency injection, DI). Даний підхід вбудований у фреймворк NestJS та дозволяє значно спростити управління залежностями між класами. Він дозволяє розробнику описати клас в одному місці й використати в іншому, імпортувавши лише його тип. Фреймворк сам створить об’єкт класу і помістить його у відповідний конструктор іншого. Також можна вказати час життя такого об’єкта – час життя запиту, сесії чи перманентний. Приклад застосування цього методу можна побачити на рисунку 2.12.

Для імплементації цього підходу NestJS вводить концепт модуля. Модуль – набір класів, поєднаних логічним чином. Це дозволяє розробнику розділити застосунок на частини з метою зменшення когнітивної складності архітектури.

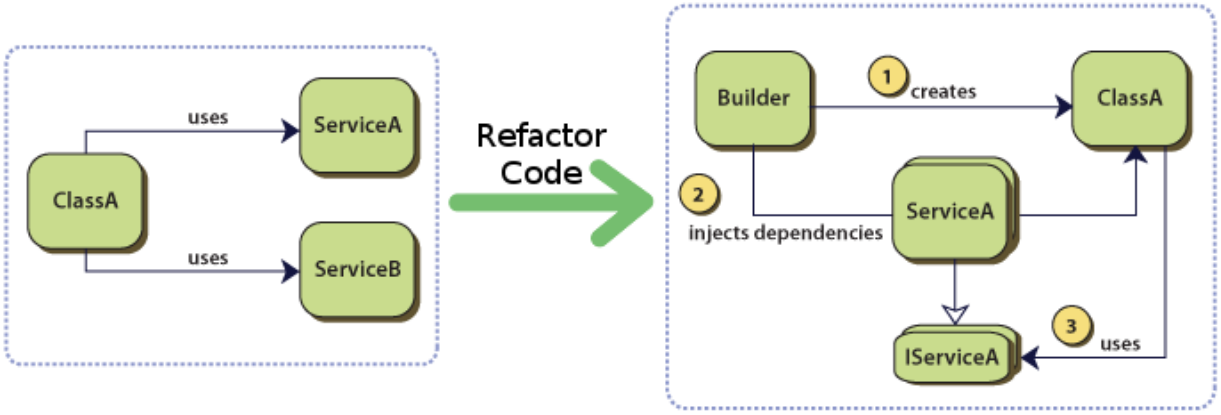

Рисунок 2.12 – Застосування ін’єкції залежностей [38]

Кожен модуль складається з наступних елементів:

– списку модулів або класів що він імпортує;

– списку модулів або класів що він експортує;

– списку класів що він створює;

– списку контролерів.

Приклад якісного розділення на модулі можна побачити на рисунку 2.13.

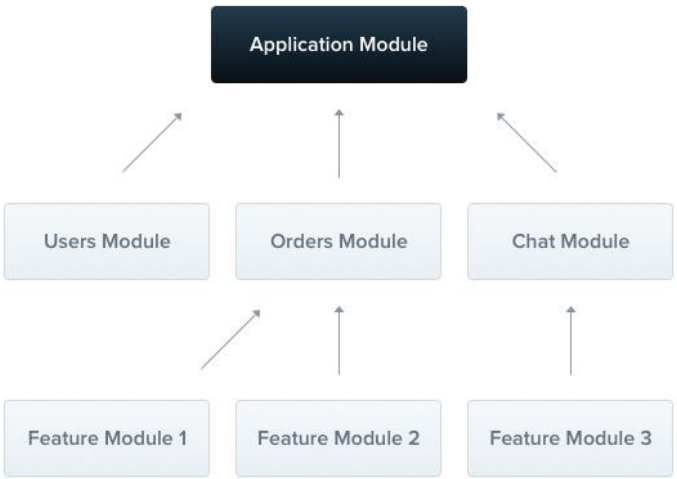

Рисунок 2.13 – Декомпозиція кодової бази на модулі [39]

2.3.3 Опис рівня інфраструктури

Як було зазначено раніше, рівень інфраструктури відповідає за зв’язок додатка із зовнішнім світом і містить наступні модулі:

а) модуль для роботи зі хмарою;

б) модуль для роботи із базою даних;

в) модуль для роботи з http та WebSocket запитами.

Розглянемо кожен модуль окремо.

2.3.3.1 Опис модуля для роботи зі хмарою

Для роботи зі хмарою AWS пропонується використання офіційної бібліотеки AWS SDK версії 3.0.

Дана бібліотека надає єдиний інтерфейс для взаємодії зі хмарою за допомогою коду. Вона надає підтримку повного життєвого циклу для взаємодії із хмарним арі, а саме: управління обліковими даними, автоматичні повторні спроби запиту та серіалізацію даних.

Починаючи з 3-ї версії, бібліотека є модульною, що дозволяє додавати залежності лише тих сервісів, що дійсно використовуються додатком. При розробці було використано наступні модулі:

– @aws-sdk/client-dynamodb – для роботи з базою даних;

– @aws-sdk/client-s3 – для роботи із файловим сховищем, де будуть зберігатися зображення;

– @aws-sdk/client-sagemaker – для виклику натренованої моделі, розгорнутої в AWS Sagemaker.

2.3.3.2 Опис модуля для роботи із базою даних

Хмарний провайдер AWS надає великий вибір різноманітних баз даних:

– AWS RDS підтр;мує основні SQL рушії такі як Postgres, Oracle тощо.

– AWS Redshift – сховище даних (data warehouse), що розраховано на аналітичне навантаження;

– AWS DynamoDB – лінійно масштабована база даних типу ключ значення, що побудована за допомогою serverless підходу;

– AWS Nuptune – графова база даних, що побудована за допомогою serverless підходу;

– AWS ElastiCache – база даних, що зберігається в оперативній пам’яті та має швидкі операції читання та запису.

В даній роботі було обрано базу даних AWS DynamoDB через її масштабованість та простоту налаштування. Розглянемо її детальніше, адже вона має нетрадиційну архітектуру збереження та організації даних.

Кожний рядок зі значеннями в даній базі даних містить одне обов’язкове поле – ключ патриції (partition key). Цей ключ використовується для розподілу рядків в різні підрозділи (partitions) бази даних, що можуть знаходитись на різних серверах. Завдяки цьому досягається масштабованість системи, адже чим більше патрицій, тим більше даних база даних може записувати за одиницю часу. Однак при збільшенні кількості підрозділів зростає складність послідовного читання даних, адже необхідно переглянути всі підрозділи. Для подолання проблеми послідовного читання використовують вторинні індекси.

Вторинний індекс (secondary index) – структура даних, що дозволяє зберігати весь масив даних у відсортованому вигляді. Це дозволяє пришвидшити деякі записи на читання, адже їм не треба буде збирати дані отримані з різних підрозділів. Візуально даний індекс можна побачити на рисунку 2.14. Важливо зазначити, що кожний вторинний індекс повторює дані і потребує виділення окремих потужностей.

Окрім ключа патриції кожний рядок бази даних також може мати ключ сортування (sort key). На основі цього відбувається автоматичне сортування записів всередині однієї патриції. Завдяки цьому розширюється спектр

запитів, що може обробити DynamoDB. Співвідношення між ключем розділу та ключем сортування зображено на рисунку 2.15.

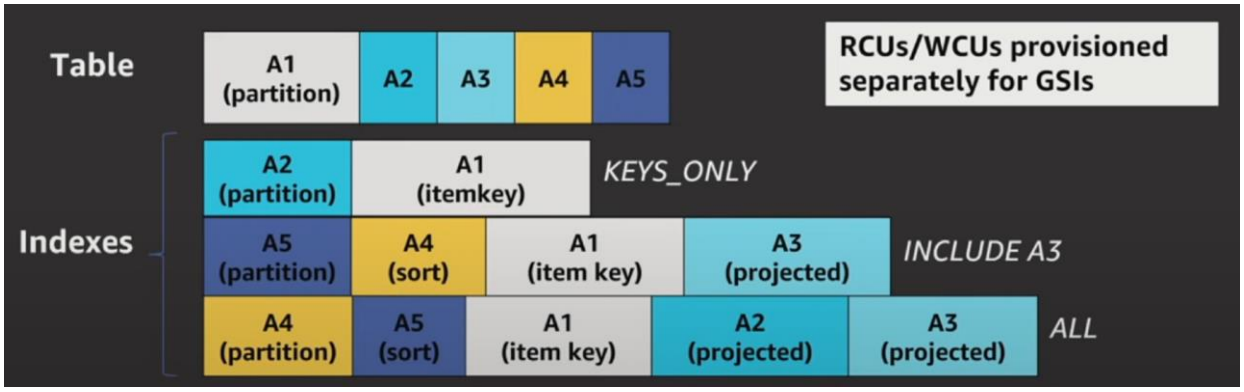

Рисунок 2.14 – Вторинний індекс у DynamoDB [40]

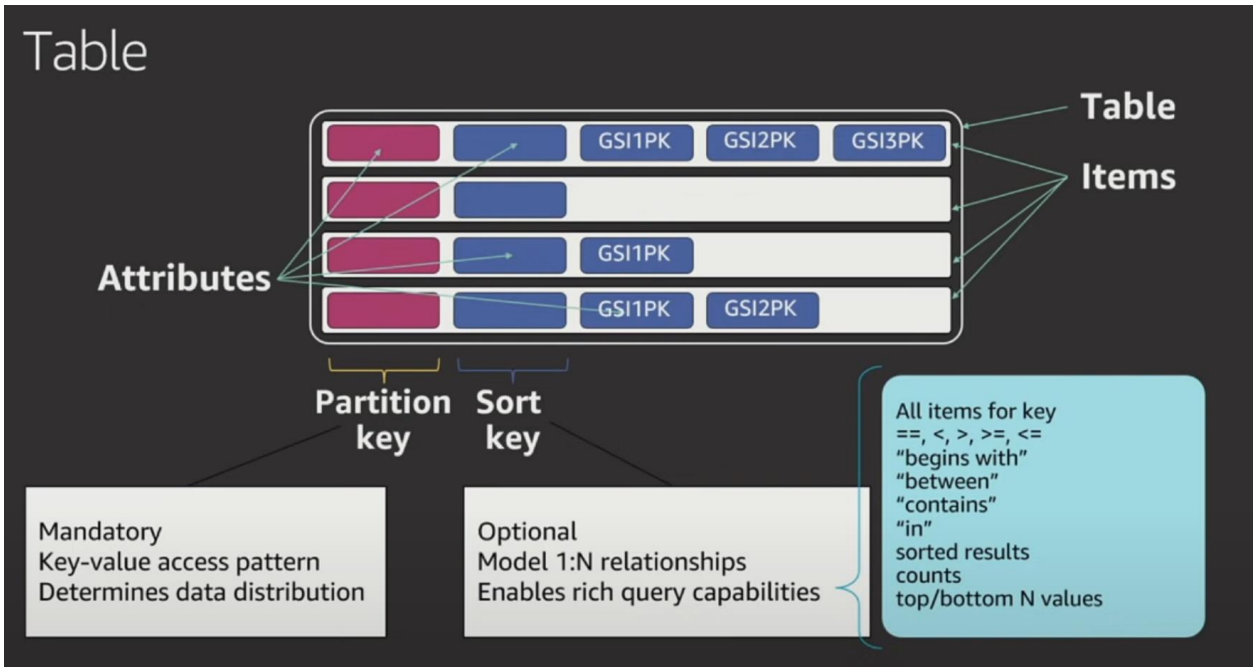

Рисунок 2.15 – Опис ключових елементів таблиці DynamoDB [40]

Головною відмінністю цієї бази даних від традиційної sql бази є те, що у ній відсутні операції єднання (join). Розглянемо на прикладі, як можна змоделювати об'єднання даних при таких обмеженнях на прикладі користувача та історії його покупок.

Нехай присутня база даних, яка зберігає наступну інформацію: дані про користувача та його замовлення. Для того, щоб отримати список покупок та всю інформацію про користувача в sql можна написати наступний запит:

```
SELECT o.*, c.* FROM orders o
LEFT JOIN  customers c ON c.id = o.customerId;
```

Для того, щоб відтворити даний запит у DynamoDB необхідно обрати ідентифікатор користувача як ключ підрозділу та комбінацію рядка <тип рядка>#<ідентифікатор замовлення або користувача> як ключ сортування. Візуально це зображено на рисунку 2.16.

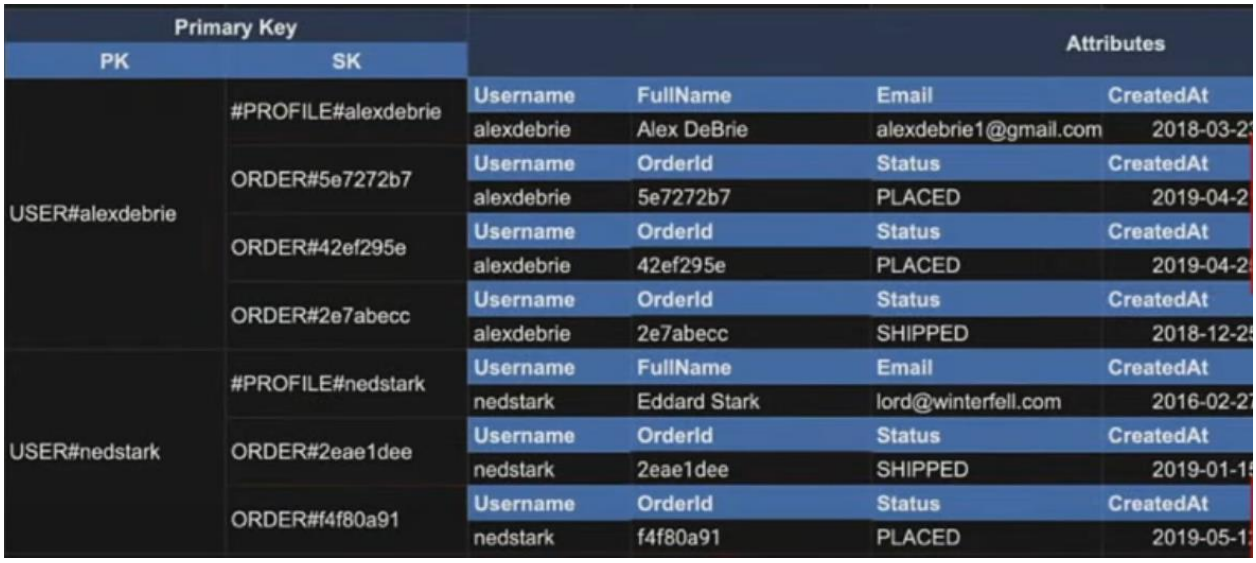

Рисунок 2.16 – Об’єднання двох таблиць у DynamoDB [41]

Беручи до уваги згадані вище обмеження та особливості розглянемо детальніше таблиці що присутні у додатку:

а) chats – для роботи з чатами та повідомленнями;

б) communities – для роботи з громадами та її членами;

в) posts – для роботи із запитаннями, відповідями та коментарями;

г) fairs – для роботи із чергою користувачів, що хочуть стати менторами;

д) users – для роботи із менторами та студентами.

Схеми таблиць подані у таблицях 2.11 – 2.15.

Таблиця 2.11 – Схема таблиці Chats

Таблиця	Назва поля	Тип даних	Опис
Chats	pk	string	Ключ підрозділу у вигляді Chat#<восьми значний ідентифікатор чату>
	sk	string	Ключ сортування у вигляді Identity якщо це рядок з даними користувача або Message#<дата і час створення>#<пошта користувача>
	author	string	Пошта користувача. Значення присутнє тільки якщо це рядок, що відповідає за повідомлення.
	chat	string	Ідентифікатор чату. Значення присутнє тільки якщо це рядок, що відповідає за повідомлення.

Продовження таблиці 2.11

	content	string	Текст повідомлення. Значення присутнє тільки якщо це рядок, що відповідає за повідомлення.
	createdAt	timestamp	Дата і час створення повідомлення. Значення присутнє тільки якщо це рядок, що відповідає за повідомлення.
	id	string	Ідентифікатор сутності (чату або повідомлення)
	coach	string	Пошта ментора, що є членом чату. Значення присутнє тільки якщо це рядок, що відповідає за чат.
	student	string	Пошта студента, що є членом чату. Значення присутнє тільки якщо це рядок, що відповідає за чат.

Таблиця 2.12 – Схема таблиці Communities

Таблиця	Назва поля	Тип даних	Опис
Communities	pk	string	Ключ розділу у вигляді Community#<назва громади>
	sk	string	Ключ сортування у вигляді Identity якщо це рядок з даними громади або Member#<пошта користувача> якщо це рядок з даними члена громади
	description	string	Опис громади. Значення присутнє тільки якщо це рядок, що відповідає за громаду.
	email	string	Ідентифікатор члена громади. Значення присутнє тільки якщо це рядок, що відповідає за члена громади.
	membersCount	number	Кількість користувачів, що перебувають у громаді. Значення присутнє тільки якщо це рядок, що відповідає за громаду.

Продовження таблиці 2.12

	name	string	Назва громади. Значення присутнє тільки якщо це рядок, що відповідає за громаду.

Таблиця 2.13 – Схема таблиці Fairs

Таблиця	Назва поля	Тип даних	Опис
Fairs	pk	string	Ключ розділу у вигляді Fair#<ідентифікатор черги>
	sk	string	Ключ сортування у вигляді Identity якщо це рядок з даними черги або Coach#<пошта ментора>
	community	string	Назва громади до якої належить черга. Значення присутнє тільки якщо це рядок, що відповідає за чергу.
	email	string	Ідентифікатор ментора. Значення присутнє тільки якщо це рядок, що відповідає за члена черги.

Продовження таблиці 2.13

	joinedAt	timestamp	Дата та час, коли ментор доєднався до черги. Значення присутнє тільки якщо це рядок, що відповідає за члена черги.
	mentor	object	Серіалізоване значення даних ментора. Значення присутнє тільки якщо це рядок, що відповідає за члена черги.

Таблиця 2.14 – Схема таблиці Posts

Таблиця	Назва поля	Тип даних	Опис
Posts	pk	string	Ключ розділу у вигляді Post#<восьми значний ідентифікатор запитання>
	sk	string	Ключ сортування у вигляді Identity якщо це рядок з даними запитання або Comment#<ідентифікатор відповіді> якщо це рядок з даними відповіді

Продовження таблиці 2.14

	community	string	Назва громади до якої належить запитання або відповідь.
	content	string	Текст запитання або відповіді.
	createdAt	timestamp	Дата та час створення запитання або відповіді.
	createdBy	string	Пошта користувача що створив дане запитання або відповідь
	id	string	Восьми значний ідентифікатор якщо це рядок з даними запитання або набір ідентифікаторів що утворюють абсолютний шлях відповіді (наприклад postid#comment1#comment2)
	postId	string	Ідентифікатор запитання. Значення присутнє тільки якщо це рядок, що відповідає за відповідь.

Продовження таблиці 2.14

	replyTo	string	Ідентифікатор попередньої відповіді. Значення присутнє тільки якщо це рядок, що відповідає за відповідь.
	title	string	Заголовок запитання. Значення присутнє тільки якщо це рядок, що відповідає за запитання.

Таблиця 2.15 – Схема таблиці Users

Таблиця	Назва поля	Тип даних	Опис
Users	pk	string	Ключ розділу у вигляді Student#<пошта користувача>
	sk	string	Ключ сортування у вигляді Identity якщо це рядок з даними користувача або Applicant#<пошта студента> якщо це рядок з даними заяви студента
	email	string	Пошта користувача

Продовження таблиці 2.15

	role	string	Роль користувача (Student якщо це студент або Coach якщо це ментор)
	languages	string[]	Список мов, якими володіє користувач
	location	object	Об’єкт з полями city, country що відповідають за місто та країну знаходження користувача
	name	string	Ім’я користувача
	programmingLanguages	string[]	Список мов програмування, якими володіє користувач
	workExperience	object[]	Масив об’єктів з полями company, start, end, що відповідають за місце, початок та завершення роботи на конкретній позиції

2.3.3.3 Опис модуля для роботи HTTP та WebSocket запитами

Для обробки HTTP та WebSocket запитів було обрано вбудовані механізми фреймворку NestJS – контролери та шлюзи відповідно. Для обробки запитів WebSocket було обрано бібліотеку ws.

2.3.4 Опис допоміжних утиліт

При розробці даного програмного забезпечення було використано цілий ряд допоміжних програм та утиліт. Опишемо їх детальніше.

Visual Studio Code – головне програмне забезпечення розробки всіх частин застосунку.

Javascript & Typescript extension for VSCode – розширення Visual Studio Code, що дозволяє запускати та компілювати файли написані мовою програмування typescript.

AWS Toolkit – розширення Visual Studio Code, що дозволяє працювати зі хмарою AWS прямо з консолі.

Postman – програмне забезпечення для запуску власних HTTP та WebSocket запитів. Використовувалось для тестування серверної частини застосунку.

DBeaver – програмне забезпечення, що дозволяє працювати з базою даних DynamoDB через клієнтський інтерфейс, а також зберігати та тестувати запити до бази даних.

