Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: Розроблення платформи для онлайн-співпраці в командних проєктах з використанням ASP.NET та Angular

Виконав: студент 4 курсу групи КН-41
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Кий В. В.
(прізвище та ініціали)

Керівник Сало М. Ф.
(прізвище та ініціали)

Керівник Яцишин С. І.
(прізвище та ініціали)

Рецензент Флуд Л. О.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Борецька І. Б.
"10" червня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Кию Владиславу Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення платформи для онлайн-співпраці в командних проєктах з використанням ASP.NET та Angular
керівник роботи *Сало Микола Федорович, старший викладач*,
керівник роботи *Яцишин Світлана Іванівна, к.т.н., доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *15.11.2024 року № С-882*
2. Термін подання студентом роботи *10.06.25р.*
3. Вихідні дані до роботи Технічне завдання для розробки платформи для онлайн-співпраці в командних проектах
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1) *Стан проблемної області*
2) *Інформаційне* та математичне забезпечення
3) *Програмне та технічне забезпечення*
4) *Висновки*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до відповіді.

6. Дата видачі завдання *18.11.2024р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд наукових публікацій та джерел	19.11.24 – 20.02.25	Виконано
2	Вивчення проблеми та формулювання завдань дослідження	20.02.25 – 02.03.25	Виконано
3	Створення інформаційної та математичної основи системи	02.03.25 – 12.03.25	Виконано
4	Програмна реалізація проекту	12.03.25 – 28.04.25	Виконано
5	Забезпечення стабільності програмного функціоналу	28.04.25 – 16.05.25	Виконано
6	Складання пояснювальної записки	16.05.25 – 08.05.25	Виконано

Студент _____ **Кий В. В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Сало М. Ф.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Яцишин С. І.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота містить 62 сторінки, 28 рисунків, 4 додатки, 17 джерел.

Дана робота присвячена розробці платформи для онлайн-співпраці в командних IT-проектах. Головною метою роботи є створення веб-застосунку, яка забезпечить взаємодію між розробниками та менеджерами, спростить процес управління проектами та сприятиме отриманню практичного та комерційного досвіду молодим спеціалістам. Для реалізації платформи використано такі сучасні технології, як ASP.NET для розробки серверної частини та Angular для створення інтерфейсу користувача. Результати роботи можуть бути використані для створення комерційних сервісів або платформ, спрямованих на інтеграцію молодих спеціалістів, та пошуку першої роботи та досвіду в командах однодумців.

Ключові слова: управління командними проектами, Angular, ASP.NET, REST API, Layered (слоїста) архітектура, SPA, SQL, Ngrok.

ABSTRACT

Бакалаврська дипломна робота містить 62 pages, 28 illustrations, 4 appendices and 17 sources

This work is dedicated to the development of a platform for online collaboration in team IT projects. The main goal of the project is to create a web application that facilitates interaction between developers and managers, simplifies project management, and provides young specialists with opportunities to gain practical and commercial experience. Modern technologies were used for the platform's implementation, including ASP.NET for the server-side development and Angular for the user interface design. The results of this work can be utilized for creating commercial services or platforms aimed at integrating young specialists into professional teams, helping them gain their first work experience and collaborate with like-minded individuals.

Keywords: team project management, Angular, ASP.NET, REST API, single-page application (SPA), SQL, Ngrok.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити платформу для онлайн співпраці в командних проектах. Під час розробки використати: мову програмування C#, використовуючи технологію ASP.NET від компанії Microsoft, фреймворк ASP.NET Identity для керування автентифікацією та авторизацією користувачів, ORM-технологію Entity Framework для доступу до даних, Ngrok для створення локальних портів, авторизація з використанням JWT токенів, клієнтська частина з використанням Angular. Інформаційна система має забезпечувати виконання таких основних функцій:

Реалізація механізмів реєстрації та авторизації користувачів. Забезпечити функціонал надання користувачам можливості створювати облікові записи в системі та виконувати безпечний вхід із використанням надійних методів захисту паролів.

Реалізація механізму вибору ролей. Забезпечити функціонал вибору ролі користувачам платформи та розділити функціонал відповідно до їх прав.

Забезпечити розробку функціоналу для створення проектів. Надання користувачам з особливою роллю можливості створювати проекти. Під час створення задати основні вимоги, щодо нього.

Забезпечити реалізацію повного управління проекту. Створення та керування команди, керування часовими рамками та завданнями під час роботи над продуктом.

Впровадження інтерактивної дошки. Спроектувати інтерактивну дошку для проекту, де розробники можуть створювати завдання в ході роботи, а менеджер отримувати звіти виконаного.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ASP.NET – платформа для розробки веб-застосунків від компанії Microsoft.

Angular – фреймворк для створення односторінкових веб-додатків, заснований на TypeScript.

EF – об’єктно-реляційний ORM-фреймворк для роботи з базами даних у середовищі .NET.

SQL – мова структурованих запитів, що використовується для управління реляційними базами даних.

UI – користувацький інтерфейс.

UX – досвід користувача під час взаємодії з інтерфейсом або системою.

API – інтерфейс прикладного програмування, що забезпечує взаємодію між програмними компонентами.

HTTP – протокол передачі гіпертексту, що використовується для обміну даними у веб-застосунках.

JSON – текстовий формат для обміну структурованими даними.

БД – база даних.

Kanban – методологія візуального управління завданнями

ВСТУП

Сучасний IT-ринок, попри стрімке зростання попиту, зіштовхується з парадоксом: з одного боку, авторитетні компанії постійно шукають кваліфікованих спеціалістів, з іншого – молодим фахівцям складно отримати перший комерційний досвід. Випускники навчальних закладів, курсів або самоуки нерідко стикаються з відмовами від роботодавців через нестачу практичних навичок. Така ситуація зумовлена високою конкуренцією та перенасиченістю ринку кадрами.

Актуальність проблеми зумовлена стрімким розвитком IT-сфери, зростаючим попитом на кваліфікованих фахівців та одночасними труднощами працевлаштування молодих спеціалістів, які не мають достатнього практичного досвіду, що ускладнює їх інтеграцію в професійне середовище на фоні високої конкуренції на ринку праці.

Об'єктом дослідження є процес професійної адаптації та працевлаштування молодих IT-спеціалістів в умовах сучасного ринку праці.

Предметом дослідження є умови, чинники та інструменти, що впливають на успішне працевлаштування та набуття практичного досвіду молодими фахівцями в IT-сфері.

Метою роботи є розробка веб-платформи для онлайн-співпраці в командних IT-проектах, яка сприятиме ефективній взаємодії між учасниками, забезпечить можливості для набуття практичного досвіду молодими спеціалістами та покращить процес організації спільної роботи в умовах сучасного IT-ринку.

Новизна роботи полягає у створенні інтерактивної платформи, яка поєднує можливості онлайн-співпраці, управління командними IT-проектами та розвитку професійних навичок молодих спеціалістів. Відмінністю розробленої системи є фокус на реальній взаємодії між початківцями та досвідченими фахівцями через спільну роботу над проектами. Платформа також передбачає механізми оцінювання внеску учасників і формування цифрового портфоліо, що сприяє підвищенню їх конкурентоспроможності на ринку праці.

Практична значимість роботи полягає в тому, що розроблена платформа може бути використана як інструмент для залучення молодих IT-спеціалістів до реальних

командних проектів, що дозволяє їм здобувати практичний досвід у наближених до комерційних умовах. Вона також надає можливість проектним менеджерам ефективно організовувати співпрацю, розподіляти завдання та контролювати хід виконання проекту. Застосування платформи сприятиме зменшенню розриву між теоретичними знаннями та практичними навичками, що є актуальним завданням сучасної ІТ-освіти.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Актуальність платформи для онлайн-співпраці.

Можливість набуття практичного досвіду для молодих спеціалістів є однією з найбільших проблем на сучасному ринку праці, зокрема в IT-галузі. Більшість молодих фахівців, які не мають досвіду роботи в реальних проектах, часто стикаються з труднощами при пошуку першого місця роботи. Це пов'язано з тим, що роботодавці вимагають від кандидатів не тільки теоретичних знань, а й практичних навичок, здобутих у процесі роботи над реальними завданнями. Зазвичай, молоді спеціалісти отримують відмови через відсутність практичного досвіду, навіть якщо вони мають сильну теоретичну підготовку.

Це веде до парадоксальної ситуації: на ринку праці є велика кількість молодих фахівців, які готові працювати, але не можуть здобути досвід, необхідний для влаштування на повноцінну роботу. Платформи для онлайн-співпраці, які забезпечують участь у реальних командних проектах, є ідеальним рішенням цієї проблеми. Вони дають можливість молодим спеціалістам здобувати цінний досвід у розв'язанні реальних задач, працюючи з досвідченими колегами, що, в свою чергу, покращує їх шанси на успішне працевлаштування. Такі платформи сприяють розвитку професійних навичок, створюючи сприятливі умови для кар'єрного зростання та інтеграції в професійну спільноту.

Підтримка формування професійних зв’язків та розвитку кар’єри є однією з ключових складових успіху в сучасному IT-середовищі. В умовах високої конкуренції на ринку праці, наявність міцних професійних контактів значно підвищує шанси на успішне працевлаштування та кар’єрне зростання. Для молодих спеціалістів, які тільки починають свій професійний шлях, створення та підтримка таких зв’язків є важливим фактором у досягненні кар'єрних цілей.

Ці зв’язки можуть стати основою для подальшого кар'єрного зростання, оскільки вони забезпечують доступ до нових можливостей на ринку праці, створюють умови для участі в більш складних проектах і надають рекомендації, які можуть значно вплинути на вибір роботодавця. Платформи також можуть включати механізми

для створення професійних портфоліо, що дозволяють учасникам продемонструвати свої досягнення, покращуючи таким чином їх видимість серед потенційних роботодавців. У підсумку, такі платформи не лише допомагають молодим спеціалістам здобути досвід, а й активно сприяють розвитку їхніх професійних зв'язків і кар'єрних перспектив.

1.2. Переваги веб-платформи для онлайн-співпраці.

Використання веб-застосунку для онлайн-співпраці стає досить актуальним в нас час, надаючи чудові умови роботи для віддалених команд. Ці вебзастосунки мають низку переваг, котрі відіграють важливу роль у співпраці між користувачами. Ось які є плюси використання веб-платформи для співпраці над спільними проектами:

- **Доступність і зручність**: Веб-платформи дозволяють користувачам працювати з будь-якого пристрою, підключеного до інтернету, без необхідності встановлення додаткового програмного забезпечення. Це особливо важливо для віддалених команд або фрілансерів, які працюють з різних локацій.
- **Спільна робота в реальному часі**: Веб-застосунки підтримують можливість одночасної роботи над документами, проектами та завданнями. Це забезпечує миттєву взаємодію між членами команди та дозволяє оперативно вносити зміни, що полегшує процес співпраці.
- **Гнучкість і масштабованість**: Веб-платформи для онлайн-співпраці можуть бути легко адаптовані до різних потреб команди та проекту. Вони дозволяють налаштовувати доступи, організовувати робочі простори, а також масштабувати інструменти в міру росту проекту.
- **Підвищення продуктивності**: Завдяки зручній організації роботи, автоматизації рутинних процесів та можливості оперативно спілкуватися в рамках проекту, веб-платформи допомагають значно підвищити продуктивність команди. Всі учасники можуть стежити за хід роботи, виконувати завдання, отримувати оновлення та відповідати на запитання в реальному часі.
- **Зниження витрат**: Використання веб-платформ знижує потребу в традиційному офісному обладнанні, оскільки вся робота виконується онлайн. Це

дозволяє компаніям зменшити витрати на фізичні ресурси та оптимізувати витрати на інфраструктуру.

