Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Вебплатформа для колективного розв'язання проблем та задач

ПОЯСНЮВАЛЬНА ЗАПИСКА

08-23.МКР.009.00.000 ПЗ

Виконав студент 2 курсу, групи 1КІ-21м
спеціальності123 — Комп'ютерна інженерія
_____Майданюк І. Г.

Керівник к.т.н., доцент, професор каф. ОТ
_____Захарченко С. М.
"____" _____2022 р.

Опонент д.т.н., професор каф. ПЗ
_____ Ліщинська Л. Б.
"____" _____2022 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
_____
"19" 12 2022 р.

ВНТУ 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
"15" 09 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту **Майданюку Івану Григоровичу**

1 Тема роботи «Вебплатформа для колективного розв'язання проблем та задач» керівник роботи Захарченко Сергій Михайлович к.т.н., доцент, професор, затверджено наказом вищого навчального закладу від 15.09.2022 року № 205-А

2 Строк подання студентом роботи 19.12.2022.

3 Вихідні дані до роботи: розроблена серверна та клієнтська частини вебплатформи для колективного розв'язання проблем та задач

4 Зміст розрахунково-пояснювальної записки: вступ, аналітичний огляд проблематики колективного розв'язання проблем та задач, технологій реалізації системи, налаштування монорепозиторію, реалізація серверної та клієнтської частини вебплатформи, економічна частина.

5 Перелік графічного матеріалу технічне завдання, схематичне зображення роботи платформи питання-відповідь, схематичне зображення роботи чаті, схема роботи клієнт-серверної взаємодії, лістинг основних фрагментів програми.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1— Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	к.т.н., доц., професор Захарченко Сергій Михайлович	24.09.2022	19.12.2022
5	к.е.н., професор Небава Микола Іванович	24.11.2022	18.12.2022

7 Дата видачі завдання 24.09.2022 .

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів МКР	Строк виконання	Підпис
1	Постановка задачі	24.09.2022	
2	Аналіз переваг та недоліків існуючих рішень	30.90.2022	
3	Складання вимог до рішення що враховує недоліки аналогів	7.10.2022	
4	Аналіз та вибір технологій для реаліації запропонованої платформи	14.10.2022	
5	Планування схеми роботи платформи	21.10.2022	
6	Налаштування монорепозиторію	28.10.2022	
7	Реалізація серверної частини	11.11.2022	
	Реалізація клієнтської частини	25.11.2022	
8	Розрахунок економічної частини	18.12.2022	
9	Оформлення пояснювальної записки та ілюстративного матеріалу	18.12.2022	
10	Виконання магістерської кваліфікаційної роботи	9.12.2022	
11	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	5.12.2022	
12	Підписи супроводжувальних документів у керівника, опонента, нормоконтролера	16.12.2022	
13	Перевірка «антиплагіат»	16.12.2022	
14	Попередній захист	19.12.2022	

Студент _____ Майданюк І. Г.

Керівник _____ к.т.н., доцент, професор Захарченко С. М.

АНОТАЦІЯ

УДК 004.42

Майданюк І. Г. Вебплатформа для колективного розв’язання проблем та задач. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна Інженерія, Вінниця: ВНТУ, 2022, 105 с.

Українською. Бібліографія: 51 назв; 17 рисунків; 11 таблиць.

У магістерській роботі проаналізовано існуючих інструментів для колективного розв’язання проблем та задач. Виділено принципи роботи, особливості, переваги та недоліки існуючих платформ. Запропоновано створення вебплатформи що враховує недоліки аналогів — вебплатформи питання-відповідь з вбудованими чатами. Проаналізовано та обрано технології для розробки запропонованого рішення. Розраховано оптимальну кількість розв’язаних задач на одиницю часу. Налаштовано середовище для розробки у вигляді монорепозиторію, згенерованого за допомогою Nx. Реалізовано клієнтську та серверну частини застосунку за допомогою Angular, NestJS, TypeORM, PostgreSQL, Socket.IO.

Ключові слова: платформа питання-відповідь, чат, інструменти для колективного розв’язання задач, колективне розв’язання задач.

ABSTRACT

Maidaniuk I. G. Web-platform for collective solving of problems and tasks. Master's thesis on the specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2022, 105 p.

In the Ukrainian language. Bibliography: 51 titles; 17 figures; 11 tables.

The master work analyzed existing tools for collective problem solving. The working principles, features, advantages and disadvantages of existing platforms are highlighted. It is proposed to create a web platform that takes into account the shortcomings of analogues — a question-answer web platform with built-in chats. Technologies for the development of the proposed solution were analyzed and selected. The optimal number of solved tasks per unit of time is calculated. The development environment is set up as a monorepository generated with Nx. Implemented the client and server part of the use using Angular, NestJS, TypeORM, PostgreSQL, Socket.IO.

Keywords: question-answer platform, chat, tools for collective problem solving, collective problem solving.

ЗМІСТ

ВСТУП

Вирішення проблем — це процес досягнення мети шляхом подолання перешкод, частини більшості видів діяльності. Проблеми, які потребують вирішення, варіюються від простих особистих завдань до складних питань у бізнесі та техніці.

Постійно створюються нові задачі та постійно люди прагнуть їх вирішити. Часто одні й ті ж задачі вирішуються людьми паралельно та незалежно. Більше того, існують певні задачі які вже вирішені однією групою людей, але інші особи про це рішення не мають відома та починають свої розв’язання, хоча до них цю задачу могли вже вирішувати раніше. В результаті певні задачі вирішуються повторно незалежними групами людей. Ефективність такого незалежного вирішення одних і тих же задач різними групами осіб досить низька.

Щоб підвищити ефективність процесу розв’язання задач можливо використовувати спеціальні інструменти. Наприклад, спеціальні вебплатформи для колективного розв’язання задач. Основна ідея таких платформ передбачає надання користувачам можливості створювати питання до яких інші користувачі можуть залишати свої відповіді. Користувачі мають доступ до питань та відповідей інших користувачів. Тож, якщо подібні питання виникнуть у нових користувачів, вони зможуть знайти попередні обговорення, рішення інших користувачів на вебплатформі. Умовно це база знань та рішень, яку наповнюють користувачі.

Тема створення такої вебплатформи також є **актуальною**, оскільки, у часи карантинних обмежень та в умовах прямої військової загрози, необхідно мати можливість проводити групові види діяльності віддалено. Зокрема, колективне вирішення задач також може відбуватись через мережу, без прямої необхідності групам людей збиратись разом.

Метою дослідження магістерської роботи є вдосконалення системи колективного розв’язання проблем та задач.

Задачі дослідження магістерської роботи:

— здійснити аналітичний огляд існуючих платформ для колективного розв’язання проблем та задач;

— запропонувати покращену версію платформи для колективного розв’язання проблем та задач, яка враховує вади аналогів;

— проаналізувати та обрати необхідні технології для реалізації запропонованого рішення;

— розрахувати оптимальну кількість розв’язувань питань на одиницю часу;

— реалізація клієнтської та серверної частини застосунку.

Об’єкт дослідження магістерської роботи процес колективного розв’язання задач.

Предмет дослідження магістерської роботи — платформи питання-відповідь та чати для колективного розв’язання проблем та задач.

Методи дослідження магістерської роботи:

— теорія масового обслуговування, для визначення оптимальної кількості задач, що можуть бути розв’язані на одиницю часу та для визначення середнього часу розв’язання.

— методи проектування програмного забезпечення для визначення структури програмного продукту;

— методи проектування баз даних для оптимізації структури даних.

Наукова новизна отриманих результатів магістерської роботи полягає у розширенні функціональних можливостей системи колективного розв’язання задач і проблем за рахунок інтеграції в систему спеціалізованого чату, що дозволило зменшити середній час розв’язання задач.

Практичне значення одержаних результатів магістерської роботи: розроблену вебплатформу можна використовувати для колективного розв’язання будь-яких задач у текстовому вигляді.

Апробація. У матеріалах LI регіональної науково-технічної конференції ВНТУ опубліковано статтю на тему «Розробка веб-застосунків за допомогою JavaScript».

1. АНАЛІТИЧНИЙ ОГЛЯД ПРОБЛЕМАТИКИ КОЛЕКТИВНОГО РОЗВ'ЯЗАННЯ ПРОБЛЕМ ТА ЗАДАЧ

1.1 Огляд наявних платформ

Існують декілька типів платформ для вирішення задач. Умовно їх можна поділити на 2 типи. Перший тип це платформи питання-відповідь. Другий тип це групи, групові чати у соціальних мережах.

1.1.1 Платформи питання-відповідь

Платформи формату питання-відповідь передбачають можливість користувачами створювати питання, на які інші користувачі можуть залишати відповіді (див. рисунок 1.1).

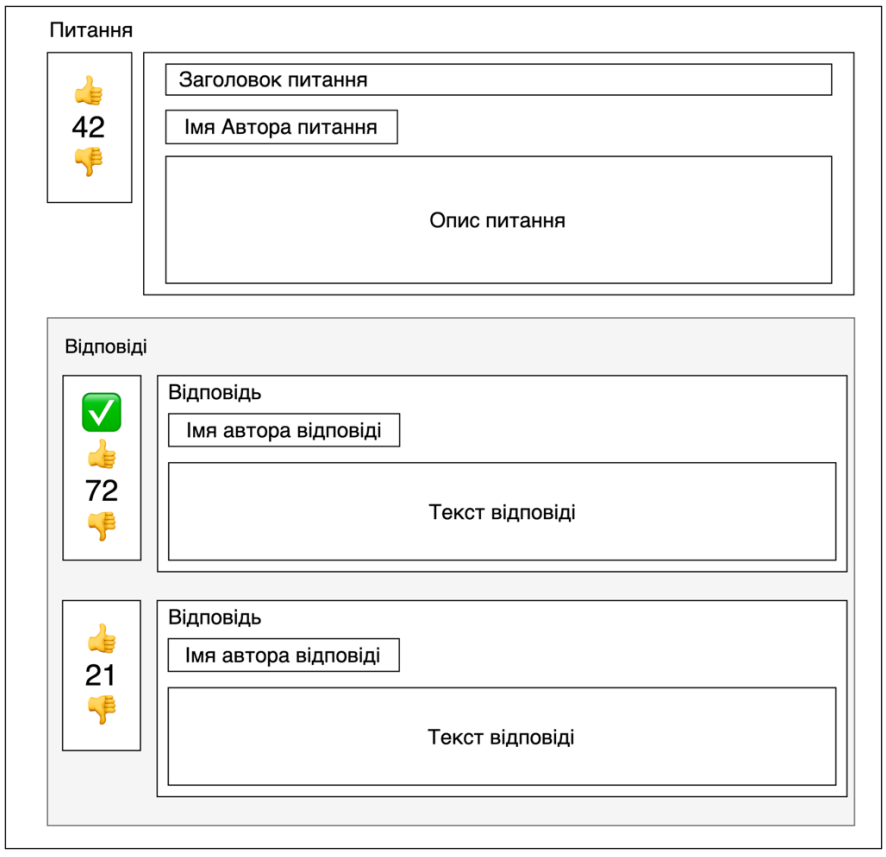

Рисунок 1.1 — Схематичне зображення типової платформи питання відповідь

Як можна побачити на рисунку 1.1, у кожного питання може бути декілька відповідей. Автор питання може відмітити слушну відповідь

відповідним прапорцем (✓). Також користувачі мають змогу впливати на рейтинг питання, надаючи питанню плюс або мінус 1 бал до рейтингу.

Найпопулярніша платформа такого типу це Stack Exchange. Понад 100 мільйонів людей щомісяця відвідують 173 спільноти питання-відповідь, якими володіє Stack Exchange, див. таблицю 1.1. [2]

Таблиця 1.1 — Спільноти компанії Stack Exchange з найбільшими рейтингами відвідувань.

Назва спільноти	Stack Overflow	Mathematics	Ask Ubuntu	Unix, Linux
Цільова аудиторія	Програмісти-професіонали та ентузіасти	Математики, професіонали у суміжних галузях	Користувачі Ubuntu та розробники	користувачі Linux, Un*x-подібних ОС
Питань всього	23 000 000	1 500 000	399 000	255 000
Відповідей всього	34 000 000	2 000 000	504 000	331 000
Вирішених (%)	69	77	64	75
Користувачів	19 000 000	972 000	1 400 000	499 000
Відвідувань щодня	8 100 000	165 000	138 000	127 000
Питань щодня	5 600	361	85	56

За поточний 2022 рік мережа компанії Stack Exchange загалом має:

— 418 800 000 відвідувачів щомісячно;

— 806 300 000 переглядів сторінок щомісячно;

— 3 100 000 заданих питань;

— 3 500 000 відповідей;

— 23 000 000 схвальних реакцій на питання чи відповіді;

— 3 000 000 не схвальних реакцій на питання чи відповіді. [2]

Особливостями платформи Stack Exchange є:

— пошук за змістом питань;

— редактор вводу тексту (WYSIWYG);

— рейтингова система питань та відповідей;

— правильні відповіді схвалені авторами питань та позначенні відповідним схвальним символом.

Загалом, типові вебплатформи питання-відповідь результативно допомагають користувачам розв’язувати задачі колективно та ділитись рішеннями. Але у платформ типу питання-відповідь не має функціоналу забезпечення користувачам місця для активної дискусії. Так, у платформ, від компанії Stack Exchange, є коментарі, які можна залишати тільки під питаннями та відповідями. Проте немає єдиного місця для проведення дискусії. Що було б слушним інструментом в умовах пошуку рішень.

1.1.2 Чат-платформи

В противагу платформам по типу питання-відповідь, яким бракує інструменту для загальних дискусій, існують спеціалізовані чати. Наприклад у чат-системах по типу Telegram, Discord, Slack, існують тематичні групи для обговорень та вирішення певних проблем та задач. Це чати спільнот, які спілкуються на теми пов’язанні з тематикою спільноти. Це можуть бути чати виключно про програмування, мистецтво, медицину тощо. Користувачі таких чатів задають питання іншим користувачам чату та намагаються колективно знайти рішення.

Коротко розглянемо кожну платформу.

Discord — це соціальна платформа VoIP і обміну миттєвими повідомленнями. Користувачі мають можливість спілкуватися за допомогою голосових дзвінків, відеодзвінків, текстових повідомлень, мультимедійних файлів і файлів у приватних чатах або в рамках спільнот, які називаються «серверами». Сервер — це набір постійних чатів і голосових каналів, які можуть отримати доступ за посиланнями для запрошень. Discord працює на Windows, macOS, Android, iOS, iPadOS, Linux і у веб-браузерах. Станом на 2021 рік сервіс має понад 350 мільйонів зареєстрованих користувачів і понад 150 мільйонів активних користувачів щомісяця. [3]

Slack — це програма обміну миттєвими повідомленнями, розроблена Slack Technologies і належить Salesforce. Хоча Slack був розроблений для професійних та організаційних комунікацій, він був прийнятий як платформа для спільнот.

Користувачі можуть спілкуватися за допомогою голосових дзвінків, відеодзвінків, текстових повідомлень, медіа та файлів у приватних чатах або в рамках спільнот, які називаються «робочими просторами». Slack також використовує функції в стилі IRC, такі як постійні кімнати чату (канали), організовані за темами, приватні групи та прямий обмін повідомленнями. На додаток до цих функцій онлайн-спілкування, Slack інтегрується з іншим програмним забезпеченням. Станом на 2021 рік сервіс має понад 18 мільйонів активних користувачів щомісяця. [4]

Telegram Messenger — це глобально доступна безкоштовна, кросплатформна, зашифрована, хмарна та централізована служба обміну миттєвими повідомленнями (IM). Послуга також надає додаткові наскрізні зашифровані чати та відеодзвінки, VoIP, обмін файлами та кілька інших функцій. Доступні різні клієнтські програми для комп’ютерних і мобільних платформ, включаючи офіційні програми для Android, iOS, Windows, macOS і Linux. У січні 2021 року Telegram перевищив 500 мільйонів активних користувачів на місяць. [5]

Структурно чати складаються з повідомлень користувачів та поля вводу нових повідомлень, див рисунок 1.2. Повідомлення складаються з тексту повідомлення, імені автора та часу публікації повідомлення. Коли один із користувачів відправляє повідомлення, то всі учасники чату отримують нове повідомлення.

Зазвичай, автори відправлених повідомлень бачать свої повідомлення візуально виділеними. Наприклад, Telegram чати виділяють повідомлення, для автора цих повідомлень, зображуючи їх по правий бік та роблять фон таких повідомлень акцентного кольору. Повідомлення інших користувачів зображують по лівий бік, використовуючи не акцентні, звичайні кольори фону цих повідомлень.

Також для покращення візуального сприйняття чатів, додатково зображують картинку автора повідомлення, так званий «аватар», біля повідомлення цього користувача.

Використання візуальних засобів, таких як: зображення повідомлень по різні боки екрану, використання зображень аватарів біля повідомлення, покращують сприйняття авторства та контенту написаних повідомлень.

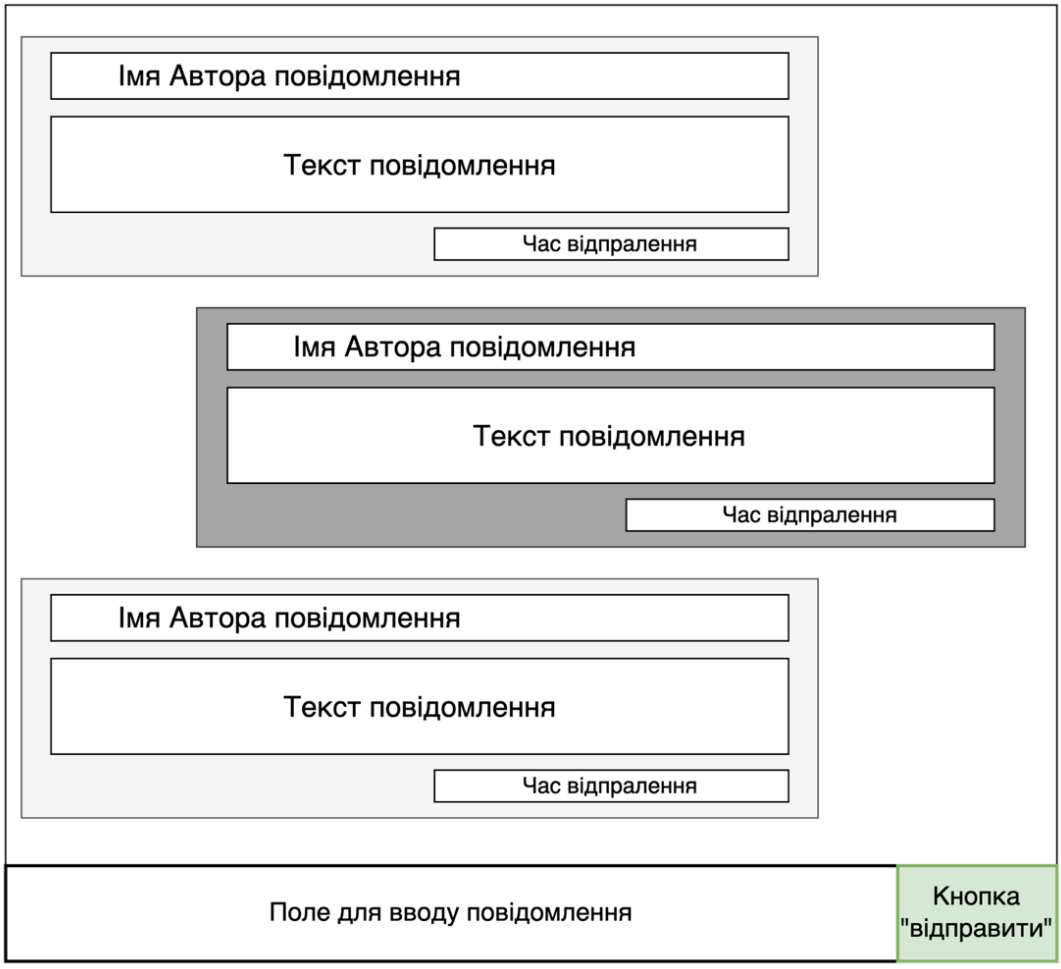

Рисунок 1.2 — Схематичне зображення типової платформи питання відповідь

З точки зору функціоналу чати по типу Discord, Slack та Telegram, окрім основного функціоналу чатів, мають різні особливості. Велика кількість таких особливостей є на декількох платформах одночасно.

Наприклад, функція реагування на повідомлення різними картинками, так званими емодзі — доступна на усіх 3 платформах. Функція робити окремі підчати стосовно окремих повідомлень, так звані треди — доступна на платформах Slack та Discord. У Telegram, така функція є тільки у групах, але не у чатах. Функція відповідей на конкретні повідомлення, так звані реплаї — доступна для 3 платформ. Також, чати Slack та Discord мають функцію

форматування коду, тобто є функція зображення комп’ютерного коду у форматованому вигляді зі збереженням відступів у тексті, та забарвленням синтаксичних конструкцій.

Останнє, але не найменш важливе, практично всі чат-платформи мають функціонал для обміну не тільки текстом, а й для обміну файлами. Певні платформи вводять обмеження на розмір файлів для обміну, а для збільшення ліміту розміру завантажуваних файлів, розробники платформи пропонують купити таку можливість у додачу до інших преміальних функцій.

Вище описаний функціонал, реакцій на повідомлень за допомогою емодзі, треди та реплаї — якісно покращує якість спілкування та зручність проведення дискусій. Особливість форматування коду стане в нагоді для проведень дискусій програмістами. Надання користувачам інструментів для обміну файлів дозволяє обмінюватись у тому числі і фото, і відео.

Всі ці інструменти допомагають якісно зобразити наявну задачу, проблему. Чим доступніше та наглядніше задача буде представлена загалу, тим простіше іншим користувачам платформи буде зрозуміти у чому саме проблематика задачі.

Також у ході дискусії, пошуку рішення якісна комунікація, коли усі учасники діалогу розуміють суть яку хотів донести автор того, чи іншого повідомлення, дійти до рішення буде простіше. Якісна комунікація суттєво покращує результати розв’язання задач. [2]

Проте коли користувачі мають спільне рішення, воно ніде не фіксується. Тому новому користувачеві буде проблематично знайти відповідь у потоці нових повідомлень у чаті де могла бути відповідь на питання.

1.2 Потенційне рішення

Аби задовільнити потреби користувачів стосовно платформ для колективного розв’язання задач, запропоновано об’єднати основний функціонал платформ питання-відповідь та спеціалізованих чатів. Рішення це вебплатформа питання-відповідь з вбудованими чатами.

1.2.1 Функціональні вимоги до рішення

Функціональні вимоги — це властивості або функції продукту, які розробники повинні реалізувати, щоб користувачі могли виконувати свої завдання. Вебплатформа питання-відповідь з вбудованими чатами має наступні функціональні вимоги:

— реєстрування нових користувачів за допомогою email, імені та паролю;

— вхід в систему користувачем за допомогою паролю та логіну, створеного під час реєстрації. Такий користувач вважається автентифікованим;

— автентифікований користувач має змогу створювати питання;

— питання має заголовок та опис;

— автентифікований користувач має змогу залишати відповіді до своїх запитань та запитань інших користувачів;

— відповіді мають текстове поле з відповіддю;

— автентифікований користувач, який створив питання, має можливість схвалити одну відповідь під своїм запитанням, якщо вважає цю відповідь коректною;

— схвалена відповідь має відповідне маркування, у вигляді прапорця (✓), яке можуть бачити усі користувачі;

— автентифікований користувач має змогу оцінювати питання та відповіді інших користувачів та свої також;

— оцінка користувача має плив на рейтинг питання чи відповіді. Позитивна оцінка — додає до рейтингу один бал, негативна — віднімає один бал;

— автентифікований користувач має змогу залишати повідомлення у чаті, що прикріплений до питання;

— у кожного питання є свій незалежний чат;

— всі користувачі мають змогу використовувати пошук по заголовку та опису питань;

— всі користувачі мають змогу переглядати питання, відповіді, їх рейтинг та наявність схвалення автором, історію чатів при питаннях;

— система повинна мати сторінку реєстрації, входу в систему, головну сторінку з переліком питань, сторінку для кожного питання на якій будуть відповіді до цього питання та чат, сторінку з інформацією про проект.