2.4 Аналіз безпеки даних

Безпека даного веб-застосування складається з трьох основних шарів: клієнт-серверної архітектури, безпеки мережі хмари та безпеки бази даних. Опишемо кожен з них.

Клієнт серверна архітектура гарантує, що клієнт не має прямого доступу до даних інших користувачів завдяки авторизації та автентифікації. Також було обрано протоколи HTTPs та WSs, що шифрують дані таким

чином що розшифрувати їх можна тільки на клієнті або сервері. Таким чином захищаються від атак типу man-in-the-middle. На сервері додатково буде налаштовано CORS, що дозволить доступ на нього лише з фіксованої адреси клієнту.

Хмарний провайдер AWS гарантує, що весь трафік, що проходить, всередині його сервісів надійно захищено. Таким чином, як тільки трафік потрапляє всередину хмари, тобто на балансир навантаження, він стає недоступним для зовнішнього світу. Також обраний хмарний провайдер дозволяє обмежити, завдяки групам безпеки, доступ до бази даних тільки своїми ip адресами. Таким чином база даних буде недоступною з ніякого адресу окрім адресу сервера.

Обрана база даних DynamoDB також гарантує надійність даних. По-перше, вона копіює дані на різні сервери з метою уникнути зникнення даних через несправність заліза. По-друге, вона шифрує дані у стані спокою (encryption at rest), тобто навіть при фізичному доступі до сховища даних, зловмисник не зможе їх розшифрувати.

Висновки до розділу

В цьому розділі було спроектовано та проаналізовано програмне забезпечення, що розробляється.

Також було детально розглянуто основні архітектурні рішення застосунку на трьох різних рівнях деталізації: високому рівні, рівні інфраструктури та низькому рівні. Було детально описано кожен архітектурний патерн, що використовується. Особливу увагу було приділено інфраструктурним сервісам хмарного провайдера та проектним шаблонам, як, наприклад, DDD.

Далі було задокументовано основні класи, що містяться в додатку, та їх взаємодію між собою.

Особливу увагу було приділено опису схеми даних обраної бази даних. Для цього були розглянуті важливі патерни проектування даних. Було описано кожну окрему таблицю зі списком їх полів.

Також було описано основні бібліотеки та утиліти, що використовувалися при розробці системи.

Останнім кроком став опис стратегії безпеки даних та додатку, в цілому. Було описано всі шари безпеки та розглянуто види атак, від який вони рятують.

На основі розробленого додатка буде описано його стратегію тестування.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Наступним кроком після проектування та розробки програмного забезпечення є його тестування. Цей етап є надзвичайно важливим, адже він дозволяє перевірити відповідність розробленого додатка як його функціональним, так і нефункціональним вимогам.

Для тестування використано класичний метод, що базується на піраміді тестування. Піраміда тестування – метафора, що говорить нам о необхідності розділяти тести на різні прошарки за ступенями зернистості. Це дозволяє описувати та виконувати різні шари незалежно один від одного та за допомогою різних фреймворків. Візуально дана піраміда зображена на рисунку 3.1.

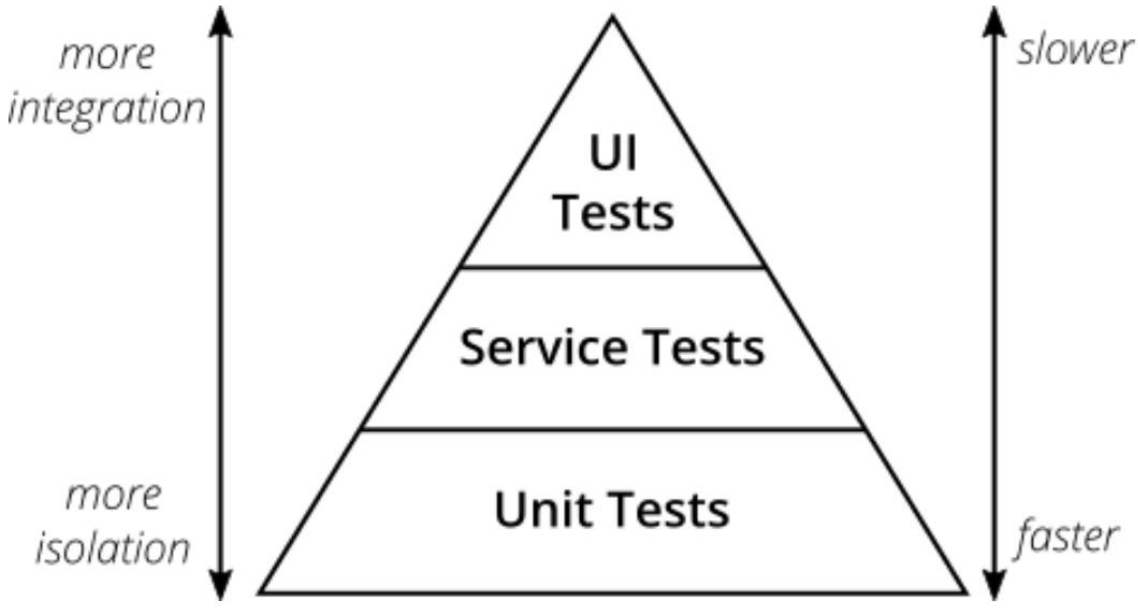

Рисунок 3.1 – Піраміда тестування [42]

Розглянемо рівні з яких вона складається та їх призначення.

Нижнім рівнем є рівень модульних тестів (unit tests). Модулем в даному контексті може бути як окрема функція, так і цілий клас. Кожен модуль, в свою чергу, тестується декілька разів – з різними вхідними даними. Особливу

увагу приділяють перевірці граничних умов. Даний рівень є зазвичай найбільшим, але, в той самий час, найшвидшим, адже запуск кожного з них займає мілісекунди.

Існує декілька правил написання якісних модульних тестів:

– тестуватися мають тільки загальнодоступні методи – для того, щоб можна було змінити їх реалізацію у будь який час;

– кожен тест має тестувати тільки один конкретний набір вхідних даних – відповідно до принципу єдиної відповідальності;

– кожен тест має бути швидким – для того, щоб їх можна було запускати якомога частіше;

– кожен тест має бути детермінованим.

Тестування окремих модулів, на жаль, не завжди дає змогу виявити потенційну проблему, адже повністю ігнорується інший важливий аспект – взаємодія даних модулів. Для подолання цієї проблеми існує сервісний рівень (service layer). Він відповідає за тестування цілого сервісу на відповідність його контракту.

Останнім і найменшим рівнем є рівень мануальних тестів інтерфейсу. Хоча таке тестування і дає найбільші гарантії щодо працездатності додатка, воно є дуже повільним, адже потребує втручання людини. Тому ці тести, зазвичай, виконуються командою тестування перед релізом нової версії програмного забезпечення.

3.2 Опис процесів тестування

3.2.1 Опис рівня модульних тестів

За допомогою модульних тестів було перевірено коректність коду шару домену додатка. Тестування виконувалося за допомогою бібліотеки Jest. Результати тестування можна побачити на рисунках 3.2 – 3.5.

```
PASS test/domain/coach.spec.ts
  Coach
    yearsOfExperience
      √ returns correct value when one company (1 ms)
      √ returns correct value when two companies
      √ returns correct value when no end date (1 ms)
    applyForMentorship
      √ returns void
      √ throws an exception if user is not student
      √ throws an exception if user is already a mentee
```

Рисунок 3.2 – Результати тестування класу ментора

```
PASS test/domain/student.spec.ts
  Student
    setProgrammingLanguages
      √ returns void when multiple languages (1 ms)
      √ returns void when no languages
    setLanguages
      √ returns void when multiple languages
      √ returns void when no languages
```

Рисунок 3.3 – Результати тестування класу студента

```
PASS test/domain/post.spec.ts
  Post
    createNew
      √ returns new post (1 ms)
    getLink
      √ returns correct url (1 ms)
    reply
      √ returns new comment
```

Рисунок 3.4 – Результати тестування класу запитання

```
PASS test/domain/community.spec.ts
  Post
    createNew
      √ returns new community (1 ms)
    editName
      √ returns void
      √ throws exception when input empty string
    createPost
      √ returns new post (1 ms)
```

Рисунок 3.5 – Результати тестування класу громади

3.2.2 Опис рівня сервісних тестів

За допомогою модульних тестів було перевірено коректність серверної частини додатка. Тестування виконувалося за допомогою програми Postman. Результати тестування можна побачити рисунку 3.6.

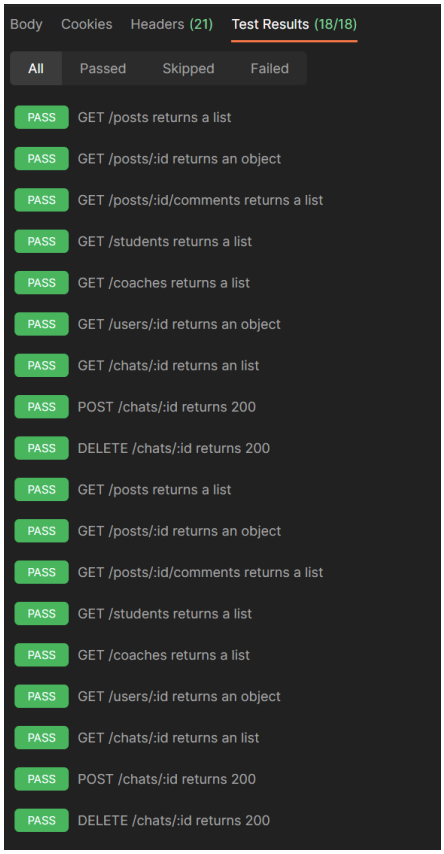

Рисунок 3.6 – Результати тестування сервіса

3.2.3 Опис рівня мануальних тестів

За допомогою мануальних тестів було перевірено коректність роботи цілого додатку. Тестування проводилося згідно з варіантами використання. Опис відповідних тестів наведено у таблицях 3.1 – 3.8.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Електронна пошта, пароль, підтвердження пароля
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 8 до 64 символів, який містить хоча б з одну англійську літеру, одне число та один спеціальний символ, підтвердження пароля, яке збігається з раніше введеним паролем. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.

Таблиця 3.2 – Тест 1.2

Тест	Авторизація користувача
Модуль	Авторизація користувача
Номер тесту	1.2
Початковий стан системи	Зареєстрованого користувача було перенаправлено на сторінку авторизації
Вхідні дані	Електронна пошта, пароль

Продовження таблиці 3.2

Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта та пароль, які до цього була зареєстровані в системі. Після цього натискається кнопка підтвердження авторизації.
Очікуваний результат	Авторизація проходить успішно, користувач перенаправляється на сторінку власного профілю.
Фактичний результат	Авторизація проходить успішно, користувач перенаправляється на сторінку власного профілю.

Таблиця 3.3 – Тест 2.1

Тест	Додавання мови програмування
Модуль	Профіль користувача
Номер тесту	2.1
Початковий стан системи	Авторизований користувач знаходиться на сторінці власного профілю
Вхідні дані	Мова програмування, яку він бажає додати
Опис проведення тесту	Натискається кнопка додавання мови програмування. У відповідне текстове поле вводиться назва даної мови від 1 до 64 символів. Після цього натискається кнопка підтвердження.
Очікуваний результат	Додавання відбувається успішно, користувач бачить нову мову у списку.
Фактичний результат	Додавання відбувається успішно, користувач бачить нову мову у списку.

Таблиця 3.4 – Тест 3.1

Тест	Приєднання до нової громади
Модуль	Громади
Номер тесту	3.1

Продовження таблиці 3.4

Початковий стан системи	Авторизований користувач знаходиться на сторінці списку громад
Вхідні дані	Громада, до якої бажає приєднатися користувач
Опис проведення тесту	Натискається кнопка «Приєднатися» праворуч від імені громади.
Очікуваний результат	Приєднання відбувається успішно, користувач перенаправляється на сторінку громади.
Фактичний результат	Приєднання відбувається успішно, користувач перенаправляється на сторінку громади.

Таблиця 3.5 – Тест 3.2

Тест	Вихід з громади
Модуль	Громади
Номер тесту	3.2
Початковий стан системи	Авторизований користувач знаходиться на сторінці списку громад
Вхідні дані	Громада, членом якої є користувач та яку він бажає покинути
Опис проведення тесту	Натискається кнопка «Покинути» праворуч від імені громади.
Очікуваний результат	Користувач покидає громаду успішно та бачить кнопку «Приєднатися» замість «Покинути».
Фактичний результат	Користувач покидає громаду успішно та бачить кнопку «Приєднатися» замість «Покинути».

Таблиця 3.6 – Тест 3.3

Тест	Створення запитання громаді
Модуль	Громади
Номер тесту	3.3
Початковий стан системи	Авторизований користувач знаходиться на сторінці громади, до якої він приєднаний
Вхідні дані	Текст запитання яке хоче поставити користувач
Опис проведення тесту	Натискається кнопка «Задати запитання» праворуч від імені громади. У формі, що з’являється заповнюються поля короткого та повного опису запитання. Текст запитання не має перевищувати 1000 символів. Після цього натискається кнопка публікації запитання.
Очікуваний результат	Користувач бачить своє запитання у списку останніх запитань громади.
Фактичний результат	Користувач бачить своє запитання у списку останніх запитань громади.

Таблиця 3.7 – Тест 3.4

Тест	Підтвердження заяви студента
Модуль	Наставництва
Номер тесту	4.1
Початковий стан системи	Авторизований наставник знаходиться на сторінці зі списком своїх студентів
Вхідні дані	Список із заявами студентів
Опис проведення тесту	Зі списку обирається студент, якого ментор готовий погодити та натискається кнопка «Підтвердити» праворуч від імені студента.
Очікуваний результат	Наставник бачить кнопку «Написати» замість «Підтвердити».

Продовження таблиці 3.7

Фактичний результат	Наставник бачить кнопку «Написати» замість «Підтвердити».

Таблиця 3.8 – Тест 3.3

Тест	Відправка повідомлення студенту
Модуль	Наставництва
Номер тесту	4.2
Початковий стан системи	Авторизований наставник знаходиться на сторінці переписки зі своїм студентом
Вхідні дані	Текст повідомлення яке хоче надіслати ментор
Опис проведення тесту	Поле вводу заповнюється текстом запитання від 1 до 200 символів. Після цього натискається кнопка «Відправити» нижче від тексту повідомлення.
Очікуваний результат	Користувач бачить своє повідомлення у списку останніх повідомлень.
Фактичний результат	Користувач бачить своє повідомлення у списку останніх повідомлень.

3.3 Опис контрольного прикладу

Розглянемо роботу застосунку на прикладі пошуку наставника.

Першим кроком є перехід на сторінку зі списком громад. На цій сторінці користувач бачить список усіх громад, що існують у системі. Він може зробити три наступні дії: приєднатися до громади, покинути громаду та перейти на сторінку громади.

Для того, щоб приєднатися до громади користувач повинен натиснути кнопку «Доєднатися» праворуч від назви громади. Ця кнопка буде доступною тільки якщо користувач не є членом громади.

Якщо користувач бажає покинути громаду, то він має натиснути кнопку «Покинути», яка буде доступна тільки членам даної громади.

Приклад обох випадків зображено на рисунку 3.7.

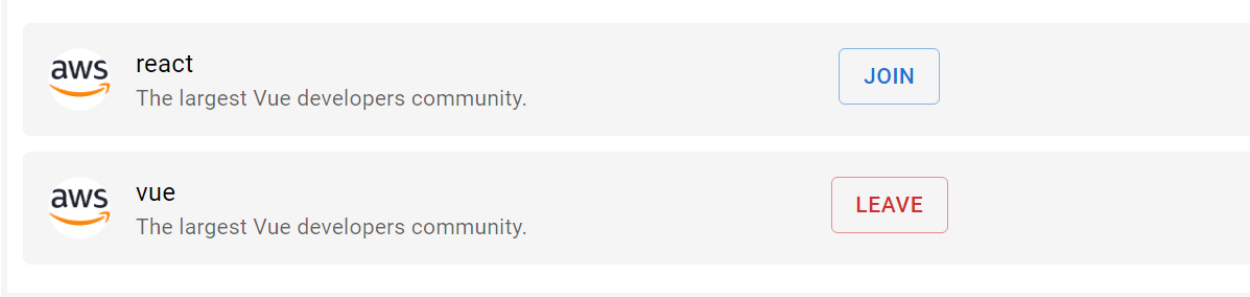

Рисунок 3.7 – Інтерфейс взаємодії з громадою.

Для того, щоб перейти до громади користувач може просто натиснути на будь-яке місце елементу зі списку. Після цього його буде перенаправлено на сторінку громади, яка зображена на рисунку 3.8.

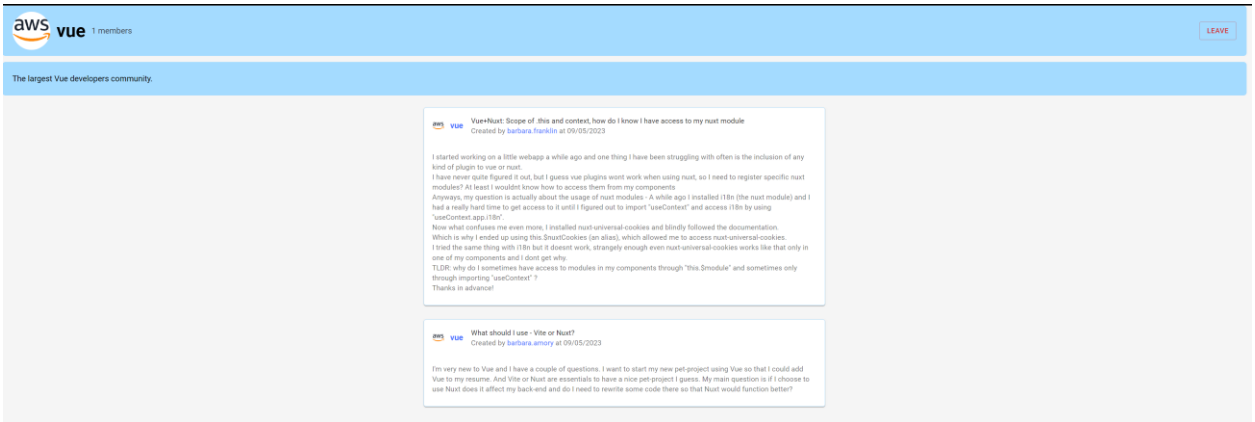

Рисунок 3.8 – Інтерфейс перегляду громади

Після того як користувач приєднався до громади він може знайти ментора серед членів громади. Для цього необхідно перейти на сторінку пошуку наставника, що подана на рисунку 3.9.

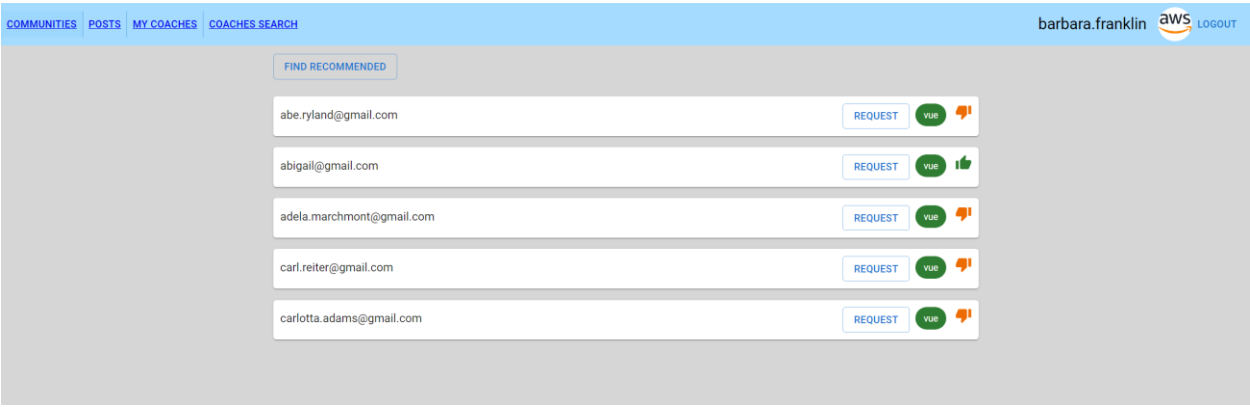

Рисунок 3.9 – Інтерфейс пошуку наставника

На сторінці пошуку користувач бачить список наставників та може запропонувати свою кандидатуру будь-якому з них. Для цього він натискає кнопку «Request» на відповідному елементі списку.

Після цього обраний наставник побачить студента у списку власних заяв та може або підтвердити заяву або відхилити. Цей процес поданий на рисунку 3.10.

Рисунок 3.10 – Інтерфейс списку заяв

У випадку погодження заяви обидва користувачі можуть почати обмін повідомленнями. Це можна побачити на рисунках 3.11 та 3.12.