Підсумовуючи, можна зазначити, що використання веб-платформ для онлайн-співпраці створює сприятливі умови для продуктивної роботи команд, оскільки такі рішення значно спрощують організацію робочих процесів і комунікацію між учасниками. Завдяки централізованому зберіганню даних, команди можуть легко відстежувати прогрес проектів, контролювати виконання завдань та оперативно реагувати на зміни. Це особливо важливо для великих проектів, де синхронізація між різними підрозділами та віддаленими командами відіграє ключову роль. Такі платформи дозволяють оптимізувати процеси управління, забезпечуючи прозорість роботи та чітке розподілення ролей і обов’язків. Це зменшує ризик виникнення непорозумінь, дублювання завдань та втрати важливої інформації. Інтеграція з іншими інструментами, такими як трекери завдань, системи зберігання документів та платформи для відеоконференцій, дозволяє централізувати всю роботу в єдиній екосистемі, що значно знижує час на перемикання між додатками.

1.3. Огляд та аналіз готових рішень.

1. Microsoft Teams

Microsoft Teams – це корпоративна платформа для спільної роботи, розроблена компанією Microsoft. Вона є частиною пакету Microsoft 365 і об'єднує функції обміну повідомленнями, відеоконференцій, спільного доступу до файлів та інтеграції з іншими сервісами.

Однією з основних переваг Microsoft Teams є його здатність централізувати робоче середовище, об'єднуючи різні інструменти для спільної роботи в одному місці. Це дозволяє командам ефективно комунікувати, обмінюватися файлами, проводити відеозустрічі, планувати завдання та спільно працювати над документами без необхідності перемикатися між різними застосунками. Така інтеграція значно скорочує час, витрачений на організаційні питання, і підвищує продуктивність, оскільки всі необхідні інструменти доступні в єдиному інтерфейсі.

Другою ключовою перевагою є гнучкість та масштабованість платформи, що дозволяє ефективно використовувати її як для малих команд, так і для великих корпорацій. Microsoft Teams підтримує різні формати спільної роботи – від простих чатів до комплексних проектних просторів із розподілом ролей і доступів (рис. 1.1). Це робить платформу ідеальним рішенням як для стартапів, так і для глобальних організацій, що працюють у гібридному форматі, де співробітники можуть перебувати в різних часових зонах і навіть країнах.

Одним помітним недоліком Microsoft Teams є обмежений функціонал для керування проектами порівняно з платформами, які спеціалізуються саме на цьому, такими як Asana, Trello або Jira [1]. Наприклад, Teams не має вбудованих інструментів для створення Gantt-діаграм, детального відстеження завантаження ресурсів чи автоматизації складних робочих процесів. Це може бути проблемою для команд, які потребують більш детальної організації проектів, глибокої аналітики або складних залежностей між завданнями. Хоча Teams можна інтегрувати з іншими інструментами для компенсації цього недоліку, це часто вимагає додаткових налаштувань і підвищує загальну складність використання платформи.

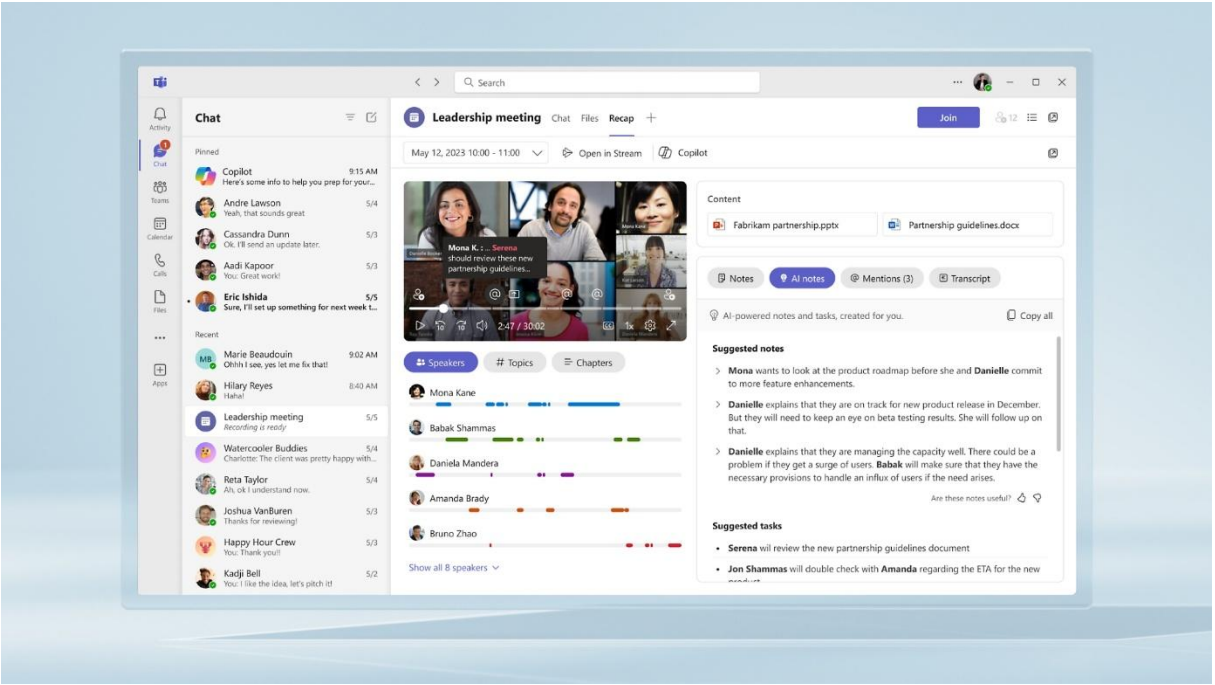


Рисунок 1.1 – Інтерфейс Microsoft Teams

2. Asana

Asana – це хмарна платформа для управління проектами та командною співпрацею, яка допомагає організовувати, відстежувати та координувати роботу команд будь-якого масштабу. Заснована у 2008 році Дастіном Московіцем та Джастіном Розенштейном, Asana [2] стала популярним інструментом для компаній, що прагнуть підвищити прозорість процесів і ефективність виконання завдань.

Користувачі можуть створювати проекти, розбивати їх на завдання та підзавдання, встановлювати дедлайни, призначати відповідальних осіб та відстежувати прогрес у реальному часі (рис. 1.2). Це дозволяє командам зберігати прозорість у виконанні завдань, зменшує ризик пропуску важливих етапів і забезпечує чітке розуміння поточного стану проекту. За допомогою функції Workflow Builder користувачі можуть створювати автоматизовані правила для рутинних завдань, таких як автоматичне призначення завдань або надсилання нагадувань.

Однак слід зазначити, що для управління великими або складними проектами може не вистачати деяких функцій, таких як розширене управління ресурсами або бюджетування. Також для невеликих команд або простих проектів функціональність Asana може виявитися надмірною, що ускладнює її використання.

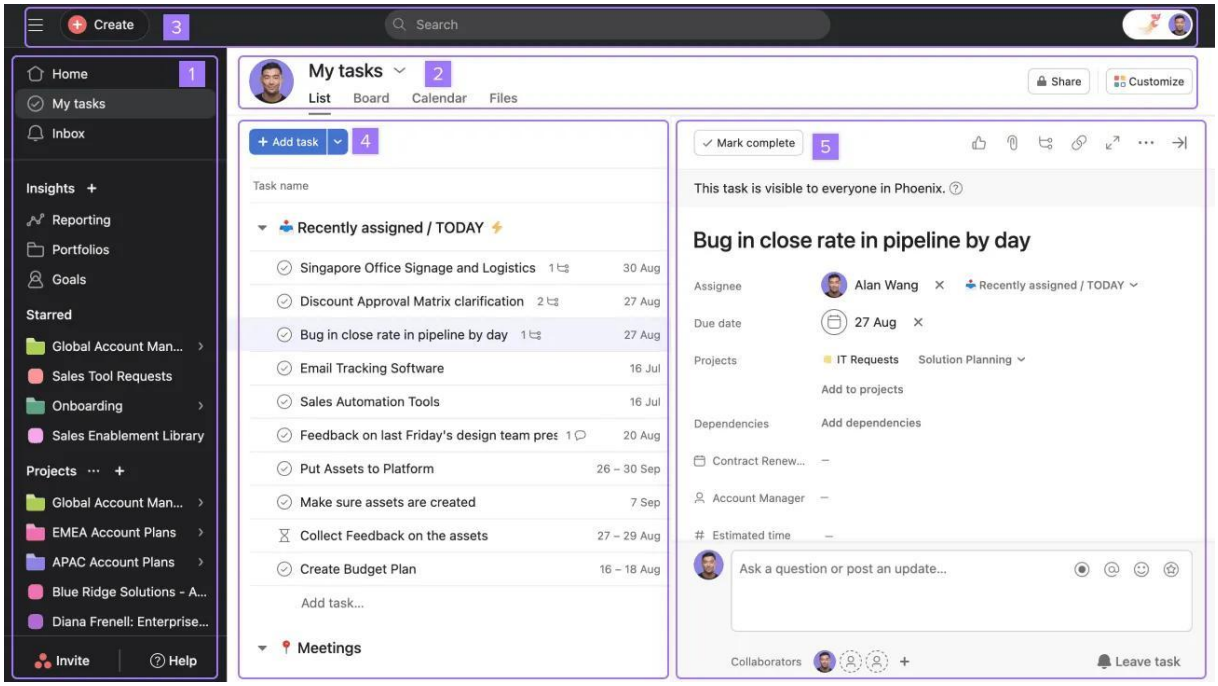


Рисунок 1.2 – Інтерфейс Asana

Отже, платформи, такі як Microsoft Teams та Asana, демонструють високу ефективність у забезпеченні командної співпраці та управлінні проектами. Вони пропонують широкий набір функцій для комунікації, організації завдань та відстеження прогресу, що робить їх потужними інструментами для сучасних команд. Завдяки своїй гнучкості, інтеграції з іншими сервісами та можливостям адаптації до різних масштабів проектів, ці платформи можуть стати відмінною альтернативою для створення онлайн-спільнот розробників, забезпечуючи ефективну координацію та підтримку розвитку професійних зв’язків.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Проектування бази даних

Проектування бази даних є одним із ключових етапів розробки веб-застосунків, оскільки саме від нього залежить ефективність зберігання, обробки та доступу до даних. Правильно спроектована база даних забезпечує стабільну роботу застосунку, високу продуктивність, масштабованість та безпеку збережених даних. Вона створює основу для реалізації бізнес-логіки, підтримки взаємодії між користувачами та надає можливість ефективного пошуку і сортування інформації.

Головною метою проектування бази даних є створення логічної та фізичної структури, яка відповідає вимогам застосунку та забезпечує цілісність даних. Це включає визначення таблиць, полів, типів даних, зв'язків між таблицями, ключів і обмежень. Крім того, важливим аспектом є забезпечення високої продуктивності запитів і оптимізація процесів читання та запису інформації.

Основна ідея проектування бази даних полягає в тому, щоб організувати дані таким чином, щоб мінімізувати дублювання, зменшити використання дискового простору та забезпечити швидкий доступ до потрібної інформації. Це досягається завдяки правильному нормуванню таблиць, використанню індексів, а також побудові ефективних відносин між таблицями, що забезпечує високу цілісність даних і стабільну роботу системи в умовах великих навантажень.

Бази даних поділяються на кілька основних видів залежно від способу організації даних, структури зберігання та принципів доступу до інформації [3-4]. Кожен вид баз даних має свої переваги та недоліки, а також підходить для різних типів застосунків. Ось основні види баз даних:

1. **Реляційні бази даних (RDBMS)**

Це найпоширеніший тип баз даних, де дані зберігаються у вигляді таблиць, які мають чітко визначену структуру з полями (колонками) та записами (рядками). Вони підтримують мову запитів SQL (Structured Query Language) і забезпечують високий рівень цілісності та консистентності даних. Основними характеристиками

реляційних баз є нормалізація даних, підтримка транзакцій та чіткі зв'язки між таблицями.

Коли вибирати:

- Коли потрібна складна структура даних з чіткими зв'язками (наприклад, CRM-системи, бухгалтерські програми).
- Коли важлива цілісність даних і підтримка транзакцій (банківські системи, системи бронювання).
- Коли потрібна складна аналітика та робота з великими обсягами структурованих даних (бізнес-аналітика, ERP-системи).
- **Приклади:** MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database.