1.2.2 Нефункціональні вимоги до рішення

Нефункціональна вимога — вимога, яка визначає критерії, за якими можна судити про роботу системи, а не конкретну поведінку. Вебплатформа відповідає наступним нефункціональним вимогам:

— система повинна мати інтуїтивно зрозумілий та зручний користувацький інтерфейс;

— система повинна працювати швидко;

— система повинна захищати данні користувачів, такі як email та паролі.

1.3 Переваги запропонованого рішення над існуючими

Запропоноване рішення, у пункті 1.2, у порівнянні з існуючими типами платформ питання-відповідь та спеціалізованими чатами, підпункти 1.1.1 та 1.1.2 відповідно, має певні переваги, див таблицю 1.2. Запропоноване рішення містить основні переваги альтернативних типів підходів. Запропонована система містить у собі пошук по змісту питань. У чатах часто є пошук, але цей пошук працює по всіх повідомленнях чату. Знайти питання у чаті, яке було поставлено тривалий час тому досить проблематично. Також, запропонована система містить рейтингову систему питань і відповідей, та можливість автору питання відмічати слушні відповіді відповідним прапорцем (✓).

Спеціалізовані чати надають користувачам можливість реагувати на повідомлення інших користувачів спеціальними знаками, емодзі. Але така система рейтингу не є репрезентативною та не є тотожною системі рейтингу платформ питання-відповідь. Оскільки, окрім схвальної та не схвальної реакції на повідомлення, інші користувачі можуть додавати ще додатково велику кількість сторонніх реакцій, мета яких не повідомити про слушність повідомлення. Тобто, система рейтингу запропонованого рішення, платформ-

питання відповідь не є тотожними з системою реакцій на повідомлення у спеціалізованих чатах.

Таблиця 1.2 — Порівняння запропонованого рішення з існуючими платформами для колективного розв’язання задач

	Платформи для колективного розв’язанню задач		
	Питання-відповідь	Спеціалізовані чати	Запропоноване рішення
Пошук по змісту питання	+	+/-	+
Рейтингова система	+	+/-	+
Відмітка схваленої автором відповіді	+	-	+
Інструмент для проведення загальної дискусії	-	+	+

У запропоноване рішення містить інструмент для проведення загальних дискусій. Саме надання такої можливості переслідують спеціалізовані чати. На відмінно від них, певні платформи питання-відповідь мають можливість залишати коментарі під конкретними питання або відповідями, проте не дають можливості проводити загальну дискусію у єдиному місці. Усі коментарі платформ питання-відповідь є розмежованими.

В результаті, запропоноване рішення одночасно містить в собі основні функціональні риси платформ питання-відповідь та спеціалізованих чатів, а саме: функції пошук по змісту питання, рейтингову систему, можливість автору питання відмічати слушне питання відповідним прапорцем (✓) та інструмент для проведення загальної дискусії. Таким чином, запропоноване рішення має функціональні переваги над існуючими аналогами та стане в нагоді для ефективного колективного розв’язання проблем та задач.

2. ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ СИСТЕМИ

Побудову запропонованої вебплатформи у пункті 1.2 можна умовно декомпозувати, тобто розділити на менші підзадачі, на 2 частини: front-end та back-end. Умовно клієнтська (font-end) та серверна (back-end) сторони — це терміни веб-розробки, які описують, де запускається код програми, див рисунок 2.1.

У розробці програмного забезпечення терміни front-end і back-end стосуються поділу проблем між рівнем представлення (front-end) і рівнем доступу до даних (back-end) частини програмного забезпечення, або фізична інфраструктура чи обладнання. У моделі клієнт-сервер клієнт зазвичай вважається front-end, а сервер зазвичай вважається back-end, навіть якщо деяка робота з репрезентацією даних фактично виконується на самому сервері. [7]

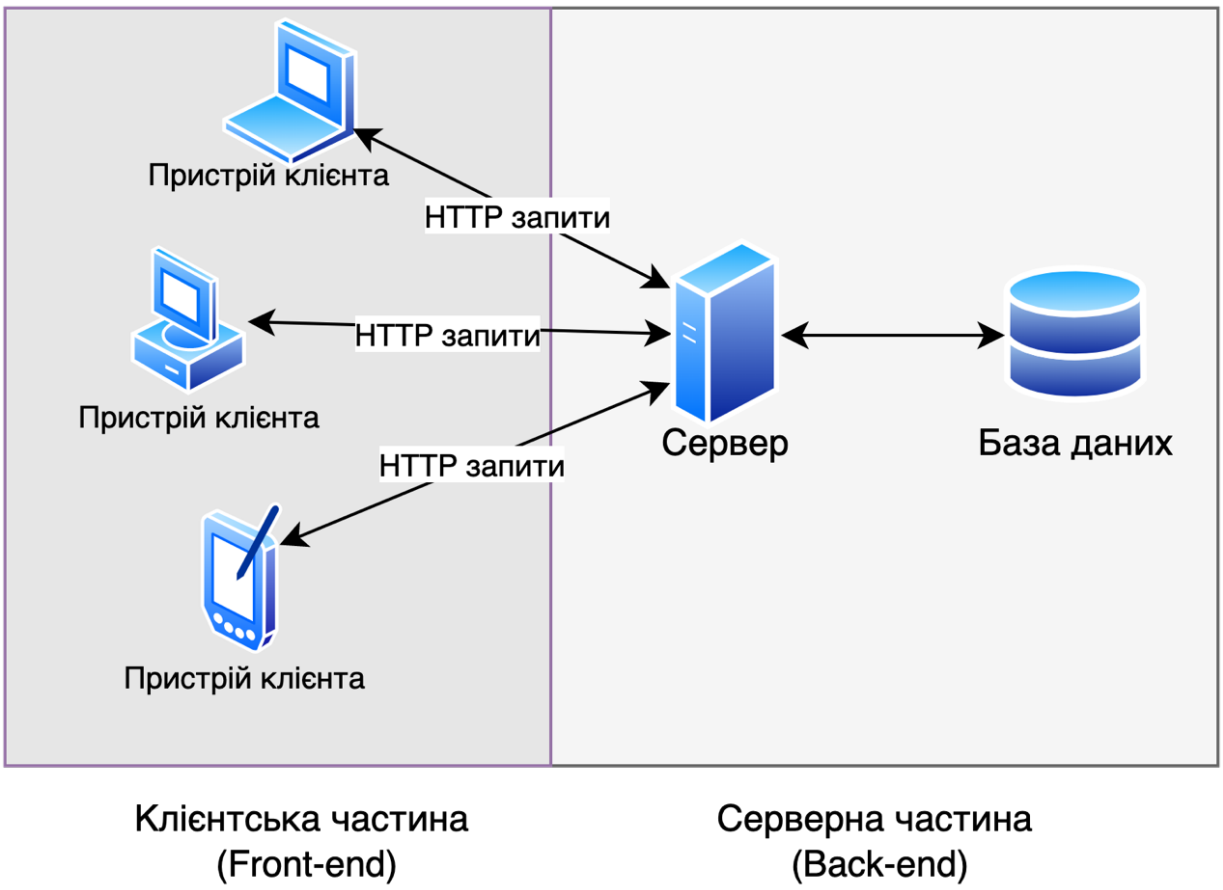

Рисунок 2.1 — Схематичне зображення взаємодії клієнтської та серверної частини застосунку

2.1 Клієнтська частина (front-end)

2.1.1 Огляд основних концепцій front-end напрямку

Front-end веброзробка — це розробка графічного інтерфейсу користувача вебсайту за допомогою HTML, CSS і JavaScript, щоб користувачі могли переглядати вебсайт і взаємодіяти з ним. [8] Для перегляду вмісту вебсайту користувачі за допомогою спеціальної програми браузеру, таких як Google Chrome, Opera, Safari, необхідно ввести URL-адресу, того чи іншого вебсайту, у спеціальне поле бразеру.

URL, уніфікований покажчик ресурсу, який у просторіччі називають веб-адресою — це посилання на веб-ресурс, яке вказує його розташування в комп'ютерній мережі та механізм його отримання. URL-адреси найчастіше зустрічаються для посилань на веб-сторінки (HTTP), але також використовуються для передачі файлів (FTP), електронної пошти (mailto), доступу до бази даних (JDBC) і багатьох інших програм. [9]

Після того як певний URL-адреса відправлена браузеру на обробку відбуваються наступні кроки:

— браузер перевіряє запис DNS, щоб знайти відповідну IP-адресу веб-сайту;

— щойно браузер отримає правильну IP-адресу, він створить з'єднання із сервером, який відповідає IP-адресі, для передачі інформації;

— браузер надсилає HTTP-запит на сервер;

— сервер обробляє запит і надсилає відповідь;

— браузер скачує весь необхідний для роботи сайту HTML, CSS та JS, запускає його та відображає вміст вебсайту;

— веб-сайт готовий для взаємодії з користувачем.

Domain Name System (DNS) подібно до телефонної книги, обробляє та спідставляє назву веб-сайту, тобто URL-адресу, і конкретну IP-адресу, на яку він посилається. Кожна URL-адреса в Інтернеті має унікальну IP-адресу комп'ютера, на якому розміщено сервер запитуваного веб-сайту. [10]

HTTP — це протокол для отримання ресурсів, наприклад документів HTML. Він є основою будь-якого обміну даними в Інтернеті та є протоколом клієнт-сервер, що означає, що запити ініціюються одержувачем, зазвичай веб-браузером. Повний документ реконструюється з різних піддокументів, наприклад, тексту, опису макета, зображень, відео, сценаріїв тощо. [11]

Клієнти та сервери спілкуються шляхом обміну окремими повідомленнями Повідомлення, які надсилає клієнт, як правило, веббраузер, називаються запитами (requests), а повідомлення, які надсилає сервер як відповідь, називаються відповідями (responses).

Розроблений на початку 1990-х років HTTP є розширюваним протоколом, який з часом розвивався. Це протокол прикладного рівня, який надсилається через TCP або через TCP-з'єднання, зашифроване TLS, хоча теоретично можна використовувати будь-який надійний транспортний протокол. Завдяки своїй розширюваності він використовується не лише для отримання гіпертекстових документів, але й для зображень і відео або для публікації вмісту на серверах, як із результатами HTML-форм. HTTP також можна використовувати для отримання частин документів для оновлення веб-сторінок на вимогу.

Загалом для розробки більшості вебсайтів використовуються HTML, CSS та JavaScript. HTML, мова розмітки гіпертексту, є стандартною мовою розмітки для документів, призначених для відображення у веб-браузері. CSS, каскадні таблиці стилів — це мова таблиць стилів, яка використовується для опису подання документа, написаного на мові розмітки, наприклад HTML або XML. JavaScript, часто скорочено JS — це динамічно типізована, прототимно базована об'єктоно орієнтована, високорівнева мова програмування з функцями першого класу. Станом на 2022 рік 98% веб-сайтів використовують JavaScript на стороні клієнта для поведінки веб-сторінок, часто включають бібліотеки сторонніх розробників. Усі основні веб-браузери мають спеціальний механізм JavaScript для виконання коду на пристроях користувачів.

Фактично HTML є каркасом сторінки, її змістом, а CSS виконує роль візуального оформлення сторінки. JavaScript відповідає за логіку вебсторінки,

обробляє дії користувача та виконує відповідний запрограмований код. JavaScript може використовуватись як для роботи простих анімацій, так і для створення повноцінних вебзастосунків, що здатні оновлювати зміст сторінки без її перезавантаження використовуючи підхід Asynchronous JavaScript and XML (AJAX).

Розробка великих та комплексних вебзастосунків передбачає вирішення частих проблем, таких як: організація, стандартизація, уніфікація підходу до розробки, кодової бази; роутинг (routing), перехід сторінками без презавантаження; система перемалювання (rerendering) та виявлення змін (change detection); управління станом додатку (state management). Вирішити ці тривіальні задачі покликані спеціалізовані бібліотеки та фреймворки. Найпопулярніші, з тих що можуть вирішити ці задачі, це React та Angular.

Вибір правильних фреймворків для розробки вебплатформи не проста задача. Особливо, коли є багато варіантів, які є надзвичайно винахідливими та здатними полегшити вимоги проекту. Але вибір того, що підходить для правильного набору проектів, вимагає знання кількох важливих аспектів.

2.1.2 Аналіз інструментів front-end розробки Angular та React

Обираючи певний фреймворк або бібліотеку, JavaScript має безліч варіантів для інтерфейсної розробки, таких як VueJS, TezJS і Svelte. Однак Angular і React зберігають свої місця у верхній частині списку, оскільки багато розробників вважають їх найкращими інтерфейсними фреймворками розробки через популярність.

Розглянемо 2 найпопулярніші технології, для побудови вебсайтів за допомогою JavaScript, фреймворк Angular та бібліотеку React. React — це інтерфейсна бібліотека JavaScript, яка дозволяє створювати інтерфейс користувача з багаторазово використовуваних компонентів інтерфейсу користувача. React використовує рендеринг на стороні сервера, щоб забезпечити гнучке та продуктивне рішення. Це дозволяє розробникам створювати бездоганний UX і складний UI. Ним керує Facebook і спільнота розробників з

відкритим кодом. Бібліотека була представлена у травні 2013 року та до тепер активно оновлюється.

Angular — це фреймворк JavaScript з відкритим вихідним кодом для веб-розробки та мобільної розробки. Він заснований на TypeScript і керується командою Google Angular і спільнотою користувачів фреймворку. Angular, також відомий як Angular 2.0, запущений у вересні 2016 року, є повністю переписаним AngularJS (Angular 1.0), який був представлений у 2010 році. Angular також можна використовувати як для односторінкових, так і для багатосторінкових веб-додатків. Angular також активно підтримується спільнотою та командою розробників, нові версії виходять регулярно, мажорні версії Angular виходять раз в пів року.

React використовується як у веб-розробці, так і в мобільній розробці. Однак для мобільної розробки його потрібно об’єднати з Cordova. Крім того, для мобільної розробки є додатковий фреймворк — React Native. React можна використовувати для створення як односторінкових, так і багатосторінкових веб-додатків. Angular підходить як для веб-розробки, так і для мобільної розробки. Однак у розробці мобільних пристроїв велика частка роботи виконується Ionic. Крім того, подібно до React, Angular має додатковий фреймворк мобільної розробки. Аналогом React Native є NativeScript.

За допомгою React, будують структуру для розробки інтерфейсу користувача, тому програми, написані за допомогою React, потребують додаткових бібліотек для використання. Наприклад, Redux, React Router або Helmet оптимізують процеси управління станом, маршрутизації та взаємодії з API. Для таких функцій, як зв’язування даних, маршрутизація на основі компонентів, створення проекту, перевірка форм або впровадження залежностей, потрібні додаткові модулі чи бібліотеки. У той час як Angular — це повноцінний фреймворк для розробки програмного забезпечення, який зазвичай не потребує додаткових бібліотек. Усі вищезазначені функції, як зв’язування даних, маршрутизація на основі компонентів, генерація проекту, перевірка

форми та впровадження залежностей, можна реалізувати за допомогою пакета Angular.

React мінімалістичний: без впровадження залежностей, без класичних шаблонів, без надто складних функцій. Фреймворк буде досить простим для розуміння, якщо ви вже добре знаєте JavaScript. Однак для того, щоб навчитися налаштовувати проект, потрібно досить багато часу, оскільки попередньо визначеної структури проекту немає. Вам також потрібно вивчити бібліотеку Redux, яка використовується в більш ніж половині програм React для управління станом. Постійні оновлення фреймворку також вимагають додаткових зусиль від розробника. Крім того, у React є чимало кращих практик, які вам потрібно буде навчитися робити все правильно.

Angular сама по собі є величезною бібліотекою, і вивчення всіх понять, пов'язаних з нею, займе набагато більше часу, ніж у випадку з React. Angular більш складний для розуміння, у ньому багато непотрібного синтаксису, а керування компонентами є складним. Деякі складні функції вбудовані в ядро фреймворку, що означає, що розробник не може уникнути їх вивчення та використання. Крім того, існує безліч способів вирішення однієї проблеми. Хоча TypeScript дуже нагадує JavaScript, для його вивчення також потрібен час. Оскільки фреймворк постійно оновлюється, розробнику потрібно докладати додаткових зусиль для навчання.

Продуктивність React значно покращилася завдяки появі віртуального DOM. Оскільки всі віртуальні дерева DOM легкі та побудовані на сервері, навантаження на браузер зменшується. Крім того, оскільки процес зв'язування даних є однонаправленим, зв'язуванням не призначаються спостерігачі, як у випадку Angular. Відповідно, додаткове навантаження не створюється. Angular працює гірше, особливо у випадку складних і динамічних веб-додатків. Двонаправлене прив'язування даних негативно впливає на продуктивність додатків Angular. Кожному зв'язуванню призначається спостерігач для відстеження змін, і кожен цикл продовжується, доки не будуть перевірені всі спостерігачі та пов'язані значення. Через це, чим більше прив'язок у вас є, тим

більше спостерігачів створюється, і тим громіздкішим стає процес. Однак одне з останіх оновлень Angular значно покращило його продуктивність, і він більше не програє React. Крім того, розмір програми Angular трохи менший за розмір програми React.

Інструменти інтерфейсу користувача для React розроблено спільнотою. На порталі React є багато безкоштовних і платних компонентів інтерфейсу користувача. Щоб використовувати компоненти матеріального дизайну в React, вам доведеться встановити додаткову бібліотеку — Material-UI Library & Dependencies. Angular має вбудований набір інструментів Material, і він пропонує безліч попередньо створених компонентів дизайну матеріалів. Є різні кнопки, макети, індикатори, спливаючі вікна та елементи керування формою. Завдяки цьому налаштування інтерфейсу користувача стає простішим і швидшим.

Angular — це повноцінний фреймворк для мобільних і веб-розробок. React — це фреймворк лише для розробки UI, який можна перетворити на повноцінне рішення за допомогою додаткових бібліотек.

На перший погляд React здається простішим, і для початку роботи над проектом React потрібно менше часу. Однак ця простота як основна перевага React нейтралізується, оскільки вам потрібно навчитися працювати з додатковими фреймворками та інструментами JavaScript.

Angular сам по собі більш складний і потребує досить багато часу для освоєння. Проте це потужний інструмент, який пропонує цілісний досвід веб-розробки, і як тільки ви навчитеся з ним працювати, ви пожинаєте плоди. Немає кращого фреймворку. Обидва постійно оновлюються, щоб не відставати від конкурентів.

Наприклад, якщо вважалося, що React переміг завдяки своєму віртуальному DOM, Angular зрівнявся, реалізувавши виявлення змін (Change Detection Mechanism). Незважаючи на те, що Angular вважався виграшним, тому що його розробляла така авторитетна компанія, як Google, величезна віддана спільнота React повністю компенсувала репутацію Google і зробила React схожим на Angular.

2.2 Серверна частина (back-end)

Back-end це пряма протилежність front-end. У веброзробці, все що стосується роботи вебсайту, до чого не має прямого доступу кінцевий користувач називається back-end. Для створення back-end використовуються різні мови програмування. Навідмінно до front-end де є монополія базових технологій розробки, у back-end може використовуватись багато різних мов програмування, навіть одночасно. Найпопулярнішими мовами програмування для back-end є JavaScript (Express, Koa, NestJS), PHP (Laravel), Python (Flask, Django), Ruby (Ruby on Rails), Java (Spring), C# (.Net Core) і т.д.

2.2.1 Аналіз основних концепцій back-end напрямку

Back-end — це всі технології, необхідні для обробки вхідного запиту (request), створення та надсилання відповіді клієнту (response). Зазвичай це включає три основні частини:

— сервер — це комп’ютер, який отримує запити;

— застосунок — програма, яка працює на сервері, яка прослуховує запити, отримує інформацію з бази даних і надсилає відповідь;

— база даних — використовують для організації та збереження даних.

Сервер — це просто комп'ютер, який прослуховує вхідні запити. Хоча ці комп’ютери, створені та оптимізовані для цієї конкретної мети, будь-який комп’ютер, підключений до мережі, може діяти як сервер. Фактично, власний комп’ютер часто може бути використаний як сервер під час розробки програмного забезпечення.

Сервер запускає програму, яка містить логіку того, як відповідати на різні запити на основі дієслова HTTP та уніфікованого ідентифікатора ресурсу (URI). Пара HTTP-дієслова та URI називається маршрутом, а їх зіставлення на основі запиту називається маршрутизацією.

Деякі з цих функцій обробки будуть проміжним програмним забезпеченням (middleware). У цьому контексті проміжне програмне забезпечення, middleware — це будь-який код, який виконується між сервером,

який отримує запит, і надсилає відповідь. Ці функції проміжного програмного забезпечення можуть змінювати об’єкт запиту, надсилати запит до бази даних або іншим чином обробляти вхідний запит. Функції проміжного програмного забезпечення зазвичай завершуються передачею керування наступній функції проміжного програмного забезпечення, а не надсиланням відповіді. Згодом буде викликана функція проміжного програмного забезпечення, яка завершує цикл запит-відповідь, надсилаючи відповідь HTTP назад клієнту.

Часто програмісти використовують такі фреймворки, як Express, Koa, NestJS, Laravel, Flask, Django, Ruby on Rails, Spring, щоб спростити логіку маршрутизації. Кожен маршрут може мати одну або багато функцій обробки, які виконуються щоразу, коли запит до цього маршруту (дієслово HTTP та URI) збігається.

Дані, які сервер надсилає назад, можуть мати різні форми. Наприклад, сервер може обслуговувати файл HTML, надсилати дані як JSON або він може надсилати назад лише код статусу HTTP. Популярним кодом відповіді є 404, цей код можна отримати якщо перейти до URI, якого не існує, але є багато інших кодів статусу, які вказують на те, що сталося, коли сервер отримав запит, див таблицю 2.1.

Таблиця 2.1 — Найбільш поширені коди HTTP відповідей.

HTTP код	Значення коду HTTP відповіді
100 Continue	Ця проміжна відповідь вказує на те, що клієнт повинен продовжити запит або проігнорувати відповідь, якщо запит уже виконано.
200 OK	Запит виконано. Значення результату "успіх" залежить від методу HTTP: GET: ресурс було отримано та передано в тілі повідомлення. PUT або POST: ресурс, що описує результат дії, передається в тілі повідомлення. HEAD: Заголовки подання включені у відповідь без будь-якого тіла повідомлення.

Продовження таблиці 2.1.

301 Moved Permanently	URL запитуваного ресурсу змінено назавжди. Нова URL-адреса вказана у відповіді.
304 Not Modified	Це використовується для цілей кешування. Він повідомляє клієнту, що відповідь не було змінено, тому клієнт може продовжувати використовувати ту саму кешовану версію відповіді.
400 Bad Request	Сервер не може або не хоче обробити запит через щось, що сприймається як помилка клієнта (наприклад, неправильний синтаксис запиту, недійсний кадр повідомлення запиту або оманлива маршрутизація запиту).
401 Unauthorized	Хоча стандарт HTTP визначає «неавторизований», семантично ця відповідь означає «неавтентифікований». Тобто клієнт повинен пройти автентифікацію, щоб отримати запитану відповідь.
403 Forbidden	Клієнт не має прав доступу до контенту; тобто він неавторизований, тому сервер відмовляється надати запитуваний ресурс. На відміну від 401 Unauthorized, особа клієнта відома серверу.
404 Not Found	Сервер не може знайти запитаний ресурс. У браузері це означає, що URL-адреса не розпізнається. В API це також може означати, що кінцева точка дійсна, але сам ресурс не існує. Сервери також можуть надсилати цю відповідь замість 403 Forbidden, щоб приховати існування ресурсу від неавторизованого клієнта. Цей код відповіді, мабуть, найвідоміший через те, що він часто зустрічається в Інтернеті.
500 Internal Server Error	Сервер зіткнувся з ситуацією, з якою він не знає, як впоратися. Зазвичай, такий код відповіді повертаються користувачеві, якщо сервер не зміг обробити помилку.