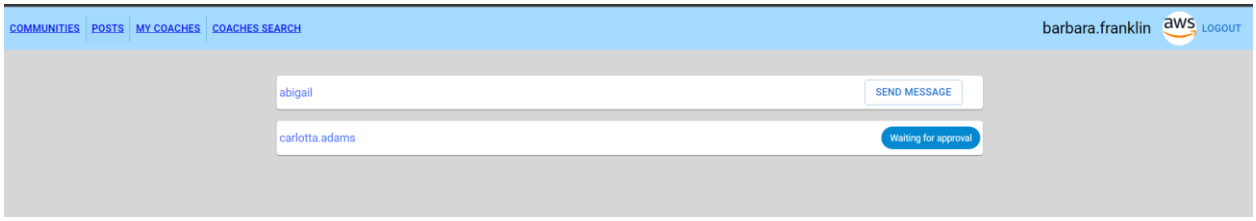

Рисунок 3.11 – Інтерфейс списку наставників

Рисунок 3.12 – Інтерфейс текстового чату між наставником та студентом

Висновки до розділу

В цьому розділі було підходи та процеси, що були використані для тестування розробленого програмного забезпечення. Всього, за допомогою запропонованої методики, було перевірено працездатність додатка на трьох різних рівнях: окремого модуля, цілого сервісу та користувацького інтерфейсу. В результаті було визначено, що розроблена онлайн платформа повністю відповідає описаним раніше функціональним вимогам.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Всі частини програмного забезпечення було розгорнуто у хмарі Amazon Web Services з використанням наступних сервісів: EC2, S3, Cloudfront, Sagemaker, DynamoDB, IAM.

Для забезпечення неперервної інтеграції та розгортання було використано сервіси GitHub Actions та AWS Elastic Beanstalk.

GitHub Actions – це сервіс, що дозволяє реагувати на останні зміни головної гілки репозиторію та виконувати різноманітні дії над ними. Окрім великої кількості вбудованих дій активно розвивається ринок дій, створених користувачами та компаніями. Для того, щоб виконати декілька операцій послідовно їх включають їх у файл робочого процесу (workflow file). В даній роботі було реалізовано три робочих процеси: процес інтеграції, процес розгортання клієнтської та серверної частин.

Процес інтеграції складається з наступних кроків:

а) встановлення усіх бібліотек, що потребуються клієнтською частиною додатка, за допомогою команди npm install;

б) запуск тестів клієнтської частини додатка за допомогою команди npm run test;

в) встановлення усіх бібліотек, що потребуються серверною частиною додатка, за допомогою команди npm install;

г) запуск тестів серверної частини додатка за допомогою команди npm run test.

Процес розгортання клієнтської частини складається з наступних кроків:

а) встановлення необхідних бібліотек за допомогою команди npm install;

б) створення фінальної версії додатку за допомогою команди npm build;

в) завантаження отриманих файлів у відповідну папку сервісу S3.

Процес розгортання основного сервісу складається з наступних кроків:

а) встановлення усіх бібліотек за допомогою команди npm install;

б) створення фінальної версії додатка за допомогою команди npm build;

в) запис отриманих файлів та бібліотек у архів;

г) завантаження отриманого архіву на сервер за допомогою Elastic Beanstalk.

Elastic Beanstalk є важливою частиною процесу, адже це хмарний сервіс, що дозволяє легко розгортати, масштабувати та контролювати стан веб-додатків, розроблених на основних мовах програмування. Це дозволяє автоматизувати процес підготовки сервера до розгортання, який включає встановлення програмного середовища NodeJS та пакетного менеджера Npm.

Діаграму розгортання можна побачити на рисунку 4.1.

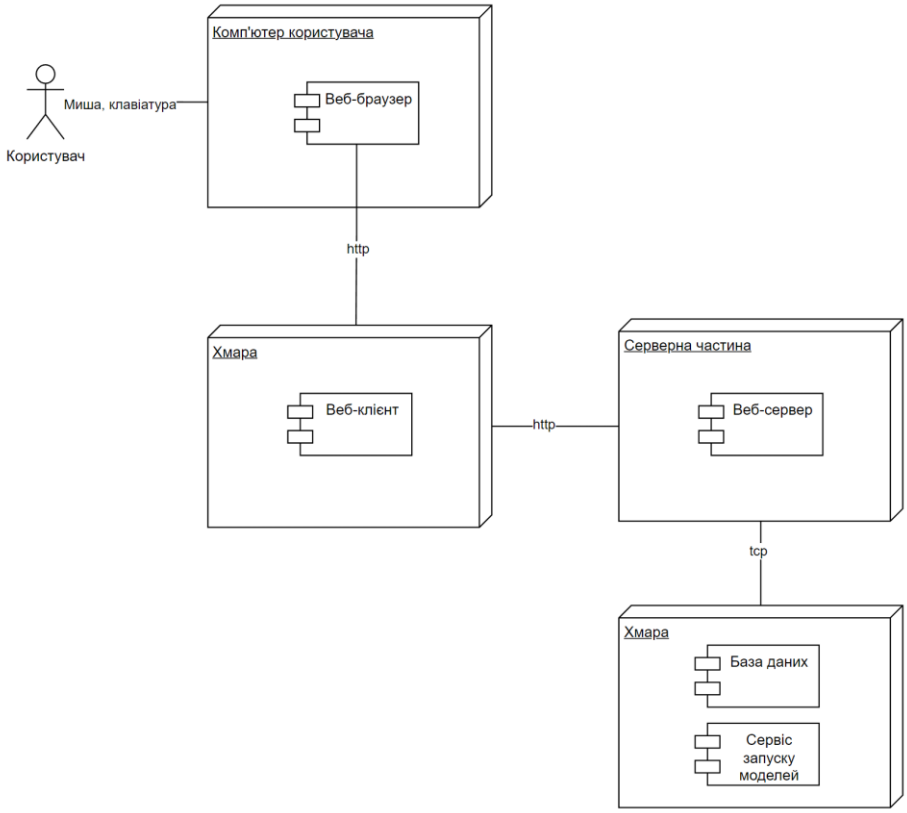

Рисунок 4.1 – Діаграма розгортання програмного забезпечення

4.2 Підтримка програмного забезпечення

Використання неперервної інтеграції та розгортання значно спрощує задачу підтримки програмного забезпечення, адже дозволяє скоротити час від моменту нової зміни, опублікованої розробником, до моменту розгортання цієї зміни на сервері.

На жаль, не всі помилки можна помітити на етапі огляду коду чи запуску автоматичних тестів. Для моніторингу розгорнутого додатка у режимі реального часу було використано ще один сервіс хмарного провайдера під назвою AWS Cloudwatch.

AWS Cloudwatch дозволяє переглядати файли журналів, створювати автоматичні повідомлення на основі метрик, що публікує сервер та будувати широкий спектр панелей з метриками, що оновлюються в режимі реального часу.

Через те, що розглянуте програмне рішення є онлайн платформою, вимоги слідкування та завантаження оновлень до користувачів не висуваються.

Висновки до розділу

В даному розділі було описано процес інтеграції та автоматичного розгортання програмного забезпечення. Також було розглянуто підхід до моніторингу та підтримки застосунку.

ВИСНОВКИ

Першим етапом виконання дипломного проекту став аналіз процесу пошуку та взаємодії із наставником. Було розглянуто основні технічні рішення, що сприяють вирішенню поставлених задач. Також було проведено порівняльний аналіз відомих аналогів. Отримані дані показали, що наявних на ринку рішень недостатньо для повного розв’язання поставленої задачі. В ході аналізу було також виокремлено основні сфери застосування додатка, а саме: галузь вищої освіти та інформаційних технологій.

На основі отриманих даних було сформовано формальний список функціональних та нефункціональних вимог до застосунку, що розробляється.

На основі цих вимог було спроектовано та успішно розроблено веб-додаток для спілкування в тематичних громадах та автоматичного пошуку наставника. Для проектування використовувалися такі сучасні підходи, як опис бізнес-процесів за допомогою BPMN-діаграм та предметно-орієнтоване проектування. Отримане програмне забезпечення, в першу чергу, дозволяє користувачам об’єднуватися у тематичні громади та обмінюватися досвідом у форматі відповідей на поширені питання. Окрім цього, воно спрощує процес пошуку та вибору оптимального наставника для кожного окремого користувача за допомогою побудови системи рекомендацій на основі глибинного навчання.

Особливу увагу при проектуванні та реалізації було приділено використанню сервісів хмарного провайдера та автоматизації процесу розгортання додатку.

Після реалізації застосунку було описано процес верифікації його працездатності та відповідності функціональним вимогам на трьох різних рівнях – рівні окремих методів та класів, рівні цілого сервісу та рівні кінцевого користувача. Окремо було розглянуто контрольний приклад. У

результаті тестування було успішно підтверджено працездатність розробленого програмного забезпечення.

Отже, розроблене програмне забезпечення повністю відповідає як функціональним, так і нефункціональним вимогам, а також впроваджує сучасні технічні підходи і рішення такі як предметно-орієнтоване програмування та використання хмари, як платформи для розгортання. Окрім цього, воно є актуальним для користувачів та простим у використанні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Akin L., Hilbun J. E-Mentoring in Three Voices. *Online Journal of Distance Learning Administration.* 2007. Т. 10, № 1. С. 1.

2) Frischer J., Larsson K. Laissez-faire in research education – An inquiry into a Swedish doctoral program. *High Education Policy*, 2000. Т. 13, № 2. С. 132-155.

3) Danielle M. E-Mentoring. *USC: Center for Excellence in Teaching*. 2004. С. 11-25.

4) Single P., Single M. E-mentoring for social equity: review of research to inform program development. *Mentoring & Tutoring*. 2005. Т. 13, № 2. С. 301–310.

5) Handbook of Youth Mentoring. 2455 Teller Road, Thousand Oaks California 91320 United States : SAGE Publications, Inc., 2005. С.130.

6) Carr C. T., Hayes R. A. Social Media: Defining, Developing, and Divining. Atlantic Journal of Communication. 2015. Т. 23, № 1. С. 8.

7) Single P., Single M. E-mentoring for social equity: review of research to inform program development. *Mentoring & Tutoring*. 2005. Т. 13, № 2. С. 301–310.

8) E-moderating: The Key to Teaching and Learning Online / G. Salmon та ін. *RoutledgeFalmer*. 2004. Т. 1, № 2. С. 1–10.

9) Bacikova M. Using social networks for supervising and mentoring. *2014 IEEE 12th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, м. Stary Smokovec, Slovakia, 4–5 груд. 2014 р. 2014.

10) Meltem O. 10 facts about Americans and Twitter. *Pew Research Center*. URL: https://www.pewresearch.org/short-reads/2022/05/05/10-facts-about-americans-and-twitter/.

11) Watch Me Code / T. Faas та ін. *Proceedings of the ACM on Human-Computer Interaction*. 2018. Т. 2, CSCW. С. 1–18.

12) Pirker J., Steinmaurer A. Beyond Gaming: The Potential of Twitch for Online Learning and Teaching. *ITiCSE '21: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education*. 2021. Т. 1, № 1. С. 74–80.

13) Tänavsuu T. How can they be so good?: The strange story of Skype. *Ars Technica. URL:* https://arstechnica.com/information-technology/2018/09/skypes-secrets/.

14) Browser Market Share Worldwide | Statcounter Global Stats. *StatCounter Global Stats*. URL: https://gs.statcounter.com/browser-market-share (дата звернення: 14.05.2023).

15) Web Sockets Now Available In Google Chrome. *Chromium Blog*. URL: https://blog.chromium.org/2009/12/web-sockets-now-available-in-google.html (дата звернення: 14.05.2023).

16) RFC ft-ietf-hybi-thewebsocketprotocol: The WebSocket Protocol. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc6455.

17) Cantisano T. 750 million downloads and 2 trillion minutes of free video calls: Skype celebrates 10 years. *Neowin*. URL: https://www.neowin.net/news/750-millionusers-and2-trillion-minutes-of-free-video-calls-skype-celebrates-10-years (дата звернення: 14.05.2023).

18) Grozev B. Efficient and scalable video conferences with selective forwarding units and webRTC : thesis. 2019. URL: http://www.theses.fr/2019STRAD055 (дата звернення: 14.05.2023).

19) *San José State University*. URL: https://www.sjsu.edu/workanywhere/docs/Zoom%20Message.pdf.

20) Google release of WebRTC source code from Harald Alvestrand on 2011-05-31 (public-webrtc@w3.org from May 2011). *W3C Public mailing list archives*. URL: https://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html.

21) What is STUN? What is STUN used for? - Huawei. *Huawei*. URL: https://info.support.huawei.com/info-finder/encyclopedia/en/STUN.html (дата звернення: 14.05.2023).

22) Collaborative Filtering Vs Content-Based Filtering for Recommender Systems. *Analytics India Magazine*. URL: https://analyticsindiamag.com/collaborative-filtering-vs-content-based-filtering-for-recommender-systems/ (дата звернення: 14.05.2023).

23) Skills2Job: A recommender system that encodes job offer embeddings on graph databases / A. Giabelli та ін. *Applied Soft Computing*. 2020. URL: https://doi.org/10.1016/j.asoc.2020.107049 (дата звернення: 14.05.2023).

24) Yesilli A. Design a Movie Recommendation System with using Graph Database (neo4j). https://medium.com/.
URL: https://medium.com/@yesilliali/design-a-movie-recommendation-system-with-using-graph-database-neo4j-part-1-2c4933f4da0a.

25) Reversed CF: A fast collaborative filtering algorithm using a k-nearest neighbor graph / Y. Park та ін. *Expert Systems with Applications*. 2015. Т. 42, № 8. С. 4022–4028.
URL: https://doi.org/10.1016/j.eswa.2015.01.001 (дата звернення: 14.05.2023).

26) Neural Collaborative Filtering / X. He та ін. *WWW '17: 26th International World Wide Web Conference*, м. Perth Australia. Republic and Canton of Geneva, Switzerland, 2017.
URL: https://doi.org/10.1145/3038912.3052569 (дата звернення: 14.05.2023).

27) What is Cloud Computing. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is-cloud-computing/ (дата звернення: 14.05.2023).

28) AWS Lambda Provisioned Concurrency AutoScaling configuration with AWS CDK. Jeroen Reijn. URL: https://www.jeroenreijn.com/2022/07/aws-lambda-provisioned-concurrency-autoscaling-configuration-with-aws-cdk.html.

29) SaaS vs PaaS vs IaaS: Examples, differences, & how to choose. eG Innovations. URL: https://www.eginnovations.com/blog/saas-vs-paas-vs-iaas-examples-differences-how-to-choose/.

30) Jenkins : ZenTimestamp Plugin. *Jenkins Wiki : Home*.
URL: https://wiki.jenkins-ci.org/display/JENKINS/ZenTimestamp+Plugin (дата звернення: 14.05.2023).

31) Nimesh C. A Comparative Analysis of Node.js (Server-Side JavaScript) : Culminating Projects in Computer Science and Information Technology. 2016.

32) State of JavaScript 2022: Front-end Frameworks. *State of JavaScript 2022*.
URL: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks (дата звернення: 14.05.2023).

33) Leonardi P. M., Meyer S. R. Social Media as Social Lubricant. *American Behavioral Scientist*. 2014. Т. 59, № 1. С. 10–34. URL: https://doi.org/10.1177/0002764214540509 (дата звернення: 14.05.2023).

34) Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Pearson Education, Limited, 2021.

35) Clean Architecture based on my open-source project. Paweł Pindel | .NET - Azure - Software Development. URL: https://netsharpdev.com/2019/12/04/clean-architecture/.

36) Building a customized recommender system in Amazon SageMaker | Amazon Web Services. *Amazon Web Services*. URL: https://aws.amazon.com/blogs/machine-learning/building-a-customized-recommender-system-in-amazon-sagemaker/ (дата звернення: 23.05.2023).

37) CQRS pattern - Azure Architecture Center. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs (дата звернення: 14.05.2023).

38) Chance E. https://elliotchance.medium.com/a-new-simpler-way-to-do-dependency-injection-in-go-9e191bef50d5. https://elliotchance.medium.com/. URL: https://elliotchance.medium.com/a-new-simpler-way-to-do-dependency-injection-in-go-9e191bef50d5.

39) Documentation | NestJS - A progressive Node.js framework. Documentation | NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/modules.

40) AWS Events. AWS re:Invent 2019: [REPEAT 1] Amazon DynamoDB deep dive: Advanced design patterns (DAT403-R1), 2019. YouTube. URL: https://www.youtube.com/watch?v=6yqfmXiZTlM

41) AWS Events. AWS re:Invent 2019: Data modeling with Amazon DynamoDB (CMY304), 2019. YouTube. URL: https://www.youtube.com/watch?v=DIQVJqiSUkE.

42) Monteiro R. Automation Testing Pyramid Today - DZone. dzone.com. URL: https://dzone.com/articles/automation-testing-pyramid-today.

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015289110

Дата перевірки:
28.05.2023 11:44:00 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
28.05.2023 11:47:25 EEST

ID користувача:
76913

Назва документа: **ІП-96_Бородаєнко_ПЗ**

Кількість сторінок: **99** Кількість слів: **14964** Кількість символів: **116163** Розмір файлу: **5.64 MB** ID файлу: **1014961450**

5.51%
Схожість

Найбільша схожість: **1.08%** з джерелом з Бібліотеки (ID файлу: **1011358348**)

3.09% Джерела з Інтернету	234	Сторінка 101
4.2% Джерела з Бібліотеки	252	Сторінка 103

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ХМАРНА ПЛАТФОРМА ДЛЯ ПОШУКУ МЕНТОРІВ З ПРОГРАМУВАННЯ

Текст програми

КПІ.ІП-9602.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Ярослав БОРОДАЄНКО

Київ – 2023

Файл Chat.ts

```
import { nanoid } from "nanoid";
import { Entity } from "../lib/Entity";
import { RemoveMethods } from "../lib/typings";
import { ChatId } from "./ChatId";
import { Message } from "./Message";
import { CoachEmail } from "../user/coach/CoachEmail";
import { StudentEmail } from "../user/student/StudentEmail";
import { MessageSent } from "./events/MessageSent.event";

export class Chat extends Entity<Chat> {
  public id: ChatId;
  public coach: CoachEmail;
  public student: StudentEmail;

  public static createNew(chat: Omit<RemoveMethods<Chat>, "id" | "messages">) {
    return new Chat({ ...chat, id: nanoid(8) })
  }

  public static initialize(chat: RemoveMethods<Chat>) {
    return new Chat({ ...chat })
  }

  public send(messageContent: string, userId: CoachEmail | StudentEmail) {
    const message = Message.createNew({
      content: messageContent,
      author: userId,
      createdAt: new Date(),
      chat: this.id,
    });

    this.events.push(new MessageSent(message));
  }
}
```

Файл Message.ts

```
import { nanoid } from "nanoid";
import { ValueObject } from "../lib/ValueObject";
import { RemoveMethods } from "../lib/typings";
import { MessageId } from "./MessageId";
import { StudentEmail } from "../user/student/StudentEmail";
import { CoachEmail } from "../user/coach/CoachEmail";
import { ChatId } from "./ChatId";
```

```
export class Message extends ValueObject<Message> {
  public id: MessageId;
  public content: string;
  public chat: ChatId;
  public createdAt: Date;
  public author: CoachEmail | StudentEmail;

  public static createNew(message: Omit<RemoveMethods<Message>, "id">) {
    return new Message({ ...message, id: nanoid(8) })
  }
}
```

Файл Comment.ts

```
import { nanoid } from "nanoid";
import { Entity } from "../lib/Entity";
import { CommentId } from "./CommentId";
import { PostId } from "../post/PostId";
import { RemoveMethods } from "../lib/typings";

export class Comment extends Entity<Comment> {
  public id: CommentId;
  public content: string;
  public createdBy: string;
  public replyTo: PostId | CommentId;
  public postId: PostId;
  public createdAt: Date;

  public static initialize(comment: RemoveMethods<Comment>) {
    return new Comment({ ...comment })
  }

  public static replyToPost(postId: PostId, content: string, user: string): Comment {
    return new Comment({
      id: `${postId}#${nanoid(8)}`,
      content,
      createdBy: user,
      replyTo: postId,
      postId,
      createdAt: new Date(),
    });
  }

  public replyToOtherComment(content: string, user: string): Comment {
```

```
    return new Comment({
      id: `${this.id}#${nanoid(8)}`,
      content,
      createdBy: user,
      replyTo: this.id,
      postId: this.postId,
      createdAt: new Date(),
    });
  }
}
```

Файл Community.ts

```
import { Fair } from "../fair/Fair";
import { Entity } from "../lib/Entity";
import { RemoveMethods } from "../lib/typings";
import { CommunityName } from "./CommunityName";
import { UserJoinedCommunityEvent } from "./events/UserJoinedCommunityEvent.event";
import { UserLeftCommunityEvent } from "./events/UserLeftCommunityEvent.event";

export class Community extends Entity<Community> {
  public name: CommunityName;
  public description: string;
  public membersCount: number;

  public static createNew(community: Omit<RemoveMethods<Community>, "membersCount">, fair: Fair) {
    return new Community({ ...community, membersCount: 0 })
  }

  public static initialize(community: RemoveMethods<Community>) {
    return new Community({ ...community })
  }

  public join(userId: string) {
    this.events.push(new UserJoinedCommunityEvent(userId, this.name));
    this.membersCount++;
  }

  public leave(userId: string) {
    this.events.push(new UserLeftCommunityEvent(userId, this.name));
    this.membersCount--;
  }
}
```