2. Документоорієнтовані бази даних

Це нереляційні (NoSQL) бази даних, які зберігають дані у вигляді документів, зазвичай у форматах JSON, BSON або XML. Документи можуть містити вкладені структури, масиви та різнорідні типи даних, що робить їх дуже гнучкими.

Коли вибирати:

- Коли потрібно працювати з напівструктурованими або неструктурованими даними (чат-боти, соціальні мережі).
- Коли важлива гнучкість моделі даних і швидка розробка (прототипи, стартапи).
- Коли потрібна висока продуктивність і горизонтальне масштабування (реальні часи, високонавантажені системи).
- **Приклади**: MongoDB, CouchDB, RavenDB.

3. Графові бази даних

Цей тип баз даних призначений для зберігання та аналізу даних, пов'язаних складними зв'язками. Дані зберігаються у вигляді графів, де вузли представляють сутності, а ребра – зв'язки між ними.

Коли вибирати:

- Коли важлива побудова соціальних графів або моделювання складних зв'язків (соціальні мережі, системи рекомендацій, управління ланцюгами постачань).
- Коли потрібен аналіз зв'язків між даними (пошук найкоротших шляхів, виявлення кластерів).

Приклади: Neo4j, Amazon Neptune, ArangoDB.

4. Бази даних типу «ключ-значення»

Це найпростіший тип баз даних, який зберігає дані у форматі пар «ключ-значення». Вони дуже швидкі, але не підходять для складних запитів або роботи зі взаємопов'язаними даними.

Коли вибирати:

- Коли потрібен простий і швидкий доступ до даних (кешування, збереження сесій, системи реального часу).
- Коли потрібно працювати з великим обсягом простих, але часто використовуваних даних (рекомендаційні системи, лічильники). Приклади: Redis, Memcached, DynamoDB.

5. Колонкові бази даних

Ці бази даних зберігають дані не рядками, а колонками, що забезпечує високу продуктивність при аналітичних запитах. Вони оптимізовані для швидкого зчитування великих обсягів даних.

Коли вибирати:

- Коли потрібно працювати з великими наборами даних, що потребують аналітики (Big Data, OLAP).

Коли важлива ефективна компресія даних (системи бізнес-аналітики, дата-центри).

Приклади: Apache Cassandra, HBase, Amazon Redshift.

6. Бази даних для обробки часових рядів (Time-series DB)

Призначені для роботи з даними, що змінюються у часі, наприклад, для моніторингу показників, збору телеметрії або обробки фінансових даних.

Коли вибирати:

- Коли основне завдання – аналіз часових рядів (моніторинг серверів, IoT, фінансовий трейдинг).
- Коли важлива висока продуктивність запису та зчитування даних у реальному часі.

Приклади: InfluxDB, TimescaleDB, Prometheus.

В межах проекту було обрано реляційну базу даних MySQL. Нижче наведено 5 основних таблиць системи. Інші таблиці можна переглянути в Додатку В.

Можна розпочати з визначення полів для таблиці Projects (рис. 2.1).

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	Title	nvarchar(MAX)	☐	☐	
=	Payment	float	☐	☐	
=	ProjectDetails	nvarchar(MAX)	☐	☐	
=	ProjectOwnerID	uniqueidentifier	☐	☐	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	
=	Type	int	☐	☐	((0))
=	IsFullTeam	bit	☐	☐	(CONVERT([bit],(0)))
=	TimeDuration	int	☐	☐	((0))
=	EndDate	datetimeoffset(7)	☐	☑	
=	PaymentType	int	☐	☐	((0))
=	StartDate	datetimeoffset(7)	☐	☑	
=	Status	int	☐	☐	((0))

Рисунок 2.1 – Структура таблиці Projects

1. **Id:** Унікальний ідентифікатор проекту, що використовується як первинний ключ таблиці.
2. **Title**: Назва проекту, яка описує його основну тему або мету.

3. **Payment**: Поле для зберігання оплати, яку отримують учасники проекту.

4. **ProjectDetails**: Детальний опис проекту, включаючи вимоги, цілі та очікувані результати.

5. **ProjectOwnerID**: Зовнішній ключ, який вказує на власника проекту у таблиці ProjectOwners.

6. **UpdatedTimestamp**: Дата і час останнього оновлення інформації про проект.

7. **CreatedTimestamp**: Дата і час створення проекту.

8. **UpdatedBy**: Ідентифікатор користувача, який останнім оновлював інформацію про проект.

9. **CreatedBy**: Ідентифікатор користувача, який створив проект.

10. **Type**: Тип проекту, що визначає його категорію або напрямок, зберігається як ціле число.

11. **IsFullTeam:** Логічне поле, яке вказує, чи є команда проекту повністю укомплектованою.

12. **TimeDuration:** Тривалість проекту (повний або неповний робочий графік), зберігається як ціле число.

13. **EndDate:** Дата завершення проекту, використовується для відстеження дедлайнів.

14. **PaymentType:** Тип оплати, що включає погодинну або фіксовану форму оплати.

15. **StartDate:** Дата початку проекту, яка використовується для планування і відстеження прогресу.

16. **Status**: Поточний статус проекту, зберігається як ціле число і може відображати різні етапи (активний, завершений, відмінений).

Далі визначимо структуру таблиці Developers (рис. 2.2):

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	FirstName	nvarchar(MAX)	☐	☐	
=	LastName	nvarchar(MAX)	☐	☐	
=	IsDeleted	bit	☐	☐	
=	Email	nvarchar(MAX)	☐	☐	
=	PhotoFileId	uniqueidentifier	☐	☑	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	
=	Bio	nvarchar(MAX)	☐	☑	
=	HourlyPayment	decimal(18,2)	☐	☐	((0.0))
=	Position	int	☐	☑	

Рисунок 2.2 – Структура таблиці Developers

1. **Id:** Унікальний ідентифікатор розробника, який використовується як первинний ключ таблиці.

2. **FirstName**: Ім'я розробника, зберігається у текстовому форматі.

3. **LastName**: Прізвище розробника, використовується для ідентифікації користувача.

4. **IsDeleted**: Логічне поле, яке вказує, чи був запис розробника видалений (1 - видалений, 0 - активний).

5. **Email**: Електронна адреса розробника, використовується для комунікації та ідентифікації.

6. **PhotoFileId**: Зовнішній ключ, який посилається на таблицю PhotoFiles для зберігання фотографії розробника.

7. **UpdatedTimestamp**: Дата і час останнього оновлення запису про розробника.

8. **CreatedTimestamp**: Дата і час створення запису про розробника.

9. **UpdatedBy**: Ідентифікатор користувача, який останнім вніс зміни до запису про розробника.

10. **CreatedBy**: Ідентифікатор користувача, який створив запис про розробника.

11. **Bio**: Коротка біографія розробника, що містить інформацію про досвід, навички та професійні досягнення.

12. **HourlyPayment**: Поле для зберігання погодинної ставки розробника, з точністю до сотих частин.

13. **Position**: Ідентифікатор, що вказує на позицію або роль розробника в системі (може використовуватися для класифікації спеціалізації).

Потім визначимо структуру таблиці ProjectOwners (рис. 2.3):

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	FirstName	nvarchar(MAX)	☐	☐	
=	LastName	nvarchar(MAX)	☐	☐	
=	IsDeleted	bit	☐	☐	
=	Email	nvarchar(MAX)	☐	☐	
=	PhotoFileId	uniqueidentifier	☐	☑	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	
=	Bio	nvarchar(MAX)	☐	☑	

Рисунок 2.3 – Структура таблиці ProjectOwners

1. **Id:** Унікальний ідентифікатор власника проекту, який використовується як первинний ключ таблиці

2. **FirstName:** Ім'я власника проекту, використовується для ідентифікації користувача.

3. **LastName**: Прізвище власника проекту, що дозволяє повноцінно ідентифікувати користувача.

4. **IsDeleted**: Логічне поле, яке вказує, чи був запис про власника проекту видалений (1 - видалений, 0 - активний).

5. **Email**: Електронна адреса власника проекту, що використовується для комунікації та автентифікації.

6. **PhotoFileId**: Зовнішній ключ, що посилається на таблицю PhotoFiles і містить ідентифікатор фотографії власника проекту.

7. **UpdatedTimestamp**: Дата і час останнього оновлення запису про власника проєкту, що допомагає відстежувати зміни.

8. **CreatedTimestamp**: Дата і час створення запису про власника проекту, використовується для логування створення користувача.

9. **UpdatedBy**: Ідентифікатор користувача, який останнім вніс зміни до запису про власника проекту.

10. **CreatedBy**: Ідентифікатор користувача, який створив запис про власника проекту.

11. **Bio**: Поле для зберігання короткої біографії власника проекту, що може містити інформацію про досвід, компетенції та професійні досягнення.

Зараз можна виначити структуру таблиці FunctionalityBlocks (рис. 2.4):

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	TaskName	nvarchar(MAX)	☐	☐	
=	Status	int	☐	☐	
=	ProjectId	uniqueidentifier	☐	☐	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	
=	Description	nvarchar(MAX)	☐	☐	(N'')
=	DeveloperId	uniqueidentifier	☐	☑	
=	Label	int	☐	☐	((0))
=	ReportContent	nvarchar(MAX)	☐	☑	

Рисунок 2.4 – Структура таблиці FunctionalityBlocks

1. **Id:** Унікальний ідентифікатор завдання проекту, який використовується як первинний ключ таблиці.

2. **TaskName:** Назва завдання, що описує його призначення та суть.

3. **Status**: Цілочисельне поле для відстеження поточного статусу завдання, наприклад, створено, в процесі виконання або завершено.

4. **ProjectId**: Зовнішній ключ, що посилається на таблицю **Projects**. Вказує, до якого проекту належить дане завдання.

5. **UpdatedTimestamp**: Дата і час останнього оновлення завдання, що дозволяє відстежувати зміни у прогресі.

6. **CreatedTimestamp**: Дата і час створення завдання, що фіксує початковий момент його створення.

7. **UpdatedBy**: Ідентифікатор користувача, який останнім вніс зміни до завдання, дозволяючи відстежувати відповідальних за редагування.

8. **CreatedBy**: Ідентифікатор користувача, який створив завдання, вказує на автора.

9. **Description**: Текстовий опис завдання, що надає детальну інформацію про його призначення та очікуваний результат. За замовчуванням порожній.

10. **DeveloperId**: Зовнішній ключ, що посилається на таблицю Developers. Вказує на розробника, відповідального за виконання цього завдання.

11. **Label:** Цілочисельне поле для маркування завдання, що може використовуватись для додаткової категоризації. За замовчуванням встановлено значення 0.

12. **ReportContent**: Поле для зберігання звіту або додаткових нотаток щодо виконання завдання, яке може містити текстові записи про історію поточної задачі.

Далі опишемо структуру таблиці ProjectRequests (рис. 2.5):

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	ProjectId	uniqueidentifier	☐	☐	
=	DeveloperId	uniqueidentifier	☐	☐	
=	IsAccepted	bit	☐	☐	
=	IsDeclined	bit	☐	☐	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	

Рисунок 2.5 – Структура таблиці ProjectRequests

1. **Id:** Унікальний ідентифікатор запиту на участь у проекті, який використовується як первинний ключ для забезпечення унікальності кожного запиту.

2. **ProjectId:** Ідентифікатор проекту, до якого розробник хоче приєднатися, створює зв'язок між запитом та конкретним проектом у таблиці Projects.

3. **DeveloperId**: Ідентифікатор розробника, який подав запит на участь у проекті, створює зв'язок між запитом та конкретним розробником у таблиці Developers.

4. **IsAccepted**: Логічне поле, що вказує, чи був запит на участь у проекті прийнятий, де значення **TRUE** означає, що запит було схвалено.

5. **IsDeclined**: Логічне поле, що вказує, чи був запит на участь у проекті відхилений, де значення **TRUE** означає, що запит було відхилено.

6. **UpdatedTimestamp**: Дата і час останнього оновлення запису, що дозволяє відстежувати останні зміни в статусі запиту.

7. **CreatedTimestamp**: Дата і час створення запису про запит, що фіксує момент подання запиту розробником.

8. **UpdatedBy**: Ідентифікатор користувача, який вніс останні зміни до запиту, що дозволяє відстежувати відповідальних за оновлення статусу запиту.