Бази даних зазвичай використовуються на сервері вебзастосунків. Ці бази даних забезпечують інтерфейс для постійного збереження даних у пам’яті. Зберігання даних у базі даних зменшує навантаження на основну пам’ять центрального процесора сервера та дозволяє отримати дані, якщо сервер виходить з ладу або втрачає живлення.

Для багатьох запитів, надісланих на сервер, може знадобитися запит до бази даних. Клієнт може запросити інформацію, яка зберігається в базі даних, або клієнт може надіслати дані разом зі своїм запитом на додавання до бази даних. Для запиту даних користувачем використовуються спеціальні API.

API — це набір чітко визначених методів зв’язку між різними програмними компонентами. Якщо говорити точніше, веб-API — це інтерфейс, створений back-end: сукупність кінцевих точок і ресурсів, які ці кінцеві точки надають. Веб-API визначається типами запитів, які він може обробляти, що визначається маршрутами, які він визначає, і типами відповідей, які клієнти можуть очікувати отримати після потрапляння на ці маршрути.

Один веб-API можна використовувати для надання даних для різних front-end. Оскільки веб-API може надавати дані, не вказуючи, як вони переглядаються, для перегляду даних із веб-API можна використовувати кілька різних HTML-сторінок або мобільних застосунків. Тобто, веб-API не має залежності від платформи кінцевого користувача.

Зазвичай сервер не може ініціювати відповіді без запитів. Кожен запит потребує відповіді, навіть якщо це лише код статусу 404, який вказує на те, що вміст не знайдено, див таблицю 2.1. Інакше ваш клієнт залишиться у стані зависання. Сервер не повинен надсилати більше однієї відповіді на запит. Це може спричинити помилки.

2.2.2 Порівняння інструментів розробки back-end застосунків

Back-end фреймворки сьогодні є незамінними для розробки будь-якої складності. Знаходження правильного серверного фреймворку може бути дуже важливим для розробників для забезпечення оптимальної продуктивності та

масштабованості. З такою кількістю доступних сьогодні варіантів вибрати відповідні може бути клопотом.

Основна мета фреймворку — автоматизувати накладні витрати, пов’язані з розробкою програмного забезпечення. Основні переваги використання фреймворку для розробки:

— масштабованість;

— економія часу;

— цілісність;

— безпека;

— інтеграції.

Крім того, більшість фреймворків є відкритими. Тобто, код фреймоворку можуть підтримувати, покращувати самі користувачі фреймворку, а не тільки команди, компанії, що їх розробили. Проаналізуємо декілька відомих фреймворків, їх області застосування та особливості.

2.2.2.1 Django

Django — це провідний фреймворк із відкритим вихідним кодом, заснований на мові програмування Python. Він відповідає шаблону контролера представлення моделі (MVC). Django підходить для розробки складних і багатофункціональних веб-сайтів, керованих базами даних, і це один із найпростіших серверних фреймворків. Django вважається одним із найкращих серверних фреймворків веб-розробки. Цей бек-енд фреймворк забезпечує оптимальне підключення, скорочення кодування, більшу можливість повторного використання та швидшу розробку. Він використовує Python для всіх операцій у Django та надає додатковий інтерфейс адміністратора для створення, читання, оновлення та видалення операцій. Django використовується багатьма відомими веб-сайтами, такими як Disqus, Mozilla та The Washington Times.

Переваги Django у тому, що він швидкий, простий у використанні, а також має низький поріг входу для нових розробників. Фреймворк з низьким часом навчання допомагає розробникам пришвидшити весь процес розробки від

початку до кінця. Він вважається простим сервером для веб-сайтів. Django є багатофункціональним, надає широкий вибір функцій, які допомагають користувачам виконувати деякі загальні вимоги веб-розробки. Це допомагає в таких завданнях, як автентифікація користувачів, карти сайту, адміністрування вмісту та багато іншого. Django — це безпечна структура, яка допомагає своїм користувачам запобігати кільком проблемам безпеки, включаючи міжсайтовий сценарій, клікджекінг, SQL інєкції і підробку запитів. Він забезпечує систему автентифікації користувачів, щоб дозволити користувачам безпечно зберігати та керувати паролями та обліковими записами. Також, Django має високу масштабованість. Це універсальний фреймворк, який можна використовувати для розробки широкого спектру типів програм. Деякі з них включають програми соціальних мереж, системи керування вмістом і обчислювальні платформи.

Проте Django має і недоліки. Наприклад, Django не підходить для малих проектів. Деякі розробники кажуть, що Django не є придатним вибором для малих проектів, оскільки він потребує багато програмування, відносно масштабу проекту. Він також потребує більшої пропускної здатності та часу обробки. Відсутність установлених конвенцій підходів до розробки — це ще один недолік використання фреймворку Django. Відсутність конвенцій створює багато проблем для розробників, включаючи низький рівень комфорту, контрастні компоненти та повільну швидкість розробки. Також, Django це монолітний фреймворк, команди розробників мають менше контролю над робочим процесом через монолітний характер фреймворка Django. Подібним чином через меншу кількість залежностей розробникам доводиться писати більше кодів.

Django має певні особливості. Наприклад, відкритий вихідний код — Django є фреймворком з відкритим кодом, який використовується для веб-додатків на основі Python. Він простий, легкий у використанні та надійний. Система імен у Django має власну систему створення всіх інструментів і функцій. Крім того, він також має просту у використанні панель адміністратора порівняно з Yii та Lavarel. Деякі з ключових функцій Django включають простий синтаксис, основну архітектуру MVC, ORM (Object Relational Mapper), підтримку

проміжного програмного забезпечення та бібліотеки HTTP. Django також має власний веб-сервер, фреймворк для модульного тестування Python і компоненти, необхідні для вирішення деяких випадків.

2.2.2.2 ASP.NET Core

ASP.NET Core — це безкоштовна платформа з відкритим вихідним кодом, яка наслідує ASP.NET, широко використовуваний серверний механізм, створений у партнерстві з .NET Foundation. ASP.NET Core — це модульний фреймворк, який може працювати в усьому .NET Framework у Windows і .NET Core.

У ядра ASP.NET є певні переваги. Наприклад, Підтримка між платформами. Розробка веб-додатків вимагає від розробників забезпечення підтримки всіх платформ. Новий ASP.NET Core — це кросплатформна основа веб-додатків, яка підтримує різні платформи. ASP.NET Core — це міжплатформне рішення для розробки веб-додатків для платформ Windows, Mac і Linux. Сервер використовує однаковий код C# на всіх платформах. Мінімальне кодування — ASP.NET Core використовує технологію, яка вимагає менше кодування. Це означає, що розробники вважають використання back-end досить зручним, оскільки їм доводиться створювати менше операторів. Менше кодування означає менше часу, необхідного для створення програми. Як наслідок, розробникам потрібно менше часу на створення програми, а процес також є економічно ефективним. Технічне обслуговування просте, менше коду також означає менше обслуговування. ASP.NET Core можна автоматично підтримувати у випадках з невеликою кількістю коду. Досвідчені розробники можуть легко зрозуміти, як скоротити зусилля з обслуговування серверної частини ASP.NET Core. Вони можуть оптимізувати код ASP.NET за допомогою лише кількох операторів.

Хороша продуктивність ASP.NET є найбільшою перевагою використання фреймворку ASP.NET Core є підвищення продуктивності, яке вона пропонує. Використання оновлень і останніх удосконалень допомагає розробникам

покращувати код і підвищувати продуктивність програми. Продуктивність також висока, оскільки користувачам не потрібно змінювати код. Вбудований компілятор ASP.NET здатний покращувати код, коли каркас ASP.NET Core перекомпілюється з кодом. Продуктивність фреймворку набагато більше, ніж його альтернативи.

Недоліки у ASP.NET також є. ASP.NET — це дорогий серверний фреймворк, особливо якщо порівнювати його з іншими варіантами. Так, вам знадобиться ліцензія на Visual Studio, Window і SQL Server, щоб використовувати цю структуру. Подібним чином хостинг-провайдери також стягують більшу плату за цю структуру. Також, Важко переносити зміни, якщо ви вносите зміни в одну версію ASP.NET, ви не знайдете їх придатними для роботи в наступній версії цього фреймворку. Дійсно, важко зберегти зміни під час оновлення. Безпека ASP.NET вважається вразливою за відсутності повідомлення про ім’я користувача, зміни пароля та підтвердження електронної пошти. Зазвичай розробникам доводиться платити додатково, щоб отримати необхідний захист для своїх програм.

До особливістей ASP.NET Core можна віднести кросплатформенність. Підтримка контейнерів — ASP.NET Core дозволяє розробникам створювати програми, які можна розгортати на платформах Windows, macOS і Linux. Бекенд найбільше підходить для платформи Linux. Асинхронність ASP.NET Core надає розробникам можливість використовувати шаблони асинхронного програмування. Async є загальною реалізацією для всіх класів.

NET Framework і багатьох зовнішніх бібліотек. Багато програм делегують значну кількість часу та циклів процесора для викликів веб-служб, запитів до бази даних і операцій введення-виведення. Висока продуктивність є однією з ключових особливостей базової системи ASP.NET Core. Веб-сервер Kestrel і ASP.NET Core, які тепер доступні розробникам, роблять ASP.NET однією з найгнучкіших структур веб-додатків. Продуктивність є одним із найважливіших факторів, який робить ASP.NET Core платформою веб-додатків, яку вибирають численні розробники.

2.2.2.3 Spring Boot

Spring Framework — це фреймворк з відкритим кодом та інверсія контейнера керування платформи Java. Програми Java можуть використовувати основні функції цього фреймворку. Користувачі також можуть використовувати багато розширень для створення веб-додатків на основі платформи Java EE.

До переваг Spring Boot відносять:

— вбудована підтримка Undertow, Jetty і Tomcat;

— відсутність необхідності шаблоної конфігурації;

— автоматичний перезапуск сервера призначений для коду, а оновлення конфігурації здійснюється за допомогою DevTools;

— просте керування залежностями;

— просте керування спеціальними властивостями профілю;

— просте налаштування властивостей програми;

— SpringBoot Starters зручні для розробників.

Проте Spring має і свої недоліки. Наприклад, менше контролю —Spring Boot створює незвичні залежності та збільшує розмір файлів розгортання. Через це створюється проблема відсутності контролю над програмою. Це однозначно неправильний вибір для манівців контролю. Також, Spring Boot не підходить для монолітних проектів. Spring Boot дуже добре працює, коли йдеться про створення мікросервісів. Однак це може бути не найкращим вибором для створення монолітних додатків. Не підходить для початківців, якщо ви не знаєте екосистему Spring і програмування AOP або не програмували за допомогою Spring, то вам не слід його використовувати. Так, як розробнику-початківцю, вам може бути складно використовувати Spring Boot.

Особливостями Spring Boot є ініціалізація, налаштування банерів, Fluent Builder API, стан життєздатності. SpringBoot допомагає розробникам виконувати відкладену ініціалізацію. Увімкнення цієї функції допомагає розробникам створювати компоненти на основі вимог, а не під час запуску програми. Отже, відкладена ініціалізація може скоротити час, необхідний для запуску програми.

Налаштування банерів — користувачі можуть змінювати банери запуску, додаючи файл banner.txt до свого шляху до класів. Банери також можна змінювати, вказуючи властивість spring.banner.location на відповідне розташування файлу. Користувачі можуть встановити spring.banner.charset для файлів, використовуючи кодування поза UTF-8. Користувачі можуть додавати зображення banner.jpg, banner.gif і banner.png разом із текстовими файлами до шляху до класів. Вони також можуть встановити spring.banner.image.location.

Fluent Builder API — розробники можуть використовувати SpringApplicationBuilder, якщо їм потрібно створити ієрархію ApplicationContext або використовувати ваш fluent Builder API. Стан життєздатності — стан життєздатності програми відповідає за сповіщення про те, чи дозволяє її внутрішній стан функціонувати або процес відновлення у разі збою. Якщо стан Liveness порушено, програма перебуває в стані, який неможливо відновити, і її інфраструктура перезапускає програму.

2.2.2.4 NestJS

NestJS — це платформа для створення ефективних, масштабованих серверних програм Node.js. Він використовує сучасний JavaScript, побудований на TypeScript (зберігає сумісність із чистим JavaScript) і поєднує елементи ООП (об’єктно-орієнтоване програмування), ФП (функціональне програмування) та FRP (функціональне реактивне програмування). Під капотом Nest використовує Express, але також забезпечує сумісність із багатьма іншими бібліотеками, як-от. Fastify, що дозволяє легко використовувати безліч доступних плагінів сторонніх розробників. Маючи понад 52,3 тисяч зірок і 6,2 тисячі розгалужень (forks) на GitHub, а також щотижневу кількість завантажень до 1 900 000, станом на кінець 2022 року.

Останніми роками, завдяки Node.js, JavaScript став «лінгва франка» Інтернету як для зовнішніх, так і для серверних додатків, створивши чудові проекти, такі як Angular, React і Vue, які покращують продуктивність розробників і дозволяють створювати швидкі, тестовані, розширювані

інтерфейсні програми. Однак на стороні сервера, хоча існує багато чудових бібліотек, помічників та інструментів для Node.js, жоден із них не вирішує ефективно головну проблему — архітектуру. NestJS прагне створити стандартну архітектуру додатків, яка дозволяє без зусиль створювати тестовані, масштабовані, слабко пов’язані та легко підтримувані програми. Архітектура значною мірою натхненна fornt-end фермворком Angular.

NestJS простий у вивченні та опануванні, особливо для розробників зі світу Angular. Це забезпечує швидкий і ефективний процес розробки, оскільки члени команди можуть легко адаптуватися до будь-яких нових принципів і структур розробки. Фреймворк відомий тим, що має чудову архітектурну структуру для корпоративних додатків прямо з коробки, що робить створення високомасштабованих і підтримуваних корпоративних додатків легким. За допомогою NestJS можна легко створювати серверні служби, починаючи від RESTful API, програм GraphQL, програм MVC, веб-сокетів, CLI та завдань Cron. Частина стандартної архітектури вже вбудована в структуру NestJS.

Оскільки NestJS використовує сучасні технології, такі як TypeScript, архітектурні шаблони, доступну документацію та просте модульне тестування, з його допомогою можна створювати масштабовані й підтримувані програми, готові для роботи в межах підприємства. NestJS створено для розробки великомасштабних монолітних і мікросервісних програм, у яких архітектура вже оброблена, і вам потрібно лише побудувати свою бізнес-логіку.

NestJS підтримує та надає великі модулі на основі спільноти, які підтримуються NestJS, для створення будь-якої конкретної функції на ваш вибір, від таких концепцій, як TypeORM, Mongoose та GraphQL до журналювання, перевірки, кешування, WebSockets та багато іншого.

Деякі з переваг використання NestJS включають наступні концепції. NestJS потужний, але зручний для розробників, достатньо простий у використанні навіть найскладнішими та потужними функціями. Команда NestJS розробила фреймворк, щоб розробники могли швидко розпочати роботу та зосередитися виключно на написанні бізнес-логіки, тоді як фреймворк піклується про інші

важливі аспекти розробки, як-от безпеку. Також NestJS має синтаксис у стилі Angular. Angular є дуже популярним font-end фреймворком, який зосереджується на архітектурі та структуруванні. NestJS діє як Angular для серверної частини, оскільки він використовує стиль і синтаксис Angular, щоб допомогти вам структурувати ваш корпоративний проект.

NestJS підтримує TypeScript безпосередньо з коробки, і це покращує продуктивність і швидко створює програми, що підходять для обслуговування, повідомляючи помилки на етапі компіляції коду. TypeScript робить його добре інтегрованим, у VSCode для доступного середовища розробки. NestJS містить доступну документацію для будь-якої структури. Це економить час налагодження, щоб швидко переглянути документацію та отримати рішення вашої проблеми. Пропрацьована архітектура та швидка розробка, Nest.js економить час розробників під час створення програми, незалежно від того, створюєтється перший MVP чи фактична програма, оскільки він дає надійну структуру та архітектуру для роботи з початкової збірки, тим самим покращуючи ваші процеси розробки.

Серед неодоліків NestJS варто відмітити, що Nest.js може бути дещо складним для розробників-початківців, які не мають досвіду Angular для вивчення та освоєння. Крім того, оскільки не кожен розробник JavaScript використовує TypeScript, структура також може бути складною для цих груп розробників. Але, як і будь-яка інша технологія, вона вимагає часу та практики. Також, якщо потрібно детально ознайомитися з деякими параметрами анотації, вам потрібно буде заглибитися у вихідний код @Nest/core. Це можна було б покращити, якщо вони з’явилися в документації NestJs API. NestJs залишається інструментом, на який поширюються деякі обмеження NodeJs. Важкі обчислення вашої програми, ймовірно, доведеться виконувати на сервері іншого типу.

NestJS — це повнофункціональний серверний фреймворк, який представляє себе як найкраща альтернатива для створення середніх і великих програм у NodeJS. Його архітектура, натхненна Angular, дозволяє вам зосередитися на вашій реальній доданій вартості, забезпечуючи при цьому якість

створеного коду. Це дозволяє створювати програми, які можна масштабувати, тестувати та підтримувати. NestJS використовує сучасні рішення та технології, тому додатки, створені з його допомогою, зручні в обслуговуванні та довговічні. Його можна підключити до GraphQL, WebSockets або використовувати для створення мікросервісів.

2.2.3 Порівняння систем керування базами даних

Система керування базами даних або СУБД — це тип програмного забезпечення, яке взаємодіє з самою базою даних, програмами та інтерфейсами користувача для отримання даних і їх аналізу. СУБД також містить ключові інструменти для управління базою даних.

В основному існує два типи СУБД: реляційні та нереляційні, які також називаються SQL та NoSQL відповідно. Перш ніж обговорювати найпопулярніші варіанти баз даних, розглянемо детальніше, чим відрізняються реляційні та нереляційні системи баз даних, враховуючи типові структури даних, продуктивність, масштабованість і безпеку.

Реляційна база даних (SQL) — це тип сховища даних, який організовує дані в таблиці, пов’язані одна з одною, що пояснює назву. Структурована мова запитів є ядром цих систем, оскільки вона використовується для зв’язку з цими базами даних і керування ними, що породило їхню другу назву — бази даних SQL. РСУБД мають попередньо визначену схему, тобто дані зберігаються в рядках (записах) і стовпцях (атрибутах) із строгою структурою.

Реляційні бази даних зазвичай масштабуються вертикально, тобто дані зберігаються на одному сервері, а масштабування здійснюється шляхом додавання додаткової потужності комп’ютера (ЦП, ГП та оперативної пам’яті) до цього одного сервера.

Однак перехід від маленьких до більших машин часто передбачає простої. Масштабування бази даних SQL між декількома серверами (горизонтальне масштабування) може бути проблемою, оскільки вимагає змін структури даних і додаткових інженерних зусиль.

Реляційні бази даних показують чудову продуктивність з інтенсивними операціями читання/запису на малих і середніх наборах даних. Вони також пропонують покращену швидкість пошуку даних завдяки додаванню індексів до полів даних для запитів і об’єднання таблиць. Однак, коли обсяг даних і запитів користувачів зростає, продуктивність може погіршитися.

Завдяки інтегрованій структурі та системі зберігання даних бази даних SQL не вимагають значних інженерних зусиль, щоб зробити їх добре захищеними. Вони є гарним вибором для створення та підтримки складних програмних рішень, де будь-яка взаємодія має ряд наслідків. Однією з основ SQL є відповідність ACID (атомність, узгодженість, ізоляція, довговічність). ACID-сумісність є кращим варіантом, якщо ви створюєте, наприклад, електронну комерцію або фінансові програми, де цілісність бази даних є критичною.

Нереляційна база даних (NoSQL) — це нетаблична база даних, яка використовує різні моделі даних для зберігання, керування та доступу до даних. Найпоширенішими моделями даних є

— документоорієнтований — для зберігання, отримання та керування даними як документами JSON;

— ключ-значення — для представлення даних у вигляді набору пар ключ-значення, де ключі — це унікальні рядки, що мають відповідні значення даних;

— граф — для зберігання даних у структурі вузол-край-вузол, де вузли є точками даних, а ребра — їхніми зв’язками;

— широкий стовпець — для зберігання даних у табличному форматі з гнучкими стовпцями, тобто вони можуть змінюватися від рядка до рядка однієї таблиці.

Оскільки ці бази даних не обмежуються структурою таблиці, вони називаються NoSQL. Вони дозволяють зберігати неструктуровані дані, такі як тексти, фотографії, відео, PDF-файли та багато інших форматів. Дані легко запитувати, але вони не завжди класифікуються за рядками та стовпцями, як у реляційній базі даних.

Коли кількість даних і запитів збільшується, нереляційні бази даних або бази даних NoSQL зазвичай масштабуються горизонтально шляхом додавання додаткових серверів до пулу. Вони обмінюються даними між різними серверами, кожен з яких містить лише частину даних, зменшуючи швидкість запитів за секунду на кожному сервері.

Нереляційні бази даних відомі своєю високою продуктивністю: вони мають розподілену структуру, яка знижує навантаження на продуктивність системи та забезпечує одночасний доступ великій кількості користувачів. Такі бази даних можуть зберігати необмежену кількість даних усіх типів і форм. Вони також досить гнучкі, коли мова йде про зміну типів даних.

На відміну від реляційних систем, бази даних NoSQL мають слабкий захист, що робить їх серйозною проблемою для багатьох інфраструктур. Хоча вони можуть надавати гарантії ACID, зазвичай вони доступні в межах одного розділу бази даних, хоча деякі СУБД пропонують розширені функції безпеки, які відповідають суворим стандартам безпеки та відповідності.

Оскільки бази даних NoSQL дозволяють резервувати різні типи даних разом і масштабувати їх шляхом розростання навколо кількох серверів, їх популярність, яка ніколи не зменшується, зрозуміла. Крім того, створення MVP — це чудовий варіант для стартапів із Agile-розробкою на основі спринту. NoSQL не вимагає підготовки до розгортання, що спрощує швидке оновлення структури даних без затримок у часі.

Для аналізу було обрано 3 популярних та часто використовуваних систем керування базами даних: MySQL, PostgreSQL, та MongoDB, див рисунок 2.2 зі Рейтинг популярності баз даних за версією 2021 Developer Survey by StackOverflow. [12] Зосереджуючись на перевагах і проблемах, також окреслено найкращі випадки використання для кожного.

2.2.3.1 MySQL

MySQL — це одна з найпопулярніших систем реляційних баз даних, див рисунок 2.2. Спочатку MySQL був відкритим вихідним кодом, а тепер належить

корпорації Oracle. Сьогодні MySQL є основою прикладного програмного забезпечення LAMP. Це означає, що він є частиною стеку Linux, Apache, MySQL і Perl/PHP/Python. Маючи C і C++ під капотом, MySQL добре працює з такими системними платформами, як Windows, Linux, MacOS, IRIX та іншими.