Файл UserJoinedCommunityEvent.ts

```
export class UserJoinedCommunityEvent {
```

```
  constructor(
    public readonly userId: string,
    public readonly communityId: string
  ) {}
}
```

Файл UserLeftCommunityEvent.ts

```
export class UserLeftCommunityEvent {
  constructor(
    public readonly userId: string,
    public readonly communityId: string
  ) {}
}
```

Файл Fair.ts

```
import { CommunityName } from "../community/CommunityName";
import { Entity } from "../lib/Entity";
import { RemoveMethods } from "../lib/typings";
import { Coach } from "../user/coach/Coach";
import { FairCoach } from "./FairCoach";
import { CoachJoinedFairEvent } from "./events/CoachJoinedFairEvent.event";
import { CoachLeftFairEvent } from "./events/CoachLeftFairEvent.event";

export class Fair extends Entity<Fair> {
  public community: CommunityName;
  public membersCount: number;

  public static createNew(fair: Omit<RemoveMethods<Fair>, "coaches" | "membersCount">) {
    return new Fair({ ...fair, membersCount: 0 })
  }

  public static initialize(fair: RemoveMethods<Fair>) {
    return new Fair({ ...fair })
  }

  public join(coach: Coach) {
    this.events.push(new CoachJoinedFairEvent(coach.email, this.community));
    this.membersCount++;
  }

  public leave(coach: Coach) {
    this.events.push(new CoachLeftFairEvent(coach.email, this.community));
    this.membersCount--;
  }
}
```

Файл Post.ts

```
import { nanoid } from "nanoid";
import { Entity } from "../lib/Entity";
import { RemoveMethods } from "../lib/typings";
import { PostId } from "./PostId";

export class Post extends Entity<Post> {
  public id: PostId;

  public title: string;
  public content: string;
  public community: string;

  public createdBy: string;
  public createdAt: Date;

  public static createNew(post: Omit<RemoveMethods<Post>, "id" | "createdAt">) {
    return new Post({ ...post, id: nanoid(8), createdAt: new Date() })
  }
}
```

Файл Coach.ts

```
import { Entity } from "../../lib/Entity";
import { RemoveMethods } from "../../lib/typings";
import { CoachEmail } from "./CoachEmail";
import { WorkExperience } from "../WorkExperience";
import { Location } from "../Location";
import { StudentEmail } from "../student/StudentEmail";
import { CoachStudent } from "./CoachStudent";
import { Student } from "../student/Student";
import { ChatId } from "src/domain/chat/ChatId";
import { BadRequestException } from "@nestjs/common";

export class Coach extends Entity<Coach> {
  public email: CoachEmail;
  public passwordHashed: string;

  public nickname: string;
  public name: string;
  public programmingLanguages: string[];
  public languages: string[];
  public workExperience: WorkExperience[];
  public location: Location;
```

```
  public mentorshipRequests: StudentEmail[];
  public students: CoachStudent[];

  public static createNew(coach: { email: string, passwordHashed: string }) {
    return new Coach({
      ...coach,
      name: "unknown",
      nickname: coach.email.split("@", 1)[0],
      programmingLanguages: [],
      languages: [],
      workExperience: [],
      location: Location.unknown(),
      mentorshipRequests: [],
      students: [],
    })
  }

  public static initialize(coach: RemoveMethods<Coach>) {
    return new Coach(coach)
  }

  public edit(coach: Omit<RemoveMethods<Coach>, "students" | "mentorshipRequests" | "passwordHashed" |
"email" | "nickname">) {
    this.name = coach.name;
    this.programmingLanguages = coach.programmingLanguages;
    this.languages = coach.languages;
    this.location = coach.location;
    this.workExperience = coach.workExperience;
  }

  public yearsOfExperience() {
    return Math.floor(this.workExperience.map(w => w.durationInMonths())
      .reduce((d1, d2) => d1 + d2, 0) / 12);
  }

  public applyForMentorship(student: StudentEmail) {
    if (this.mentorshipRequests.includes(student)) throw new BadRequestException("Student is already in the
applicants list");
    if (this.students.map(s => s.student).includes(student)) throw new BadRequestException("Student is already in
the students list");
    this.mentorshipRequests.push(student);
```

```
  }

  public approveMentorship(student: StudentEmail, chat: ChatId) {
    if (!this.mentorshipRequests.includes(student)) throw new BadRequestException("Student is not in the applicants
list");
    this.students.push(CoachStudent.createNew(this.email, student, chat));
    this.mentorshipRequests = this.mentorshipRequests.filter(s => s !== student);
  }
}
```

Файл Student.ts

```
import { ChatId } from "src/domain/chat/ChatId";
import { Entity } from "../../lib/Entity";
import { RemoveMethods } from "../../lib/typings";
import { Location } from "../Location";
import { CoachStudent } from "../coach/CoachStudent";
import { StudentEmail } from "./StudentEmail";
import { CoachEmail } from "../coach/CoachEmail";
import { BadRequestException } from "@nestjs/common";


export class Student extends Entity<Student> {
  public email: StudentEmail;
  public passwordHashed: string;

  public name: string;
  public programmingLanguages: string[];
  public languages: string[];
  public location: Location;

  public coaches: CoachStudent[];
  public mentorshipRequests: CoachEmail[];

  public static createNew(student: { email: string, passwordHashed: string }) {
    return new Student({
      ...student,
      name: "Unknown",
      programmingLanguages: [],
      languages: [],
      location: Location.unknown(),
      coaches: [],
      mentorshipRequests: [],
```

```
    })
  }

  public static initialize(student: RemoveMethods<Student>) {
    return new Student({
      ...student,
    })
  }

  public edit(student: Omit<RemoveMethods<Student>, "chatWithCoach" | "passwordHashed" | "email" | "coaches"
| "mentorshipRequests">) {
    this.name = student.name;
    this.programmingLanguages = student.programmingLanguages;
    this.languages = student.languages;
    this.location = student.location;
  }

  public applyForMentorship(coach: CoachEmail) {
    this.mentorshipRequests.push(coach);
  }

  public joinChatWithCoach(coach: CoachEmail, chat: ChatId) {
    this.coaches.push(CoachStudent.createNew(coach, this.email, chat));
    this.mentorshipRequests = this.mentorshipRequests.filter(s => s !== coach);
  }
}
```

Файл Location.ts

```
import { ValueObject } from "../lib/ValueObject";

export class Location extends ValueObject<Location> {
  public readonly country: string;
  public readonly city: string;

  public static createNew(country: string, city: string) {
    return new Location({ city, country })
  }

  public static unknown() {
    return new Location({ city: "Unknown", country: "Unknown" })
  }
```

```
}
```

Файл WorkExperience.ts

```
import { differenceInMonths } from "date-fns";
import { ValueObject } from "../lib/ValueObject";

export class WorkExperience extends ValueObject<WorkExperience> {
  public readonly company: string;
  public readonly start: Date;
  public readonly end?: Date;

  public static createFinished(company: string, start: Date, end: Date) {
    return new WorkExperience({ company, start, end })
  }

  public static createUnfinished(company: string, start: Date) {
    return new WorkExperience({ company, start })
  }

  public durationInMonths() {
    return differenceInMonths(this.end || Date.now(), this.start);
  }
}
```

Файл Entity.ts

```
import { Event } from "./Event";
import { RemoveMethods } from "./typings";

export class Entity<T> {
  public readonly events: Event[];

  constructor(entity: RemoveMethods<T>) {
    Object.assign(this, entity);
    this.events = [];
  }
}
```

Файл DynamoDbService.ts

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocument } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";

@Injectable()
export class DynamoDbService {
  readonly #client: DynamoDBDocument;
```

```
  constructor() {
    this.#client = DynamoDBDocument.from(
      new DynamoDB({}),
      {
        marshallOptions: {
          removeUndefinedValues: true,
          convertClassInstanceToMap: true,
        },
      });
  }

  public client(): DynamoDBDocument {
    return this.#client;
  }
}
```

Файл SagemakerService.ts

```
import { SageMakerRuntimeClient } from "@aws-sdk/client-sagemaker-runtime";
import { Injectable } from "@nestjs/common";

@Injectable()
export class SagemakerService {
  readonly #client: SageMakerRuntimeClient;

  constructor() {
    this.#client = new SageMakerRuntimeClient({});
  }

  public client(): SageMakerRuntimeClient {
    return this.#client;
  }
}
```

Файл ChatRepository.ts

```
import { DynamoDbService } from "../../aws/dynamodb.service";
import { Injectable } from "@nestjs/common";
import { Repository } from "./repository";
import { Chat } from "src/domain/chat/Chat";
import { DeleteCommand, GetCommand, TransactWriteCommand } from "@aws-sdk/lib-dynamodb";
import { MessageSent } from "src/domain/chat/events/MessageSent.event";
```

```
@Injectable()
export class ChatRepository implements Repository<Chat> {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async save(entity: Chat) {
    const { events, ...chat } = entity;

    await this.dynamoDb.client().send(new DeleteCommand({
      TableName: "Chats",
      Key: { pk: `Chat${chat.id}`, sk: "Identity" },
    }))

    await this.dynamoDb.client().send(new TransactWriteCommand({
      TransactItems: [
        {
          Put: {
            TableName: "Chats",
            Item: {
              pk: `Chat#${chat.id}`,
              sk: "Identity",
              ...JSON.parse(JSON.stringify(chat)),
            },
          },
        },
        ...events.filter(e => e instanceof MessageSent).map((e: MessageSent) => ({
          Put: {
            TableName: "Chats",
            Item: {
              pk: `Chat#${e.message.chat}`,
              sk: `Message#${e.message.createdAt.toISOString()}#${e.message.author}`,
              ...JSON.parse(JSON.stringify(e.message)),
            },
          },
        })),
      ],
    }));
  }

  async findOne(chatId: string): Promise<Chat> {
    const query = new GetCommand({
```

```
      TableName: "Chats",
      Key: {
        pk: `Chat#${chatId}`,
        sk: "Identity",
      },
    });

    const chat = await this.dynamoDb.client().send(query);

    // eslint-disable-next-line @typescript-eslint/no-non-null-assertion
    return new Chat(chat.Item! as any);
  }
}
```

Файл CoachRepository.ts

```
import { DeleteCommand, GetCommand, PutCommand, ScanCommand } from "@aws-sdk/lib-dynamodb";
import { DynamoDbService } from "../../aws/dynamodb.service";
import { Injectable } from "@nestjs/common";
import { Repository } from "./repository";
import { Coach } from "src/domain/user/coach/Coach";
import { CoachEmail } from "src/domain/user/coach/CoachEmail";

@Injectable()
export class CoachRepository implements Repository<Coach> {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async save(entity: Coach) {
    const { events, ...coach } = entity;

    await this.dynamoDb.client().send(new PutCommand({
      TableName: "Users",
      Item: {
        pk: `Coach#${coach.email}`,
        sk: "Identity",
        role: "Coach",
        ...JSON.parse(JSON.stringify(coach)),
      },
    }));
  }

  async findOne(id: CoachEmail): Promise<Coach> {
```

```
    const query = new GetCommand({
      TableName: "Users",
      Key: {
        "pk": `Coach#${id}`,
        "sk": "Identity",
      },
    });

    const coach = (await this.dynamoDb.client().send(query)).Item as any;
    return Coach.initialize(coach);
  }

  // async deleteOne(email: CoachEmail) {
  //   const query = new ScanCommand({
  //     TableName: "Users",
  //     FilterExpression: "#pk = :pk",
  //     ExpressionAttributeNames: { "#pk": "pk" },
  //     ExpressionAttributeValues: { ":pk": `Coach#${email}` },
  //   });

  //   const elements = (await this.dynamoDb.client().send(query)).Items;

  //   console.log(elements);

  //   for (const element of elements) {
  //     const deleteQ = new DeleteCommand({
  //       TableName: "Users",
  //       Key: {
  //         "pk": `Coach#${email}`, "sk": element.sk,
  //       },
  //     })

  //     await this.dynamoDb.client().send(deleteQ)
  //   }
  // }
}
```

Файл CommentRepository.ts

```
import { PutCommand, ScanCommand } from "@aws-sdk/lib-dynamodb";
import { DynamoDbService } from "../../aws/dynamodb.service";
import { Injectable } from "@nestjs/common";
import { Repository } from "./repository";
import { Comment } from "src/domain/comment/Comment";
```

```
import { CommentId } from "src/domain/comment/CommentId";
import { Select } from "@aws-sdk/client-dynamodb";

@Injectable()
export class CommentRepository implements Repository<Comment> {
 constructor (
  private readonly dynamoDb: DynamoDbService
 ) {}

 async save(entity: Comment) {
  const { events, ...comment } = entity;

  await this.dynamoDb.client().send(new PutCommand({
   TableName: "Posts",
   Item: {
    pk: `Post#${comment.postId}`,
    sk: `Comment#${comment.id}`,
    ...JSON.parse(JSON.stringify(comment)),
   },
  }));
 }

 async findOne(id: CommentId): Promise<Comment> {
  const query = new ScanCommand({
   TableName: "Posts",
   Select: Select.ALL_ATTRIBUTES,
   FilterExpression: "contains(#pk, :pk) and contains(#sk, :sk)",
   ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
   ExpressionAttributeValues: { ":pk": "Post#", ":sk": `Comment#${id}` },
  });

  const comment = (await this.dynamoDb.client().send(query)).Items[0] as any;
  return Comment.initialize(comment);
 }
}
```

Файл CommunityRepository.ts

```
import { DynamoDbService } from "../../aws/dynamodb.service";
import { Injectable } from "@nestjs/common";
import { Repository } from "./repository";
import { Community } from "src/domain/community/Community";
import { DeleteCommand, QueryCommand, ScanCommand, TransactWriteCommand } from "@aws-sdk/lib-
dynamodb";
```

```
import { UserJoinedCommunityEvent } from "src/domain/community/events/UserJoinedCommunityEvent.event";
import { CommunityName } from "src/domain/community/CommunityName";
import { Select } from "@aws-sdk/client-dynamodb";
import { UserLeftCommunityEvent } from "src/domain/community/events/UserLeftCommunityEvent.event";

@Injectable()
export class CommunityRepository implements Repository<Community> {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async save(entity: Community) {
    const { events, ...community } = entity;

    const leftEvents = events.filter(e => e instanceof UserLeftCommunityEvent).map(e => e as
UserLeftCommunityEvent);
    const joinEvents = events.filter(e => e instanceof UserJoinedCommunityEvent)
      .map(e => e as UserJoinedCommunityEvent);

    await this.dynamoDb.client().send(new TransactWriteCommand({
      TransactItems: [
        {
          Put: {
            TableName: "Communities",
            Item: {
              pk: `Community#${community.name}`,
              sk: "Identity",
              ...JSON.parse(JSON.stringify(community)),
            },
          },
        },
        ...joinEvents.map(e => ({
          Put: {
            TableName: "Communities",
            Item: {
              pk: `Community#${community.name}`,
              sk: `Member#${e.userId}`,
              email: e.userId,
            },
          },
        })),
        ...leftEvents.map(e => ({
```

```
        Delete : {
          TableName: "Communities",
          Key: {
            pk: `Community#${community.name}`,
            sk: `Member#${e.userId}`,
          },
        },
      })),
    ] }));
  }

  async findOne(id: CommunityName): Promise<Community> {
    const query = new QueryCommand({
      TableName: "Communities",
      Select: Select.ALL_ATTRIBUTES,
      KeyConditionExpression: "#pk = :pk and #sk = :sk",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: { ":pk": `Community#${id}`, ":sk": "Identity" },
    });

    const community = (await this.dynamoDb.client().send(query)).Items[0] as any;
    return Community.initialize(community);
  }
}
```

Файл FairRepository.ts

```
import { DynamoDbService } from "../../aws/dynamodb.service";
import { Injectable } from "@nestjs/common";
import { Repository } from "./repository";
import { Fair } from "src/domain/fair/Fair";
import { TransactWriteItemsCommand } from "@aws-sdk/client-dynamodb";
import { QueryCommand, TransactWriteCommand } from "@aws-sdk/lib-dynamodb";
import { CommunityName } from "src/domain/community/CommunityName";
import { CoachLeftFairEvent } from "src/domain/fair/events/CoachLeftFairEvent.event";
import { CoachJoinedFairEvent } from "src/domain/fair/events/CoachJoinedFairEvent.event";

@Injectable()
export class FairRepository implements Repository<Fair> {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async save(fairE: Fair) {
```

```
  const { events, ...fair } = fairE;

  const leftEvents = events.filter(e => e instanceof CoachLeftFairEvent).map(e => e as CoachLeftFairEvent);
  const joinEvents = events.filter(e => e instanceof CoachJoinedFairEvent)
   .map(e => e as CoachJoinedFairEvent);

  await this.dynamoDb.client().send(new TransactWriteCommand({
   TransactItems: [
    {
     Put: {
      TableName: "Fairs",
      Item: {
       pk: `Fair#${fair.community}`,
       sk: "Identity",
       ...JSON.parse(JSON.stringify(fair)),
      },
     },
    },
    ...joinEvents.map(e => ({
     Put: {
      TableName: "Fairs",
      Item: {
       pk: `Fair#${fair.community}`,
       sk: `Coach#${e.coachId}`,
       email: e.coachId,
      },
     },
    })),
    ...leftEvents.map(e => ({
     Delete : {
      TableName: "Fairs",
      Key: {
       pk: `Fair#${fair.community}`,
       sk: `Coach#${e.coachId}`,
      },
     },
    })),
   ] }));
 }

 async findOne(id: CommunityName): Promise<Fair> {
  const query = new QueryCommand({
```

```
      TableName: "Fairs",
      KeyConditionExpression: "#pk = :pk and #sk = :sk",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: { ":pk": `Fair#${id}`, ":sk": "Identity" },
    });

    const fair = (await this.dynamoDb.client().send(query)).Items[0] as any;
    return Fair.initialize(fair);
  }
}
```

Файл PostRepository.ts

```
import { PutCommand } from "@aws-sdk/lib-dynamodb";
import { DynamoDbService } from "../../aws/dynamodb.service";
import { Injectable } from "@nestjs/common";
import { Repository } from "./repository";
import { Post } from "src/domain/post/Post";

@Injectable()
export class PostRepository implements Repository<Post> {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async save(entity: Post) {
    const { events, ...post } = entity;

    await this.dynamoDb.client().send(new PutCommand({
      TableName: "Posts",
      Item: {
        pk: `Post#${post.id}`,
        sk: `Identity#${post.createdAt.toISOString()}`,
        ...JSON.parse(JSON.stringify(post)),
      },
    }));
  }
}
```

Файл StudentRepository.ts

```
import { QueryCommand, PutCommand } from "@aws-sdk/lib-dynamodb";
import { DynamoDbService } from "../../aws/dynamodb.service";
import { Student } from "src/domain/user/student/Student";
import { Injectable } from "@nestjs/common";
import { Repository } from "./repository";
```

```
import { StudentEmail } from "src/domain/user/student/StudentEmail";
import { Select } from "@aws-sdk/client-dynamodb";

@Injectable()
export class StudentRepository implements Repository<Student> {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async save(entity: Student) {
    const { events, ...student } = entity;

    await this.dynamoDb.client().send(new PutCommand({
      TableName: "Users",
      Item: {
        pk: `Student#${student.email}`,
        sk: "Identity",
        role: "Student",
        ...JSON.parse(JSON.stringify(student)),
      },
    }));
  }

  async findOne(id: StudentEmail): Promise<Student> {
    const query = new QueryCommand({
      TableName: "Users",
      Select: Select.ALL_ATTRIBUTES,
      KeyConditionExpression: "#pk = :pk",
      ExpressionAttributeNames: { "#pk": "pk" },
      ExpressionAttributeValues: { ":pk": `Student#${id}` },
    });

    const student = (await this.dynamoDb.client().send(query)).Items[0] as any;
    return Student.initialize(student);
  }
}
```

Файл SendMessageCommand.ts

```
import { Injectable } from "@nestjs/common";
import { ChatRepository } from "src/infrastructure/persistance/repositories/chats.repository";
import { ChatsWebSocketOutGateway } from "../chats.outgateway";
import { MessageSent } from "src/domain/chat/events/MessageSent.event";
```

```
@Injectable()
export class SendMessageCommand {
  constructor (
    private readonly chatRepository: ChatRepository,
    private readonly chatsWebSocketGateway: ChatsWebSocketOutGateway
  ) {}

  async execute(
    { chatId, message, author }:
    { chatId: string, message: string, author: string }
  ) {
    const chat = await this.chatRepository.findOne(chatId);

    chat.send(message, author);

    await this.chatRepository.save(chat);

    chat.events.filter(e => e instanceof MessageSent)
      .forEach(({ message }: MessageSent) => {
        this.chatsWebSocketGateway.emit(message);
      })
  }
}
```

Файл GetByIdQuery.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { GetCommand, ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetByIdQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(id: string) {
    const query = new GetCommand({
      TableName: "Chats",
      Key: {
        "pk": `Chat#${id}`,
        "sk": "Identity",
      },
```

```
    });

    return (await this.dynamoDb.client().send(query)).Item;
  }
}
```