9. **CreatedBy**: Ідентифікатор користувача, який створив запис про запит, що допомагає визначити початкового ініціатора запиту на участь у проекті.

2.2. Побутова об’єктно-орієнтованої моделі

2.2.1. Діаграма класів

Діаграма класів є одним з основних елементів мови моделювання UML (Unified Modeling Language) і використовується для відображення статичної структури програмного забезпечення. Вона описує класи, їх атрибути, методи, а також взаємозв'язки між ними. Основна мета діаграми класів – показати, як різні частини системи взаємодіють між собою, створюючи основу для розробки коду і забезпечуючи більш глибоке розуміння архітектури проекту.

Використання діаграм класів є важливим етапом проектування, оскільки вони дозволяють чітко структурувати об'єкти та їх відносини ще до початку програмування. Це спрощує командну роботу, забезпечуючи всім учасникам розробки єдине бачення проекту. Крім того, така візуалізація полегшує виявлення помилок в логіці взаємодії класів, що може суттєво знизити кількість можливих багів у майбутньому. Діаграми класів також допомагають у впровадженні принципів об'єктно-орієнтованого програмування, таких як наслідування, поліморфізм і інкапсуляція, що підвищує гнучкість і масштабованість системи.

Доцільність використання діаграм класів особливо помітна у великих проектах з великою кількістю взаємозалежних об'єктів, де чітке уявлення про архітектуру допомагає уникнути складностей у підтримці коду. Це також є корисним інструментом для документування системи, що робить її легшою для розуміння новими членами команди або зовнішніми аудиторами.

Діаграма класів відображає не лише самі класи, але й зв'язки між ними, що дозволяє побудувати повноцінну модель системи. Зв'язки між класами представляють реальні відносини між компонентами програми, включаючи асоціації, агрегації, композиції та наслідування. Наприклад, один клас може бути пов'язаним з іншим через асоціацію, що відображає залежність між об'єктами в час виконання, як-от відносини між користувачем і його профілем.

Інший тип зв'язків, агрегація, вказує на слабкішу форму відношення, коли один клас складається з іншого, але може існувати незалежно від нього. Наприклад, проект може складатися з кількох завдань, але ці завдання можуть продовжувати існувати, навіть якщо проект видалено. Композиція, на відміну від агрегації, представляє сильнішу форму залежності, де один клас не може існувати без іншого, як у випадку відношення між замовленням та його елементами.

Також діаграми класів включають наслідування, яке відображає ієрархію класів, дозволяючи повторно використовувати властивості та методи базових класів у похідних. Це важливо для створення масштабованих та гнучких архітектур, де спільні функціональні можливості можна реалізувати один раз у базовому класі.

Всі ці зв'язки допомагають створити чітку, логічно структуровану модель програми, що полегшує розуміння, розширення та підтримку коду в майбутньому. Побачити діаграми можна в додатку А.

2.2.2. Use Case діаграма

Діаграма варіантів використання (Use Case діаграма) [5] є важливим елементом моделювання поведінки системи, що описує, як різні користувачі (акторів) взаємодіють із системою для досягнення певних цілей. Вона дозволяє зрозуміти, які функціональні можливості має система з точки зору кінцевих користувачів, а також визначити межі системи, основні сценарії взаємодії та можливі виключення.

Цей тип діаграми є корисним на початкових етапах розробки програмного забезпечення, коли необхідно зібрати вимоги до системи та структурувати їх у зручний для аналізу формат. Завдяки діаграмам варіантів використання можна легко виявити ключові функції системи, зрозуміти взаємозв'язки між користувачами та системою, а також оцінити складність розробки. Це значно полегшує спілкування між розробниками, дизайнерами, аналітиками та замовниками, оскільки діаграми є візуально зрозумілими навіть для людей без технічної підготовки.

Основними перевагами використання діаграм варіантів використання є можливість чіткого визначення ролей користувачів, документування основних сценаріїв роботи системи та відображення взаємодій між підсистемами. Це забезпечує

більш структурований підхід до розробки програмного забезпечення, знижує ризик пропуску критичних функцій і полегшує підтримку системи у майбутньому. До прикладу, нижче наведено Use Case діаграму веб-платформи для онлайн співпраці над командними проектами для користувача з роллю «Розробник» (рис. 2.6) та користувача з роллю «Проектний менеджер» (рис. 2.7):

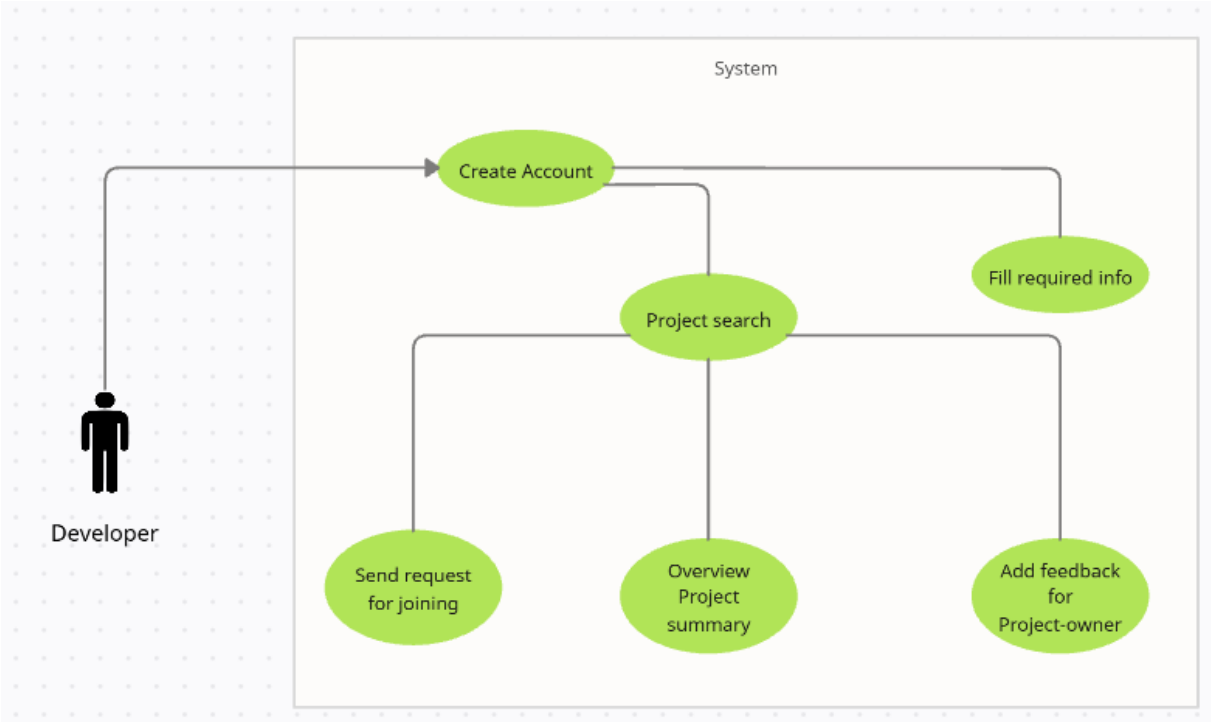


Рисунок 2.6 – Use Case діаграма для користувача з роллю «Розробник»

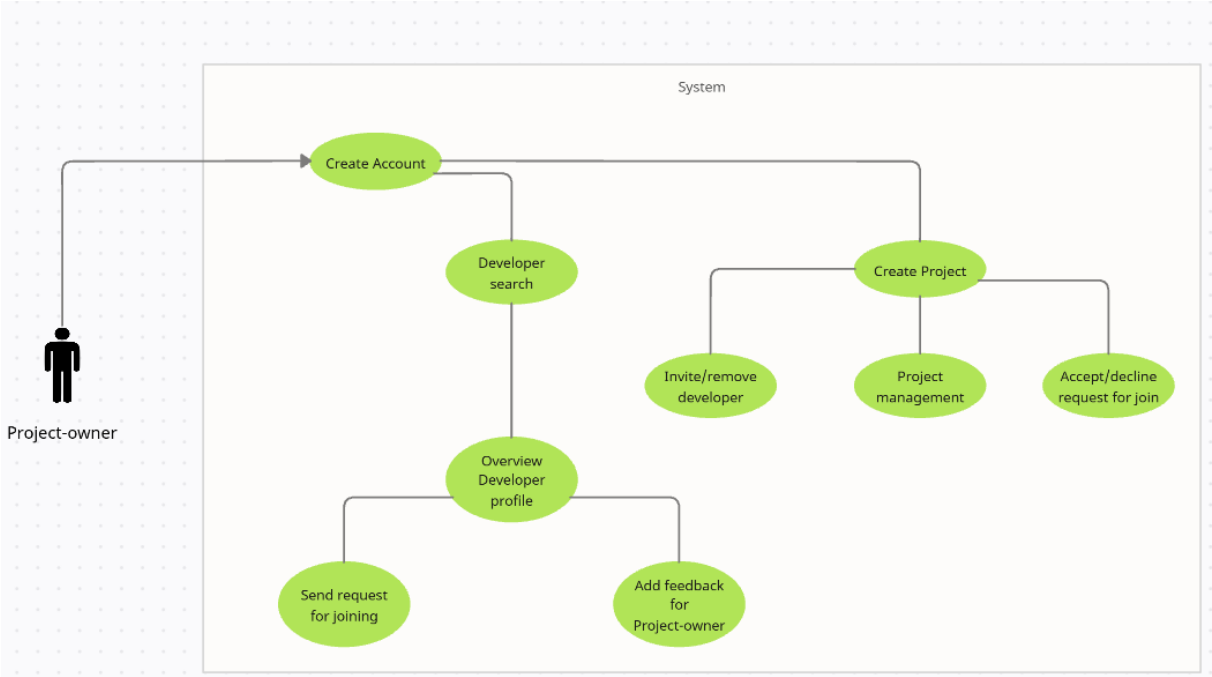


Рисунок 2.7 – Use Case діаграма для користувача з роллю «Проектний менеджер»

2.2.3. Архітектура системи

Вибір архітектури системи є одним з найважливіших етапів розробки програмного забезпечення, оскільки він визначає основну структуру майбутнього продукту, його гнучкість, продуктивність, масштабованість та можливість подальшого розвитку. Архітектура задає фундаментальні принципи побудови програмного рішення, впливає на спосіб реалізації бізнес-логіки, обробки даних, організації взаємодії між компонентами та забезпечення безпеки.

Правильно обрана архітектура дозволяє створити стабільну, ефективну та легку в підтримці систему, що може адаптуватися до змін бізнес-вимог без значних витрат. Це особливо важливо для великих проектів, які постійно розширюються і потребують внесення нових функцій. Також архітектура визначає, наскільки легко можна інтегрувати систему з іншими сервісами, масштабувати її для обробки великої кількості запитів і зберігати високу швидкодію.

Окрім цього, архітектурні рішення значно впливають на якість коду, його повторне використання та модульність, що знижує витрати на розробку, тестування та обслуговування. Обрана архітектура повинна відповідати потребам користувачів,

вимогам бізнесу та технічним обмеженням, а також враховувати фактори безпеки, продуктивності та зручності використання.

Таким чином, вибір архітектури є критичним кроком, що впливає на весь життєвий цикл програмного продукту, від початкових етапів розробки до його розгортання та підтримки в реальних умовах.

Для розробки платформи для онлайн-співпраці в командних IT-проектах мною було вибрано Layered (слоїста) архітектуру (рис. 2.8). Основними перевагами цього підходу є чітка структурованість та легкість підтримки.

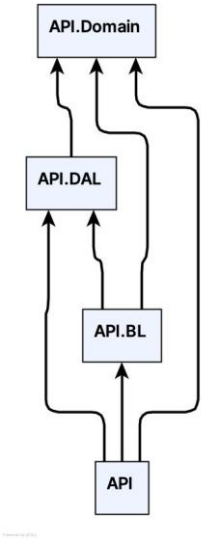


Рисунок 2.8 – Слоїста архітектура застосунку

Моя архітектура складається з таких основних шарів:

1. **API.Domain** - шар, що містить основні сутності та доменні моделі, які використовуються у всій системі. Він визначає структуру даних і основні бізнес-правила.