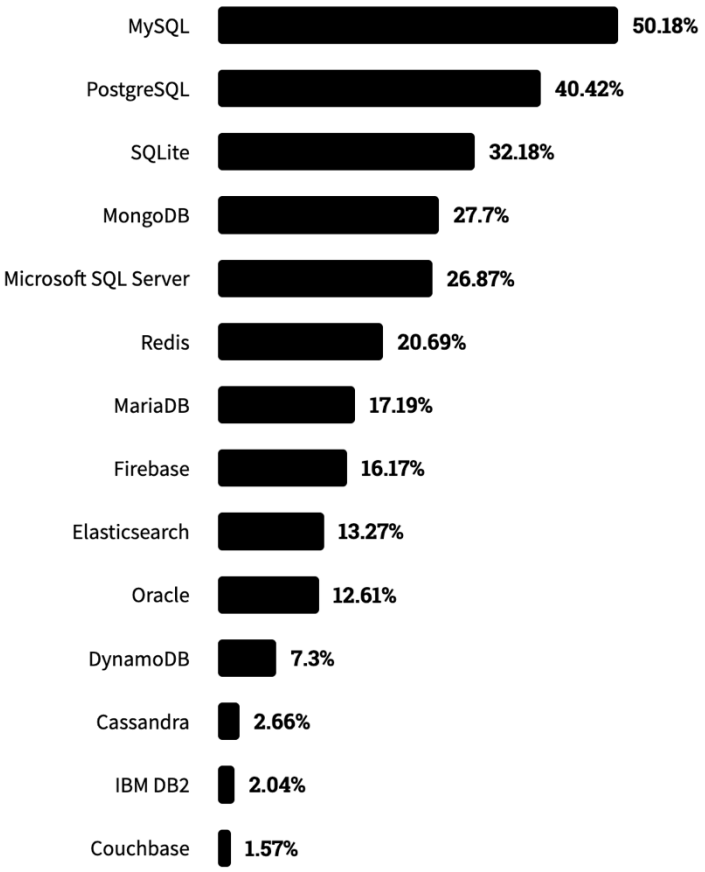

Рисунок 2.2 — Рейтинг популярності баз даних за версією 2021 Developer Survey by StackOverflow [12]

Серед плюсів MySQL можна відзначити безкоштовну установку. Видання спільноти MySQL можна завантажити безкоштовно. Завдяки базовому набору інструментів для індивідуального використання видання MySQL Community є хорошим варіантом для початку. Звичайно, є й інші опції з передоплатою для підприємств або кластерів із більшою функціональністю. Тим не менше, якщо ваша компанія занадто мала, щоб платити за один із них, безкоштовна модель є найбільш підходящою для нового старту. Також, MySQL має простий синтаксис і легку складність. Структура та стиль MySQL дуже прості. Розробники навіть вважають MySQL базою даних з людською мовою. MySQL часто

використовується в тандемі з мовою програмування PHP. Оскільки вони мають спільну криву навчання, набагато легше сформувати команду для керування вашою базою даних. Крім того, MySQL простий у використанні. Наприклад, більшість завдань можна виконувати прямо в командному рядку, скорочуючи кроки розробки.

За своєю природою бізнес-орієнтований і спочатку розроблений для Інтернету, MySQL підтримується найпопулярнішими хмарними провайдерами. Він доступний на таких провідних платформах, як Amazon, Microsoft та інших. Це робить MySQL ще більш привабливим і дає підприємствам можливість для розвитку.

Серед недоліків MySQL проблеми масштабованості. MySQL не створювався з урахуванням масштабованості, яка закладена в його коді. Теоретично ви можете масштабувати MySQL, але для цього знадобиться більше інженерних зусиль порівняно з будь-якою базою даних NoSQL. Отже, якщо ви очікуєте, що одного дня ваша база даних суттєво збільшиться, пам’ятайте про це обмеження або виберіть інший варіант СУБД.

Хоча MySQL має частину з відкритим вихідним кодом, вона здебільшого знаходиться під ліцензією Oracle. Це обмежує спільноту MySQL щодо вдосконалення СУБД.

Оскільки, коли є підтримка з повністю відкритим вихідним кодом, очікується багато реалізацій для конкретних проблем і допомоги від спільноти. Це не той випадок, коли програмне забезпечення належить корпоративним власникам і вам доведеться платити за підтримку.

Структурована мова запитів має певні стандарти. MySQL не повністю їх дотримується, тобто MySQL не підтримує деякі стандартні функції SQL. З іншого боку, MySQL має деякі розширення та відмінні функції, які не відповідають стандартам Structured Query Language. Для невеликих веб-додатків це не є великою проблемою. Проблеми можуть виникнути, коли вам доведеться перейти на інші бази даних, що, швидше за все, станеться, коли ваш бізнес почне розвиватися.

Випадки використання MySQL це невеликі веб-рішення. Система баз даних MySQL є найкращим варіантом, коли ви розробляєте невелике веб-рішення з невеликим обсягом даних. Наприклад, під час створення локального магазину електронної комерції MySQL може стати в нагоді.

2.2.3.2 PostgreSQL

PostgreSQL — це об’єктно-реляційна СУБД, у якій визначені користувачем об’єкти та підходи до таблиць поєднуються для побудови більш складних структур даних. PostgreSQL ділиться своєю популярністю з MySQL Крім того, PostgreSQL має багато схожості з MySQL. Він спрямований на посилення стандартів відповідності та розширюваності. Отже, він може обробляти будь-яке робоче навантаження, як для одномашинних продуктів, так і для складних програм. Володіє та розробляється PostgreSQL Global Development Group, але все ще залишається повністю відкритим кодом. Ця СУБД доступна для використання з такими платформами, як Microsoft, iOS, Android та багатьма іншими.

Серед переваг PostgreSQL чудова масштабованість. Вертикальна масштабованість є відмінною рисою PostgreSQL, на відміну від СУБД MySQL. Враховуючи, що майже будь-яке спеціальне програмне рішення має тенденцію до зростання, що призводить до розширення бази даних, цей конкретний варіант, безумовно, підтримує зростання та розвиток бізнесу.

PostgreSQL за умовчанням підтримує велику кількість типів даних, таких як JSON, XML, H-Store та інші. PostgreSQL використовує його переваги, будучи однією з небагатьох реляційних баз даних із сильною підтримкою функцій NoSQL. Крім того, це дозволяє користувачам визначати власні типи даних. Оскільки вашій бізнес-моделі програмного забезпечення можуть знадобитися різні типи баз даних протягом усього часу її існування для кращої продуктивності або комплексності додатків, цей варіант забезпечує покращену гнучкість таблиці.

Система керування базами даних PostgreSQL має потужну підтримку додаткових інструментів, як безкоштовних, так і комерційних. Сфера їх застосування включає розширення для покращення багатьох аспектів. Наприклад, ClusterControl надає вражаючу допомогу в управлінні, моніторингу та масштабуванні відкритих баз даних SQL і NoSQL. Щоб зробити порівняння та синхронізацію даних більш ефективними, розгляньте можливість використання DB Data Difftective. Якщо ви збираєтеся масштабувати свої дані для великих робочих навантажень, система резервного копіювання та відновлення pgBackRest стане гарним варіантом для вибору.

Підтримка з відкритим кодом і спільнотою PostgreSQL. PostgreSQL є повністю відкритим вихідним кодом і підтримується спільнотою, що зміцнює його як цілісну екосистему. Крім того, розробники завжди можуть розраховувати на безкоштовну та швидку допомогу спільноти.

До недоліків PostgreSQL відносять невідповідну документацію. Незважаючи на те, що PostgreSQL має велику спільноту та надає потужну підтримку своїм учасникам, документації все ще бракує послідовності та повноти. Оскільки спільнота PostgreSQL досить розподілена, документація не відповідає однаковим стандартам для всіх функцій Postgre.

Відсутність інструментів звітності та аудиту відносять до недоліків. Істотним недоліком PostgreSQL є відсутність інструментів ревізії, які б показували поточний стан бази даних. Ви повинні постійно перевіряти, чи не йде щось не так. Завжди існує ризик, що інженери БД помітять збій занадто пізно.

Завдяки складним запитам і широкому вибору настроюваних інтерфейсів із попередньо визначеними функціями PostgreSQL ідеально підходить для аналізу та зберігання даних. Якщо ви створюєте інструмент автоматизації бази даних, PostgreSQL найкраще підходить для нього завдяки потужним аналітичним можливостям, сумісності з ACID і потужному механізму SQL. Все в одному це значно прискорює обробку величезних обсягів даних. Ця СУБД користується популярністю серед фінансових установ і телекомунікаційних систем.

2.2.3.3 MongoDB

MongoDB — безкоштовна нереляційна СУБД із відкритим кодом. MongoDB також включає комерційну версію. Хоча MongoDB спочатку не призначався для обробки структурованих даних, його можна використовувати для програм, які використовують як структуровані, так і неструктуровані дані. У MongoDB бази даних підключаються до програм через драйвери бази даних. Вони широко доступні в системі керування базами даних. Кілька типів даних обробляються одночасно, для цього використовується внутрішній кеш.

До плюсів MongoDB відносять Простий доступ до даних, зберігання, введення та пошук. Однією з переваг MongoDB, яка випливає з її природи NoSQL, є швидка та проста робота з даними. Тобто дані можна вводити, зберігати та вилучати з бази даних швидко й без додаткового підтвердження. Як і в будь-якій іншій нереляційній базі даних, у ній наголошується на використанні оперативної пам’яті, тому записами можна маніпулювати дуже швидко та без жодних наслідків для цілісності даних.

Легку сумісність з іншими моделями даних MongoDB відносять до переваг. MongoDB легко поєднується з різними системами керування базами даних, як SQL, так і NoSQL. Крім того, він має підключаються API механізму зберігання. Коротше кажучи, ця опція дозволяє третім сторонам створювати власні механізми зберігання даних для MongoDB. З комерційної точки зору це створює додаткову цінність для програмного забезпечення для бізнесу.

Масштабованість, коли дані розподіляються по розподіленій мережі керованих серверів, є аспектом фундаментальної природи MongoDB. Це стає ще важливішим для підприємств, які працюють із застосуваннями великих даних. Крім того, база даних може розподіляти дані між кластером машин. Як це може тобі допомогти? Дані розподіляються швидше та рівномірно, без громіздкості. Оскільки це призводить до швидшої обробки даних, продуктивність програми також прискорюється.

До неодліків MongoDB відносять велике споживання пам'яті. Процес денормалізації, коли раніше нормалізовані дані в базі даних групуються для

підвищення продуктивності, зазвичай призводить до значного споживання пам’яті. Крім того, ця СУБД зберігає в пам'яті всі імена ключів для кожної пари значень. Крім того, через відсутність підтримки об’єднань бази даних Mongo мають надлишок даних, що призводить до великої втрати пам’яті та зниження продуктивності програми.

Оскільки MongoDB, як і будь-яка інша СУБД NoSQL, орієнтована на швидку роботу з даними, не має захисту даних. Оскільки автентифікація користувача не є опцією Mongo за умовчанням, а вищий захист доступний лише в комерційній версії, ви не можете вважати її повністю безпечною. Крім того, існують постійні випуски оновлень MongoDB, без гарантії, що всі поправки чи зміни даних працюватимуть, як раніше. Майте на увазі, що всі маніпуляції повинні формуватися навколо цих оновлень, покриваючись додатковими тестами. Складний процес інтерпретації на інші мови запитів. Оскільки MongoDB спочатку не розроблявся для роботи з реляційними моделями даних, у таких випадках продуктивність може сповільнитися. Крім того, переклад запитів SQL у MongoDB вимагає додаткових дій для використання механізму, що може затримати розробку та розгортання.

MongoDB найкраще працює в інтеграції даних у реальному часі та масштабованості бази даних. Наприклад, це правильний варіант для каталогів продуктів завдяки його здатності зберігати безліч об’єктів із різноманітними колекціями атрибутів. Також розгляньте тут аналітичні платформи, оскільки швидкість MongoDB забезпечує динамічну продуктивність, яка може допомогти відстежувати поведінку користувача в режимі реального часу.

2.3 Вибір технологій для реалізації запропонованого рішення

Опираючись аналіз бібліотеки React та фреймворку Angular, див. підпункт 2.1.2, було сформовано таблицю 2.2. В результаті, було надано перевагу фреймворку Angular, оскільки він пропонує більш структурний підхід до побудови великих вебзастосунків, ніж React. React це лише бібліотека для ефективного відображення. Щоб React мав стільки ж функціоналу, як Angular,

до нього потрібно додати з півдесятка і більше бібліотек. Ці додаткові бібліотеки перетворюють кожен React проект на унікальний набір технологій, через що потім важко наймати нових членів команди, для розробки продукту. Тому те, що Angular надає усі необхідні інструменти та модулі без додаткових бібліотек, на відміну від React, це значний плюс.

Також проекти на Angular мають зручні інструменти для розробки та оновлення проекту, які надає Angular CLI. У ролі допоміжної UI бібліотеки обрано Angular Material UI, оскільки ця бібліотека створена розробниками Angular та містить усі необхідні для побудови запропонованої вебплатформи компоненти та інструменти (CDK).

Таблиця 2.2 — Порівняння front-end фреймворку Angular та бібліотеки React

Характеристика	Angular	React
Наявність інструментів розробки форм, анімацій, роутингу, роботи з HTTP і т. д. прямо з коробки	+	-
Підтримка TypeScript	+	+/-
Наявність CLI	+	+/-
Data Binding	Two-way	One-way
Активна спільнота розробників	+	+
Шаблон	HTML + TypeScript	JSX/TSX
Зірок на GitHub	85 200	198 000
Відгілкувань на GitHub	22 600	41 200

Опираючись на аналіз інструментів для розробки back-end, див підпункт 2.2.2, перевагу було надано фреймворку NestJS. Оскільки NestJS має стандартну архітектуру додатків, яка дозволяє без зусиль створювати тестовані,

масштабовані, слабко пов’язані та легко підтримувані модулі програми. Архітектура у NestJS значною мірою натхненна fornt-end фермворком Angular та має спільну мову програмування TypeScript, що спрощує розробку та розуміння структури проекту. Також, NestJS сумісний з бібліотеками для роботи з WebSocket, наприклад Socket.IO, яка необхідна для розробки чату. NestJS має рішення для роботи з різними базами данних, зокрема TypeORM дає можливість працювати NestJS з різними типами SQL та NoSQL баз даних.

Відповідно до аналізу систем керування базами даних, див підпункт 2.2.3, було сформовано порівняльну таблицю 2.3 та було надано перевагу СУБД PostgreSQL. Оскільки, у випадку розробки системи що має структуровані дані з великою кількістю зв’язків, варто використовувати реляційні бази даних, а не NoSQL рішення. PostgreSQL має відкритий вихідний код та безкоштовна у використанні, на відміну від Oracle та MSSQL. Відповідно до аналізу PostgreSQL має інструменти для масштабування, на відміну від MySQL. Також MySQL має обмежену відповідність SQL стандартам, MySQL має деякі розширення та відмінні функції, які не відповідають стандартам Structured Query Language.

Таблиця 2.3 — Порівняння СУБД PostgreSQL та MySQL

Характеристика	PostgresSQL	MySQL
Архітектура	Об'єктно-реляційний; багатопроцесна	Реляційна; однопроцесна
Підтримувані типи даних	Числовий, дата/час, символьний, логічний, нумерований, геометричний, мережева адреса, JSON, XML, HSTORE, масиви, діапазони, композитні	Числові, дата/час, символьні, просторові, JSON
Продуктивність	Підходить для додатків із великим обсягом як читання, так і запису	Підходить для додатків із великим обсягом читань

Продовження таблиці 2.3

Безпека	Контроль доступу, кілька варіантів зашифрованого підключення	Контроль доступу, зашифровані з'єднання
Підтримка	Підтримка громади. Компанії, які мають власний випуск PostgreSQL, можуть запропонувати підтримку навколо нього.	Підтримка спільноти, а також контракти на підтримку від постачальників

Отже, в результаті аналізу було обрано наступний набір технологій для розробки запропонованої вебплатформи для колективного розв’язання проблем та задач:

— front-end: Angular фреймворк, Angular Material UI;

— back-end: NestJS фреймворк, TypeORM, Socket.IO;

— data base: PostgreSQL.

Для організації роботи модулів проекту між собою, зокрема Angular та NestJS, у яких використовується спільна мова програмування TypeScript, варто використовувати монорепозиторій. Популярним інструментом для створення монорепозиторів з підтримкою Angular та NestJS є Nx (nrwl/nx). За допомогою Nx можна організувати монорепозиторій, перевагою якого є те, що у проекту буде одна єдина конфігурація package.json та спільні модулі (node_modules). Nx також відповідає за кешування білдів, що пришвидшує час компіляції проектів.

2.4 Планування схеми бази даних

Оскільки у аналізі, з пункту 2.3, вже було визначено, що використовуватись буде саме реляційна SQL база даних, то варто проаналізувати структуру майбутньої бази даних вебплатформи.

У процесі розробки бази даних керуються певними принципами. Перший принцип полягає в тому, що дублювання інформації це погано, оскільки вона витрачає простір бази даних і збільшує ймовірність помилок та невідповідностей. Другий принцип полягає в тому, що важлива правильність і

повнота інформації. Якщо база даних містить неправильну інформацію, будь-які звіти, які отримують інформацію з бази даних, також міститимуть неправильну інформацію. У результаті будь-які рішення, які прийняте на основі цих звітів, будуть невірні.

Щоб формувати базу даних для застосунку, варто добре розуміти які сутності є у додатку та як вони пов’язані та взаємодіють. З вимог описаних у пункті 1.2 можна виокремити такі сутності даних, як: користувач (user), питання або тема обговорення (question or topic), відповідь (answer), повідомлення (message), tag (tag). Взаємозв’язки сутностей баз даних передано на рисунку 2.3. Сутності з’єднані лініями посередині яких описано як саме пов’язані чи взаємодіють сутності. Також застосовано тип запису UML, щоб показати тип відношення за допомогою позначень «1» — один, «0..*» — один та більше.

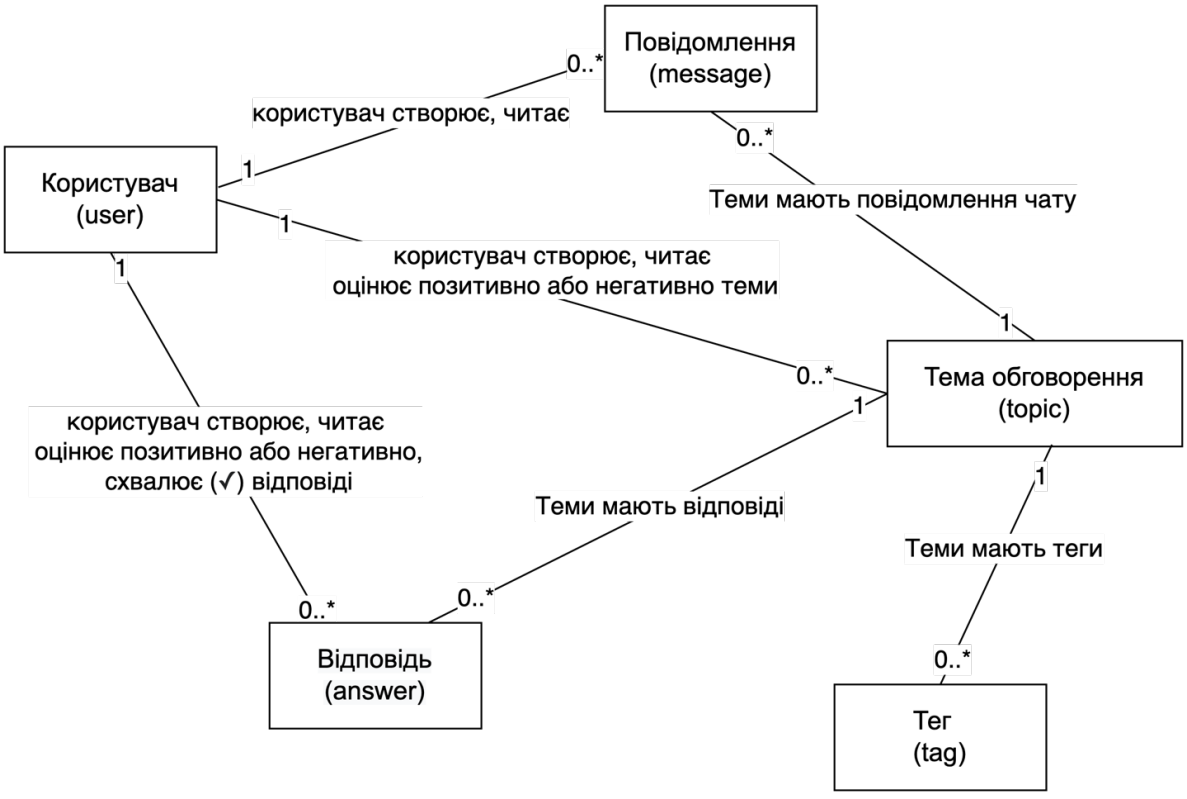

Рисунок 2.3 — Схематична візуалізація взаємозв’язків сутностей даних застосунку

Будувати таблиці бази даних варто застосовуючи правила нормалізації. Нормалізація схеми бази даних — покроковий процес розбиття одного

відношення, таблиці відповідно до алгоритму нормалізації на декілька відношень на базі функціональних залежностей. Всього виділяють 11 нормальних форм, проте неофіційно зв'язок реляційної бази даних часто описується як "нормалізований", якщо він відповідає третій нормальній формі. Більшість зв’язків 3NF не мають аномалій вставки, оновлення та видалення. Тому розробляти базу даних є сенс не вище 3 нормальної форми.

Перша нормальна форма стверджує, що на перетині кожного рядка та стовпця в таблиці існує одне значення, а не список значень. Наприклад, не може бути поле з іменем Ціна, у якому можна розмістити більше ніж одну ціну. Якщо розглядати кожен перетин рядків і стовпців як клітинку, кожна клітинка може містити лише одне значення. [14]

Друга нормальна форма вимагає, щоб кожен неключовий стовпець повністю залежав від усього первинного ключа, а не лише від частини ключа. Це правило застосовується, якщо у вас є первинний ключ, який складається з кількох стовпців. Наприклад, припустімо, що є таблиця, яка містить такі стовпці, де ID замовлення та ID продукту утворюють первинний ключ:

— ID замовлення (первинний ключ);

— ID продукту (первинний ключ);

— назва продукту.

Цей дизайн порушує другу нормальну форму, оскільки назва продукту залежить від ідентифікатора продукту, але не від ідентифікатора замовлення, тому воно не залежить від усього первинного ключа. Тому потрібно видалити назву продукту з таблиці, він належить до іншої таблиці (Продукти). [14]

Третя нормальна форма вимагає, щоб не тільки кожен неключовий стовпець залежав від усього первинного ключа, але й щоб неключові стовпці були незалежними один від одного. Іншими словами, кожен неключовий стовпець повинен залежати від первинного ключа і нічого, крім первинного ключа. Наприклад, припустімо, що у вас є таблиця, що містить такі стовпці:

— ID продукту (первинний ключ);

— ім'я;

— рекомендована роздрібна ціна (РРЦ);

— знижка.

Припустимо, що знижка залежить від рекомендованої роздрібної ціни (РРЦ). Ця таблиця порушує третю нормальну форму, оскільки неключовий стовпець знижка залежить від іншого неключового стовпця РРЦ. Незалежність стовпця означає, що має бути можливість змінювати будь-який неключовий стовпець, не впливаючи на інші стовпці. Якщо ви зміните значення в полі РРЦ, знижка зміниться відповідно, порушуючи це правило. У цьому випадку сповпець знижка слід перемістити до іншої таблиці, яка введена в РРЦ. [14]

Згідно з вище визначеними сутностями даних та з дотриманням правил 3 нормальної форми розроблено базу даних зображену на рисунку 2.4.