Файл GetMessagesQuery.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetMessagesQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(chatId: string) {
    const query = new ScanCommand({
      TableName: "Chats",
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "#pk = :pk and contains(#sk, :sk)",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: { ":pk": `Chat#${chatId}`, ":sk": "Message#" },
    });

    return (await this.dynamoDb.client().send(query)).Items;
  }
}
```

Файл ChatsController.ts

```
import { Controller, Get, Param } from "@nestjs/common";
import { GetByIdQuery } from "./queries/get-by-id.query";
import { GetMessagesQuery } from "./queries/get-messages.query";

@Controller("chats")
export class ChatsController {
  constructor(
    private readonly getByIdQuery: GetByIdQuery,
    private readonly getMessagesQuery: GetMessagesQuery
  ) {}

  @Get(":id/messages")
```

```
  getMessages(@Param("id") chatId: string) {
    return this.getMessagesQuery.execute(chatId);
  }

  @Get(":id")
  getById(@Param("id") chatId: string) {
    return this.getByIdQuery.execute(chatId);
  }
}
```

Файл ApproveMentorshipRequestCommand.ts

```
import { Injectable } from "@nestjs/common";
import { Chat } from "src/domain/chat/Chat";
import { ChatRepository } from "src/infrastructure/persistance/repositories/chats.repository";
import { CoachRepository } from "src/infrastructure/persistance/repositories/coach.repository";
import { StudentRepository } from "src/infrastructure/persistance/repositories/student.repository";

@Injectable()
export class ApproveMentorshipRequestCommand {
  constructor (
    private readonly coachRepository: CoachRepository,
    private readonly chatRepository: ChatRepository,
    private readonly studentRepository: StudentRepository
  ) {}

  async execute(authenticatedCoachId: string, studentId: string) {
    const chat = Chat.createNew({ coach: authenticatedCoachId, student: studentId });
    await this.chatRepository.save(chat);

    const coach = await this.coachRepository.findOne(authenticatedCoachId);
    coach.approveMentorship(studentId, chat.id);
    await this.coachRepository.save(coach);

    const student = await this.studentRepository.findOne(studentId);
    student.joinChatWithCoach(coach.email, chat.id);
    await this.studentRepository.save(student);

    return coach.students.filter(s => s.student === studentId)[0];
  }
}
```

Файл EditCoachCommand.ts

```
import { Injectable } from "@nestjs/common";
import { Location } from "src/domain/user/Location";
```

```
import { CoachRepository } from "src/infrastructure/persistance/repositories/coach.repository";

@Injectable()
export class EditCoachCommand {
  constructor (
    private readonly coachRepository: CoachRepository
  ) {}

  async execute(
    authenticatedCoachId: string,
    coach: {
      languages: string[],
      programmingLanguages: string[],
      location: { city: string, country: string },
      name: string
    }
  ) {
    const c = await this.coachRepository.findOne(authenticatedCoachId);

    const { languages, programmingLanguages, name, location } = coach;

    c.edit({
      languages,
      programmingLanguages,
      name,
      location: Location.createNew(location.country, location.city),
      workExperience: c.workExperience,
    });

    await this.coachRepository.save(c);
  }
}
```

Файл RequestMentorshipCommand.ts

```
import { Injectable } from "@nestjs/common";
import { CoachRepository } from "src/infrastructure/persistance/repositories/coach.repository";
import { StudentRepository } from "src/infrastructure/persistance/repositories/student.repository";

@Injectable()
export class RequestMentorshipCommand {
  constructor (
    private readonly coachRepository: CoachRepository,
    private readonly studentRepository: StudentRepository,
```

```
  ) {}

  async execute(authenticatedStudentId: string, coachId: string) {
    const coach = await this.coachRepository.findOne(coachId);
    coach.applyForMentorship(authenticatedStudentId);
    await this.coachRepository.save(coach);

    const student = await this.studentRepository.findOne(authenticatedStudentId);
    student.applyForMentorship(coachId);
    await this.studentRepository.save(student);
  }
}
```

Файл GetAllCoachesQuery.ts

```
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Select } from "@aws-sdk/client-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetAllCoachesQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute() {
    const query = new ScanCommand({
      TableName: "Users",
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "contains(#pk, :pk)",
      ExpressionAttributeNames: { "#pk": "pk" },
      ExpressionAttributeValues: { ":pk": "Coach#" },
      Limit: 10,
    });

    return (await this.dynamoDb.client().send(query)).Items;
  }
}
```

Файл GetCoachByIdQuery.ts

```
import { GetCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";
```

```
@Injectable()
export class GetCoachByIdQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(id: string) {
    const query = new GetCommand({
      TableName: "Users",
      Key: { "pk": `Coach#${id}`, "sk": "Identity" },
    });

    return (await this.dynamoDb.client().send(query)).Item;
  }
}
```

Файл CoachesController.ts

```
import { Controller, Get, Param, Post, Body } from "@nestjs/common";
import { GetAllCoachesQuery } from "src/modules/coaches/queries/get-all.query";
import { GetCoachByIdQuery } from "./queries/get-by-id.query";
import { ApproveMentorshipRequestCommand } from "./commands/approve-mentorship-request.command";
import { AuthenticatedUser } from "../auth/decorators/authenticated-user.decorator";
import { EditCoachCommand } from "./commands/edit-coach.command";
import { RequestMentorshipCommand } from "./commands/request-mentorship.command";

@Controller("coaches")
export class CoachesController {
  constructor(
    private readonly getAllCoaches: GetAllCoachesQuery,
    private readonly getCoachByIdQuery: GetCoachByIdQuery,
    private readonly approveMentorshipRequestCommand: ApproveMentorshipRequestCommand,
    private readonly editCoachCommand: EditCoachCommand,
    private readonly requestMentorshipCommand: RequestMentorshipCommand,
  ) {}

  @Get()
  getAll() {
    return this.getAllCoaches.execute();
  }

  @Get(":id")
  getById(@Param("id") id: string) {
    return this.getCoachByIdQuery.execute(id);
```

```
  }

  @Post(":id/requestMentorship")
  requestMentorship(
    @Param("id") coach: string,
    @AuthenticatedUser() authenticatedStudent: string
  ) {
    return this.requestMentorshipCommand.execute(authenticatedStudent, coach);
  }

  @Post("approveMentorshipRequest")
  approveMentorshipRequest(
    @Body("student") student: string,
    @AuthenticatedUser() authenticatedUser: string
  ) {
    return this.approveMentorshipRequestCommand.execute(authenticatedUser, student);
  }

  @Post("self")
  edit(
    @AuthenticatedUser() authenticatedCoach: string,
    @Body("name") name: string,
    @Body("languages") languages: string[],
    @Body("programmingLanguages") programmingLanguages: string[],
    @Body("location") location: any
  ) {
    return this.editCoachCommand.execute(
      authenticatedCoach,
      {
        name,
        languages,
        programmingLanguages,
        location,
      }
    );
  }
}
```

Файл ReplyToCommentCommand.ts

```
import { Injectable } from "@nestjs/common";
import { CommentRepository } from "src/infrastructure/persistance/repositories/comment.repository";

@Injectable()
```

```
export class ReplyToCommentCommand {
  constructor (
    private readonly commentRepository: CommentRepository
  ) {}

  async execute(authenticatedUserId: string, commentId: string, content: string) {
    const comment = await this.commentRepository.findOne(commentId);

    const reply = comment.replyToOtherComment(content, authenticatedUserId);

    return await this.commentRepository.save(reply);
  }
}
```

Файл CommentsController.ts

```
import { Body, Controller, Get, Param, Post, Query } from "@nestjs/common";
import { ReplyToCommentCommand } from "./commands/reply-to-comment.command";
import { AuthenticatedUser } from "../auth/decorators/authenticated-user.decorator";

@Controller("comments")
export class CommentsController {
  constructor(
    private readonly replyToCommentCommand: ReplyToCommentCommand
  ) {}

  @Post(":id/reply")
  reply(
    @Body("content") content: string,
    @Param("id") id: string,
    @AuthenticatedUser() authenticatedUser: string
  ) {
    return this.replyToCommentCommand.execute(authenticatedUser, id, content);
  }
}
```

Файл JoinCommunityCommand.ts

```
import { Injectable } from "@nestjs/common";
import { CommunityRepository } from "src/infrastructure/persistance/repositories/community.repository";

@Injectable()
export class JoinCommunityCommand {
  constructor (
    private readonly communityRepository: CommunityRepository,
  ) {}
```

```
  async execute(userId: string, communityId: string) {
    const community = await this.communityRepository.findOne(communityId);

    community.join(userId);

    await this.communityRepository.save(community);
  }
}
```

Файл LeaveCommunityCommand.ts

```
import { Injectable } from "@nestjs/common";
import { CommunityRepository } from "src/infrastructure/persistance/repositories/community.repository";

@Injectable()
export class LeaveCommunityCommand {
  constructor (
    private readonly communityRepository: CommunityRepository,
  ) {}

  async execute(userId: string, communityId: string) {
    const community = await this.communityRepository.findOne(communityId);

    community.leave(userId);

    await this.communityRepository.save(community);
  }
}
```

Файл GetAllCommunitiesQuery.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetAllCommunitiesQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService,
  ) {}

  async execute(authenticatedUser: string) {
    const query = new ScanCommand({
      TableName: "Communities",
```

```
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "contains(#pk, :pk) and #sk = :sk",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: { ":pk": "Community#", ":sk": "Identity" },
    });

    const communities = (await this.dynamoDb.client().send(query)).Items;

    for (const community of communities) {
      const query = new ScanCommand({
        TableName: "Communities",
        Select: Select.ALL_ATTRIBUTES,
        FilterExpression: "#pk = :pk and #sk = :sk",
        ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
        ExpressionAttributeValues: {
          ":pk": `Community#${community.name}`,
          ":sk": `Member#${authenticatedUser}`,
        },
      });

      const isJoined = (await this.dynamoDb.client().send(query)).Items.length > 0;
      community.isJoined = isJoined;
    }

    return communities;
  }
}
```

Файл GetByIdCommunityQuery.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetByIdCommunityQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(authenticatedUser: string, id: string) {
    const query = new ScanCommand({
      TableName: "Communities",
```

```
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "contains(#pk, :pk) and #sk = :sk",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: { ":pk": `Community#${id}`, ":sk": "Identity" },
    });

    const community = (await this.dynamoDb.client().send(query)).Items[0];

    const queryIsJoined = new ScanCommand({
      TableName: "Communities",
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "#pk = :pk and #sk = :sk",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: {
        ":pk": `Community#${community.name}`,
        ":sk": `Member#${authenticatedUser}`,
      },
    });

    const isJoined = (await this.dynamoDb.client().send(queryIsJoined)).Items.length > 0;
    community.isJoined = isJoined;

    return community;
  }
}
```

Файл CommunitiesController.ts

```
import { Controller, Get, Post, Body, Param } from "@nestjs/common";
import { GetAllCommunitiesQuery } from "src/modules/communities/queries/get-all.query";
import { JoinCommunityCommand } from "./commands/join-community.command";
import { LeaveCommunityCommand } from "./commands/leave-community.command";
import { GetByIdCommunityQuery } from "./queries/get-by-id.query";
import { AuthenticatedUser } from "../auth/decorators/authenticated-user.decorator";

@Controller("communities")
export class CommunitiesController {
  constructor(
    private readonly getAllCommunities: GetAllCommunitiesQuery,
    private readonly getByIdCommunityQuery: GetByIdCommunityQuery,
    private readonly joinCommunity: JoinCommunityCommand,
    private readonly leaveCommunity: LeaveCommunityCommand,
  ) {}
```

```
  @Get()
  getAll(@AuthenticatedUser() authenticatedUser: string) {
    return this.getAllCommunities.execute(authenticatedUser);
  }

  @Get(":id")
  getById(@Param("id") id: string, @AuthenticatedUser() authenticatedUser: string) {
    return this.getByIdCommunityQuery.execute(authenticatedUser, id);
  }

  @Post(":id/join")
  join(@Param("id") id, @AuthenticatedUser() authenticatedUser: string) {
    return this.joinCommunity.execute(authenticatedUser, id);
  }

  @Post(":id/leave")
  leave(@Param("id") id, @AuthenticatedUser() authenticatedUser: string) {
    return this.leaveCommunity.execute(authenticatedUser, id);
  }
}
```

Файл JoinFairCommand.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";
import { CoachRepository } from "src/infrastructure/persistance/repositories/coach.repository";
import { FairRepository } from "src/infrastructure/persistance/repositories/fair.repository";

@Injectable()
export class JoinFairCommand {
  constructor (
    private readonly fairRepository: FairRepository,
    private readonly coachRepository: CoachRepository
  ) {}

  async execute(authenticatedCoachId: string, community: string) {
    const fair = await this.fairRepository.findOne(community);
    const coach = await this.coachRepository.findOne(authenticatedCoachId);

    fair.join(coach);

    await this.fairRepository.save(fair);
```

```
  }
}
```

Файл LeaveFairCommand.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";
import { CoachRepository } from "src/infrastructure/persistance/repositories/coach.repository";
import { FairRepository } from "src/infrastructure/persistance/repositories/fair.repository";

@Injectable()
export class LeaveFairCommand {
  constructor (
    private readonly fairRepository: FairRepository,
    private readonly coachRepository: CoachRepository
  ) {}

  async execute(authenticatedCoachId: string, community: string) {
    const fair = await this.fairRepository.findOne(community);
    const coach = await this.coachRepository.findOne(authenticatedCoachId);

    fair.leave(coach);

    await this.fairRepository.save(fair);
  }
}
```

Файл GetAllMembersRankedQuery.ts

```
import { GetCommand, ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";
import { GetAllMembersQuery } from "./get-all-members.query";
import { InvokeEndpointCommand } from "@aws-sdk/client-sagemaker-runtime";
import { CoachRepository } from "src/infrastructure/persistance/repositories/coach.repository";
import { StudentRepository } from "src/infrastructure/persistance/repositories/student.repository";
import { Student } from "src/domain/user/student/Student";
import { SagemakerService } from "src/infrastructure/aws/sagemaker.service";
import * as fs from "fs";
import * as path from "path";
import { parse } from "csv-parse/sync";

type FairCoach = { email: string, communities: [] };
```

```
type RankedCoach = FairCoach & { recommended: boolean };

@Injectable()
export class GetAllMembersRankedQuery {
 constructor (
  private readonly getAllMembersQuery: GetAllMembersQuery,
  private readonly coachRepository: CoachRepository,
  private readonly studentRepository: StudentRepository,
  private readonly sagemakerService: SagemakerService
 ) {}

 async execute(studentEmail: string) {
  const coaches: FairCoach[] = await this.getAllMembersQuery.execute(studentEmail);
  const student = await this.studentRepository.findOne(studentEmail);

  if (process.env.USE_SAGEMAKER === "true") {
   try {
    return await this.rankUsingSagemaker(coaches, student);
   } catch (e) {
    console.error(e);
    return await this.rankUsingCustomLogic(coaches, student);
   }
  } else {
   return await this.rankUsingCustomLogic(coaches, student);
  }
 }

 private async rankUsingSagemaker(coaches: FairCoach[], student: Student): Promise<RankedCoach[]> {
  const actions = this.readActions();
  const input = this.oneHotEncode(coaches, student, actions);

  const body = {
   "instances": input.map(([studentEncoded, coachEncoded]) => ({
    "input_1": studentEncoded, "input_2": coachEncoded,
   })),
  }

  console.dir(body, { depth: null });

  const command = new InvokeEndpointCommand({
   EndpointName: "sagemaker-soln-crs-js-xd5uponeural-collab-filtering-endpoint",
   Body: Buffer.from(JSON.stringify(body)),
```

```
    ContentType: "application/json",
  })

  const response = await this.sagemakerService.client().send(command);

  const { predictions } = JSON.parse(Buffer.from(response.Body).toString("utf8"));

  console.log(predictions);

  return coaches.map((coach, index) => ({
    ...coach,
    recommended: Number.parseFloat(predictions[index]) >= 0.5,
  }));
}

private readActions() {
  const actions = fs.readFileSync(path.join(__dirname, "../../../resources/actions.csv")).toString();

  const records = parse(actions, {
    columns: true,
    delimiter: ",",
  });

  return records;
}

private oneHotEncode(coaches: FairCoach[], student: Student, actions: any[]) {
  const studentIdToMap = new Map<string, number>();
  const coachIdToMap = new Map<string, number>();
  for (const { student_id: studentId, coach_id: coachId } of actions) {
    if (!studentIdToMap.has(studentId)) {
      studentIdToMap.set(studentId, studentIdToMap.size + 1);
    }

    if (!coachIdToMap.has(coachId)) {
      coachIdToMap.set(coachId, coachIdToMap.size + 1);
    }
  }

  for (const record of actions) {
    record.studentId = studentIdToMap.get(record.student_id);
    record.studentIdString = record.student_id;
```

```
      record.coachId = coachIdToMap.get(record.coach_id);
      record.coachIdString = record.coach_id;
      record.relevance = Number.parseInt(record.relevance);
      delete record.student_id;
      delete record.coach_id;
    }

    return coaches.map(coach => {
      const studentEncoded = Array(studentIdToMap.size).fill(0);
      studentEncoded[studentIdToMap.get(student.email) - 1] = 1;

      const coachEncoded = Array(coachIdToMap.size).fill(0);
      coachEncoded[coachIdToMap.get(coach.email) - 1] = 1;

      return [studentEncoded, coachEncoded];
    });
  }

  private async rankUsingCustomLogic(coaches: FairCoach[], student: Student): Promise<RankedCoach[]> {
    const results: RankedCoach[] = [];

    for await (const fairCoach of coaches) {
      const coach = await this.coachRepository.findOne(fairCoach.email);

      const haveCommonProgrammingLanguages = coach.programmingLanguages
        .some(r=> student.programmingLanguages.includes(r));

      console.log(coach.programmingLanguages, student.programmingLanguages,
haveCommonProgrammingLanguages);

      results.push({ ...fairCoach, recommended: haveCommonProgrammingLanguages });
    }

    return results;
  }
}
```

Файл GetAllMembersQuery.ts

```
import { GetCommand, ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
```

```
export class GetAllMembersQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(student: string) {
    const queryCommunities = new ScanCommand({
      TableName: "Communities",
      FilterExpression: "#sk = :sk",
      ExpressionAttributeNames: { "#sk": "sk" },
      ExpressionAttributeValues: {
        ":sk": `Member#${student}`,
      },
    });

    const studentCommunities = (await this.dynamoDb.client().send(queryCommunities)).Items.map(c =>
c.pk.split("#")[1]);

    const coaches = [];

    for (const community of studentCommunities) {
      const query = new ScanCommand({
        TableName: "Fairs",
        FilterExpression: "#pk = :pk and contains(#sk, :sk)",
        ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
        ExpressionAttributeValues: {
          ":pk": "Fair#vue",
          ":sk": "Coach#",
        },
      });

      const results = (await this.dynamoDb.client().send(query)).Items.map(coach => ({ coach, community }));
      coaches.push(...results);
    }

    const groupedCoaches = [];

    for (const { coach, community } of coaches) {
      if (groupedCoaches.some(c => c.email === coach.email)) {
        groupedCoaches.find(c => c.email === coach.email).communities.push(community)
      } else {
        coach.communities = [community];
```

```
      groupedCoaches.push(coach);
     }
    }

    return groupedCoaches;
  }
}
```

Файл GetByCommunityFairQuery.ts

```
import { GetCommand, ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetByCommunityFairQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(authenticatedUserId: string, community: string) {
    const query = new GetCommand({
      TableName: "Fairs",
      Key: {
        "pk": `Fair#${community}`,
        "sk": "Identity",
      },
    });

    const fair = (await this.dynamoDb.client().send(query)).Item;

    const queryJ = new ScanCommand({
      TableName: "Fairs",
      FilterExpression: "#pk = :pk and #sk = :sk",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: {
        ":pk": `Fair#${community}`,
        ":sk": `Coach#${authenticatedUserId}`,
      },
    });

    const isJoined = (await this.dynamoDb.client().send(queryJ)).Items.length > 0;
    fair.isJoined = isJoined;
```

```
    return fair;
  }
}
```

Файл FairsController.ts

```
import { Controller, Get, Param, Post } from "@nestjs/common";
import { AuthenticatedUser } from "../auth/decorators/authenticated-user.decorator";
import { GetByCommunityFairQuery } from "./queries/get-by-community.query";
import { JoinFairCommand } from "./commands/join-fair.command";
import { LeaveFairCommand } from "./commands/leave-fair.command";
import { GetAllMembersQuery } from "./queries/get-all-members.query";
import { GetAllMembersRankedQuery } from "./queries/get-all-members-ranked.query";