2. **API.DAL** - цей шар відповідає за доступ до бази даних. Він включає контекст бази даних, міграції, конфігурації зв’язків та операції збереження даних. Його основна задача - забезпечити надійний і швидкий доступ до даних, ізолюючи бізнес-логіку від деталей реалізації сховища даних.

3. **API.BL** - шар бізнес-логіки, в якому реалізуються всі CRUD-операції та сервіси. Він містить основну логіку обробки даних, включаючи перевірку правил бізнесу, управління транзакціями та виклики до шару доступу до даних.

4. **API** - зовнішній шар, який відповідає за прийом запитів від клієнтів (фронтенду) та передачу результатів. Він забезпечує маршрутизацію, валідацію вхідних даних і обробку виключень, тісно взаємодіючи з шаром бізнес-логіки.

Окрім слоїстої архітектури, під час роботи я розглянув також клієнт-серверну модель та мікросервісну архітектуру. Клієнт-серверна модель виявилася надто простою для забезпечення гнучкої масштабованості та чіткого розділення компонентів, а мікросервісний підхід вимагав значних зусиль на організацію міжсервісної взаємодії та складного DevOps-процесу. Зважаючи на ці фактори, я обрав саме слоїсту архітектуру, оскільки вона поєднує достатню модульність і зрозумілість реалізації з відносно простою організацією розгортання та підтримки, що є оптимальним рішенням для платформи, котра повинна балансувати між швидкістю розвитку та надійністю.

По-перше, така архітектура забезпечує розділення відповідальностей, де кожен шар виконує свою чітко визначену функцію. Це спрощує процес розробки та тестування, оскільки кожен компонент системи може бути розроблений та перевірений окремо.

По-друге, слоїста архітектура сприяє гнучкості в модернізації та масштабуванні системи. Завдяки чіткій ізоляції шарів, зміни в одному з них мінімально впливають на інші, що дозволяє легше впроваджувати нові функціональні можливості або змінювати існуючі компоненти без ризику порушити роботу всієї системи.

2.3. Архітектура підкомпонентів

У складі платформи присутній окремий модуль безпеки, в якому повністю інтегровано та налаштовано можливості Microsoft Identity. Цей підмодуль відповідає за реєстрацію та автентифікацію користувачів, управління ролями й правами доступу, а також за генерацію та валідацію JWT-токенів для захищеного звернення до API. Завдяки використанню вбудованих сервісів UserManager і SignInManager, усі операції з обліковими записами та паролями виконуються централізовано й стандартизовано, що забезпечує високу консистентність даних і мінімізує ризики помилок у процесі аутентифікації. Конфігурація модуля зберігається у файлі налаштувань, де визначено

симетричний ключ для підпису токенів і правила їхнього терміну дії, а також параметри політик паролів і підтвердження електронної пошти. Така архітектурна ізоляція механізмів безпеки дозволяє легко адаптувати або розширювати модуль із мінімальним впливом на інші компоненти системи.

2.3.1. Структура бази даних

У базі даних платформи передбачено окрему схему для зберігання даних (див. рис. 2.25), пов’язаних із механізмами автентифікації та авторизації, реалізованими за допомогою Microsoft Identity. У центрі цієї схеми лежить таблиця AspNetUsers, складовою частиною якої є модель IdentityUser, що містить усі стандартні поля для облікових записів (ідентифікатор, ім’я користувача, електронна пошта, хеш пароля тощо). Поряд із нею присутні таблиці AspNetRoles для зберігання ролей, AspNetUserRoles – для встановлення зв’язків «користувач–роль», AspNetUserClaims та AspNetRoleClaims для додаткових прав та претензій (claims), а також AspNetUserLogins, AspNetUserTokens і AspNetUserTokens для підтримки зовнішніх входів і керування токенами безпеки. Кожна з цих таблиць має зовнішні ключі, які гарантують цілісність даних між собою, а також індекси на полях, що використовуються для пошуку за іменем користувача або електронною поштою, що забезпечує високу швидкодію запитів аутентифікації. Така структура дозволяє централізовано зберігати та обробляти всі аспекти роботи з обліковими записами, від підтвердження електронної пошти до налаштувань двофакторної автентифікації, без необхідності визначати власні таблиці чи ключі поза рамками стандартної моделі Microsoft Identity (рис. 2.9).

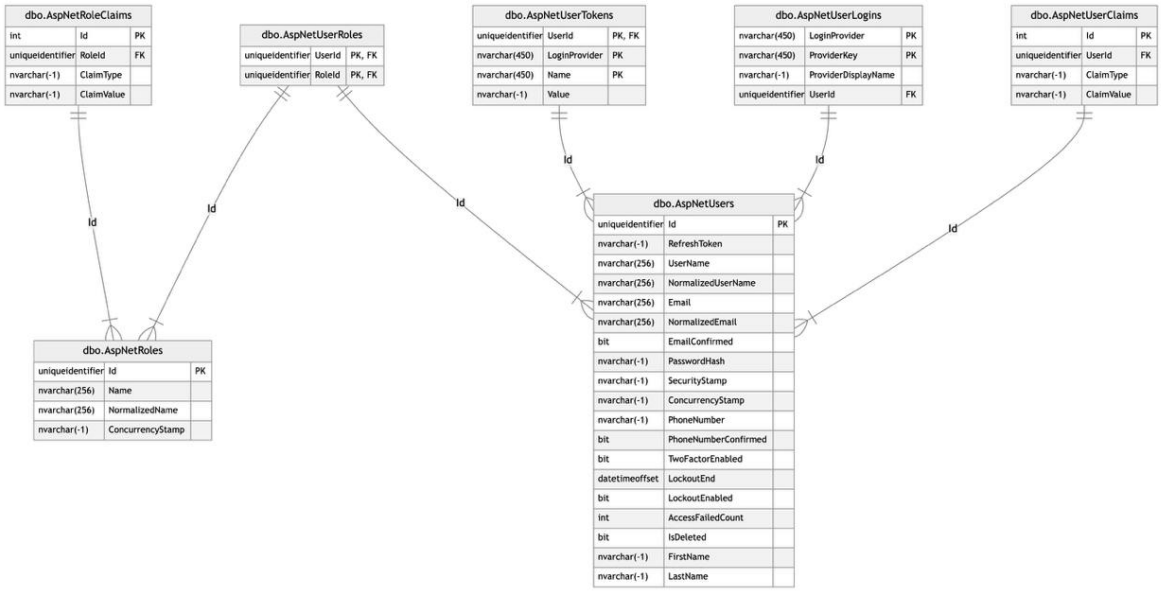


Рисунок 2.9 – Схема бази даних Microsoft Identity.

2.4. Алгоритмічне забезпечення

Хешування паролів є важливим елементом безпеки будь-якої сучасної інформаційної системи. Воно дозволяє зберігати паролі у зашифрованому вигляді, що зменшує ризик компрометації у випадку несанкціонованого доступу до бази даних. Це особливо важливо, оскільки навіть якщо зловмисник отримає доступ до хешованих паролів, він не зможе легко відновити початковий текст паролю без знання секретного ключа.

Для забезпечення безпеки паролів у моєму проекті було використано алгоритм хешування HMAC-SHA-256 (Hash-based Message Authentication Code з використанням SHA-256). Цей алгоритм забезпечує високу стійкість до атак завдяки поєднанню криптографічного хешування з секретним ключем. Він працює шляхом застосування хеш-функції SHA-256 для створення цифрового підпису, що забезпечує цілісність і автентичність даних. HMAC-SHA-256 використовує секретний ключ разом з повідомленням для створення унікального хешу, який складно підробити без знання цього ключа. Це робить його особливо ефективним для захисту паролів, оскільки навіть незначна зміна у вхідних даних призводить до суттєво іншого результату хешування.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Інструменти та технології розробки

3.1.1 Мова програмування C#

C# - це сучасна, об'єктно-орієнтована мова програмування, розроблена компанією Microsoft як частина платформи .NET. Вона поєднує в собі простоту написання коду, високу продуктивність і потужні можливості для створення різноманітних програмних рішень, від простих консольних додатків до великих корпоративних систем та хмарних сервісів. Основна мета розробки C# полягала в тому, щоб створити мову, яка буде легка у використанні, але водночас достатньо потужна для вирішення складних завдань.

C# широко використовується у розробці веб-додатків (ASP.NET), настільних застосунків (WPF, Windows Forms), мобільних додатків (Xamarin, .NET MAUI), ігор (Unity) та мікросервісів. Завдяки інтеграції з платформою .NET, ця мова забезпечує високий рівень продуктивності, безпеки та кросплатформеності, що дозволяє створювати додатки для Windows, macOS, Linux, iOS та Android.

Однією з ключових переваг C# є його об'єктно-орієнтована природа, яка підтримує такі концепції, як інкапсуляція, наслідування, поліморфізм та абстракція. Це спрощує розробку великих і складних програм, підвищуючи їхню підтримуваність і масштабованість. Крім того, C# підтримує сучасні функціональні можливості, такі як асинхронне програмування (async/await), розширювальні методи, делегати, події та лямбда-вирази, що робить його зручним для роботи з потоками даних і асинхронними операціями.

Важливою перевагою цієї мови є автоматичне керування пам'яттю через механізм збирача сміття (garbage collector), що мінімізує ризик виникнення витоків пам'яті. Також варто відзначити високу безпеку коду завдяки суворій типізації та перевірці типів під час компіляції, що дозволяє уникати багатьох поширених помилок.

Таким чином, C# є універсальною мовою програмування, яка поєднує у собі простоту написання коду, високий рівень безпеки, продуктивності та

кросплатформені можливості, що робить її відмінним вибором для створення різноманітних програмних рішень.

3.1.2. .NET Framework

.NET Framework – це потужна платформа для розробки програмного забезпечення, створена компанією Microsoft. Вона забезпечує розробникам широкий набір інструментів і бібліотек для створення додатків різної складності, від простих консольних програм до великих корпоративних систем. Основна мета .NET Framework полягає в тому, щоб спростити розробку, розгортання та підтримку програмного забезпечення, пропонуючи єдине інтегроване середовище для різних типів додатків, включаючи десктопні (Windows Forms, WPF), веб-додатки (ASP.NET), служби Windows та консольні додатки.

.NET Framework складається з кількох ключових компонентів. Основою є Common Language Runtime (CLR) – середовище виконання, яке керує виконанням коду, забезпечуючи автоматичне керування пам'яттю, збір сміття та безпеку типів. CLR також забезпечує сумісність між різними мовами програмування, такими як C#, Visual Basic .NET та F#. Іншою важливою частиною є .NET Framework Class Library (FCL) – великий набір готових класів і функцій, які значно полегшують розробку додатків, включаючи роботу з файлами, мережевими протоколами, базами даних та графічними інтерфейсами користувача.

Однією з основних причин вибору .NET Framework є його глибока інтеграція з операційною системою Windows, що робить його ідеальним для створення високопродуктивних і надійних Windows-додатків. Крім того, .NET Framework підтримує об'єктно-орієнтоване програмування, що сприяє створенню масштабованих і зрозумілих додатків з високим рівнем повторного використання коду. Завдяки використанню загальної мови проміжного коду (CIL) і JIT-компіляції, .NET забезпечує високу продуктивність і ефективність виконання додатків.

Серед ключових переваг .NET Framework можна виділити стабільність, високу продуктивність, простоту у використанні та багату екосистему бібліотек і компонентів, що робить його популярним вибором для створення корпоративних

додатків. Це також включає інтеграцію з потужними засобами розробки, такими як Visual Studio, підтримку сучасних протоколів безпеки та можливість легкого розширення функціональності через сторонні бібліотеки та фреймворки.

Таким чином, .NET Framework залишається одним з найпопулярніших середовищ розробки для створення надійних, масштабованих і продуктивних додатків під платформу Windows.