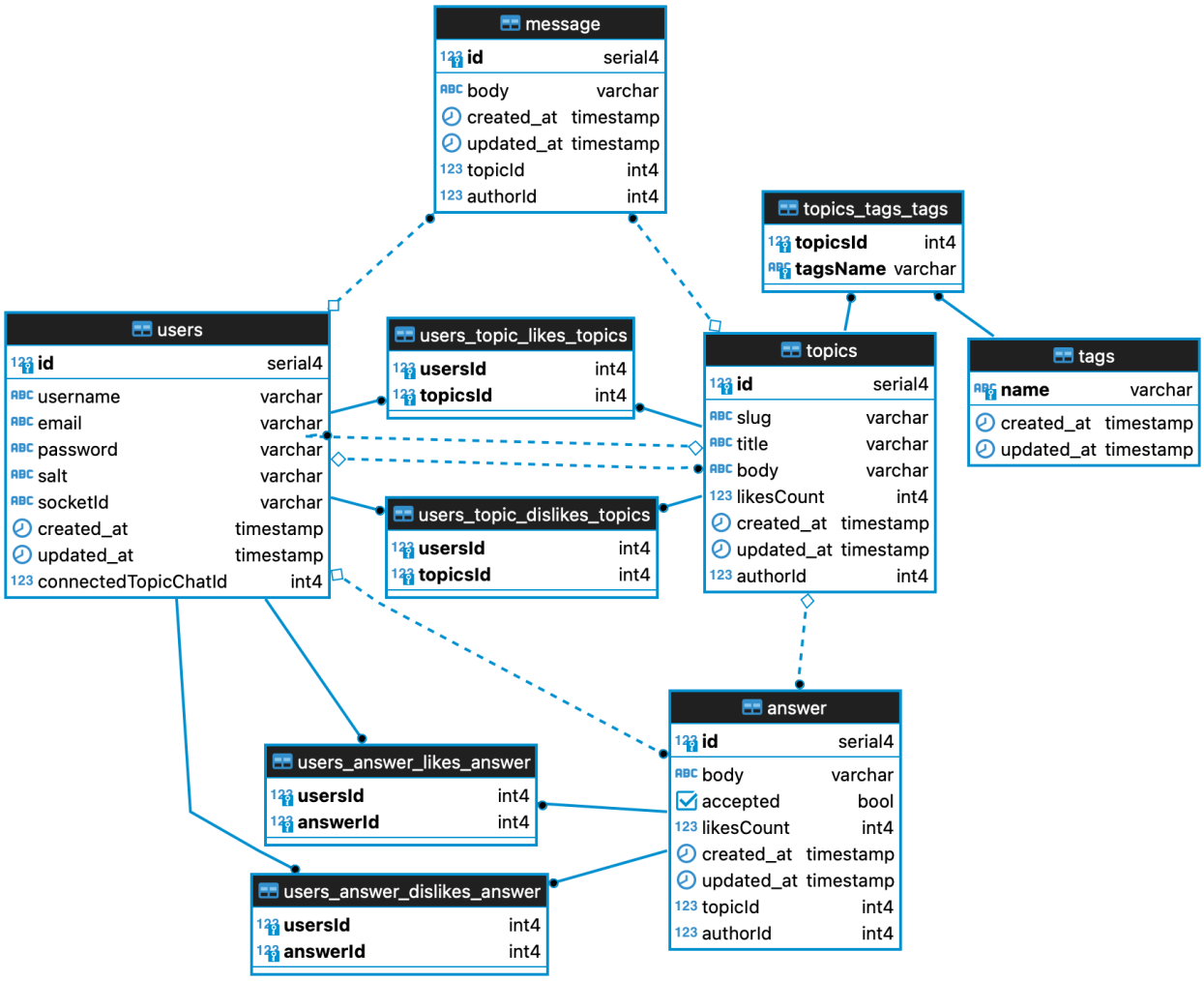

Рисунок 2.4 — Візуалізація взаємозв’язків сутностей даних застосунку у таблицях

У ній описано сутності та їх взаємозв’язки на рівні таблиць бази даних. Суцільна лінія означає, що стовпець зовнішнього ключа в одній таблиці є первинним ключем в іншій таблиці. Пунктирна лінія означає, що стовпець зовнішнього ключа не є первинним ключем, а звичайним стовпцем в іншій таблиці зображення. Чорна точка використовується як початок лінії. Ромб використовується в кінці посилання, коли стовпець у вихідній таблиці є необов’язковим (немає NOT NULL).

У кожні таблиці біля назви поля вказано тип цього поля. Детальніше про кожен тип:

— serial4 — чотирибайтове ціле число, що автоматично збільшується, використовується для позначення ID;

— varchar — символьний рядок змінної довжини;

— timestamp — дата і час;

— int4 — чотирибайтове ціле число зі знаком;

— bool — логічний логічний (true/false). [15]

Для відображення зв’язків many-to-many використовуються проміжні таблиці з комбінованими назвами, наприклад users_topic-dislikes_topics. Такі назви таблиць були автоматично згенеровані бібліотекою TypeORM, про TypeOrm більше у 3 розділі.

2.5 Розрахунок оптимальної кількості розв’язувань питань на одиницю часу

Час очікування відповіді на питання поставлені через запропоновану платформу, див. пункт 1.2, може бути проаналізовано за допомогою теорії масового обслуговування. Теорія масового обслуговування — це математичне дослідження черг очікування, або черг. Модель черги побудована так, що можна передбачити довжину черги та час очікування. Теорія масового обслуговування зазвичай вважається розділом дослідження операцій, оскільки результати часто використовуються під час прийняття бізнес-рішень щодо ресурсів, необхідних для надання послуги.

У теорії масового обслуговування, дисципліні в рамках математичної теорії ймовірності, нотація Кендалла, або іноді нотація Кендалла, є стандартною системою, яка використовується для опису та класифікації вузла масового обслуговування. Кендалл запропонував описати моделі масового обслуговування за допомогою трьох факторів, написаних A/S/c, у 1953 році, де A — позначає час між надходженнями в чергу, S — розподіл часу обслуговування, а c — кількість каналів обслуговування, відкритих у вузлі. З тих пір його було розширено до A/S/c/K/N/D, де K — ємність черги, N — розмір популяції завдань, які потрібно обслуговувати, а D — дисципліна черги. Якщо останні три параметри не вказано (наприклад, M/M/1 черга), припускається, що K = ∞, N = ∞ і D = FIFO.

Проаналізуємо конкретно запропоновану платформу питання-відповідь, то відповідно до формули 2.2. A та S, час між надходженнями в чергу та час розподілу обслуговування — M, процес Пуассона (або випадковий) процес надходження, тобто експоненціальний час між надходженнями.

Значення c, кількість каналів обслуговування — s, це кількість користувачів системи, що готові брати участь у розв’язанні задач. Значення K та N, ємність черги та розмір популяції завдань, які треба обслуговувати — ∞, безкінечність. Значення D, дисципліна черги — скоріше PQ, пріоритетна черга, оскільки у застосунку є рейтингова система питань. Тобто, запропонована система питання-відповідь має нотацію Кендадла M/M/s/∞/∞/PQ.

Коли є одна черга з більш, ніж 1 паралельним сервером, маємо так звану систему масового обслуговування M/M/s. Важливо зрозуміти різницю між системою масового обслуговування M/M/s і системою масового обслуговування s разів M/M/1. У системі масового обслуговування M/M/s ми маємо s паралельних серверів з 1 чергою, тоді як у системі масового обслуговування s*M/M/1 ми маємо ту саму кількість серверів, але ми змінюємо конфігурацію, щоб мати s кількість черг, див рисунок 2.5.

Щоб обчислити параметри системи масового обслуговування, спочатку потрібно обчислити коефіцієнт інтенсивності трафіку та ймовірність того, що

система простоює P_0. Якщо відомі швидкість надходження та рейтинг обслуговування системи, то параметри системи можна обчислити за наступними формулами. [13]

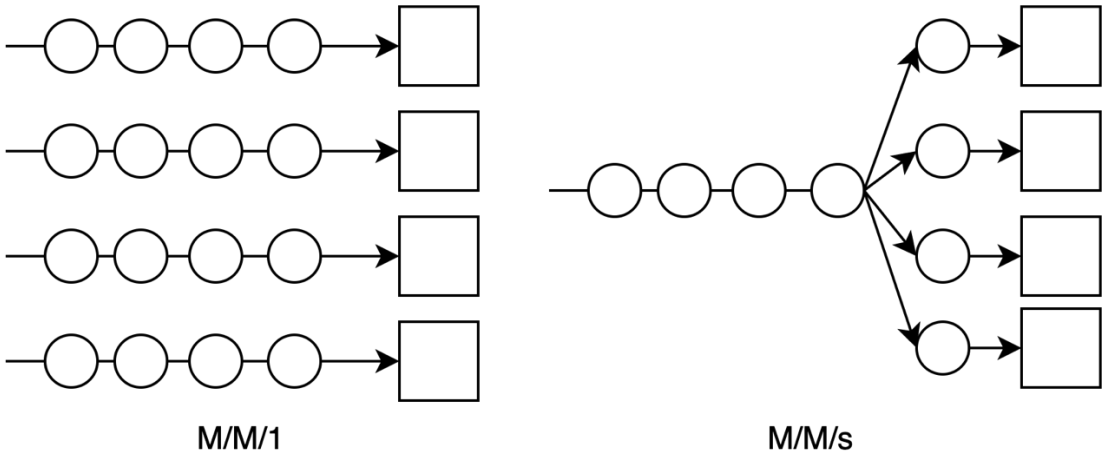

Рисунок 2.5 — Схема систем масового обслуговування M/M/1 та M/M/s, кола — користувачі, квадрати — сервіси.

Формула обчислення коефіцієнту інтенсивності трафіку ρ:

$$\rho=\frac{\lambda}{\mu}, \tag{2.1}$$

де λ — швидкість надходження, кількість клієнтів/одиницю часу, μ — рейтинг обслуговування, кількість клієнтів/одиницю часу.

Формула коефіцієнту використання U — відсоток часу, коли всі сервери зайняті:

$$U=\frac{\rho}{s}=\frac{\lambda}{s\mu}. \tag{2.2}$$

Формула ймовірності того, що в системі немає клієнтів P_0:

$$P_0=\left(\sum_{i=0}^{s-1}\frac{\rho^i}{i!}+\frac{\rho^s}{s!}\left(\frac{s\mu}{s\mu-\lambda}\right)\right)^{-1}. \tag{2.3}$$

Формула середнього часу, який клієнт проводить в черзі в очікуванні обслуговування W_q:

$$W_q = \frac{L_q}{\lambda}. \quad (2.4)$$

Формула середнього часу, який клієнт проводить у системі, в черзі та на обслуговуванні W:

$$W = W_q + \frac{1}{\mu} = \frac{L}{\lambda}. \quad (2.5)$$

Формула середньої кількості клієнтів в черзі на обслуговування L_q:

$$L_q = \frac{P_0 \rho^{s+1}}{(s-1)!(s-\rho)^2}. \quad (2.6)$$

Формула середньої кількості клієнтів системи в черзі та обслуговуютьс:

$$L = \lambda W. \quad (2.7)$$

Відповідно до вказаних вище формул можна побудувати таблицю залежності рейтингу надходження та обслуговування питань користувачів системи за одиницю часу і кількості можливо розв'язаних задач, див таблицю 2.4. Згідно з даними з таблиці 2.4, якщо 11% / 13% / 21% користувачів розв'язуватимуть 10% / 8% / 5 % задач відповідно, що надійшли за певну одиницю часу, то цього буде достатньо для розв'язання всіх задач що прийшли за цю одиницю часу. Якщо показники будуть менші за вказані вище, то кількість нерозв'язаних задач буде невпинно рости.

Таблиця 2.4 — Параметри запропонованої системи питання-відповідь згідно з теорією обслуговування M/M/s

λ	μ	s	ρ	L	L_q	W	W_q
100	5	21	0.95	35.2	15.2	0.35	0.15
100	5	25	0.8	20.8	0.8	0.2	0.008
100	5	35	0.5	20	0.002	0.2	0
100	5	50	0.4	20	0	0.2	0
100	8	13	0.96	33.6	21.1	0.33	0.21
100	8	16	0.7	13.4	0.94	0.13	0.0094
100	10	11	0.9	16.8	6.8	0.17	0.068
100	10	15	0.6	10	0.2	0.1	0.002

Варто згадати практичне правило черги (QROT), що представляє математичну формулу, відома як рівняння обмеження черги, коли вона використовується для визначення наближеної кількості серверів, необхідних для обслуговування черги. Формула записується як нерівність:

$$s > \frac{Nr}{T}, \tag{2.8}$$

яка пов’язує кількість серверів — s, загальну кількість запитувачів обслуговування — N, час обслуговування — r і максимальний час для спорожнення черги — T. [13]

Відповідно до цього практичного правила черги (QROT) побудуємо таблицю рівню обмеження черги. З таблиці 2.5, можна простежити схожу залежність параметрів, як у таблиці 2.4.

Таблиця 2.5 — Рівень обмеження черги платформи

N	r	T	$\frac{Nr}{T}$
100	1/2	1	50
100	1/3	1	33.3
100	1/5	1	20
100	1/8	1	12.5
100	1/10	1	10
100	1/12	1	8.3
100	1/15	1	6.6

Чим більше завдань може бути розв’язано за одинцю часу користувачами, що розв’язують задачі — параметр r, тим менше таких користувачів необхідно на платформі для розв’язання усіх задач платформи.

3. НАЛАШТУВАННЯ МОНОРЕПОЗИТОРІЮ ТА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ВЕБПЛАТФОРМИ

Реалізація включає в себе опис імплементації вимог до вебплатформи для колективного розв’язання проблем та задач сформованих у пунткі 1.2. Для реалізації вимог необхідно організувати середовище розробки — монорепозиторій на Nx, серверну частину — back-end на NestJS, TypeORM, Socket.IO, базу даних на PostgreSQL та клієнтську частину — front-end на Angular, Angular Material, Socket.IO. Детальніше про вибір технологій у пункті 2.3.

3.1 Налаштування монорепозиторію

Популярним інструментом для створення монорепозиторів з підтримкою Angular та NestJS є Nx (nrwl/nx). За допомогою Nx можна організувати монорепозиторій, перевагою якого є те, що у такому проекті одна єдина конфігурація package.json та спільні модулі (node_modules). Nx також відповідає за кешування збірки проекту, що пришвидшує час компіляції проектів для розробки та продакшину.

Для розгортання Nx монорепозиторію необхідно мати встановленим на комп’ютер файловий менеджер yarn. Далі запустити команду yarn create nx-workspace question-answer --preset=nest та зайти в новостворену директорію question-answer — cd question-answer. У цій папці Nx створив базовий NestJS проект. Щоб встановити Angular необхідно додати @nrwl/angular пакет до проекту за допомогою команди — yarn add -D @nrwl/angular, та згенерувати клієнтську частину — yarn nx g @nrwl/angular:app client.

Далі необхідно налаштувати проксі на клієнтському застосунку. Щоб HTTP запити надіслані з клієнтського додатку на шлях «/api» перенаправлялись на певний порт серверного застосунку. Для цього необхідно створити proxy.conf.json файл у apps/client директорії та зареєструвати цей проскі файл у apps/client/project.json.

Якщо все налаштовано правильно, то запустити клієнтський та серверний застосунок можна командами yarn nx serve client та yarn nx serve server відповідно. Завдяки налаштованому проксі на клієнті можна відправляти HTTP запити з клієнтського застосунку на сервер. Фінальну файлову структуру створеного застосунку зображено на рисунку 3.1.

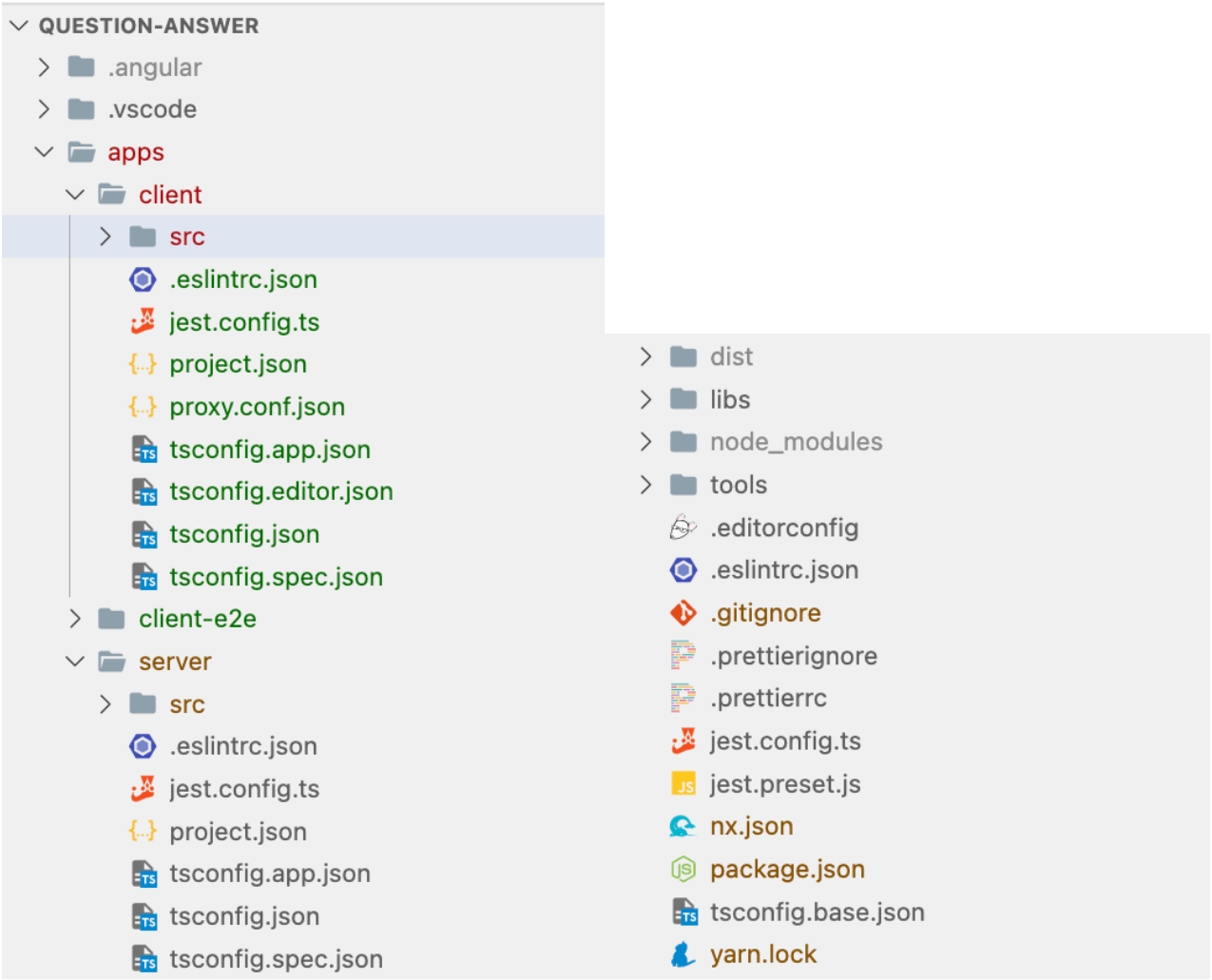

Рисунок 3.1 — Файлова структура згенерованого застосунку через Nx

Щоб використовувати спільні модулі між клієнтським та серверним застосунками можна створити бібліотеку. Для цього в Nx також є інструменти та є можливість згенерувати бібліотеки монорепозиторію, які можна використовувати у різних частинах програми. Наприклад, у випадку Angular та NestJS застосунків, у яких мова програмування однакова — TypeScript, можна створити спільну бібліотеку що буде містити спільні інтерфейси для 2

застосунків. Таким чином можна реалізувати контракт API для клієнтського та серверного застосунку, оскільки там і там, будуть використовуватись однакові моделі, інтерфейси опису, імплементації моделей Data Transfer Object (DTO) для клієнт-серверної взаємодії.

Також для того щоб код, що розробляється, був оформлений в одному стилі написання, варто налаштувати ESLint. Ця підпрограма буде аналізувати написаний код та сигналізуватиме чи код відповідає вказаним правилам. Наявність такого інструменту на проекті покращує структурованість коду, та змушує розробників слідувати певним правилам оформлення.

3.2 Серверна частина

Базова конфігурація NestJS застосунку згенерована за допомогою Nx у пункті 3.1. Для забезпечення взаємодії з базою даних, у нашому випадку PostgreSQL, із NestJS застосунком, необхідно встановити ORM бібліотеку. Об’єктно-реляційне відображення (ORM) — в інформатиці це техніка програмування для перетворення даних між системами типів за допомогою об’єктно-орієнтованих мов програмування. Це фактично створює «віртуальну базу даних об’єктів», яку можна використовувати з мови програмування. [16]

Для NestJS у ролі ORM використаємо TypeORM бібліотеку. Її можна використовувати з TypeScript і JavaScript (ES5, ES6, ES7, ES8). Її мета полягає в тому, щоб завжди підтримувати найновіші функції JavaScript і надавати додаткові функції, які допомагають розробляти будь-які програми, які використовують бази даних. Від невеликих програм з кількома таблицями до великих корпоративних програм з кількома базами даних. [17]

Структурно NestJS складається з модулів. Кожна програма має принаймні один модуль, кореневий модуль. Кореневий модуль є відправною точкою, яку Nest використовує для побудови графіка застосунку — внутрішньої структури даних, яку Nest використовує для вирішення зв’язків і залежностей між модулем і провайдером. Хоча дуже маленькі програми теоретично можуть мати лише

кореневий модуль, це не типовий випадок. Модулі настійно рекомендуються як ефективний спосіб організації компонентів.

Таким чином, для більшості програм результуюча архітектура буде використовувати кілька модулів, кожен з яких інкапсулює тісно пов’язаний набір можливостей. Так на кожен об’єкт платформи питання-відповідь: питання, відповідь, користувач, тег, повідомлення, варто створити свій модуль. Кожен модуль, якщо дуже сильно спрощувати, може містити декілька контролерів (controller), сервісів (service), сутності (entity) та інші модулі, щоб використовувати експортні сервіси та сутності цих модулів.

Контролери відповідають за обробку вхідних запитів і повернення відповідей клієнту. Метою контролера є отримання конкретних запитів для програми. Механізм маршрутизації контролює, який контролер отримує які запити. Часто кожен контролер має більше одного маршруту, і різні маршрути можуть виконувати різні дії. Щоб створити базовий контролер, використовуються класи та декоратори TypeScript. Декоратори пов’язують класи з необхідними метаданими та дозволяють NestJS створювати карту маршрутизації, прив’язувати запити до відповідних контролерів.

Оскільки у застосунку повинна бути реєстрація, як зазначено у 1.2, то варто реалізувати якийсь механізм автентифікації користувача та утримання сесії. Популярним рішенням цієї задачі є використання JSON Web Token (JWT). Веб-токен JSON (JWT) — це відкритий стандарт (RFC 7519), який визначає компактний і самодостатній спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Цю інформацію можна перевірити та довіряти їй, оскільки вона має цифровий підпис. JWT можна підписати за допомогою секрету (з алгоритмом HMAC) або пари відкритих/приватних ключів за допомогою RSA або ECDSA. [18]

Підписані токени можуть підтвердити цілісність даних, що містяться в ньому. Коли токени підписуються за допомогою пар відкритих/приватних ключів, підпис також засвідчує, що лише сторона, яка володіє закритим ключем, є тією, яка його підписала. Таким чином підписаний серверною стороною JWT,

так званим секретом (secret), містить відкриті дані, цілісність яких гарантується підписом. Якщо дані будуть спотворені, то при зворотному розшифруванні JWT буде визнано сфабрикованим. Умовно JWT це наче підписана перепустка з якою користувач може підтвердити свою особистість для системи (серверу). У разі спотворення даних перепустки (JWT) система (сервер) розпізнає фальсифікацію та визнає перепустку не дійсною.