@Controller("fairs")
export class FairsController {
  constructor(
    private readonly getByCommunityFairQuery: GetByCommunityFairQuery,
    private readonly getAllMembersQuery: GetAllMembersQuery,
    private readonly getAllMembersRankedQuery: GetAllMembersRankedQuery,
    private readonly joinFairCommand: JoinFairCommand,
    private readonly leaveFairCommand: LeaveFairCommand,
  ) {}

  @Get(":community")
  getByCommunity(@AuthenticatedUser() user: string, @Param("community") community: string) {
    return this.getByCommunityFairQuery.execute(user, community);
  }

  @Get("all/members")
  getAllMembers(@AuthenticatedUser() student: string) {
    return this.getAllMembersQuery.execute(student);
  }

  @Get("all/members/ranked")
  getAllMembersRanked(@AuthenticatedUser() student: string) {
    return this.getAllMembersRankedQuery.execute(student);
  }

  @Post(":community/join")
  join(@Param("community") community: string, @AuthenticatedUser() authenticatedUser: string) {
    return this.joinFairCommand.execute(authenticatedUser, community);
  }
```

```
  @Post(":community/leave")
  leave(@Param("community") community: string, @AuthenticatedUser() authenticatedUser: string) {
    return this.leaveFairCommand.execute(authenticatedUser, community);
  }
}
```

Файл GetAllPostsQuery.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetAllPostsQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute() {
    const query = new ScanCommand({
      TableName: "Posts",
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "contains(#pk, :pk) and contains(#sk, :sk)",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: { ":pk": "Post#", ":sk": "Identity#" },
    });

    const allPosts = (await this.dynamoDb.client().send(query)).Items;

    allPosts.sort((a, b) => Date.parse(b.createdAt) - Date.parse(a.createdAt));

    return allPosts;
  }
}
```

Файл GetByCommunityIdPostsQuery.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetByCommunityIdPostsQuery {
  constructor (
```

```
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(communityId: string) {
    const query = new ScanCommand({
      TableName: "Posts",
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "contains(#pk, :pk) and contains(#sk, :sk)",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: { ":pk": "Post#", ":sk": "Identity#" },
    });

    const allPosts = (await this.dynamoDb.client().send(query)).Items;

    allPosts.sort((a, b) => Date.parse(b.createdAt) - Date.parse(a.createdAt));

    return allPosts.filter(p => p.community === communityId);
  }
}
```

Файл GetByIdPostQuery.ts

```
import { Select } from "@aws-sdk/client-dynamodb";
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetByIdPostQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(postId: string) {
    const query = new ScanCommand({
      TableName: "Posts",
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "#pk = :pk and contains(#sk, :sk)",
      ExpressionAttributeNames: { "#pk": "pk", "#sk": "sk" },
      ExpressionAttributeValues: { ":pk": `Post#${postId}`, ":sk": "Identity#" },
    });

    return (await this.dynamoDb.client().send(query)).Items[0];
  }
```

```
}
```

Файл PostsController.ts

```
import { Body, Controller, Get, Param, Post, Query } from "@nestjs/common";
import { GetAllPostsQuery } from "./queries/get-all.query";
import { GetCommentsByPostQuery } from "./queries/get-comments-by-post.query";
import { GetByCommunityIdPostsQuery } from "./queries/get-by-community-id.query";
import { GetByIdPostQuery } from "./queries/get-by-id.query";
import { ReplyToPostCommand } from "./commands/reply-to-post.command";
import { AuthenticatedUser } from "../auth/decorators/authenticated-user.decorator";
import { CreateNewPostCommand } from "./commands/create-new.command";

@Controller("posts")
export class PostsController {
  constructor(
    private readonly getAllPostsQuery: GetAllPostsQuery,
    private readonly getByCommunityIdPostsQuery: GetByCommunityIdPostsQuery,
    private readonly getCommentsByPostQuery: GetCommentsByPostQuery,
    private readonly getByIdPostQuery: GetByIdPostQuery,
    private readonly replyToPostCommand: ReplyToPostCommand,
    private readonly createNewPostCommand: CreateNewPostCommand
  ) {}

  @Get("feed")
  getAll() {
    return this.getAllPostsQuery.execute();
  }

  @Get()
  getByCommunityId(@Query("communityId") communityId: string) {
    return this.getByCommunityIdPostsQuery.execute(communityId);
  }

  @Get(":id")
  getById(@Param("id") id: string) {
    return this.getByIdPostQuery.execute(id);
  }

  @Get(":id/comments")
  getByPost(@Param("id") id: string) {
    return this.getCommentsByPostQuery.execute(id);
  }
```

```
  @Post("")
  create(
    @AuthenticatedUser() authenticatedUser: string,
    @Body("title") title: string,
    @Body("content") content: string,
    @Body("community") community: string) {
    return this.createNewPostCommand.execute(authenticatedUser, title, content, community);
  }

  @Post(":id/reply")
  reply(
    @AuthenticatedUser() authenticatedUser: string,
    @Param("id") id: string,
    @Body("content") content: string) {
    return this.replyToPostCommand.execute(authenticatedUser, id, content);
  }
}
```

Файл EditStudentCommand.ts

```
import { Injectable } from "@nestjs/common";
import { StudentRepository } from "src/infrastructure/persistance/repositories/student.repository";
import { Location } from "src/domain/user/Location";

@Injectable()
export class EditStudentCommand {
  constructor (
    private readonly studentRepository: StudentRepository
  ) {}

  async execute(
    authenticatedStudentId: string,
    student: {
      languages: string[],
      programmingLanguages: string[],
      location: { city: string, country: string },
      name: string
    }
  ) {
    const s = await this.studentRepository.findOne(authenticatedStudentId);

    const { languages, programmingLanguages, name, location } = student;
```

```
    s.edit({ languages, programmingLanguages, name, location: Location.createNew(location.country, location.city)
});

    await this.studentRepository.save(s);
  }
}
```

Файл GetAllStudentsQuery.ts

```
import { ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Select } from "@aws-sdk/client-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetAllStudentsQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute() {
    const query = new ScanCommand({
      TableName: "Users",
      Select: Select.ALL_ATTRIBUTES,
      FilterExpression: "contains(#pk, :pk)",
      ExpressionAttributeNames: { "#pk": "pk" },
      ExpressionAttributeValues: { ":pk": "Student#" },
      Limit: 10,
    });

    return (await this.dynamoDb.client().send(query)).Items;
  }
}
```

Файл GetStudentByIdQuery.ts

```
import { GetCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetStudentByIdQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}
```

```
  async execute(id: string) {
    const query = new GetCommand({
      TableName: "Users",
      Key: {
        "pk": `Student#${id}`,
        "sk": "Identity",
      },
    });

    return (await this.dynamoDb.client().send(query)).Item;
  }
}
```

Файл StudentsController.ts

```
import { Body, Controller, Get, Param, Post } from "@nestjs/common";
import { GetAllStudentsQuery } from "./queries/get-all.query";
import { GetStudentByIdQuery } from "./queries/get-by-id.query";
import { EditStudentCommand } from "./commands/edit-student.command";
import { AuthenticatedUser } from "../auth/decorators/authenticated-user.decorator";

@Controller("students")
export class StudentsController {
  constructor(
    private readonly getAllStudents: GetAllStudentsQuery,
    private readonly getStudentByIdQuery: GetStudentByIdQuery,
    private readonly editStudentCommand: EditStudentCommand
  ) {}

  @Get(":id")
  getById(@Param("id") id: string) {
    return this.getStudentByIdQuery.execute(id);
  }

  @Get()
  getAll() {
    return this.getAllStudents.execute();
  }

  @Post("self")
  edit(
    @AuthenticatedUser() authenticatedStudent: string,
    @Body("name") name: string,
    @Body("languages") languages: string[],
```

```
    @Body("programmingLanguages") programmingLanguages: string[],
    @Body("location") location: any
  ) {
    return this.editStudentCommand.execute(
      authenticatedStudent,
      {
        name,
        languages,
        programmingLanguages,
        location,
      }
    );
  }
}
```

Файл GetUserByIdQuery.ts

```
import { GetCommand, ScanCommand } from "@aws-sdk/lib-dynamodb";
import { Injectable } from "@nestjs/common";
import { DynamoDbService } from "src/infrastructure/aws/dynamodb.service";

@Injectable()
export class GetUserByIdQuery {
  constructor (
    private readonly dynamoDb: DynamoDbService
  ) {}

  async execute(userEmail: string) {
    const queryStudent = new GetCommand({
      TableName: "Users",
      Key: {
        pk: `Student#${userEmail}`,
        sk: "Identity",
      },
    });

    const student = (await this.dynamoDb.client().send(queryStudent)).Item;

    if (student) return student;

    const queryCoach = new GetCommand({
      TableName: "Users",
      Key: {
        pk: `Coach#${userEmail}`,
```

```
        sk: "Identity",
      },
    });

    const coach = (await this.dynamoDb.client().send(queryCoach)).Item;

    return coach;
  }
}
```

Файл UsersController.ts

```
import { Controller, Get, Param } from "@nestjs/common";
import { GetUserByIdQuery } from "./queries/get-by-id.query";
import { AuthenticatedUser } from "../auth/decorators/authenticated-user.decorator";

@Controller("users")
export class UsersController {
  constructor(
    private readonly getUserByIdQuery: GetUserByIdQuery
  ) {}

  @Get("current")
  getCurrent(@AuthenticatedUser() authenticatedUser: string) {
    return this.getUserByIdQuery.execute(authenticatedUser);
  }

  @Get(":id")
  getById(@Param("id") id: string) {
    return this.getUserByIdQuery.execute(id);
  }
}
```

Файл Api.ts

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import axios, { AxiosError, AxiosInstance, AxiosResponse } from "axios";
import { Auth } from "common/auth/auth-context";
import { Toast } from "common/toast/toast-context";

const BASE_URL = process.env.REACT_APP_API_URL || "/api";

export class Api {
  private readonly instance: AxiosInstance;

  constructor() {
```

```
  this.instance = axios.create({
    baseURL: BASE_URL,
    headers: {
      "Content-Type": "application/json",
    },
  });

  this.instance.interceptors.request.use(config => {
    config.headers.Authorization = btoa(`Bearer ${Auth.getToken()}`);
    return config;
  });

  this.instance.interceptors.response.use(response => {
    console.log("DASsa", response.status, response.data);
    if (response.status !== 200) {
      console.log(response.data);
    }
    return response;
  });
}

async get<Response>(url: string, params?: object): Promise<Response> {
  return await this.instance
    .get(url, { params })
    .then(this.parseResponse<Response>)
    .catch(this.parseError<Response>);
}

async post<Response>(url: string, payload: object): Promise<Response> {
  return await this.instance
    .post(url, payload)
    .then(this.parseResponse<Response>)
    .catch(this.parseError<Response>);
}

async put<Response>(url: string, payload: object): Promise<Response> {
  return await this.instance
    .put(url, payload)
    .then(this.parseResponse<Response>)
    .catch(this.parseError<Response>);
}
```

```
  async delete<Response>(url: string): Promise<Response> {
    return await this.instance
      .delete(url)
      .then(this.parseResponse<Response>)
      .catch(this.parseError<Response>);
  }

  private parseResponse<Response>(response: AxiosResponse): Response {
    return response.data;
  }

  private parseError<Response>(error: AxiosError): Response {
    const data = error.response?.data as any;
    const message = data?.message;
    Toast.registerError(message);
    return error.response?.data as Response;
  }
}

export default new Api();
```

Файл ChatsService.ts

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import api from "./api";
import { Chat } from "domain/chat/Chat";
import { ChatMember } from "domain/chat/ChatMember";
import ws from "./ws";
import coachesService from "./coaches.service";
import studentsService from "./students.service";

class ChatsService {
  async getById(chatId: string): Promise<Chat> {
    const chat = await api.get<any>(`/chats/${chatId}`);

    const coach = await coachesService.getById(chat.coach);
    const student = await studentsService.getById(chat.student);

    return new Chat(
      chatId,
      new ChatMember(student!.email, student!.avatar()),
      new ChatMember(coach!.email, coach!.avatar()));
  }
```

```
  async emitMessage(chatId: string, message: string, author: string): Promise<void> {
    ws.emit(
      "chats/messages",
      { chatId, message, author });
  }
}

export default new ChatsService();
```

Файл CoachesService.ts

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import api from "./api";
import { Location } from "domain/user/location";
import { Coach } from "domain/user/coach/coach";
import { WorkExperience } from "domain/user/work-experience";
import { CoachStudent } from "domain/user/coach/coach-student";

const endpoint = "/coaches";

class CoachesService {
  async getById(email: string): Promise<Coach | null> {
    const coach = await api.get<any>(`${endpoint}/${email}`);
    if (!coach?.role) return null;
    const {
      role, name, location, languages, programmingLanguages, mentorshipRequests, students, workExperience,
    } = coach;
    return new Coach(
      email,
      role,
      name,
      new Location(location.city, location.country),
      languages,
      programmingLanguages,
      workExperience.map((w: any) =>
        new WorkExperience(w.company, new Date(Date.parse(w.start)), w.end && new Date(Date.parse(w.end)))),
      mentorshipRequests,
      students.map((s: any) =>
        new CoachStudent(s.coach, s.student, s.chat)),
    );
  }

  async approveMentorship(coach: string, student: string): Promise<CoachStudent> {
    const { chat } = await api.post<any>(`${endpoint}/approveMentorshipRequest`, { student });
```

```
    return new CoachStudent(coach, student, chat);
  }

  async requestMentorship(coach: string) {
    await api.post(`${endpoint}/${coach}/requestMentorship`, {});
  }

  async edit(coach: Coach): Promise<void> {
    await api.post<any>(`${endpoint}/self`, coach);
  }
}

export default new CoachesService();
```

Файл CommunitiesService.ts

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import { Community } from "domain/community";
import api from "./api";

const endpoint = "/communities";

class CommunitiesService {
  async getAll(): Promise<Community[]> {
    const communities = await api.get<any[]>(endpoint);
    return communities.map(({ name, description, isJoined, membersCount }) =>
      new Community(name, description, isJoined, membersCount));
  }

  async getById(id: string): Promise<Community> {
    const { name, description, isJoined, membersCount } = await api.get<any>(`${endpoint}/${id}`);
    return new Community(name, description, isJoined, membersCount);
  }

  async joinCommunity(communityId: string): Promise<void> {
    await api.post(`${endpoint}/${communityId}/join`, { });
  }

  async leaveCommunity(communityId: string): Promise<void> {
    await api.post(`${endpoint}/${communityId}/leave`, { });
  }
}

export default new CommunitiesService();
```

Файл FairsService.ts

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import { FairCoach } from "domain/fair/FairCoach";
import api from "./api";
import { Fair } from "domain/fair/Fair";

const endpoint = "/fairs";

class FairsService {
  async getByCommunity(community: string): Promise<Fair> {
    const { membersCount, isJoined } = await api.get<any>(`${endpoint}/${community}`);
    return new Fair(
      community,
      membersCount,
      isJoined,
    );
  }

  async getMembers(): Promise<FairCoach[]> {
    const coaches = await api.get<any[]>(`${endpoint}/all/members`);
    return coaches.map(({ email, communities }) => new FairCoach(email, communities));
  }

  async getMembersRanked(): Promise<FairCoach[]> {
    const coaches = await api.get<any[]>(`${endpoint}/all/members/ranked`);
    return coaches.map(({ email, communities, recommended }) => new FairCoach(email, communities,
recommended));
  }

  async join(communityId: string): Promise<void> {
    await api.post(`${endpoint}/${communityId}/join`, { });
  }

  async leave(communityId: string): Promise<void> {
    await api.post(`${endpoint}/${communityId}/leave`, { });
  }
}

export default new FairsService();
```

Файл PostsService.ts

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import { Comment } from "domain/comment";
```

```
import api from "./api";
import { Post } from "domain/post";

const endpoint = "/posts";

class PostsService {
  async getFeed(): Promise<Post[]> {
    const posts = await api.get<any[]>(`${endpoint}/feed`);
    return posts.map((
      { id, title, content, community, createdBy, createdAt }) =>
      new Post(id, title, content, community, createdBy, new Date(Date.parse(createdAt))));
  }

  async getByCommunity(communityId: string): Promise<Post[]> {
    const posts = await api.get<any[]>(endpoint, { communityId });
    return posts.map((
      { id, title, content, community, createdBy, createdAt }) =>
      new Post(id, title, content, community, createdBy, new Date(Date.parse(createdAt))));
  }

  async getById(postId: string): Promise<Post> {
    const { id, title, content, community, createdBy, createdAt } = await api.get<any>(`${endpoint}/${postId}`);
    return new Post(id, title, content, community, createdBy, new Date(Date.parse(createdAt)));
  }

  async getComments(postId: string): Promise<Comment[]> {
    const comments = await api.get<any[]>(`${endpoint}/${postId}/comments`);
    return comments.map((
      { id, content, createdBy, createdAt, replyTo }) =>
      new Comment(id, content, createdBy, new Date(Date.parse(createdAt)), replyTo));
  }

  async replyToPost(postId: string, content: string): Promise<void> {
    await api.post<any[]>(`${endpoint}/${encodeURIComponent(postId)}/reply`, { content });
  }

  async create(title: string, content: string, community: string): Promise<Post> {
    const newPost = await api.post<any>(endpoint, { title, content, community });
    return await this.getById(newPost.id);
  }
}
```

```
export default new PostsService();
```

Файл StudentsService.ts

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import { Student } from "domain/user/student";
import api from "./api";
import { Location } from "domain/user/location";

const endpoint = "/students";

class StudentsService {
  async getById(email: string): Promise<Student | null> {
    const student = await api.get<any>(`${endpoint}/${email}`);
    if (!student?.role) return null;
    const {
      role, name, location, languages, programmingLanguages, coaches, mentorshipRequests,
    } = student;
    return new Student(
      email,
      role,
      name,
      new Location(location.city, location.country),
      languages,
      programmingLanguages,
      coaches,
      mentorshipRequests,
    );
  }

  async edit(student: Student): Promise<void> {
    await api.post<any>(`${endpoint}/self`, student);
  }
}

export default new StudentsService();
```

Файл ws.ts

```
import { io } from "socket.io-client";

const ws = io("ws://localhost:8001", { transports: ["websocket"] });

export default ws;
```

Файл Auth.ts

```
import coachesService from "api/coaches.service";
```

```
import studentsService from "api/students.service";
import { Coach } from "domain/user/coach/coach";
import { Student } from "domain/user/student";
import { makeAutoObservable, runInAction } from "mobx";
import { createContext } from "react";

export class Auth {
  authenticatedCoach?: Coach;
  authenticatedStudent?: Student;
  isAuthenticated: boolean = false;

  constructor() {
    makeAutoObservable(this);
  }

  async initialize() {
    if (Auth.getToken()) {
      const email = Auth.getToken()!.split(":")[0];

      const coach = await coachesService.getById(email);
      runInAction(() => {
        if (coach) {
          this.authenticatedCoach = coach;
          this.isAuthenticated = true;
        }
      });

      const student = await studentsService.getById(email);
      runInAction(() => {
        if (student) {
          this.authenticatedStudent = student;
          this.isAuthenticated = true;
        }
      });
    }
  }

  async login(email: string, password: string) {
    localStorage.setItem("accessToken", `${email}:${password}`);
    await this.initialize();
  }
```

```
  async logout() {
    localStorage.removeItem("accessToken");
    window.location.reload();
  }

  static getToken() {
    return localStorage.getItem("accessToken");
  }

  isCoach() {
    return this.authenticatedCoach;
  }

  authenticatedUser(): Student | Coach {
    return (this.authenticatedStudent ?? this.authenticatedCoach)!;
  }
}

export const auth = new Auth();

export const AuthContext = createContext(auth);
```

Файл AppRoute.ts

```
export enum AppRoute {
  BASE = "/",

  COMMUNITIES = "/communities",
  COMMUNITY = "/communities/:id",

  POSTS = "/posts",
  POST = "/posts/:id",

  USERS = "/users",
  USER_MY = "/users/my",
  USER = "/users/:id",
  LOGIN = "/login",

  STUDENTS = "/students",

  CHAT = "/chats/:id",

  COACHES = "/coaches",
  COACHES_MY = "/coaches/my",
```

```
}
```

Файл Page.tsx

```
import React, { PropsWithChildren } from "react";

import styles from "./styles.module.scss";
import { PageHeader } from "../page-header";

interface IProps {
  withHeader?: boolean;
  classes?: {
    root?: string;
  },
}

export const Page: React.FC<PropsWithChildren<IProps>> = ({
  children,
  classes = {},
  withHeader = true,
}) => {
  return (
    <div className={`${styles.page} ${classes.root || ""}`}>
      {withHeader && <PageHeader />}
      <div className={styles.content}>
        {children}
      </div>
    </div>
  );
};
```