3.1.3. ASP.NET Framework

ASP.NET Framework – це високопродуктивна платформа для розробки веб-додатків, створена компанією Microsoft як частина більшої .NET екосистеми. Вона призначена для створення динамічних, інтерактивних та високопродуктивних веб-додатків, веб-сервісів і API. Основою ASP.NET [6] є модель програмування, яка використовує CLR (Common Language Runtime), що дозволяє розробникам використовувати об'єктно-орієнтовані мови, такі як C#, Visual Basic .NET або F#, для створення сучасних веб-рішень.

ASP.NET забезпечує розробників широким спектром інструментів та бібліотек для швидкої розробки веб-додатків. Основні технології, які входять до ASP.NET, включають Web Forms, MVC (Model-View-Controller), Web API та SignalR [7]. Web Forms дозволяє створювати додатки з використанням серверних контролів, що значно спрощує побудову складних веб-інтерфейсів. MVC, у свою чергу, забезпечує чітке розділення логіки додатку, інтерфейсу користувача та управління даними, що покращує тестованість і розширюваність коду. Web API дозволяє створювати легкі HTTP-сервіси для клієнтських додатків, а SignalR додає підтримку реального часу для таких додатків, як чати чи живі сповіщення.

Однією з головних причин використання ASP.NET є його висока продуктивність та безпека, що досягається завдяки попередній компіляції, оптимізації коду і вбудованим засобам безпеки, таким як аутентифікація, авторизація та захист від атак типу XSS і CSRF. Це робить ASP.NET популярним вибором для розробки корпоративних додатків, які вимагають високої надійності, безпеки та продуктивності. Крім того, ASP.NET має тісну інтеграцію з іншими сервісами

Microsoft, такими як Azure, що спрощує розгортання та масштабування додатків у хмарному середовищі.

Завдяки розвинутій екосистемі компонентів, розширеній підтримці стандартів HTTP, RESTful сервісів та легкій інтеграції з базами даних через Entity Framework, ASP.NET залишається одним з найпопулярніших виборів для створення веб-додатків, здатних обслуговувати тисячі одночасних користувачів.

3.1.4. Entity Framework Core

Entity Framework Core (EF Core) – це сучасний об'єктно-реляційний відображувач (ORM) для .NET, який дозволяє розробникам працювати з базами даних, використовуючи об'єктно-орієнтований підхід. Він забезпечує абстракцію над реляційними сховищами даних, дозволяючи взаємодіяти з базою даних через типізовані класи і методи, а не через прямі SQL-запити. Це значно спрощує процес розробки, зменшуючи кількість рутинного коду і підвищуючи читабельність.

EF Core [8] є кросплатформенним рішенням, що підтримує широкий набір баз даних, таких як Microsoft SQL Server, MySQL, PostgreSQL, SQLite та інші. Завдяки підтримці LINQ (Language Integrated Query) і потужним механізмам відстеження змін, він автоматично обробляє CRUD-операції, що знижує ризик помилок і забезпечує високу продуктивність розробки.

Цей фреймворк також підтримує міграції, що дозволяє легко керувати версіями схем бази даних і оновлювати її структуру без втрати даних. Це особливо корисно для великих проектів, де необхідно постійно вносити зміни в модель даних.

EF Core робить код додатків більш чистим і підтримуваним, дозволяючи розробникам фокусуватися на бізнес-логіці, а не на деталях роботи з базою даних. Завдяки цьому, його доцільно використовувати в сучасних корпоративних застосунках, які потребують масштабованості, високої продуктивності та стабільності.

3.1.5. ASP.NET Core Identity

ASP.NET Core Identity – це потужна система аутентифікації та авторизації для веб-додатків, побудованих на платформі ASP.NET Core. Вона надає розробникам комплексний набір інструментів для управління користувачами, їхніми ролями, паролями та правами доступу, забезпечуючи безпеку і зручність розробки. Завдяки інтеграції з ASP.NET Core, система Identity легко налаштовується і масштабовується, що робить її ідеальним вибором для проектів будь-якої складності.

Ця система дозволяє розробникам уникати створення власних механізмів аутентифікації, використовуючи надійне рішення з підтримкою двофакторної автентифікації, токенів доступу і відновлення паролів. ASP.NET Core Identity [9] також підтримує інтеграцію з зовнішніми постачальниками аутентифікації, такими як Google, Facebook, Microsoft і Twitter, що значно полегшує процес реєстрації і входу для кінцевих користувачів.

Серед ключових переваг ASP.NET Core Identity – висока безпека, зокрема захист від атак на паролі, таких як brute force, завдяки використанню хешування паролів і токенів безпеки. Крім того, система дозволяє гнучко налаштовувати правила авторизації, що забезпечує точний контроль доступу до ресурсів. Це робить ASP.NET Core Identity оптимальним вибором для корпоративних додатків і сервісів з високими вимогами до безпеки.

IdentityUser - це базовий клас, що використовується в ASP.NET Core Identity для представлення користувача в системі аутентифікації. Він містить основні властивості, необхідні для зберігання ідентифікаційних даних користувача, такі як унікальний ідентифікатор (ID), ім'я користувача (UserName), електронна пошта (Email), хешований пароль (PasswordHash), статус підтвердження електронної пошти (EmailConfirmed), номер телефону (PhoneNumber) та інші атрибути. Це робить IdentityUser центральним компонентом для управління користувачами у веб-додатках на базі ASP.NET Core.

Використання IdentityUser доцільне, коли необхідно забезпечити надійну і масштабовану систему аутентифікації користувачів з мінімальними зусиллями. Оскільки цей клас включає всі необхідні поля для роботи з обліковими записами,

розробникам не потрібно створювати власні моделі користувачів з нуля. Це значно спрощує процес створення систем реєстрації, входу, зміни паролів і відновлення доступу, що дозволяє зосередитись на основних функціональних можливостях додатку.

Серед основних переваг IdentityUser - тісна інтеграція з базою даних через Entity Framework Core, підтримка токенів безпеки для відновлення паролів і двофакторної автентифікації, а також можливість розширення і кастомізації для створення складних схем авторизації з ролями, правами доступу і політиками.

UserManager – це основний сервіс у бібліотеці ASP.NET Core Identity, призначений для управління користувачами в додатках. Він забезпечує широкий набір методів для роботи з обліковими записами, таких як створення користувачів, встановлення паролів, перевірка автентифікаційних даних, блокування облікових записів, додавання ролей та генерування токенів для відновлення паролів або підтвердження електронної пошти. Клас UserManager підтримує роботу з користувачами, що реалізують клас IdentityUser або його похідні, що дозволяє зберігати всю необхідну інформацію про користувача в структурованому вигляді, включаючи ролі, статус підтвердження акаунта та інші властивості.

Використання UserManager доцільне, коли потрібно організувати систему автентифікації і авторизації, яка враховує сучасні вимоги безпеки. Він дозволяє уникнути ручного управління користувацькими даними, надаючи високорівневий API для всіх типових операцій, таких як перевірка паролів, відправка підтверджувальних листів та активація облікових записів. Завдяки цьому можна значно скоротити час розробки, зменшити кількість помилок і підвищити загальний рівень безпеки додатку.

Основні переваги UserManager включають тісну інтеграцію з Entity Framework Core для роботи з базами даних, можливість кастомізації поведінки через настройки та інтерфейси, підтримку асинхронних методів для високої продуктивності, а також готові рішення для двофакторної автентифікації і захисту від brute-force атак.

3.1.6. TypeScript

TypeScript – це строго типізована мова програмування, розроблена компанією Microsoft як надбудова над JavaScript. Вона забезпечує розширені можливості типізації, що дозволяє створювати масштабовані, надійні та легше підтримувані додатки. Основною ідеєю TypeScript є компіляція до стандартного JavaScript, що забезпечує сумісність з усіма сучасними браузерами, платформами та бібліотеками. Його використання значно покращує продуктивність розробників завдяки інтеграції статичної типізації, автоматичного завершення коду, перевірки типів і покращеної підтримки рефакторингу в IDE.

Застосування TypeScript доцільне в проектах, де важлива передбачуваність і стабільність коду, особливо для великих команд або довготривалих проектів. Він зменшує кількість помилок на ранніх стадіях розробки, оскільки компілятор перевіряє типи до виконання коду, що дозволяє уникнути багатьох типових проблем JavaScript, таких як виклики невизначених методів або доступ до неіснуючих властивостей. Це також допомагає забезпечити кращу інтеграцію з сучасними фреймворками, такими як Angular, React або Vue, які активно використовують TypeScript.

Основні переваги TypeScript включають потужну систему типів, підтримку сучасних стандартів ECMAScript, відмінну інтеграцію з інструментами розробки, а також можливість поступового впровадження в існуючі JavaScript-проекти, що дозволяє плавно мігрувати до більш типізованої і структурованої архітектури коду.

3.1.7. Angular (web framework)

Angular – це високопродуктивний фреймворк для розробки клієнтських веб-додатків, створений командою Google. Він забезпечує комплексний підхід до створення сучасних односторінкових додатків (SPA), поєднуючи TypeScript, HTML і CSS у єдину структуру. Angular [10-14] відомий своєю компонентною архітектурою, яка сприяє кращій організації коду та його повторному використанню. Завдяки потужному механізму двостороннього зв'язування даних і реактивному оновленню інтерфейсу, Angular дозволяє створювати високодинамічні та інтерактивні користувацькі інтерфейси. Його багата екосистема включає в себе модулі для

маршрутизації, керування станом, форм, анімацій, авторизації та тестування, що робить його потужним інструментом для комплексних веб-рішень.

Використання Angular є доцільним у проектах, де важлива структурованість і модульність коду, а також швидкий розвиток і підтримка надійних рішень. Цей фреймворк ідеально підходить для великих корпоративних додатків, які вимагають високої продуктивності та стабільності. Його сувора типізація через TypeScript допомагає уникнути типових помилок JavaScript, забезпечуючи кращу передбачуваність коду. Крім того, Angular має розвинену систему інжекції залежностей, що спрощує тестування компонентів і сприяє створенню легко підтримуваних додатків.

Основні переваги Angular включають високу продуктивність, розширені можливості тестування, інтеграцію з інструментами розробки, ефективне управління станом додатка і активну підтримку з боку спільноти. Його реактивний підхід до зміни стану додатку, розподілена система модулів та інструменти для оптимізації продуктивності роблять його одним з найпопулярніших виборів для створення сучасних клієнтських додатків.

3.1.8. Hangfire

Hangfire – це потужна бібліотека для фонових обчислень у .NET, яка дозволяє розробникам створювати, планувати та виконувати довготривалі задачі у фоновому режимі без складної конфігурації. Вона базується на парадигмі task scheduling, що забезпечує надійне та ефективне виконання завдань із автоматичним керуванням потоками, повторними спробами та можливістю відстеження стану кожного завдання. Завдяки інтеграції з популярними сховищами даних, такими як SQL Server, Redis або PostgreSQL, Hangfire [15] надійно зберігає всі заплановані завдання, що гарантує їх виконання навіть у випадку несподіваних збоїв або перезапусків серверів.

Доцільність використання Hangfire проявляється у проектах, де необхідно виконувати задачі у фоновому режимі, наприклад, обробку великих масивів даних, надсилання електронних листів, створення резервних копій чи автоматичне очищення баз даних. Hangfire підтримує різні типи завдань – від одноразових (fire-and-forget) до

періодичних (recurring) із детальним налаштуванням інтервалів виконання. Завдяки цьому, система стає гнучкішою, а продуктивність основного потоку додатку не знижується через затримки у виконанні важких задач.

Hangfire пропонує кілька ключових переваг, серед яких висока надійність і масштабованість. Завдяки вбудованій підтримці розподіленого виконання завдань, вона легко інтегрується у розподілені системи, розширюючи можливості обробки великих обсягів даних. Додатково, розробники отримують зручний інформаційний веб-інтерфейс (рис. 3.1, рис. 3.2) для моніторингу стану завдань, статистики їх виконання та налаштування повторних спроб у разі помилок. Завдяки таким можливостям, Hangfire суттєво спрощує керування фоновими процесами, забезпечуючи гнучкість, стабільність і легкість у розгортанні навіть для складних розподілених систем.