Тоді реєстрація на платформу питання-відповідь буде працювати наступним чином. Користувач вводить свої дані у спеціальній формі на клієнтській стороні, ці дані передаються у зашифрованому вигляді через POST HTTP запит на «/api/users». Контролер серверного застосунку обробляє запит користувача, див лістинг 3.1.

Лістинг 3.1 — Фрагмент обробки POST HTTP запиту на реєстрацію користувача

```
@Controller() export class UserController {
  public constructor(private userService: UserService) {}
  @Post('users') public async create(@Body() createUserDTO: CreateUserDto): Promise<UserAuthResponseDto> {
    const user = await this.userService.create(createUserDTO);
    return this.userService.buildCreateUserResponse(user);
  }
 // …
}
```

У контролері обробляються отримані дані createUserDTO та викликається метод create сервісу користувача (userService) що створить, зареєструє нового користувача. Попередньо дані що надіслав користувач createUserDTO валідуються, перевіряються чи відповідають вони певним вимогам. Наприклад, чи пароль має необхідну довжину, відповідний тип даних і т. д. У методі create аналізується чи вже існує користувач з таким користувацьким ім’ям (username) та електродною поштою (email). Якщо користувач з такими даним вже існує, то

користувачеві буде повідомлено про HTTP помилку 422 UNPROCESSABLE_ENTITY. Якщо з вхідними даними все гаразд, то такий користувач буде збережено у базі даних.

При реєстрації, після збереження даних користувача у базі даних, користувачеві відправляються його публічні дані з JWT назад. Надалі, якщо користувач при наступному запиті на сервер використовуватиме Authorization заголовок HTTP з вказаним токеном (Authorization: Bearer <вміст JWT>), то після підтвердження достовірності цього токену на стороні серверу, вважатиметься що запит надіслав саме автентифікований користувач.

Проте, чутливі дані як пароль користувача, небезпечно зберігати у базі даних у відкритому вигляді. Такі дані потрібно шифрувати. У випадку з NodeJS, який має багато вбудованих модулів, в нагоді стане модуль Crypto, а зокрема метод pbkdf2Sync. З допомогою цього методу можна односторонньо захешувати стрічку, в нашому випадку пароль. Таким чином, в базі даних варто зберігати не пароль у звичайному вигляді а його зашифровану версію. При наступній перевірці чи пароль користувача збігається з паролем збереженим у базі даних, необхідно провести стрічку, яку хочемо перевірити на відповідність паролю (надалі початкова стрічка), через ті ж маніпуляції з pbkdf2Sync, що і з оригінальним паролем. Якщо кінцевий результат шифрування початкової стрічки збігається з зашифрованим паролем, який збережено у базі даних, то початкова стрічка тотожна первинному паролю до шифрування.

Процедура входу в систему досить схожа на процедуру реєстрації. Після відправки даних користувача на сервер, їх також аналізують. Якщо дані валідні, та за такими даними існує зареєстрований користувач, його пароль збігається з надісланим, то користувачеві, який зробив HTTP запит на логін, відправлять його публічні дані з JWT. Цей JWT, як перепустка, підтверджуватиме автентифікованість такого користувача при наступних HTTP запитах.

Для збереження даних використовуються спеціальні сутності, так звані entity. Цей інструмент надає TypeORM. За допомогою сутностей можна

описувати структуру таблиць бази даних, типи даних колонок, їх назви та зв’язки таблиць між собою.

Сутності це програмне представлення, відображення таблиць у коді. Через TypeORM також можна маніпулювати даними таблиць та робити SQL прямо з серверної частини застосунку. Фрагмент оформлення основних полів сутності користувач (UserEntity) зображено у лістингу 3.2.

Такі сутності створюються для кожного об’єкту програми який має представлення у базі даних. Саме за допомогою таких сутностей та опису зв’язків між ними побудована таблиці бази даних на рисунку 2.4. За допомогою декораторів в описі полів сутності, таких @OneToMany, @ManyToOne, @ManyToMany та параметрів цих декораторів описано взаємозв’язки між таблицями бази даних, сутностями.

Лістинг 3.2 — Фрагмент опису сутності користувач UserEntity

```
@Entity({ name: 'users' })
export class UserEntity {
  @PrimaryGeneratedColumn('increment') public id: number;
  @Column({ nullable: false }) public username: string;
  @Column({ unique: true, nullable: false })  public email: string;
  @Column({ select: false, unique: true, nullable: false }) public password: string;
  //…
}
```

Наприклад у сутності теми обговорення (TopicEntity), за допомогою @OneToMany(() => AnswerEntity, (answer) => answer.topic) public answers: AnswerEntity[] створюється відношення до сутності відповіді (AnswerEntity). Одночасно з тим у сутності відповіді (AnswerEntity) аналогічна конструкція тільки з іншого боку @ManyToOne(() => TopicEntity, (topic) => topic.answers) public topic: TopicEntity, де замість @OneToMany використовується @ManyToOne. Таким чином за допомогою сутностей TypeORM створюється

зв’язок таблиці теми обговорення (topic) з таблицею відповідь (answer) one-to-may. Одна тема для обговорення може мати декілька відповідей.

Для збереження позитивних на негативних реакцій користувачів до тем обговорень (питань) та відповідей на сутності користувача створено спеціальні поля topicLikes, topicDislikes, answerLikes, answerDislikes, ці поля декоровано @ManyToMany функцією. Тобто, користувач може мати декілька позитивно та негативно оцінених питань та відповідей. Питання та відповіді можуть мати реакції багатьох користувачів. Для many-to-many відношень TypeORM самостійно генерує проміжні таблиці між користувачем та питанням чи відповіддю, наглядно зображено на рисунку 2.4.

Схвалення відповіді (✓) може робити тільки автор питання. Тому при HTTP POST «/api/:answerId» запиті перевіряється чи даний користувач, який зробив HTTP запит, є автором питання до якого належить дана відповідь. . Якщо користувач не має прав на схвалення даної відповіді, то йому повертається відповідь з HTTP кодом 401 UNAUTHORIZED. Якщо користувач має право на схвалення даної відповіді, то у відповідь йому повертаються усі відповіді до питання з актуальною інформацією. Якщо користувач уже раніше схвалив якусь іншу відповідь, то у неї забирають позначку схвалення (✓) і на дають її останній зазначеній у запиті схвалення відповіді. Якщо відправлений запит на схвалення було надіслано стосовно відповіді яка вже має схвалення, то її позначу схвалення буде нівельовано.

Для реалізації чату використано бібліотеку Socket.IO. Socket.IO — це бібліотека, яка забезпечує двонаправлений зв’язок між клієнтом і сервером на основі подій із низькою затримкою. Він створений на основі протоколу WebSocket і надає додаткові гарантії, як-от повернення до тривалого HTTP опитування (HTTP long-polling) або автоматичне повторне підключення. Кожне повідомлення також має сутність TypeORM для представлення у базі даних.

Наприклад, сутність повідомлення чату має поля:

— id, ідентифікатор повідомлення;

— body, зміст повідомлення, безпосередньо текст повідомлення;

— topic, зв’язок з темою обговорення через декоратор @ManyToOne. Одне обговорення може мати багато повідомлень;

— author, зв’язок з автором даного повідомлення через декоратор @ManyToOne. Один автор може мати багато повідомлень;

— createdAt, часова мітка з часом останьої зміни будь-якого поля;

— updatedAt, часова мітка створення даного повідомлення.

Для опису обробників подій WebSocket використовуються декоратори @SubscribeMessage(<назва події>). Такі декоратори вказуються перед функціями обробниками. Наприклад, у застосунку використано 2 функції обробники.

Одна така функція для обробки приєднання користувача до чату, назва події ChatActions.JoinChat. При відправлені події на приєднання також вказується topicId, що вказує до якого саме чату питання користувач хоче приєднатись. В результаті цього обробника буде створено нову подію ChatActions.ChatHistory з інформацією про всі попередні повідомлення цього чату. Таким чином користувач, який приєднався до чату після того як в чаті була дискусія зможе побачити історію обговореня.

Також, у даних запиту ChatActions.JoinChat міститься дані користувача що робить запит. Якщо користувач автентифікований, то на його сутності в базі даних, зберігається topicId та socketId. Ці дані зберігаються, щоб в подальшому, якщо будуть в чаті нові повідомлення, то було відомо кому повідомляти про це нове повідомлення та на який socketId це нове повідомлення надсилати. При перезапуску сервера всі збережені на сутності користувача topicId та socketId для спілкування в чаті очищуються.

Друга функція ChatActions.AddMessage перевіряє чи користувач автентифікований. Якщо ні, то розриває з’єднання з сокетом, оскільки спілкуватись в чаті можуть тільки автентифіковані користувачі. Якщо з автентифікацією все гаразд, то всім під’єднаним до цього чату користувачам надсилається повідомлення з подією ChatActions.AddedMessage.

Для спілкування між клієнтом та сервером використовується програмний інтерфейс застосунку (API). У даному випадку ця комунікація вебклієнта та сервера відбувається через HTTP. Відповідно до правил REST API було розроблено інтрефейс взаємодії клієнта та сервера.

Загалом для застосунку було реалізовано 17 REST API ендпоїнтів, з яких 12 захищені, тобто потребують автентифікації користувача. Та як вже було згадано вище, 2 шлюзи (Gateway) для Socket.IO комунікації чатів, та 4 події (4 events) для роботи чатів. Детальніше про кожен едпоїнт можна знайти інформацію на сторінці самого застосунку за шляхом «/api». На цій сторінці автоматично згенерована документація ендпоїнтів, описана за допомогою Swagger Open API, див рисунок 3.2.

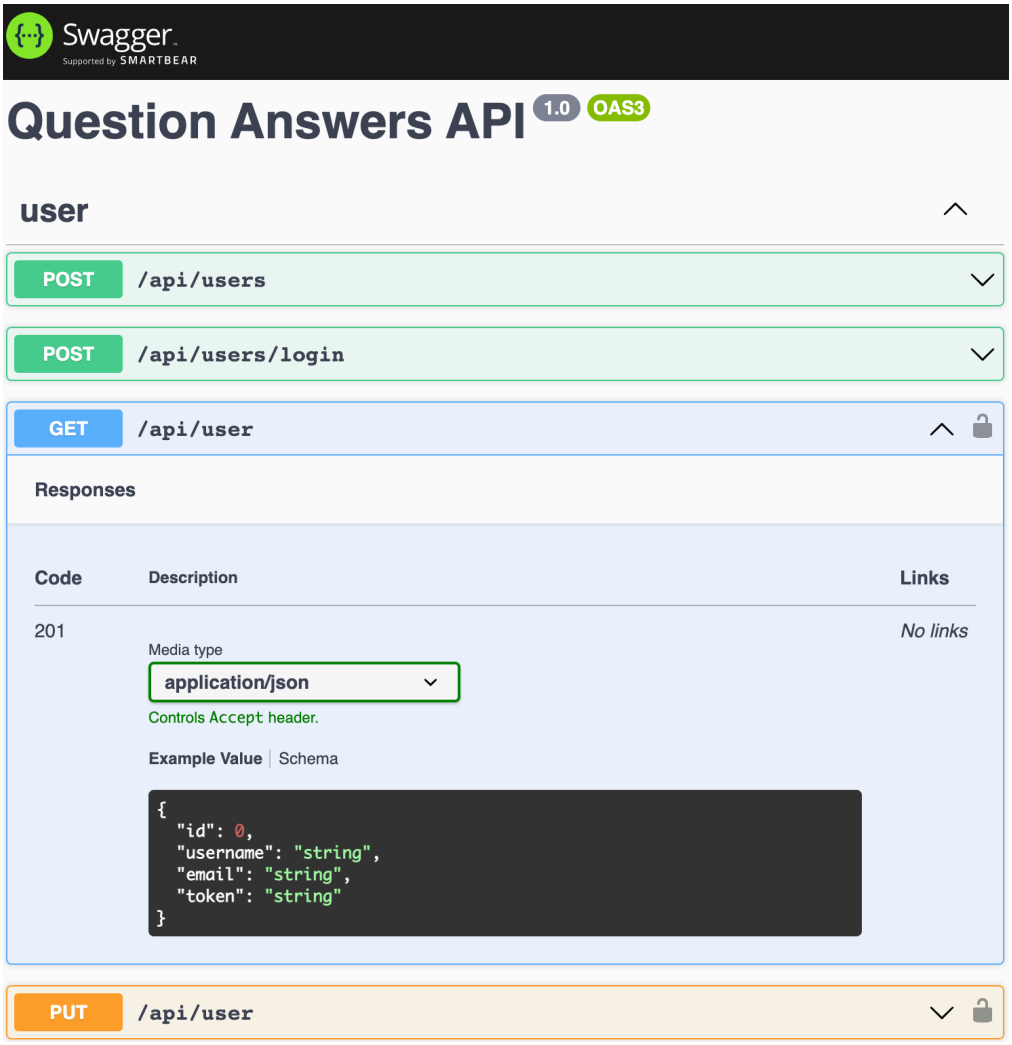

Рисунок 3.2 — Фрагмент сторінки з описом REST API ендпоїнтів згенерованох за допомогою Swagger OpenAPI

Наприклад для створення нового користувача необхідно відправити HTTP POST запит на «/api/users» з даними про username, email, password. Для логіну POST запит на «/api/users/login» з email та password. Для отримання даних про автентифікованого користувача GET «/api/users» та для оновлення даних користувача PUT «/api/users» з даними про оновлені поля.

Останні 2 запити вимагають щоб користувач був автентифікованим. Для перевірки автентифікації користувача на певні ендпоїнти ставлять вартові декоратори, @UseGuards(AuthGuard). Такі вартові автентифікаційні (AuthGuard) перевіряють чи юзер автентифікований. Якщо не автентифікований користувач надсилає запит на певний едпоїнт, то сервер повертає помилку такому користувачеві з кодом 401 UNAUTHORIZED.

4. РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБПЛАТФОРМИ

Клієнтська сторона застосунку не важлива та не менш складна. Розглянемо організацію модулів проекту, методи обробки JWT, маршрутизацію, візуалізацію компонентів і їх станів, використання методу віртуальних списків.

4.1 Модулі системи

Проекти на Angular мають модульну структуру. Angular має власну систему модульності під назвою NGModules. NGModules — це контейнери для пов’язаних блоків коду, присвяченого спільному домену, робочого процесу або тісно пов'язаного набору можливостей. Вони можуть містити компоненти, постачальники послуг та інші файли коду. Вони можуть імпортувати функціональність, яка експортується з інших NGModules, та експортувати вибрані функціональні можливості для використання іншими NGModules.

Кожен Angular додаток має щонайменше один клас NGModules, кореневий модуль, який умовно названий AppModule і знаходиться у файлі під назвою app.module.ts. Програми запускаються шляхом завантаження кореневого NGModule. Хоча невелика програма може мати лише один NGModules, у більшості застосунків є набагато більше модулів функцій. Кореневий NGModules для програми так названий, оскільки він може включати підмодулі у ієрархії будь-якої глибини. [19]

Модулі Angular та NestJS дещо схожі за своєю структурою, оскільки розробники NestJS надихались структурою Angular. [20] Подібно до того як в NestJS було створено модуль для кожої сутності, в Angualr також необхідно стврити модуль для кожної частини програми яка має спільний сенс. Таким чином можна винести наступні модулі:

— app, корінний модуль, батьківський для всіх підмодулів;

— core, основа застосунку, де зареєстровані основні вартові, сервіси, компоненти та сторінки;

— about, для сторінки про проект;

— topic, для компонентів теми обговорення;

— answer, для компонентів відповіді;

— chat, для компонентів чату;

— modules, для загальних модулів, які можна перевикористовувати, але вони вживаються не настільки часто, щоб переносити їх у shared. Момент передчасної оптимізації;

— shared, для спільних модулів, які перевикористовуються дуже часто.

4.2 Автентифікація

Після реєстрації чи логіну в систему необхідно, згадані у попередньому розділі, JWT обробляти на клієнті та прикріплювати для наступних HTTP запитів на сервер. Для такої мети у Angular є спеціальний механізм під назвою перехоплювач (interceptor). Це спеціальний клас, що реєструється у модулі як HTTP_INTERCEPTORS, таким чином усі HTTP запити та відповіді перехоплюються таким зареєстрованим класом. У зареєстрованому класі і відбувається перевірка чи є у користувача збережений власний JWT. Якщо токен є, то він записується у HTTP заголовку Authorization.

Якщо користувач отримує з серверу JWT після реєстрації чи логіну в систему, то поки JWT має актуальний термін дії, такий користувач вважається автентифікованим, тобто таким, який підтвердив свою особистість. У застосунку є певні сторінки та операції які може робити тільки автентифікований користувач. Для перевірки чи користувач автентифікований та які його публічні дані (користувацьке ім’я, email) використовується AuthService. Цей сервіс має функції для реєстрації, входу та виходу з системи, перевірки чи користувач автентифікований.

4.3 Маршрутизація

Маршрутизація застосунку зазвичай також описується на рівні модулів. Так у AppModule маршрутизація включає опис шляху на власно зареєстровані сторінки (AboutComponent), так і на дочірню маршрутизацію підмодулів, наприклад authRoutes, topicRoutes, див. лістинг 4.1.

Кожен маршрут у цьому масиві є об’єктом JavaScript, який містить дві властивості. Перша властивість шлях (path), визначає URL-шлях для маршруту. Друга властивість компонент (component), визначає компонент, який Angular має використовувати для відповідного шляху. Або замість component може використовуватись children, який містить масив дочірніх об’єктів маршрутів. Що стосується відображення, то в більшості випадків використовувались власно розроблені компоненти з використанням елементів Angular Material, а саме: кнопки, іконки, поля вводу, картки, таби і т. д. Кольорова тема теж базувалась на палітрі кольорів Angular Material. Обраний був пурпуровий (deep purple), але кольорову схему платформи легко змінити через файли конфігурації теми.

Лістинг 4.1 — Фрагмент декларації модуля маршрутизація корінного модуля AppModule

```
const routes: Routes = [
  { path: '', redirectTo: 'topics', pathMatch: 'full' },
  { path: 'about', component: AboutComponent },
  { path: 'auth', children: authRoutes },
  { path: 'topics', children: topicRoutes },
  { path: '**', redirectTo: 'topics' },
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule { }
```

4.4 Основні компоненти системи

Доступ до вебплатформи не вимагає обов’язкової автентифікації, встановлення особистості користувача. Усі питання, відповіді, історії чатів, реакції на питання чи відповіді, а також схвалення відповіді доступні без

реєстрації. На рисунку 4.1 зображено який вигляд має головна, початкова сторінка реалізованого застосунку відповідно до вимог з пункту 1.2. З верху сторінки з ліва на право зображено логотип платформи, основні сторінки (теми та про проект), кнопка для створення власної теми обговорення, кнопки логіну та реєстрації.

Також на рисунку 4.1 нижче шапки сайту (header) відображено пошукову стрічку. Якщо ввести в неї пошуковий запит та натиснути кнопку пошук (search), то система відображатиме лише ті відповіді у яких знайшла збіги пошукового запиту із заголовком або описом питання.

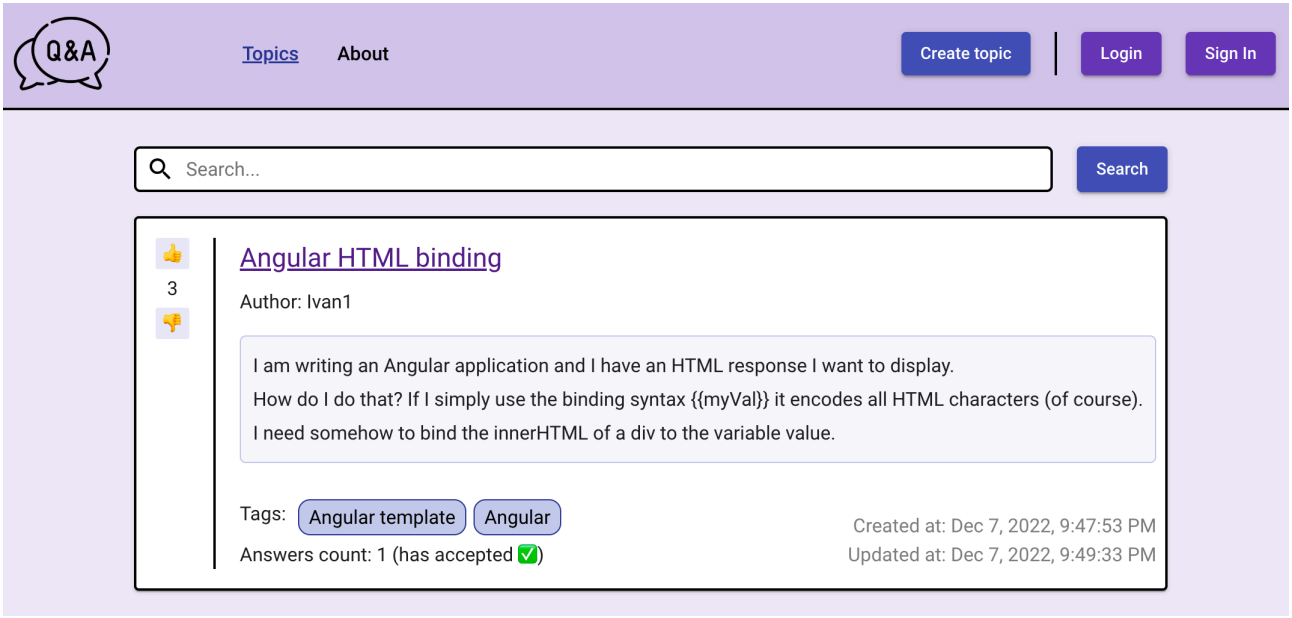

Рисунок 4.1 — Фрагмент головної сторінки вебплатформи, вигляд тем для обговорення для не автентифікованого користувача

Під пошуковою стрічкою зображено саме питання. Тема для обговорення складається з лівої частини, де зображено рейтинг реакцій та кнопки для самих реакцій, та з головної правої частини. На правій частині по порядку з верху до низу відображено заголовок теми, автор теми, опис питання, теги до яких належить тема, статус відповідей — кількість відповідей та чи є у теми схвалена відповідь, та дати створення питання та дати редагування питання. Якщо неавтентифікований користувач спробує здійснити дію, яка вимагає від користувача бути автентифікованим, то він побачить відповідне повідомлення у

лівому нижньому куті екрану. Повідомлення вказуватиме користувачеві, що для виконання таких дій необхідно бути зареєстрованим у системі.

Перейти на саму сторінку питання користувач може натиснувши на заголовок питання, який підкреслено лінією та зображено як покликання на вебсторінку. Вигляд сторінки питання зображено на рисунку 4.2.

На сторінці питання сам компонент питання, який вже було розібрано вище, має такий же самий вигляд, як на головній сторінці. Безпосередньо під самим питанням зображено 2 вкладки, таби. Перша вкладка Answers — це питання до відповіді, див рисунок 4.2, друга вкладка Chat Conversation — це чат для колективного пошуку рішення, див рисунок 4.3.