Файл PageHeader.tsx

```
import React, { PropsWithChildren, useContext } from "react";

import styles from "./styles.module.scss";
import { Auth, AuthContext } from "common/auth/auth-context";
import { Avatar, ButtonGroup, Button } from "@mui/material";
import { Link, generatePath } from "react-router-dom";
import { UserName } from "pages/profile/user-name";
import { AppRoute } from "common/enums/app-route.enum";

interface IProps {
  classes?: {
    root?: string;
  },
```

```
}

export const PageHeader: React.FC<PropsWithChildren<IProps>> = ({
  classes = {},
}) => {
  const auth = useContext(AuthContext);
  const user = auth.authenticatedUser();

  const links = [
    { name: "Communities", link: AppRoute.COMMUNITIES },
    { name: "Posts", link: `${AppRoute.POSTS}` },
  ];

  if (auth.isCoach()) {
    links.push({ name: "My students", link: AppRoute.STUDENTS });
  }

  if (!auth.isCoach()) {
    links.push({ name: "My coaches", link: AppRoute.COACHES_MY });
  }

  if (!auth.isCoach()) {
    links.push({ name: "Coaches search", link: AppRoute.COACHES });
  }

  return (
    <div className={`${styles.root} ${classes.root || ""}`}>
      <div className={styles.links}>
        <ButtonGroup variant="text" aria-label="text button group">
          {links.map(({ link, name }) => (
            <Button key={link}>
              <Link to={link}>{name}</Link>
            </Button>
          ))}
        </ButtonGroup>
      </div>
      <div className={styles.user}>
        <UserName email={user.email} className={styles.name} />
        <Avatar src={user.avatar()} sx={{ height: 48, width: 48 }} />
        <Button onClick={() => auth.logout()}>Logout</Button>
      </div>
    </div>
```

```
  );
};
```

Файл PageList.tsx

```
import React, { PropsWithChildren } from "react";
import { List } from "@mui/material";

export const PageList: React.FC<PropsWithChildren> = ({
  children,
}) => {
  return (
    <List sx={{ width: "100%", maxWidth: 1000 }} >
      {children}
    </List>
  );
};
```

Файл MessageInput.tsx

```
import React, { useContext, useState } from "react";

import styles from "./styles.module.scss";

import { observer } from "mobx-react-lite";
import { Chat } from "domain/chat/Chat";
import { TextField, Button } from "@mui/material";
import { AuthContext } from "common/auth/auth-context";

interface IProps {
  chat: Chat;
}

export const MessageInput: React.FC<IProps> = observer(({ chat }) => {
  const auth = useContext(AuthContext);
  const [message, setMessage] = useState<string>("");

  const sendMessage = () => {
    chat.send(message, auth.authenticatedUser().email);
  };

  return (
    <div className={`${styles.root}`}>
      <TextField
        id="message-input1"
        label="Message text"
```

```
        variant="outlined"
        value={message}
        onChange={e => setMessage(e.target.value)}
      />
      <Button variant="outlined" onClick={sendMessage}>Send</Button>
    </div>
  );
});
```

Файл MessageListItem.tsx

```
import React, { useContext } from "react";

import styles from "./styles.module.scss";

import { observer } from "mobx-react-lite";
import { Message } from "domain/chat/Message";
import { PageListItem } from "components/page-list-item";
import { Chat } from "domain/chat/Chat";
import { Avatar } from "@mui/material";
import { AuthContext } from "common/auth/auth-context";

interface IProps {
  message: Message;
  chat: Chat;
}

export const MessageListItem: React.FC<IProps> = observer(({ message, chat }) => {
  const auth = useContext(AuthContext);

  const authorAvatar = chat.coach.email === message.author ? chat.coach.avatar : chat.student.avatar;
  const isMine = auth.authenticatedUser().email === message.author;

  return (
    <div className={`${styles.root} ${isMine ? styles.mine : styles.other}`}>
      {!isMine && <Avatar sx={{ width: 64, height: 64, marginRight: 2 }} src={authorAvatar} />}
      <PageListItem className={styles.content}>
        <div className={styles.content}>
          <p>{message.content}</p>
        </div>
      </PageListItem>
    </div>
  );
});
```

Файл ChatPage.tsx

```
import React, { useEffect, useState } from "react";
import { Page } from "components/page";
import { CircularProgress } from "@mui/material";
import { MessageListItem } from "./message-list-item";
import { observer } from "mobx-react-lite";
import { Message } from "domain/chat/Message";
import { useParams } from "react-router-dom";
import messagesService from "api/messages.service";
import { PageList } from "components/page-list";
import { Chat } from "domain/chat/Chat";
import chatsService from "api/chats.service";
import { MessageInput } from "./message-input";
import ws from "api/ws";

export const ChatPage: React.FC = observer(() => {
  const { id } = useParams();

  const [messages, setMessages] = useState<Message[]>([]);
  const [chat, setChat] = useState<Chat>();
  const [isLoading, setIsLoading] = useState<boolean>(true);

  useEffect(() => {
    const fetch = async () => {
      setMessages(await messagesService.getByChatId(id!));
      setChat(await chatsService.getById(id!));
      setIsLoading(false);
    };

    fetch();
  }, []);

  useEffect(() => {
    ws.on("chats/messages", ({ id, content, chat, author }) => {
      setMessages(messages => [...messages, new Message(id, content, chat, author)]);
    });

    return () => {
      ws.off("chats/messages");
    };
  }, []);
```

```
  if (isLoading) return (<CircularProgress />);

  return (
    <Page>
      <PageList>
        {messages.map(message => (
          <MessageListItem key={message.id} message={message} chat={chat!} />
        ))}
      </PageList>
      <MessageInput chat={chat!} />
    </Page>
  );
});
```

Файл CoachesPage.tsx

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import React, { useEffect, useState } from "react";
import { observer } from "mobx-react-lite";
import { Page } from "components/page";
import { Community } from "domain/community";
import { CoachListItem } from "./coach-list-item";
import communitiesService from "api/communities.service";
import { PageList } from "components/page-list";
import fairsService from "api/fairs.service";
import { FairCoach } from "domain/fair/FairCoach";
import { Button } from "@mui/material";

export const CoachesPage: React.FC = observer(() => {
  const [coaches, setCoaches] = useState<FairCoach[]>([]);

  useEffect(() => {
    const fetch = async () => {
      const coaches = await fairsService.getMembers();
      setCoaches(coaches);
    };

    fetch();
  }, []);

  const findRecommended = async () => {
    const coaches = await fairsService.getMembersRanked();
    setCoaches(coaches);
  };
```

```
  return (
    <Page>
      <div>
        <Button variant="outlined" onClick={findRecommended}>Find recommended</Button>
      </div>
      <PageList>
        {coaches.map(coach => (
          <CoachListItem key={coach.email} coach={coach} />
        ))}
      </PageList>
    </Page>
  );
});
```

Файл Communities.tsx

```
import React, { useEffect, useState } from "react";
import { observer } from "mobx-react-lite";
import { Page } from "components/page";
import { Community } from "domain/community";
import { CommunityListItem } from "./community-list-item";
import communitiesService from "api/communities.service";
import { PageList } from "components/page-list";

export const Communities: React.FC = observer(() => {
  const [communities, setCommunities] = useState<Community[]>([]);

  useEffect(() => {
    const fetch = async () => {
      const communities = await communitiesService.getAll();
      setCommunities(communities);
    };

    fetch();
  }, []);

  return (
    <Page>
      <PageList>
        {communities.map(community => (
          <CommunityListItem key={community.name} community={community} />
        ))}
      </PageList>
```

```
    </Page>
  );
});
```

Файл CommunityPage.tsx

```
import React, { useEffect, useState } from "react";
import { Page } from "components/page";
import { Post } from "domain/post";
import postsService from "api/posts.service";
import { useParams } from "react-router-dom";
import { Community } from "domain/community";
import { CircularProgress } from "@mui/material";
import communitiesService from "api/communities.service";
import { CommunityHeader } from "./community-header";
import { PageList } from "components/page-list";
import { PostListItem } from "pages/posts-feed/post-list-item";
import { CreatePostForm } from "./create-post-form";
import { CommunityFair } from "./community-fair";
import { observer } from "mobx-react-lite";

export const CommunityPage: React.FC = observer(() => {
  const { id } = useParams();

  const [community, setCommunity] = useState<Community>();
  const [posts, setPosts] = useState<Post[]>([]);
  const [isPostFormOpened, setIsPostFormOpened] = useState(false);

  useEffect(() => {
    const fetchPosts = async () => {
      setPosts(await postsService.getByCommunity(id!));
    };

    const fetchCommunity = async () => {
      setCommunity(await communitiesService.getById(id!));
    };

    fetchPosts();
    fetchCommunity();
  }, []);

  const savePost = (post: Post) => {
```

```
    setPosts(posts => [post, ...posts]);
    setIsPostFormOpened(false);
  };

  if (!community) return (<CircularProgress />);

  return (
    <Page>
      <CommunityHeader community={community} openPostForm={() => setIsPostFormOpened(true)} />
      <CommunityFair community={community.name} />
      <PageList>
        {posts.map(post => (
          <PostListItem key={post.id} post={post} />
        ))}
      </PageList>
      <CreatePostForm open={isPostFormOpened} community={community} close={savePost} />
    </Page>
  );
});
```

Файл MyCoachesPage.tsx

```
/* eslint-disable @typescript-eslint/no-explicit-any */
import React, { useContext, useEffect } from "react";
import { observer } from "mobx-react-lite";
import { Page } from "components/page";
import { CoachListItem } from "./coach-list-item";
import { PageList } from "components/page-list";
import { AuthContext } from "common/auth/auth-context";
import { MentorshipRequestListItem } from "./mentorship-request-list-item";

export const MyCoachesPage: React.FC = observer(() => {
  const auth = useContext(AuthContext);

  const student = auth.authenticatedStudent!;

  console.log(student);

  return (
    <Page>
      <PageList>
        {student.coaches.map(coach => (
```

```
          <CoachListItem key={coach.coach} coach={coach} />
        ))}
        {student.mentorshipRequests.map(coach => (
          <MentorshipRequestListItem coach={coach} key={coach} />
        ))}
      </PageList>
    </Page>
  );
});
```

Файл PostPage.tsx

```
import React, { useEffect, useState } from "react";
import { Page } from "components/page";
import { Post } from "domain/post";
import postsService from "api/posts.service";
import { useParams } from "react-router-dom";
import { Comment } from "domain/comment";
import { CircularProgress, Divider } from "@mui/material";
import { CommentsThread } from "./comments-thread";
import { PostDetailed } from "./post-detailed";

export const PostPage: React.FC = () => {
  const { id } = useParams();

  const [post, setPost] = useState<Post>();
  const [comments, setComments] = useState<Comment[]>([]);
  const [isLoading, setIsLoading] = useState<boolean>(true);

  useEffect(() => {
    const fetch = async () => {
      setComments(await postsService.getComments(id!));
      setPost(await postsService.getById(id!));
      setIsLoading(false);
    };

    fetch();
  }, []);

  if (isLoading) return (<CircularProgress />);

  return (
    <Page>
      <PostDetailed post={post!} />
```

```
      <Divider style={{ marginTop: "12px", marginBottom: "12px" }}>Comments</Divider>
      <CommentsThread parent={post!.id} comments={comments} level={0} />
    </Page>
  );
};
```

Файл PostsFeedPage.tsx

```
import React, { useEffect, useState } from "react";
import { Page } from "components/page";
import { Post } from "domain/post";
import postsService from "api/posts.service";
import { PageList } from "components/page-list";
import { PostListItem } from "./post-list-item";

export const PostsFeedPage: React.FC = () => {
  const [posts, setPosts] = useState<Post[]>([]);

  useEffect(() => {
    const fetch = async () => {
      const feed = await postsService.getFeed();
      setPosts(feed);
    };

    fetch();
  }, []);

  return (
    <Page>
      <PageList>
        {posts.map(post => (
          <PostListItem key={post.id} post={post} />
        ))}
      </PageList>
    </Page>
  );
};
```

Файл MyProfilePage.tsx

```
import React, { useEffect, useState, useContext } from "react";
import { Page } from "components/page";
import { useParams } from "react-router-dom";
import { CircularProgress } from "@mui/material";
import usersService from "api/users.service";
import { ProfileHeader } from "./profile-header";
```

```
import { Languages } from "./languages";
import { ProgrammingLanguages } from "./programming-languages";
import styles from "./styles.module.scss";
import { Coach } from "domain/user/coach/coach";
import coachesService from "api/coaches.service";
import { AuthContext } from "common/auth/auth-context";
import { StudentInfo } from "./student-info";
import { CoachInfo } from "./coach-info";
import { EditStudentForm } from "./edit-student-form";
import { EditCoachForm } from "./edit-coach-form";

export const MyProfilePage: React.FC = () => {
  // const { id } = useParams();

  // const [coach, setCoach] = useState<Coach>();
  // const [isLoading, setIsLoading] = useState<boolean>(true);

  // useEffect(() => {
  //   const fetch = async () => {
  //     const user = await usersService.getById(id!);
  //     if (user.role === "Coach") {
  //       setCoach(await coachesService.getById(id!));
  //     }
  //     setIsLoading(false);
  //   };

  //   fetch();
  // }, []);

  // if (isLoading) return (<CircularProgress />);

  const auth = useContext(AuthContext);

  const user = (auth.authenticatedCoach || auth.authenticatedStudent)!;

  const [isEditFormOpen, setIsEditFormOpen] = useState(false);

  return (
    <Page>
      <ProfileHeader user={user} openEditingForm={() => setIsEditFormOpen(true)} />
      {auth.isCoach()
        ? <CoachInfo coach={auth.authenticatedCoach!} />
```

```
        : <StudentInfo student={auth.authenticatedStudent!} />}
      {auth.isCoach()
        ? <EditCoachForm
          coach={auth.authenticatedCoach!}
          open={isEditFormOpen}
          close={() => setIsEditFormOpen(false)}
        />
        : <EditStudentForm
          student={auth.authenticatedStudent!}
          open={isEditFormOpen}
          close={() => setIsEditFormOpen(false)}
        />}
    </Page>
  );
};
```

Файл OtherProfilePage.tsx

```
import React, { useEffect, useState, useContext } from "react";
import { Page } from "components/page";
import { useParams } from "react-router-dom";
import { CircularProgress } from "@mui/material";
import usersService from "api/users.service";
import { ProfileHeader } from "./profile-header";
import { Languages } from "./languages";
import { ProgrammingLanguages } from "./programming-languages";
import styles from "./styles.module.scss";
import { Coach } from "domain/user/coach/coach";
import coachesService from "api/coaches.service";
import { AuthContext } from "common/auth/auth-context";
import { StudentInfo } from "./student-info";
import { CoachInfo } from "./coach-info";
import { EditStudentForm } from "./edit-student-form";
import { EditCoachForm } from "./edit-coach-form";
import { Student } from "domain/user/student";
import studentsService from "api/students.service";

export const OtherProfilePage: React.FC = () => {
  const { id } = useParams();

  const [user, setUser] = useState<Coach | Student>();
  const [isLoading, setIsLoading] = useState<boolean>(true);

  useEffect(() => {
```

```
    const fetch = async () => {
      const coach = await coachesService.getById(id!);
      const student = await studentsService.getById(id!);

      if (coach) setUser(coach);
      else setUser(student!);

      setIsLoading(false);
    };

    fetch();
  }, []);

  if (isLoading) return (<CircularProgress />);

  return (
    <Page>
      <ProfileHeader user={user!} openEditingForm={() => {}} />
      {user?.role.toLocaleLowerCase() === "coach"
        ? <CoachInfo coach={user as Coach} />
        : <StudentInfo student={user as Student} />}
    </Page>
  );
};
```

Файл StudentsPage.tsx

```
import React, { useEffect, useState, useContext } from "react";
import { Page } from "components/page";
import { CircularProgress } from "@mui/material";
import { StudentListItem } from "./student-list-item";
import { Coach } from "domain/user/coach/coach";
import coachesService from "api/coaches.service";
import { observer } from "mobx-react-lite";
import { AuthContext } from "common/auth/auth-context";
import { MentorshipRequestListItem } from "./mentorship-request-list-item";
import { PageList } from "components/page-list";

export const StudentsPage: React.FC = observer(() => {
  const auth = useContext(AuthContext);

  // const [coach, setCoach] = useState<Coach>();
  // const [isLoading, setIsLoading] = useState<boolean>(true);
```

```
  // useEffect(() => {
  //   const fetch = async () => {
  //     setCoach((await coachesService.getById(auth.authenticatedCoach!.email))!);
  //     setIsLoading(false);
  //   };

  //   fetch();
  // }, []);

  // if (isLoading) return (<CircularProgress />);

  return (
    <Page>
      <PageList>
        {auth.authenticatedCoach?.students.map(student => (
          <StudentListItem key={student.student} student={student} />
        ))}
        {auth.authenticatedCoach?.mentorshipRequests.map(applicant => (
          <MentorshipRequestListItem coach={auth.authenticatedCoach!} key={applicant} applicant={applicant} />
        ))}
      </PageList>
    </Page>
  );
});
```

Файл App.tsx

```
import { BrowserRouter, Route, Routes } from "react-router-dom";

import { AppRoute } from "common/enums/app-route.enum";
import { Communities } from "pages/communities";

import { PostsFeedPage } from "pages/posts-feed";
import { CommunityPage } from "pages/community";
import { PostPage } from "pages/post";
import { AuthContext, auth } from "common/auth/auth-context";
import { StudentsPage } from "pages/students";
import { ChatPage } from "pages/chat";
import ws from "api/ws";
import { useEffect, useState } from "react";
import { LoginPage } from "pages/auth";
import { PrivateRoute } from "components/private-route";
import { Alert, CircularProgress } from "@mui/material";
import { CoachesPage } from "pages/coaches";
```

```
import { MyProfilePage } from "pages/profile/my";
import { OtherProfilePage } from "pages/profile/other";
import { MyCoachesPage } from "pages/my-coaches";
import { Toast } from "components/toast";
import api, { Api } from "api/api";
import { observer } from "mobx-react-lite";

const App = observer(() => {
  const [isConnected, setIsConnected] = useState(ws.connected);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    ws.on("connect", () => setIsConnected(true));
    ws.on("disconnect", () => setIsConnected(false));
    auth.initialize().then(() => setIsLoading(false));

    return () => {
      ws.off("connect");
      ws.off("disconnect");
    };
  }, []);

  if (!isConnected || isLoading) return (<CircularProgress />);

  return (
    <AuthContext.Provider value={auth}>
      <BrowserRouter>
        <Routes>
          <Route path={AppRoute.COMMUNITIES} >
            <Route path={AppRoute.COMMUNITY} element={<PrivateRoute><CommunityPage
/></PrivateRoute>}/>
            <Route path={""} element={<PrivateRoute><Communities /></PrivateRoute>}/>
          </Route>

          <Route path={AppRoute.POSTS} >
            <Route path={AppRoute.POST} element={<PrivateRoute><PostPage /></PrivateRoute>}/>
            <Route path={""} element={<PrivateRoute><PostsFeedPage /></PrivateRoute>}/>
          </Route>

          <Route path={AppRoute.USERS} >
            <Route path={AppRoute.USER_MY} element={<PrivateRoute><MyProfilePage /></PrivateRoute>}/>
            <Route path={AppRoute.USER} element={<PrivateRoute><OtherProfilePage /></PrivateRoute>}/>
```

```
          </Route>

          <Route path={AppRoute.LOGIN} element={<LoginPage />}/>

          <Route path={AppRoute.STUDENTS} >
            <Route path={""} element={<PrivateRoute><StudentsPage /></PrivateRoute>} />
          </Route>

          <Route path={AppRoute.CHAT} >
            <Route path={""} element={<PrivateRoute><ChatPage /></PrivateRoute>} />
          </Route>

          <Route path={AppRoute.COACHES} >
            <Route path={AppRoute.COACHES_MY} element={<PrivateRoute><MyCoachesPage
/></PrivateRoute>} />
            <Route path={""} element={<PrivateRoute><CoachesPage /></PrivateRoute>} />
          </Route>

          <Route path={AppRoute.BASE} element={<PrivateRoute><PostsFeedPage /></PrivateRoute>} />
        </Routes>
        <Toast />
      </BrowserRouter>
    </AuthContext.Provider>
  );
});

export default App;
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ХМАРНА ПЛАТФОРМА ДЛЯ ПОШУКУ МЕНТОРІВ З ПРОГРАМУВАННЯ

Програма та методика тестування

КПІ.ІП-9602.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Ярослав БОРОДАЄНКО

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб-додаток для пошуку менторів з програмування.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка безпеки даних;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox, …);

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;

– функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

– тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;

– тестування «білої скриньки» – об’єктом тестування тут є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується у три етапи.

Першим етапом є тестування окремих методів та класів.

Другим етапом є тестування серверної частини додатку як єдиного цілого.

Останнім етапом є ручне тестування клієнтської частини на відповідність функціональним вимогам та простоту використання.

Всього було проведено наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на різних веб-браузерах;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ХМАРНА ПЛАТФОРМА ДЛЯ ПОШУКУ МЕНТОРІВ З ПРОГРАМУВАННЯ

Керівництво користувача

КПІ.ІП-9602.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Ярослав БОРОДАЄНКО

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«CodeCoach» – це веб-застосунок для спілкування в тематичних громадах. Він дозволяє користувачам обмінюватися досвідом у форматі відповідей на поширені запитання. Також він автоматизує процес пошуку індивідуального наставника то спілкування з ним.