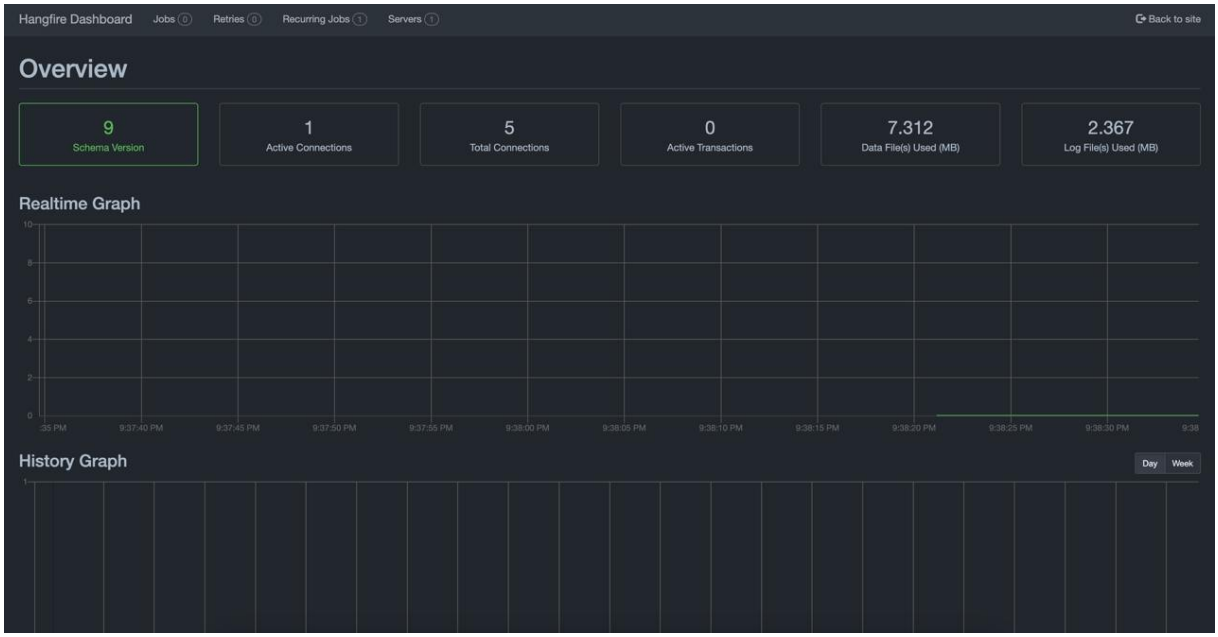


Рисунок 3.1 – Веб-інтерфейс Hangfire

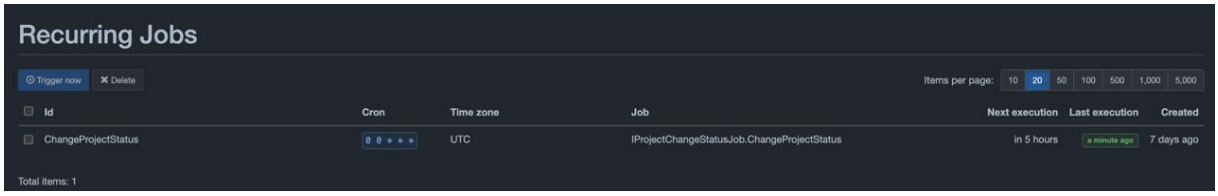


Рисунок 3.2 – Список стану завдань

3.1.9. Патерн проектування WEB API

Патерн проектування Web API базується на ідеї створення легковагових HTTP-сервісів, які надають доступ до функціональності додатку через добре визначені URI та стандартні методи протоколу (GET, POST, PUT, DELETE). Цей підхід орієнтований на відокремлення клієнтської і серверної частин, що дозволяє фронтенд-додаткам або будь-яким іншим споживачам сервісу звертатися до бекенду незалежно від платформи чи технологій, на яких вони реалізовані.

Вибір шаблону Web API [16] стає виправданим у ситуаціях, коли потрібно забезпечити зручний та уніфікований інтерфейс для доступу до даних і бізнес-логіки з різних джерел: веб-браузерів, мобільних пристроїв, IoT-пристроїв або мікросервісів. Така архітектура полегшує розробку, оскільки клієнтська частина може еволюціонувати окремо від сервера, а оновлення одного з компонентів не вимагає зміни іншого. Крім того, REST-орієнтований підхід Web API сприяє кращому розумінню та документуванню сервісів завдяки використанню стандартних форматів (JSON, XML) і зрозумілих URL.

Основна перевага застосування шаблону Web API полягає в його масштабованості та гнучкості: завдяки безстанному характеру запитів сервер може легко балансувати навантаження і горизонтально масштабуватися, не зберігаючи стан сесій між запитами. Іншим вагомим позитивом є висока сумісність із сучасними інструментами розробки й тестування: інтеграція з засобами автоматизованого тестування, такими як Postman чи Swagger, робить процес перевірки якості сервісів прозорим і швидким. Врешті-решт, Web API створює чіткий контракт між клієнтом і сервером, що сприяє стабільності проекту та полегшує подальшу підтримку й розширення функціоналу.

3.1.10. Архітектурний стиль REST API

Архітектура REST API базується на принципах ресурсно-орієнтованого підходу, за якого кожен елемент системи (користувачі, проекти, завдання тощо) представлений як окремий ресурс із власним унікальним URI. Взаємодія з такими ресурсами здійснюється через стандартизовані HTTP-методи – це дозволяє клієнтам з будь-якої

платформи або мови програмування звертатись до серверу однорідним чином. У процесі обміну повідомленнями REST API не зберігає стан сесії між запитами, що значно спрощує балансування навантаження та горизонтальне масштабування системи.

Застосування REST API [17] особливо виправдане, коли проект передбачає мультиплатформену екосистему клієнтів – від веб-інтерфейсів до мобільних додатків та зовнішніх сервісів. Завдяки суворій роздільності клієнт-сервер та принципу «stateless», нові компоненти або версії клієнтів можуть вводитись в експлуатацію незалежно один від одного, без необхідності синхронного оновлення бекенду. Крім того, можливість використовувати HTTP-кешування та стандартизовані заголовки відповіді підвищує швидкодію роботи, оскільки повторні запити можуть оброблятись із мінімальним завантаженням серверних ресурсів.

Головною перевагою REST API є його гнучкість та адаптивність до еволюції бізнес-логіки: відкритий контракт через URI й єдиний інтерфейс робить архітектуру прозорою та добре документованою. Використання уніфікованих форматів даних (зазвичай JSON) забезпечує високу сумісність із різноманітними клієнтами, а підтримка принципу шаруватості дозволяє впроваджувати проксі-сервери та балансувальники навантаження без змін у логіці застосунку. У підсумку REST API створює надійну, масштабовану та легко розширювану основу для побудови сучасних веб-сервісів.

3.2. Потреби щодо апаратного устаткування та програмного забезпечення

Платформа реалізована як веб-додаток і забезпечує крос-платформену роботу без необхідності встановлення спеціалізованого клієнтського ПЗ. Тоді як для кінцевого користувача достатньо сучасного браузера та інтернет-з’єднання, розробник повинен мати набір інструментів для побудови та підтримки серверної та фронтенд-частин.

Для кінцевого пристрою:

- Веб-браузер: будь-який сучасний браузер із підтримкою HTML5, CSS3 та JavaScript (Chrome, Firefox, Safari, Edge).

- Підключення до Інтернету: стабільне з’єднання для коректного завантаження даних і роботи інтерфейсу.
- Операційна система: незалежність від ОС; сумісність з Windows, macOS, Linux тощо.

Для середовища розробника:

- Серверна частина: встановлений .NET SDK (версії, сумісної з проектом).
- Фронтенд-частина: Node.js та менеджер пакетів NPM для роботи з Angular.
- База даних: доступ до Microsoft SQL Server (локально або віддалено).
- IDE / Редактори: середовище розробки з підтримкою .NET і TypeScript (Visual Studio, VS Code, JetBrains Rider).
- Підключення до Інтернету: для завантаження та оновлення бібліотек і залежностей.

Система має забезпечувати реалізацію наступних функціональних можливостей:

- Створення та налаштування нових проектів із зазначенням ключових параметрів (назва, опис, терміни, бюджет).
- Реєстрацію та автентифікацію користувачів із підтримкою захищеного зберігання паролів і валідації облікових записів.
- Редагування персональних даних у профілі користувача, включаючи контактну інформацію, біографію та фотографію.
- Формування команд для роботи над проектами з можливістю запрошення та прийняття розробників.
- Управління проектом на рівні призначення завдань, відстеження прогресу та зміни статусів.
- Отримання сповіщень у реальному часі про нові запити на участь у проекті.
- Надання можливості залишати зворотний зв’язок (фідбек) щодо роботи розробників або менеджерів, що сприяє підвищенню якості командної взаємодії та відповідальності.

- Забезпечення окремого функціоналу для адміністратора, який має розширені повноваження щодо керування всією платформою.

3.3. Тестування програми

3.3.1 Види тестувань

Існує кілька основних підходів до перевірки якості програмного забезпечення, кожен із яких орієнтується на свій рівень деталізації та мету.

На найнижчому рівні знаходиться модульне тестування – перевірка окремих функцій або методів у відриві від усього проекту. Такий підхід дозволяє переконатися, що кожен компонент виконує свою задачу згідно з очікуваннями, і швидко виявити помилки ще на етапі реалізації бізнес-логіки.

Кроком вище розташоване інтеграційне тестування, яке оцінює взаємодію між двома й більше модулями. Тут перевіряють, як об’єднані частини системи працюють разом: чи коректно передаються дані, чи не виникають конфлікти залежностей, чи відпрацьовують транзакції в єдиному потоці.

Далі розгортається системне тестування – воно охоплює всю програму цілком і перевіряє її відповідність функціональним вимогам. У ході таких випробувань тестувальники моделюють реальні сценарії використання, перевіряючи, що всі підсистеми коректно обробляють вхідні дані, успішно завершують операції й видають очікуваний результат.

Прийомне тестування, або тестування користувацьких сценаріїв, здійснюється вже замовником або кінцевими користувачами. Воно служить підтвердженням того, що система відповідає бізнес-вимогам, зручна в експлуатації та готова до запуску в продакшн-середовище.

Окремо виділяють нефункціональні види тестування. Тестування продуктивності перевіряє, чи витримує система заплановані навантаження, чи не погіршується час відповіді при збільшенні числа одночасних користувачів. Безпекове тестування шукає вразливості, спроби SQL-ін’єкцій або XSS-атак, щоб гарантувати захист конфіденційних даних. Тестування на зручність використання (usability)

оцінює інтерфейс і зручність взаємодії, а стрес-тестування змушує систему працювати під максимальним навантаженням, щоб виявити межі її можливостей.

3.3.2 Модульне тестування (Unit testing)

Модульне тестування зосереджене на найменших одиницях коду – окремих методах, функціях або класах – і покликане перевірити їхню правильність незалежно від решти системи. Під час розробки кожен модуль супроводжується наборами тестів, які автоматично виконуються після внесення змін у код. Завдяки цьому можна упевнитися, що функціонал, закладений у конкретному компоненті, працює саме так, як передбачено вимогами й дизайном, і не порушується внаслідок рефакторингу або додавання нової логіки.

Ключовим елементом модульного тестування є ізоляція перевірюваного коду від зовнішніх залежностей. Для цього застосовують заглушки (stubs), фейкові об’єкти або мок-об’єкти (mocks), які імітують поведінку інших компонентів, з якими модуль взаємодіє – наприклад, сервіси доступу до бази даних або зовнішні API. Такий підхід гарантує, що тест фокусується виключно на внутрішній логіці модуля й не «ліматиме» на недоступність зовнішніх ресурсів або змінну поведінку зовнішньої системи.

У практиці розробки модульні тести зазвичай організовують таким чином, щоб вони виконувалися автоматично в рамках процесів CI/CD. Інструменти на кшталт xUnit, NUnit чи MSTest у середовищі .NET дозволяють легко писати, групувати й запускати такі тести, а також інтегруються з конвеєрами побудови проекту. Якщо якийсь тест не проходить, система збирає детальний звіт про збій, що дозволяє розробникам оперативно виявити й виправити проблему ще до потрапляння коду в загальну гілку або на продакшн.