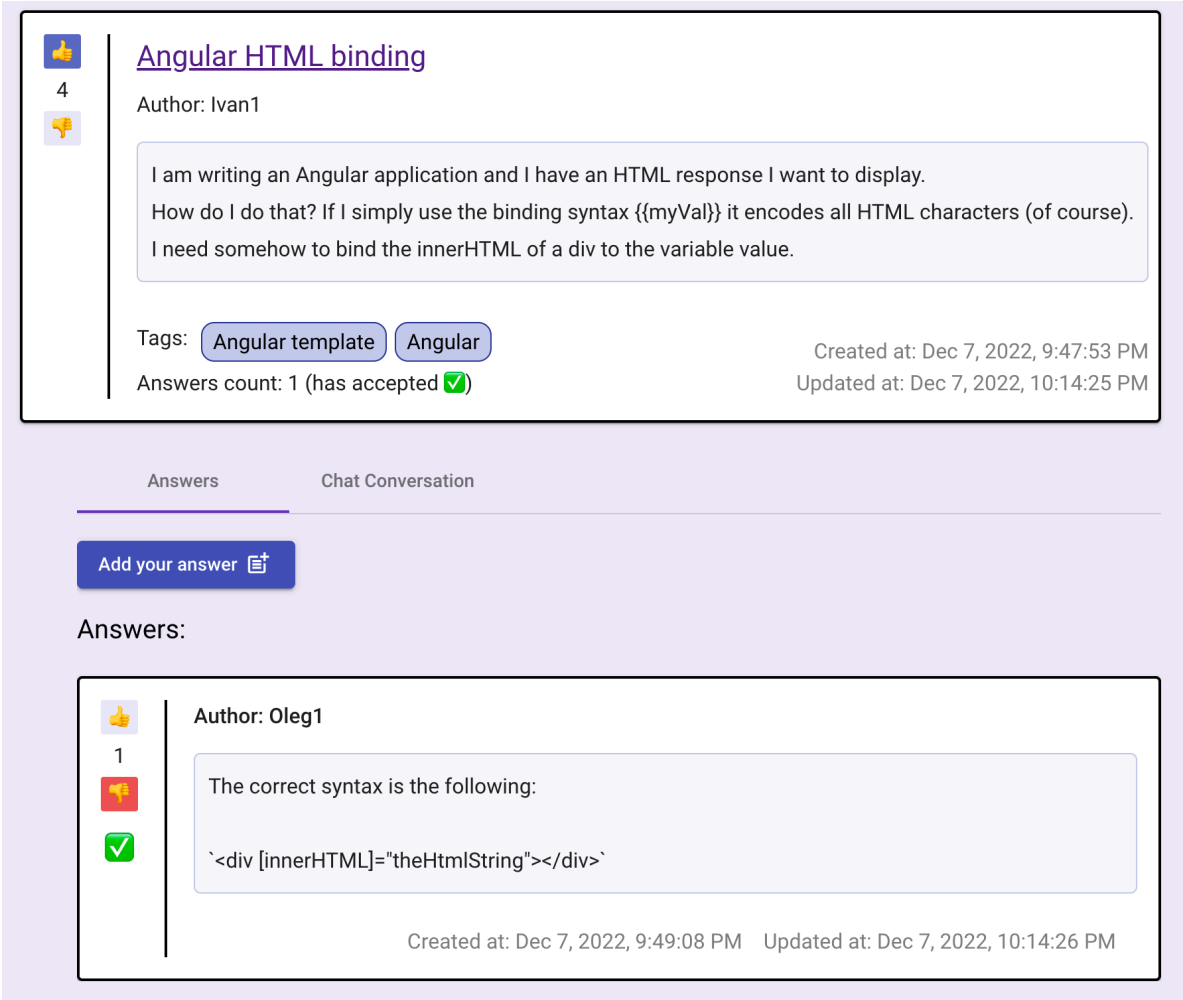

Рисунок 4.2 — Фрагмент сторінки питання з відповідями до нього

Розглянемо кожну вкладку окремо. На вкладці з відповідями зображено список відповідей до питання одна за одною з верху в низ. Розглянемо приклад

відповіді наведений на рисунку 4.2. Структура схожа до компоненту питання. Зліва рейтинг, справа основна частина.

Звернімо увагу на стан реакції зліва компоненту. Число рейтингу це загальна сума усіх реакцій, позитивна реакція це +1 бал рейтингу, негативна реакція це -1 бал рейтингу. Автентифіковані користувачі можуть залишати свої реакції на питання та відповіді. Якщо користувач автентифікований, то він може бачити стан своїх реакцій.

Наприклад якщо реакція позитивна, то значок з великим пальцем вгору підсвічено пурпуровим кольором, якщо реакція негативна, то великий палець вниз підсвічено червоним. Див. рисунок 4.2, де питання (компонент зверху), має рейтинг 4 та позитивну реакцію з відміченим пальцем у верх. Також зверніть увагу на відповідь з рисунку 4.2, де рейтинг 1 та відмічена червоним негативна реакція з великим пальцем донизу.

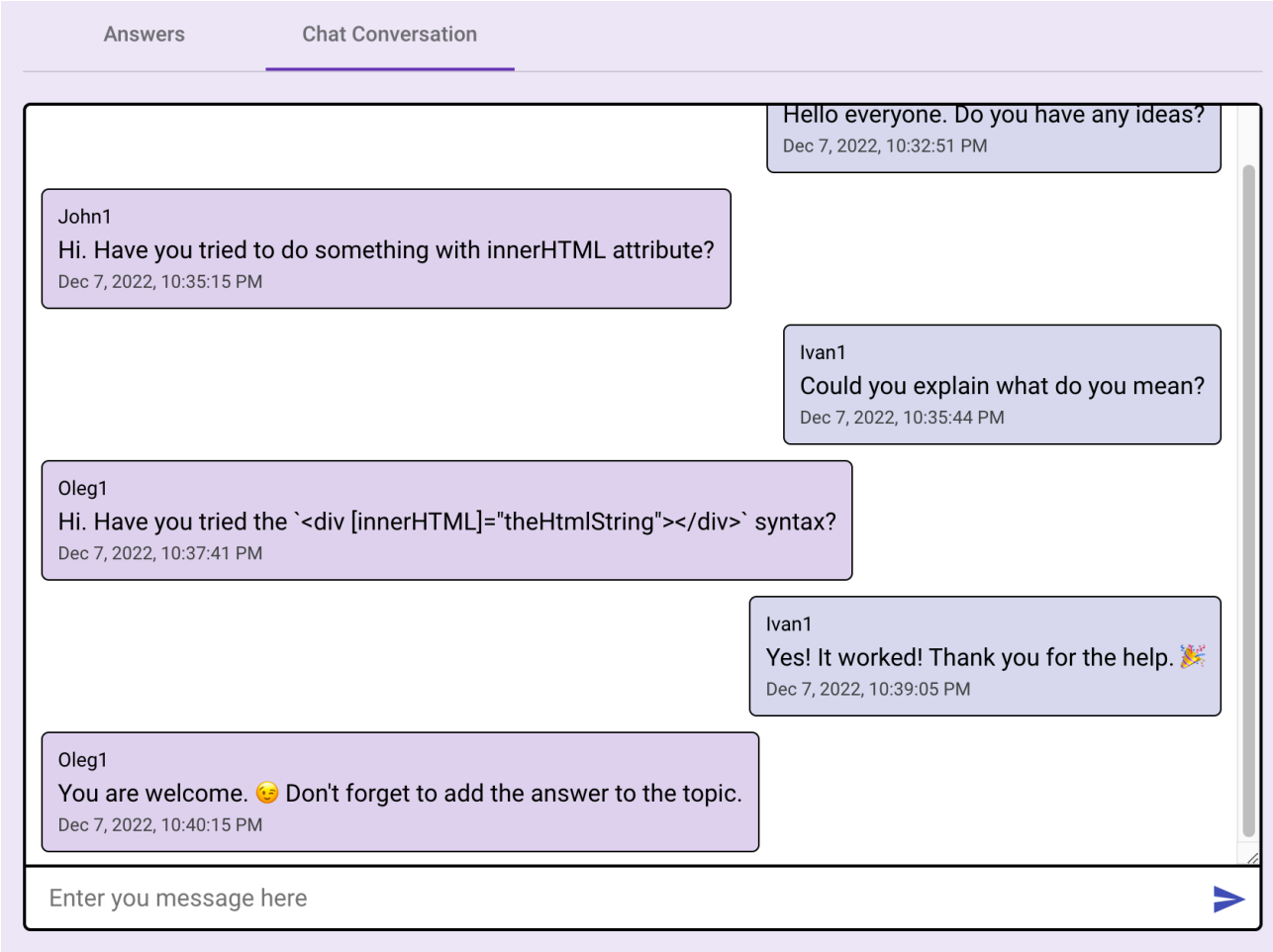

Рисунок 4.3 — Компонент чату для колективного розв’язання питання

Структура правої частини відповіді схожа до структури правої частини питання. З верху до низу у компоненті відповіді, див рисунок 4.2, зображено користувацьке ім’я автора, текст відповіді, дата написання та останнього редагування.

Автентифіковані користувачі мають змогу створювати питання та відповідних до питань за допомогою спеціальних форм. Також автори питань та відповідей мають змогу редагувати свої публікації. І лише автор питання може схвалити будь-яку 1 відповідь під своїм запитанням. Приклад відмітки такого схвалення зображено на рисунку 4.2 у лівій частині відповіді, іконка прапорця (✓) на зеленому фоні.

Вкладка чатів для дискусії, див. рисунок 4.3, надають змогу учасникам платформи проводити колективні дискусії у спробі спільно розв’язати певну задачу. Таким чином, автентифіковані користувачі можуть брати участь у дискусіях в чаті, а неавтентифіковані користувачі можуть отримати доступ до історії попередніх повідомлень чату. Повідомлення чату умовно поділені на праві та ліві. Зліва зображено повідомлення інших користувачів, справа повідомлення самого користувача, з перспективи якого розглядається чат. Структурно повідомлення складається користувацького імені автора повідомлення, тексту повідомлення та часу публікації повідомлення. Чати реалізовані з використанням бібліотеки Socket.IO та з використанням подій описаних у пункті 3.2, у частині про події для роботи чату.

Для входу та реєстрації в систему створені 2 окремих сторінки. На цих сторінках зображено форми для вводу даних користувача. Після успішного входу в систему ім’я користувача зображено у шапці (header) платформи. У нижній частині (footer) платформи вказано покликання на публічний репозиторій проекту на GitHub та особисту сторінку автора вебплатформи у GitHub та рік створення першої версії платформи.

Також звернемо увагу на те, що для забезпечення прийнятної продуктивності чатів при відображенні великої кількості повідомлень одночасно була використана спеціальна технологія віртуального прокручування (virtual

scroll). Цей інструмент, представлений набором для розробки компонентів (CDK) від бібліотеки Angular Material Design. Директива cdkScrollable і служба ScrollDispatcher разом дозволяють компонентам реагувати на прокручування в будь-якому з попередніх контейнерів прокручування. Директиву cdkScrollable слід застосовувати до будь-якого елемента, який діє як контейнер для прокручування. Це позначає елемент як Scrollable і реєструє його в ScrollDispatcher. Тоді диспетчер дозволяє компонентам ділитися як слухачами подій, так і знаннями про всі прокручувані контейнери в додатку. [21] З використанням віртуального прокручування досягається більш оптимізоване відображення компонентів. Відображаються, та ініціалізуються програмно лише ті компоненти які безпосередньо у полі зору користувача, та ті що дуже близько за областю видимості контейнеру, що може прокручуватись.

Варто відмітити що розроблена платформа є повнофункціональною початковою версією. Усі заявлені вимоги у пункті 1.2 виконані, проте є простір для розвитку та масштабування проекту. Наприклад, потенційно можна додати додаткові версії локалізації інтерфейсу вебплатформи, щоб до міжнародної мови англійської додати ще українську мову. Можна ще додатково покращувати платформу з точки зору оптимізації використання ресурсів пам’яті; масштабування проекту для роботи з великою кількістю користувачів; покращення системи пошуку; виводу програмного коду в тексі питань, відповідей платформи; додати пагінації відображення для питань, відповідей, повідомлень. Це потенційні вектори покращення, але все ж базовий функціонал з вимог описаних у пункті 1.2 повністю працює.

5. ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку веб-платформи для колективного розв’язання проблем та задач.

Особливістю програми є те, що дана технологія є вебплатформою з питаннями-відповідями з чатами для кожного питання, що вирізняє її серед аналогів, які, як правило мають подібні платформи, але без окремих чатів для кожного питання.

Аналогом може бути вебплатформа Stack Exchange, яку недавно купила Prosus за $1.8 мільярдів.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Крите-рій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах
Ринкові переваги					
2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 5.1

2	Багато аналогів на малому ринку	Ринкові п Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження табл. 5.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виро-
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій	Термін реалізації ідеї від 3-х до 5-ти років. Термін	Термін реалізації ідеї менше 3-х років. Термін окупності	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій
12	Необхідна розробка регламентних документів та отримання великої кількості до-	Необхідно отримання великої кількості дозвільних документів на виробництво та	Процедура отримання дозвільних документів для виробництва	Необхідно тільки повідомлення відповідним органам	Відсутні будь-які регламентні обмеження на виробництво та реалізацію

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	3	3
Наявність аналогів на ринку	4	4	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	2	3	3
Експлуатаційні витрати	3	4	3
Ринок збуту	2	3	3
Конкурентоспроможність	3	3	3
Фахівці з технічної і комерційної реалізації	4	3	4

Продовження табл. 5.2

Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	4	4	4
Сума	40	42	41
Середньоарифметична сума балів	(40+42+41) / 3 = 41		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 - 1 0	Низький
1 1 - 2 0	Нижче середнього
2 1 - 3 0	Середній
3 1 - 4 0	Вище середнього
4 1 - 4 8	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок того, що програмний продукт відрізняється від існуючих тим, що представляє собою веб-платформи для колективного розв’язання проблем та задач. Це платформа питання-відповідь з чатами під кожним питанням. Існують окремо платформи питання-відповідь та чати. Особливістю програми є те, що дана технологія є веб-платформою з питаннями-відповідями з чатами для кожного питання, що вирізняє її серед аналогів, які, як правило мають подібні платформи, але без окремих чатів для кожного питання.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{М}{Т_р} t, \quad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ — число робочих днів в місяці, 22 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.3.

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Таблиця 5.3 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	21500	977,27	34	33227,273
Програміст	20000	909,09	34	30909,091
Всього				64136,36

5.2.2 Додаткова заробітна плата розробників, які приймали участь в розробці обладнання

Додаткова заробітна плата прийнято розраховувати як 10 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 10\ \% / 100\ \% \quad (5.2)$$

$$З_д = (64136,36 \cdot 10\ \% / 100\ \%) = 6413,64 \text{ (грн.)}$$

5.2.3 Нарахування на заробітну плату розробників

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$$Н_з = (64136,36 + 6413,64) \cdot 22\ \% / 100\ \% = 15521,00\ (грн.)$$

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді амортизація обладнання, що використовувалась для розробки розраховується за формулою:

$$А = \frac{Ц}{Т_в} \frac{t_{вик}}{12}\ [грн.], \qquad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

$Т_в$ — термін корисного використання обладнання згідно податкового законодавства, років;

$t_{вик}$ — термін використання під час розробки, місяців.

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 47000 грн., термін його корисного використання згідно податкового законодавства — 2 роки, а термін його фактичного використання — 1,55 міс.

$$А_{обл} = \frac{47000}{2} \frac{1,55}{12} = 3026,515\ грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.4. Так як вартість одного спеціалізованого ліцензійного нематеріального ресурсу є безкоштовною (Visual

Studio Code, PostgreSQL), іншого місячна підписка складає 160 грн/міс (Digitalocean), а ресурс використовувався 1,55 місяці, то $В_{нем.ак.}$ = 248 грн.

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас).

Таблиця 5.4 — Амортизаційні відрахування матеріальних і нематеріальних ресурсів для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахування-ня, грн.
Комп'ютер та комп'ютерна периферія (Ноутбук MacBook Pro 13-inch 2020)	47000	2	1,55	3026,515
Офісне обладнення (меблі)	20000	4	1,55	643,939
Приміщення	650000	20	1,55	4185,606
Всього				7856,06

Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства, В = 6,2 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,35 кВт;

Ф — фактична кількість годин роботи обладнання, годин;

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9;

$$В_е = 0,9 \cdot 0,35 \cdot 8 \cdot 34 \cdot 6,2 = 531,216 \text{ (грн.)}.$$

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$І_в = (З_о + З_р) \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$І_в = 64136,36 * 50\% / 100\% = 32068,18 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 64136,36 * 100\ \% / 100\ \% = 64136 \text{ (грн.)}$$

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$В_{заг}$ = 64136,36+6413,64+15521,00+7856,06+248+531,22+32068,18+64136 = 190910,82 грн.

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються ЗВ, визначається за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

ЗВ = 190910,82 / 0,5 = 381822 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_o \, N + Ц_0 \, \Delta N)_i \, \lambda \, \rho \, (1 - \frac{\vartheta}{100}), \qquad (5.10)$$

де $\pm\Delta Ц_o$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

Цб — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

ρ — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2022 році ϑ =18%.

Припустимо, що при прогнозованій річній підписці в 120 грн. термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на 10 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року — на 50000 шт., протягом другого року — на 75000 шт., протягом третього року на 10000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*10 + (120 + 10)*50000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 1434999{,}943 \text{ грн.}$$

$$\Delta П_2 = (0*10 + (120 + 10)*(50000+75000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 3886458{,}178 \text{ грн.}$$

$$\Delta П_3 = (0*10 + (120 + 10)*(50000+75000+10000)* 0{,}8333* 0{,}35) * (1 - 0{,}18) = 4197374{,}832 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 9518832,95 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_1^T \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.11)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0{,}05…0{,}15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо починаючи з першого року:

$$ПП = (1434999{,}943/(1+0{,}1)^1)+(3886458{,}178/(1+0{,}1)^2)+(4197374{,}832/(1+0{,}1)^3)$$
$$= 1304545{,}40 + 3211948{,}907 + 3153549{,}836 = 7670044{,}146 \text{ грн.}$$

Далі розраховують величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.12)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 381822 = 763643{,}29 \text{ грн.}$$

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$Е_{абс} = ПП - PV, \tag{5.13}$$

$$Е_{абс} = 7670044{,}146 - 763643{,}29 = 6906400{,}86 \text{ грн.}$$

Оскільки $Е_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1, \tag{5.14}$$

де $T_{ж}$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{1 + \frac{6906400{,}86}{763643{,}29}} - 1 = 1{,}158$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f \tag{5.15}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{min} = 0,14 + 0,05 = 0,19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} \quad (5.16)$$

$$T_{ок} = 1 / 1,158 = 0,86 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,86 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 381822 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,86 роки.

ВИСНОВКИ

В рамках магістерської дипломної роботи досягнуто мети та задач дослідження. У результаті роботи було розроблено вебплатформу для колективного розв’язання проблем та задач, що враховує недоліки рішення платформ-аналогів.

У першому розділі було проведено аналітичний огляд наявних на ринку інструментів для колективного розв’язання задач. Виявлено основні переваги та недоліки таких інструментів. Сформовано функціональні та не функціональні вимоги до технічного рішення, яке враховує вади аналогів.

У другому розділі було проаналізовано основні технології необхідні для реалізації такої вебплатформи. В результаті огляду та порівняння технологій було обґрунтовано вибір технологій для розробки. Для клієнтської частини для розробки було обрано Angualr, Angualr Material. Для серверної частини було обрано NestJs, TypeORM, PostgreSQL, Socket.IO. Також, було сплановано структуру реляційної бази даних. Та було розраховано оптимальну кількість розв’язання задач на одиницю часу за теорією масового обслуговування.

У третьому розділі було створено та налаштовано середовище розробки у в вигляді монорепозиторію, згенерованого за допомогою Nx. Також, було описано структуру серверної частини застосунку. Описано як модулі серверної частини взаємодіють між собою, як влаштована автентифікація через JWT, як працюють обробка HTTP запитів, як влаштований REST API застосунку.

У четвертому розділі створено клієнтську частину застосунку. Описано структуру моделей застосунку, роботу перехоплювачів HTTP запитів, принцип роботи JWT автентифікації з клієнтської перспективи, вигляд та принцип роботи компонентів застосунку та як вони взаємодіють один з одним.

У рамках розділу економічної частини було виконані економічні розрахунки із обґрунтування доцільності реалізації запропонованої вебплатформи.

Створена платформа має потенціал для покращення та додавання нових функцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Frensch, Peter A.; Funke, Joachim, eds. (2014-04-04) Complex Problem Solving [Електронний ресурс] — режим доступу: https://doi.org/10.4324%2F9781315806723

2. Stack Exchange Official Website [Електронний ресурс] — режим доступу: https://stackexchange.com/

3. Discord Official Website [Електронний ресурс] — режим доступу: https://discord.com/

4. Slack Official Website [Електронний ресурс] — режим доступу: https://slack.com/

5. Telegram Official Website [Електронний ресурс] — режим доступу: https://telegram.org/

6. Rothwell R. Robertson A. The role of communications in technological innovation [Електронний ресурс] — режим доступу: https://doi.org/10.1016/0048-7333(73)90003-6

7. Frontend and backend [Електронний ресурс] — режим доступу: https://en.wikipedia.org/wiki/Frontend_and_backend

8. Front-end web development [Електронний ресурс] — режим доступу: https://en.wikipedia.org/wiki/Front-end_web_development

9. Uniform Resource Locator [Електронний ресурс] — режим доступу: https://en.wikipedia.org/wiki/URL

10. URL and DNS [Електронний ресурс] — режим доступу: https://dev.to/namitmalasi/url-and-dns-explained-4aef

11. An overview of HTTP [Електронний ресурс] — режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

12. StackOverflow survey 2021 [Електронний ресурс] — режим доступу: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-databases

13. Teknomo, K. (2012). Queuing Rule of Thumb based on M/M/s Queuing Theory with Applications in Construction Management. Civil Engineering Dimension,

14(3), 139-146. [Електронний ресурс] — режим доступу: https://doi.org/10.9744/ced.14.3.139-146

14. Database design basics [Електронний ресурс] — режим доступу: https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5

15. PostgreSQL Documentation [Електронний ресурс] — режим доступу: https://www.postgresql.org/docs/current/datatype.html

16. Object–relational mapping [Електронний ресурс] — режим доступу: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping

17. TypeORM Documentation [Електронний ресурс] — режим доступу: https://typeorm.io/

18. JWT Documentation [Електронний ресурс] — режим доступу: https://jwt.io/

19. Angular Documentation [Електронний ресурс] — режим доступу: https://angular.io/guide/architecture-modules

20. NestJS Documentation [Електронний ресурс] — режим доступу: https://docs.nestjs.com/#philosophy

21. Angular Material CDK Documentation [Електронний ресурс] — режим доступу: https://material.angular.io/cdk/scrolling/overview

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д.

_____

"____" _____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

“Вебплатформа для колективного розв’язання проблем та задач”

08-23.МКР.009.00.000.ТЗ

Науковий керівник: к.т.н., доц. проф.

_____ Захарченко С.М.

Студент групи 1КІ-21м

_____ Майданюк І.Г.

ВНТУ 2022

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 У часи карантинних обмежень та в умовах прямої військової загрози, необхідно мати можливість проводити групові види діяльності віддалено. Зокрема, колективне вирішення задач також може відбуватись через мережу, без прямої необхідності групам людей збиратись разом.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Мета роботи — вдосконалення системи колективного розв’язання проблем та задач.

2.2 Призначення розробки — розробка вдосконаленої вебплатформи для розв’язання проблем та задач, яка враховує недоліки аналогів.