Додаток має простий користувацький інтерфейс, який розроблено на основі відомих соціальних мереж, що дозволяє об’єднуватися користувачам у тематичні громади, задавати та відповідати на запитання. Но основі громад та інформації про кожного користувача побудовано рекомендаційну систему, що використовує сучасні методи глибинного навчання. Після успішного пошуку наставника користувач може спілкуватися з ним в режимі реального часу за допомогою приватного текстового чату.

Окремою перевагою розробленого додатку є його інтеграція з популярним хмарним провайдером – Amazon Web Services. Це дозволяє спростити процес розгортання, масштабування та моніторингу. Також це дозволяє значно спростити реалізацію деяких критичних елементів системи, як, наприклад, запуск моделей глибинного навчання.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

- наявність веб-браузеру Chrome версії 80 та вище;
- наявність доступу до Інтернету;

2.2 Завантаження застосунку

Застосунок не потребує завантаження. Для того щоб його переглянути необхідно перейти за відповідним посиланням.

2.3 Перевірка коректної роботи

Після завантаження сторінки користувач має побачити форму логіну.

3 ВИКОНАННЯ ПРОГРАМИ

При запуску застосунку користувач потрапляє на сторінку авторизації (рис. 3.1), де користувач має ввести свою пошту та пароль.

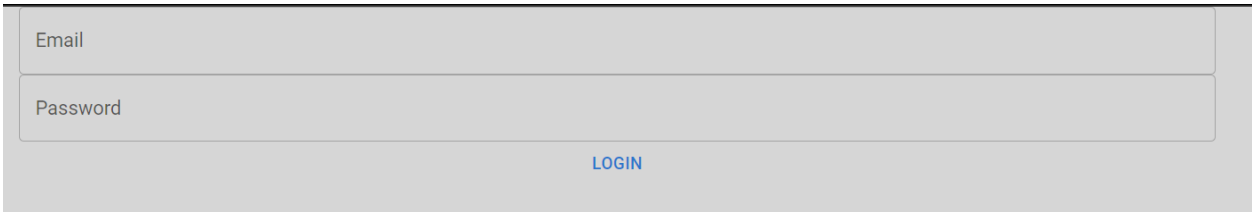

Рисунок 3.1 – Сторінка авторизації

Після натискання кнопки «Логін» користувач потрапляє на сторінку громад (рис. 3.2). На цій сторінці користувач бачить список усіх громад, що містяться у системі. Кожен елемент списку складається з назви, головного зображення та коткого опису. Також користувачу доступні 2 кнопки – «Приєднатися», якщо користувач ще не є членом громади та «Покинути», в іншому випадку. Окрім цього користувач може перейти на окрему сторінку громади. Для цього достатньо натиснути на будь-яку частину елементу відповідної громади.

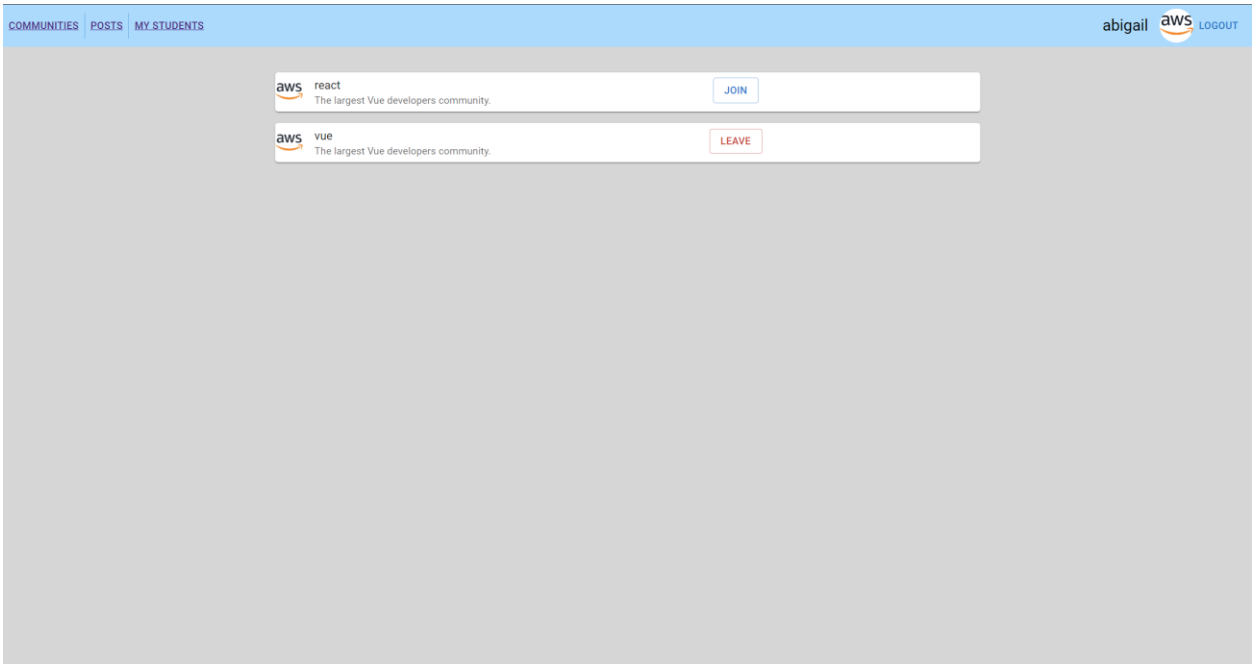

Рисунок 3.2 – Сторінка списку громад

Окрема сторінка громади подана на рис. 3.3 і має наступні елементи: назву, логотип, короткий опис, кількість учасників, кількість наставників, що

шукають студентів та список усіх запитань, відсортованих за спаданням часу публікації. Також користувач може покинути громади натиснувши кнопку «Покинути» або створити нове запитання за допомогою кнопки «Задати запитання».

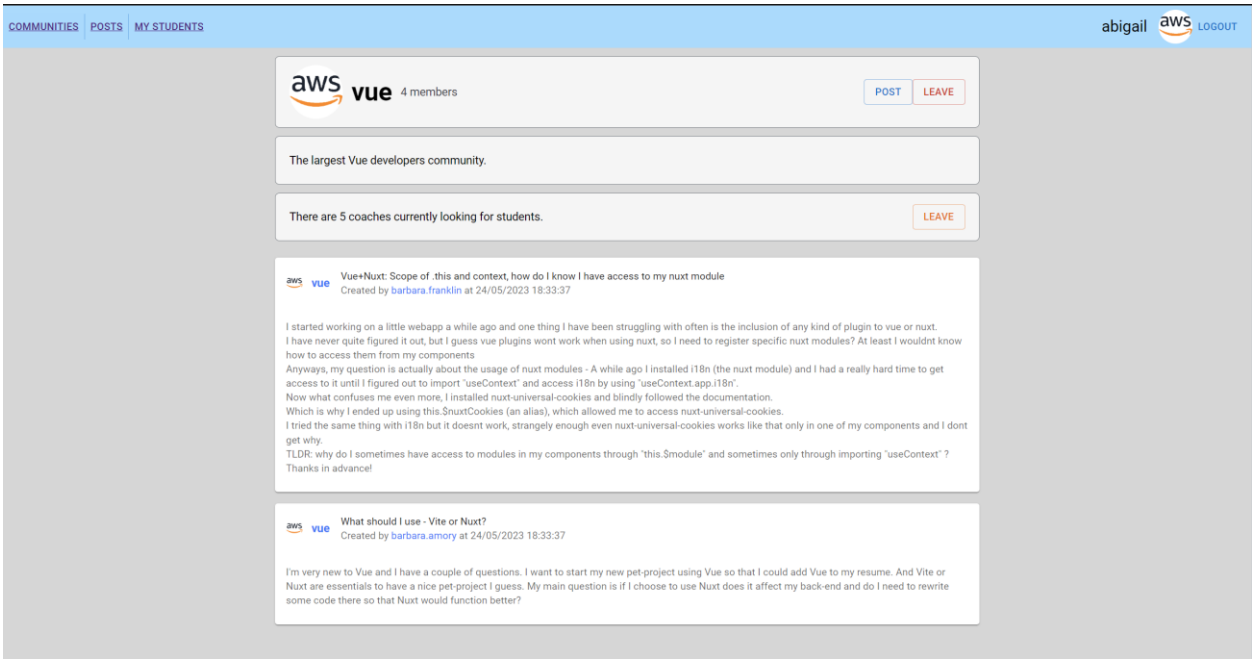

Рисунок 3.3 – Сторінка громади

Форма створення запитання знаходиться на рис. 3.4 і має всього два поля: коротка назва та повний текст запитання.

Рисунок 3.4 – Форма створення запитання

Для того щоб перейти на сторінку окремого запитання користувачу достатньо натиснути на будь-яку частину елементу відповідного запитання.

Сторінка запитання (рис. 3.5) складається з тексту запитання та списку відповідей до нього.

Відповіді представлені у деревоподібному вигляді, де кожне запитання має інформацію про автора та текст відповіді. Користувач може надати коментарі щодо відповіді іншого користувача натиснувши кнопку «Відповісти» відповідної відповіді.

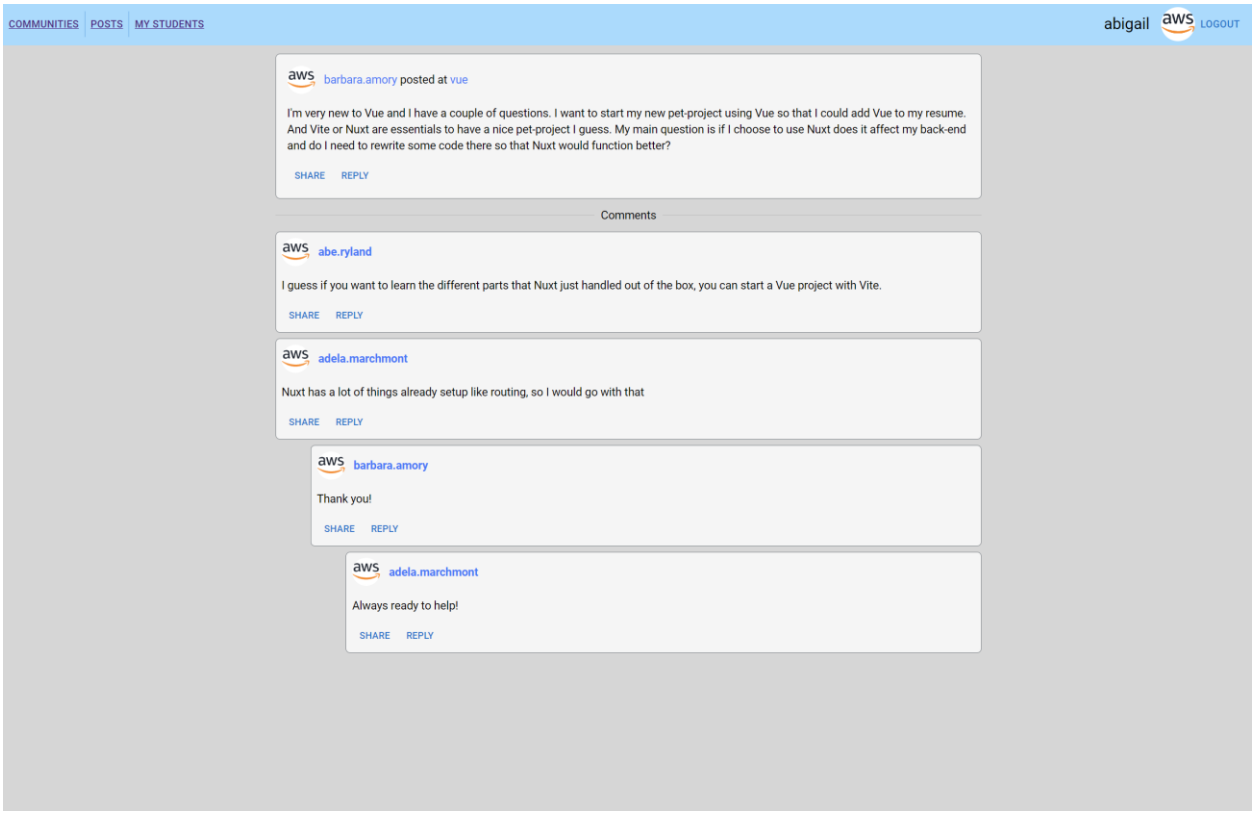

Рисунок 3.5 – Сторінка запитання

Щоб переглянути сторінку користувача (рис. 3.6) достатньо натиснути на ім’я користувача, що виділено синім. На ній можна побачити ім’я, фото, місце знаходження, роль, мови програмування, якими володіє та попередній досвід роботи. Якщо цей користувач є наставником, то йому можна запропонувати стати ментором. Для цього існує кнопка «Запропонувати стати наставником». Після натискання кнопки відповідний студент з’являється на сторінці заяв (рис. 3.7). На даній сторінці наставник може переглянути список своїх поточних студентів та заяв потенційних. Також він

може або підтвердити заяву, натиснувши на кнопку «Підтвердити» або перейти на приватного чату із студентом завдяки кнопці «Написати у приватні повідомлення», праворуч від імені.

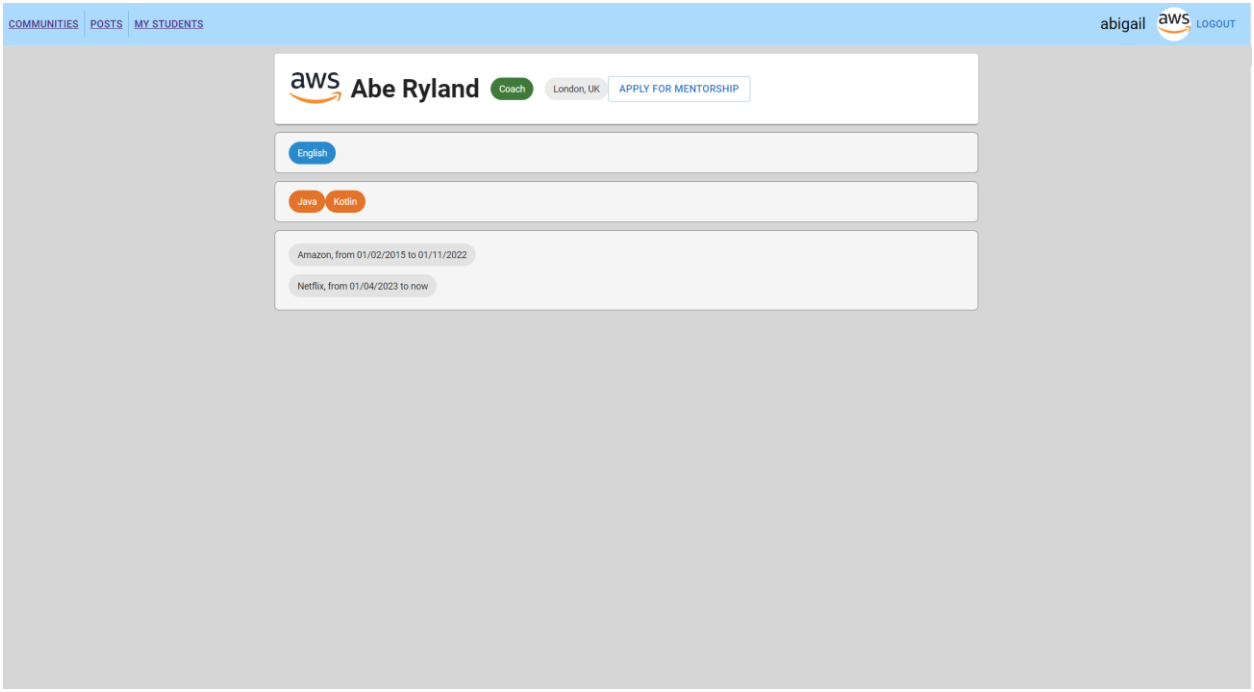

Рисунок 3.6 – Сторінка користувача

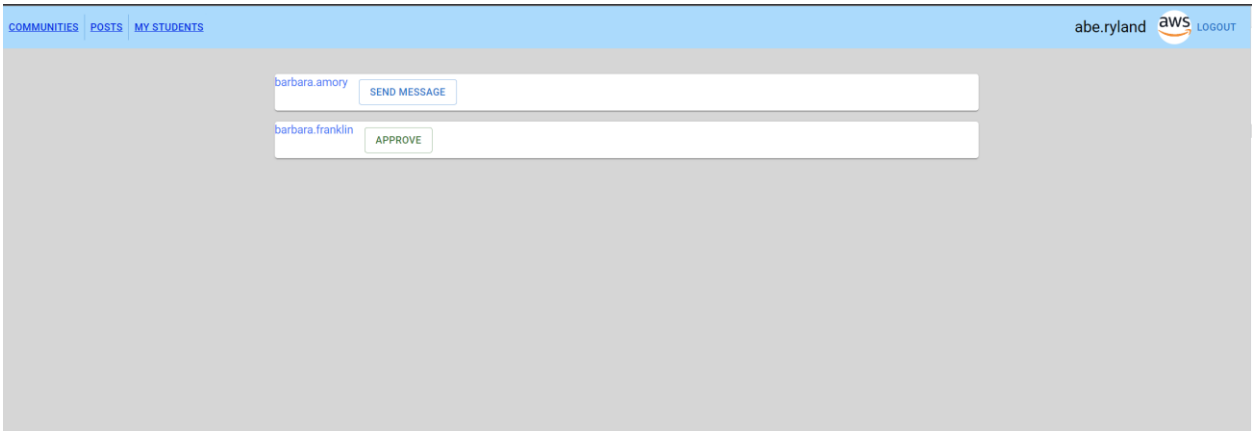

Рисунок 3.7 – Сторінка заяв

Приватний чат поданий на рис. 3.8 і складається з повідомлень обох сторін. Також користувач може відправити нове повідомлення написавши його у відповідне текстове поле і натиснувши клавішу «Відправити».

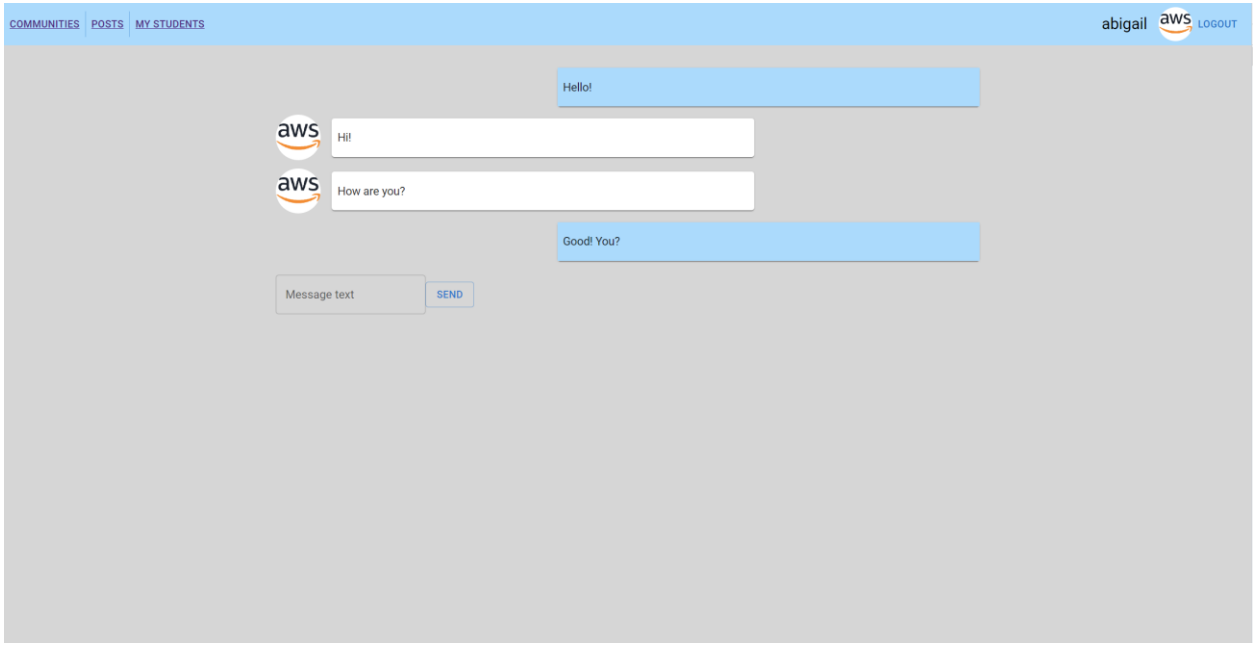

Рисунок 3.8 – Приватний чат

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ОНЛАЙН ПЛАТФОРМА ДЛЯ СПІЛКУВАННЯ В ТЕМАТИЧНИХ ГРОМАДАХ ТА АВТОМАТИЧНОГО ПОШУКУ НАСТАВНИКА НА ОСНОВІ ХМАРИ

Графічний матеріал

КПІ.ІП-9602.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Ярослав БОРОДАЄНКО

Київ – 2023