3 Вихідні дані для виконання МКР

Аналітичний огляд існуючих платформ для колективного розв’язання задач. Аналіз технологій реалізації. Розроблена серверна та клієнтська частини вебплатформи для колективного розв’язання проблем та задач

4 Вимоги до виконання МКР

МКР повинна задовольняти такі вимоги:

— здійснити аналітичний огляд існуючих платформ для колективного розв’язання проблем та задач;

— запропонувати покращену версію платформи для колективного розв’язання проблем та задач, яка враховує вади аналогів;

— проаналізувати та обрати необхідні технології для реалізації запропонованого рішення;

— розрахувати оптимальну кількість розв’язувань питань на одиницю часу;

— реалізація клієнтської та серверної частини застосунку.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналітичний огляд проблематики наявних платформ для колективного розв'язання проблем та задач, формування вимог до розробки покращеної вебплатформи, що враховує недоліки аналогів	24.09.22	14.10.22	Розділ 1
2	Дослідження технологій для реалізації вебплатформи. Планування реалізації. Розрахунок оптимальної кількості розв'язувань питань на одиницю часу	14.10.22	21.10.22	Розділ 2
3	Налаштування монорепозиторію та реалізація серверної частини	21.10.22	28.10.22	Розділ 3
4	Реалізація клієнтської частини	28.10.22	11.11.22	Розділ 4
	Підготовка економічної частини	11.11.22	25.11.22	Розділ 5
5	Апробація та впровадження результатів дослідження	25.11.22	30.11.22	Тези доповідей
6	Опублікування результатів досліджень	30.11.22	12.12.22	Стаття
7	Оформлення пояснювальної записки, графічного матеріалу і презентації	12.12.22	19.12.22	Пояснювальна записка, графічний матеріал і презентація
8	Підготовка супроводжуючих документів, їх підписування, проходження нормоконтролю та тесту на плагіат	12.12.22	19.12.22	Оформлені документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до

МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— методичні вказівки. Кафедра обчислювальної техніки 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

Вигляд сторінок автентифікації вебплатформи

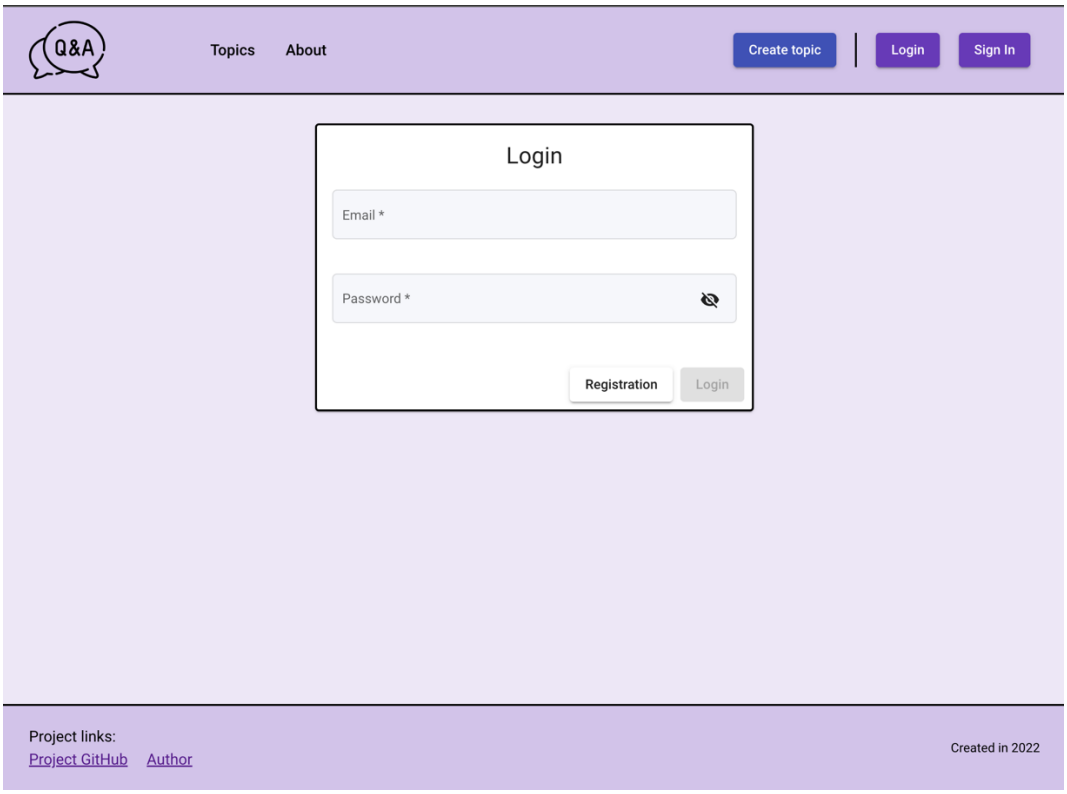

Рисунок Б.1 — Сторінка входу в систему

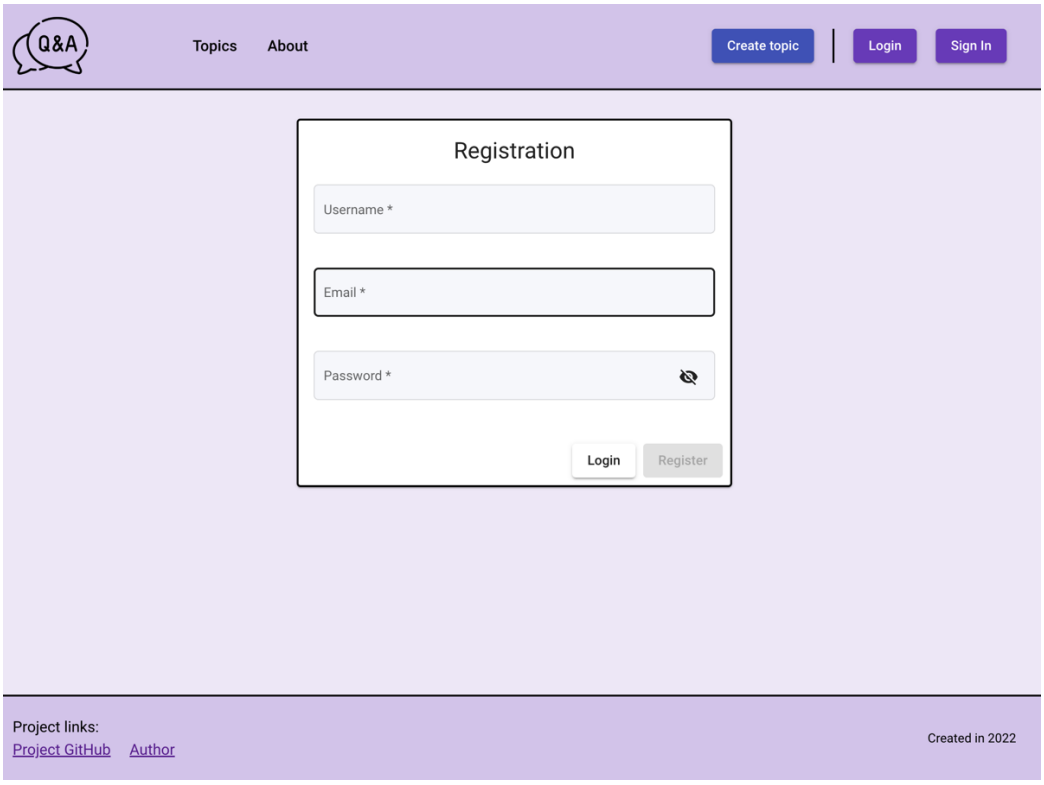

Рисунок Б.2 — Сторінка реєстрації в систему

ДОДАТОК В

Вигляд головної сторінки входу в систему

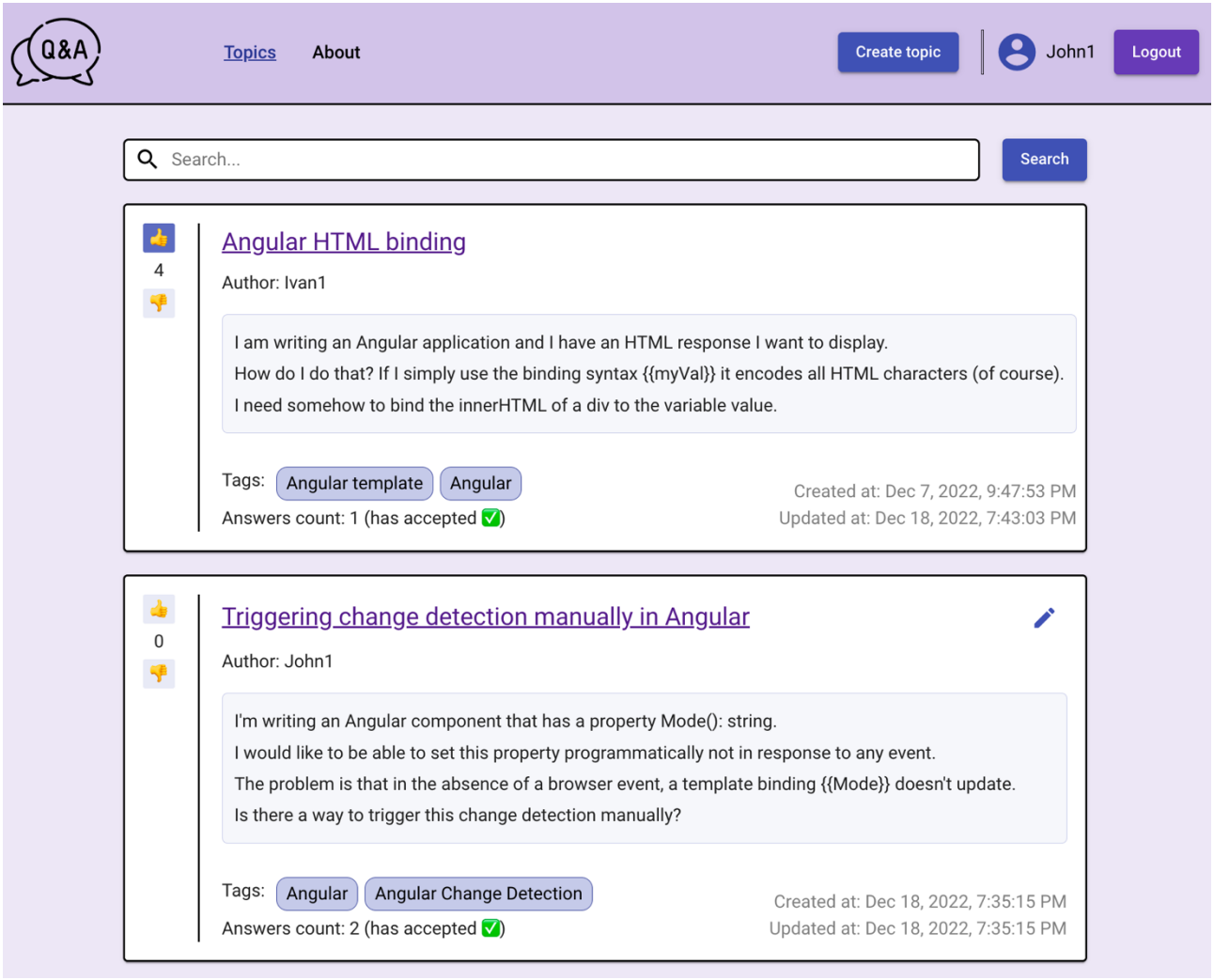

Рисунок В.1 — Головна сторінка платформи. Вигляд з сторони автентифікованого користувача з користувацьким іменем John1

ДОДАТОК Г

Вигляд сторінки теми тля обговорення з активною вкладкою відповідей

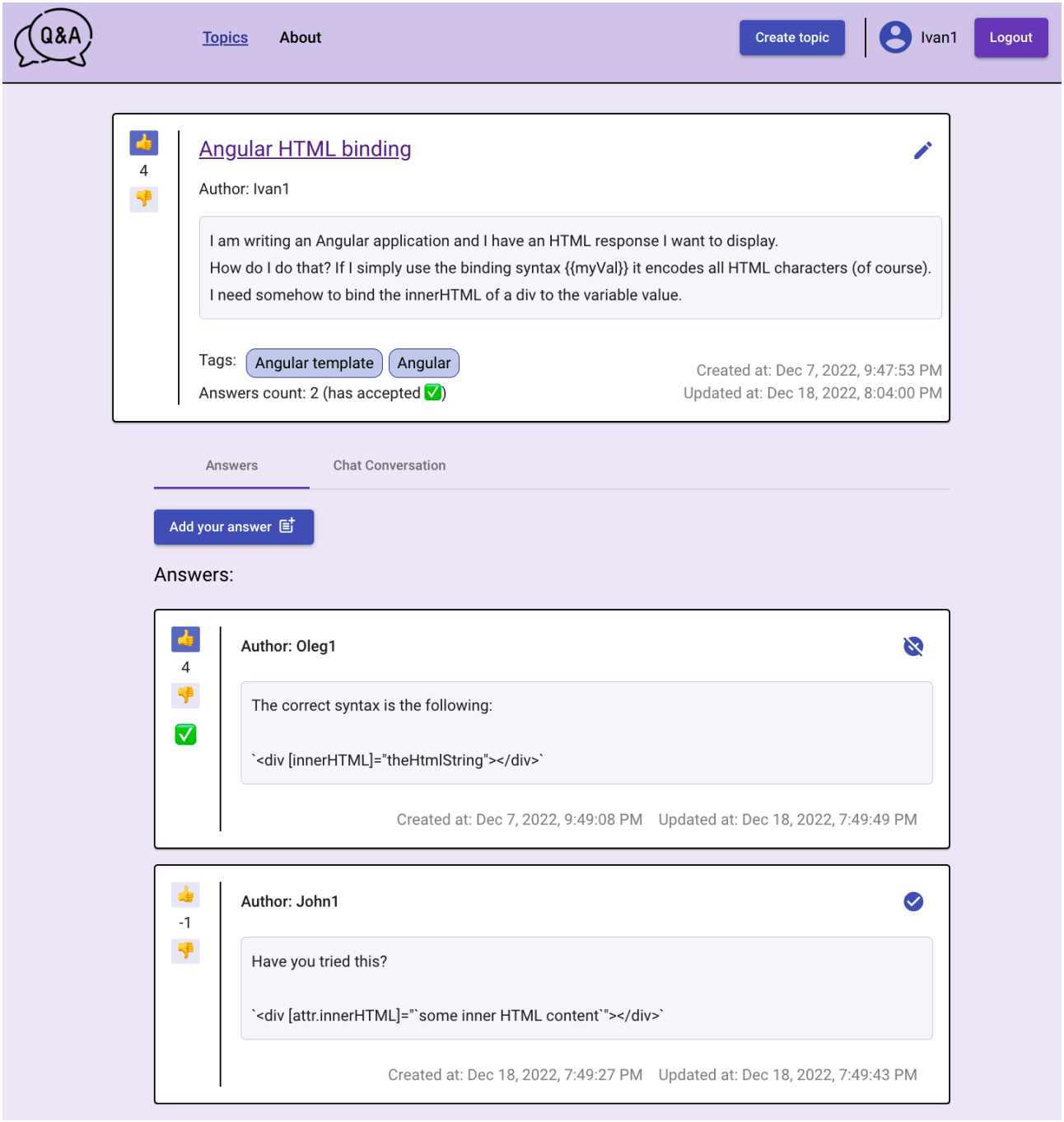

Рисунок Г.1 — Сторінка теми обговорення з активною вкладкою відповіді, що містить тему обговорення та 2 відповіді

ДОДАТОК Д

Вигляд сторінки теми тля обговорення з активованою вкладкою чату

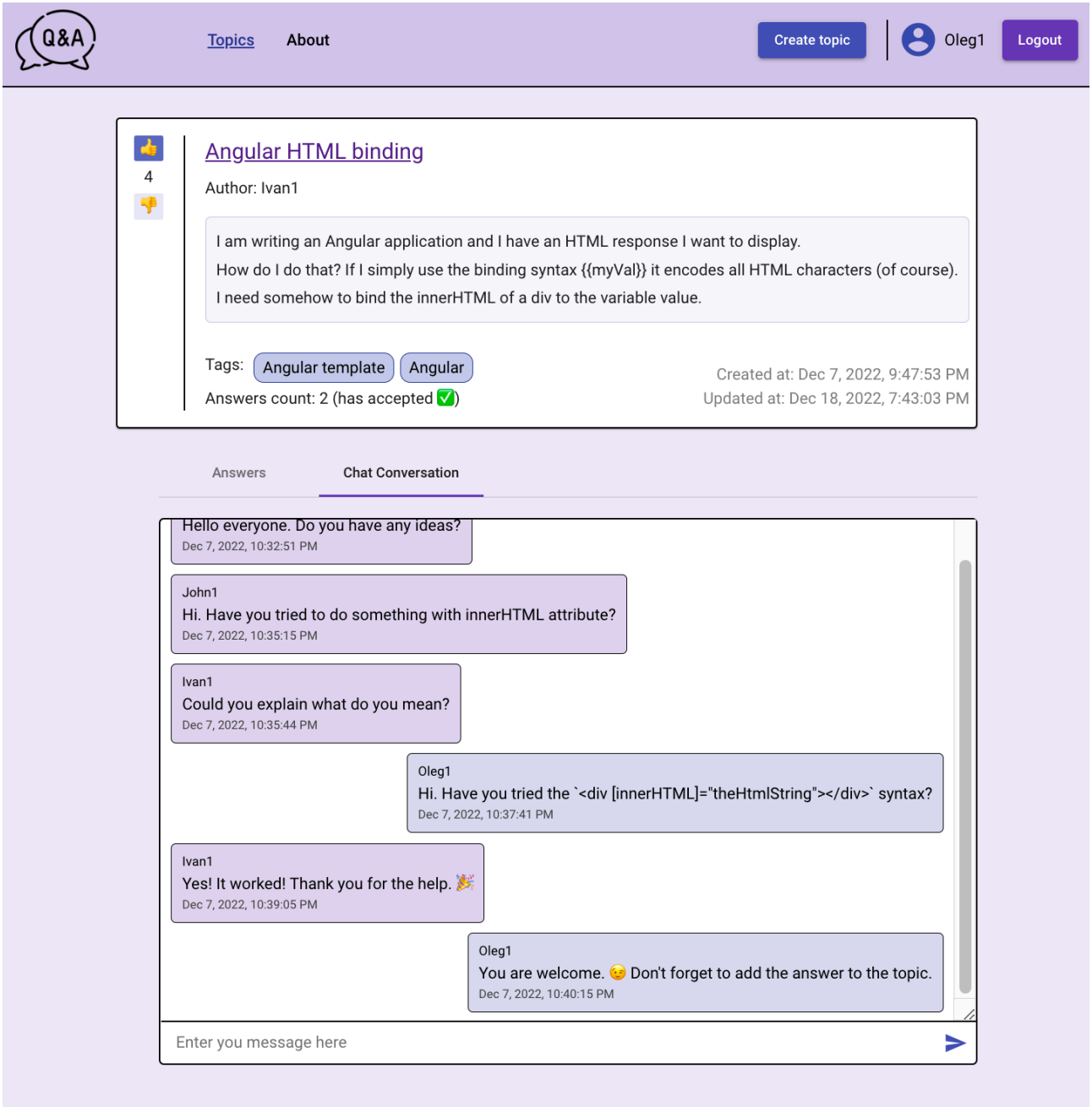

Рисунок Д.1 — Сторінка теми обговорення з активною вкладкою чату, що містить тему обговорення та чат для колективного розв’язання задачі

ДОДАТОК Е

Лістинг файлу package.json

```
{
  "name": "question-answer",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "nx",
    "start:client": "yarn nx serve client",
    "start:server": "yarn nx serve server",
    "build:client": "yarn nx run client:build:production",
    "build:server": "yarn nx run server:build:production",
    "postinstall": "node ./decorate-angular-cli.js && ngcc --properties es2020 browser module main",
    "typeorm": "cross-env NODE_ENV=production ts-node -r tsconfig-paths/register --project ./tsconfig.base.json -O '{\"module\":\"commonjs\"}' ./node_modules/typeorm/cli.js --config ./apps/server/src/typeorm/ormconfig.ts",
    "db:drop": "yarn typeorm schema:drop",
    "db:create": "yarn typeorm migration:generate",
    "db:migrate": "yarn typeorm migration:run",
    "db:seed": "cross-env NODE_ENV=production ts-node -r tsconfig-paths/register --project ./tsconfig.base.json -O '{\"module\":\"commonjs\"}' ./node_modules/typeorm/cli.js --config ./apps/server/src/typeorm/ormseedconfig.ts migration:run"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "14.1.3",
    "@angular/cdk": "14.1.3",
    "@angular/common": "14.1.3",
```

"@angular/compiler": "14.1.3",
"@angular/core": "14.1.3",
"@angular/forms": "14.1.3",
"@angular/material": "14.1.3",
"@angular/platform-browser": "14.1.3",
"@angular/platform-browser-dynamic": "14.1.3",
"@angular/router": "14.1.3",
"@nestjs/common": "9.0.11",
"@nestjs/core": "9.0.11",
"@nestjs/platform-express": "9.0.11",
"@nestjs/platform-socket.io": "^9.1.6",
"@nestjs/swagger": "6.1.0",
"@nestjs/typeorm": "^8.0.3",
"@nestjs/websockets": "^9.1.6",
"@nrwl/angular": "14.5.10",
"class-transformer": "^0.5.1",
"class-validator": "^0.13.2",
"cross-env": "^7.0.3",
"dotenv": "^16.0.0",
"express-session": "^1.17.3",
"jsonwebtoken": "^8.5.1",
"ngx-socket-io": "^4.3.1",
"pg": "^8.7.3",
"reflect-metadata": "^0.1.13",
"rxjs": "7.5.6",
"slugify": "^1.6.5",
"socket.io": "^4.5.3",
"swagger-ui-express": "^4.3.0",
"ts-node": "10.9.1",
"tslib": "^2.0.0",

```
  "typeorm": "^0.2.42",
  "zone.js": "~0.11.4"
 },
 "devDependencies": {
  "@angular-devkit/build-angular": "14.1.3",
  "@angular-eslint/eslint-plugin": "14.0.3",
  "@angular-eslint/eslint-plugin-template": "14.0.3",
  "@angular-eslint/template-parser": "14.0.3",
  "@angular/cli": "~14.1.0",
  "@angular/compiler-cli": "14.1.3",
  "@angular/language-service": "14.1.3",
  "@nestjs/schematics": "9.0.1",
  "@nestjs/testing": "9.0.11",
  "@nrwl/cli": "14.5.10",
  "@nrwl/cypress": "14.5.10",
  "@nrwl/eslint-plugin-nx": "14.5.10",
  "@nrwl/jest": "14.5.10",
  "@nrwl/linter": "14.5.10",
  "@nrwl/nest": "14.5.10",
  "@nrwl/node": "14.5.10",
  "@nrwl/nx-cloud": "14.6.0",
  "@nrwl/workspace": "14.5.10",
  "@types/express-session": "^1.17.4",
  "@types/jest": "27.4.1",
  "@types/node": "18.7.1",
  "@typescript-eslint/eslint-plugin": "5.35.1",
  "@typescript-eslint/parser": "5.35.1",
  "cypress": "^9.1.0",
  "eslint": "8.15.0",
  "eslint-config-prettier": "8.1.0",
```

```
    "eslint-plugin-cypress": "^2.10.3",
    "jest": "27.5.1",
    "jest-preset-angular": "11.1.2",
    "nx": "14.5.10",
    "prettier": "2.7.1",
    "ts-jest": "27.1.4",
    "typescript": "4.7.4"
  }
}
```

ДОДАТОК Ж

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Вебплатформа для колективного розв'язання проблем та задач

Тип роботи: магістерська кваліфікаційна робота

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет)

Показники звіту подібності
Unicheck

Оригінальність 98.1% Схожість 1.9%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.

(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Майданюк І.Г.

(підпис) (прізвище, ініціали)

Керівник роботи _____ Захарченко С.М.

(підпис) (прізвище, ініціали)