Перевагою модульного тестування є швидкий зворотний зв’язок: тести виконуються за лічені секунди, і розробник миттєво дізнається, чи не порушив він існуючий функціонал. Це стимулює сміливіше проведення змін – рефакторинг, оптимізацію та додавання нових можливостей – без страху зламати вже реалізовані частини системи. В результаті, зростає стабільність і підтримуваність коду, а співпраця в команді стає більш безпечною та передбачуваною.

У рамках перевірки коректності роботи механізмів автентифікації та авторизації я реалізував набір юніт-тестів (рис. 3.3) для відповідних ендпоінтів Microsoft Identity, використовуючи фреймворк NUnit. Ці тести автоматично імітують виклики реєстрації, входу в систему, підтвердження електронної пошти та генерації JWT-токенів, перевіряючи, що кожен сценарій обробляється відповідно до очікувань і забезпечує належний рівень безпеки.

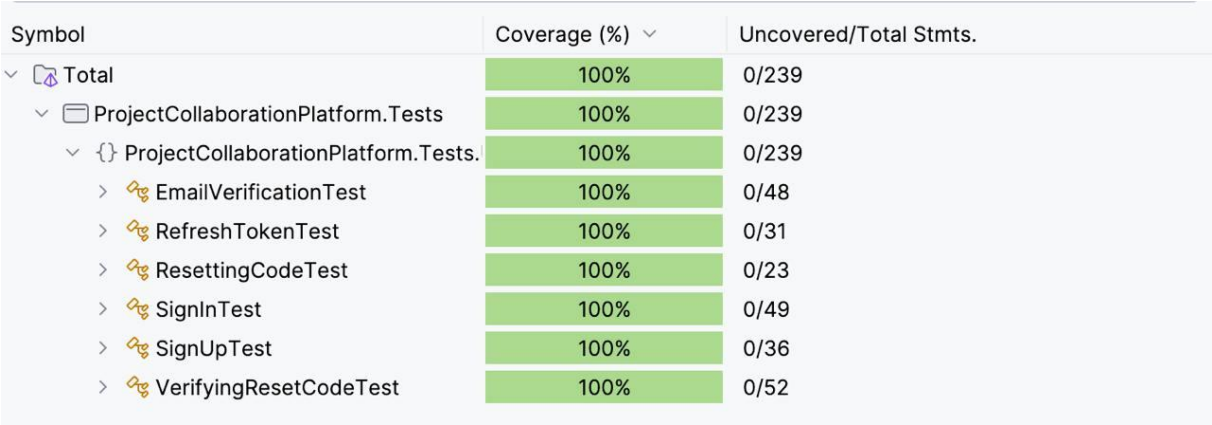


Рисунок 3.3 – Покриття коду юніт-тестами

3.3. Робота з функціоналом системи

У межах створеного застосунку передбачено окремі сторінки для авторизації (рис. 3.4) та реєстрації користувачів (рис. 3.5), що є невід'ємною частиною будь-якої системи з керуванням доступом. Дані сторінки забезпечують безпечний вхід до системи та створення нового облікового запису з подальшим доступом до функціоналу відповідно до призначеної ролі.

На сторінці реєстрації реалізовано можливість вибору ролі користувача за допомогою перемикача, що дозволяє одразу визначити, яким буде призначення облікового запису в системі – наприклад, проектний менеджер або розробник. Це рішення спрощує процес початкового налаштування профілю, забезпечує більш гнучкий розподіл прав доступу та дає змогу надалі адаптувати інтерфейс і можливості під потреби кожної ролі.

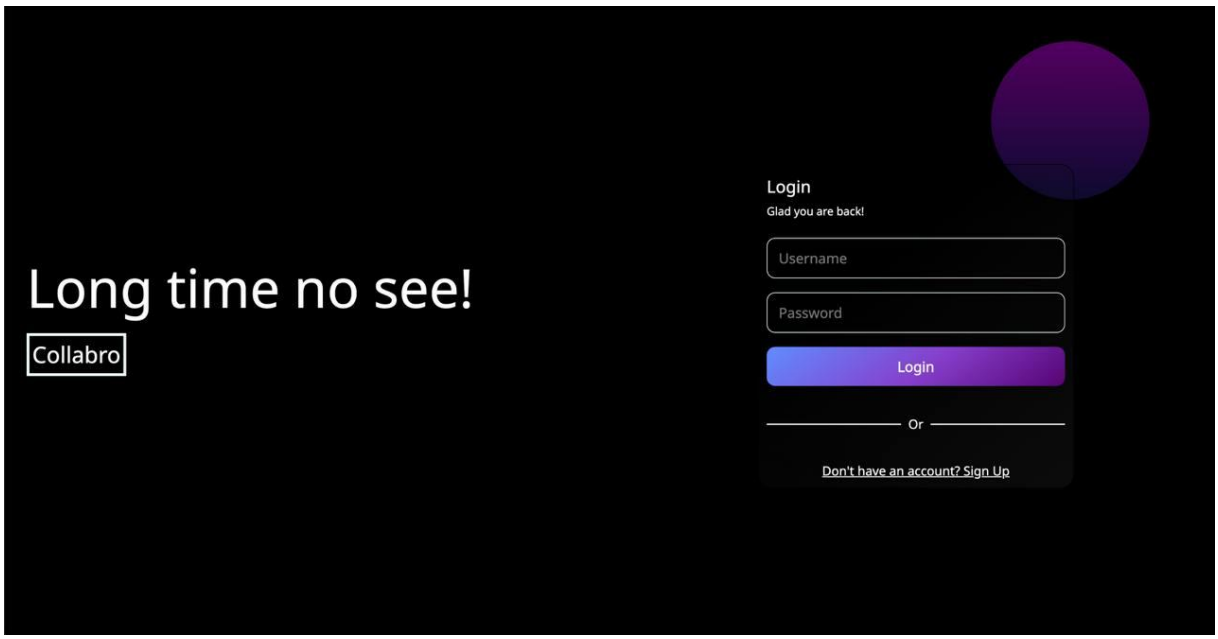


Рисунок 3.4 – Сторінка для авторизації

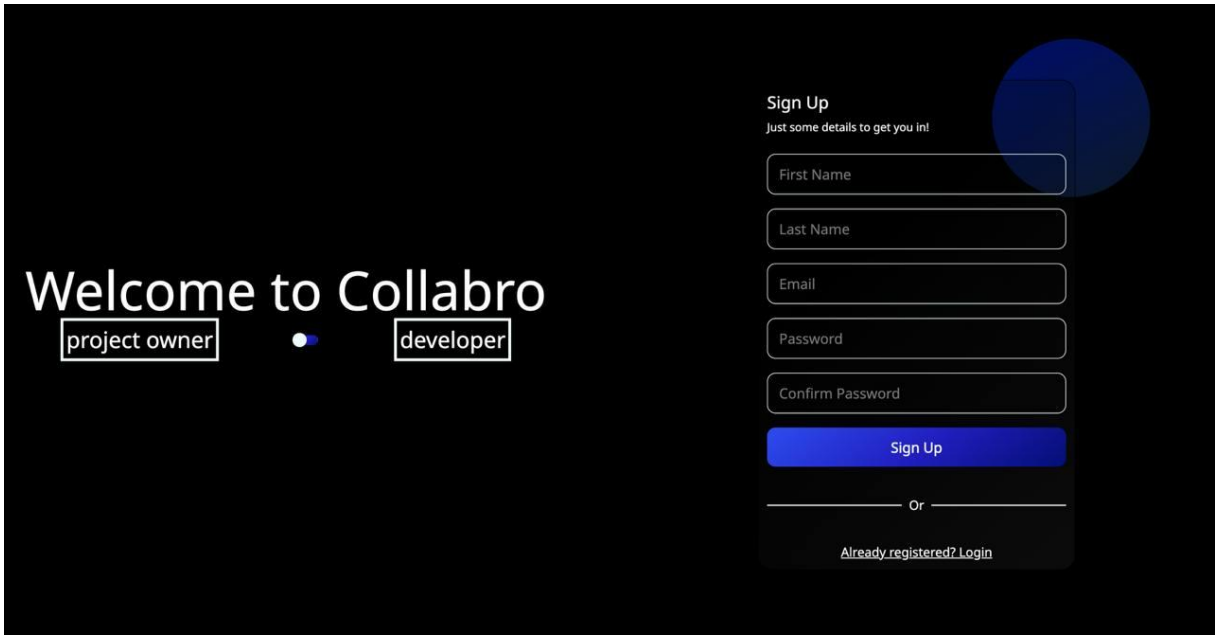


Рисунок 3.5 – Сторінка для реєстрації

У розробленому застосунку передбачено окремі профільні сторінки для кожного типу користувачів, при цьому їхній зміст та функціональність залежать від обраної під час реєстрації ролі. Такий підхід забезпечує адаптацію інтерфейсу до конкретних потреб користувача та підвищує зручність у взаємодії із системою.

Сторінка профілю проектного менеджера (рис. 3.6) містить розділ із персональними даними, включаючи ім’я, електронну пошту, а також інформацію про адресу проживання. Ці дані слугують основою для ідентифікації користувача та його участі в керуванні проектами.

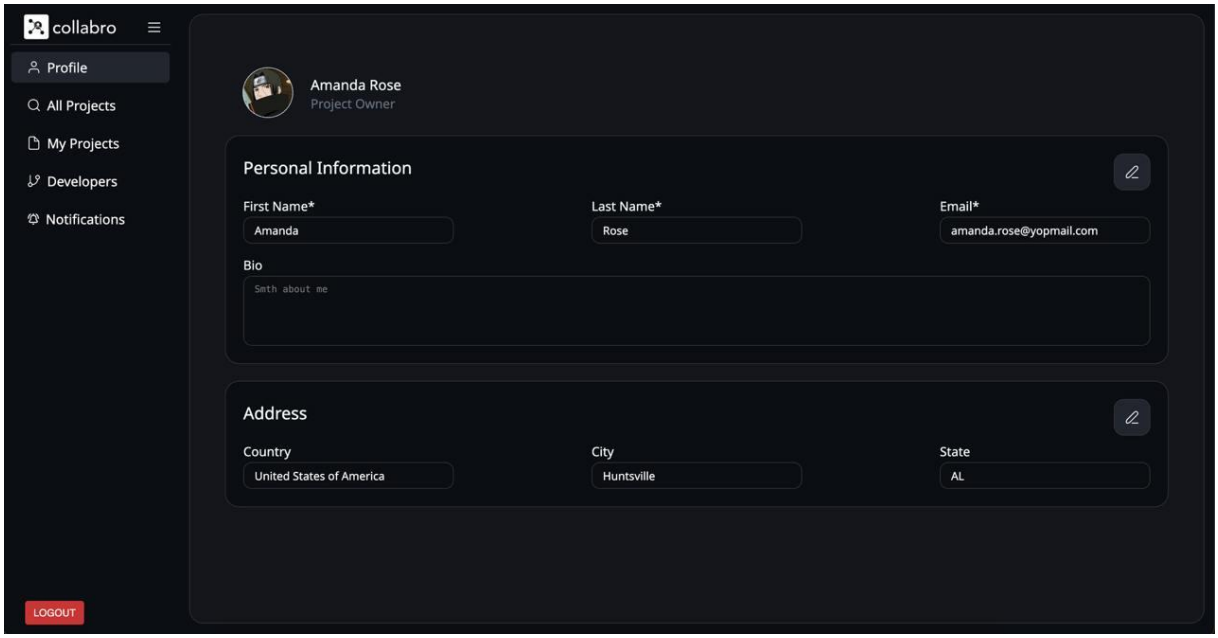


Рисунок 3.6 – Сторінка профілю проектного менеджера

Для розробника структура профілю (рис. 3.7) доповнена функціоналом, що дозволяє зазначити перелік власних технологій і фреймворків, з якими він працює. Цей список не лише зберігається, а й відображається безпосередньо в інтерфейсі профілю, що дає змогу проектним менеджерам переглядати технічні компетенції потенційних учасників проектів і формувати команди з урахуванням їхніх навичок.

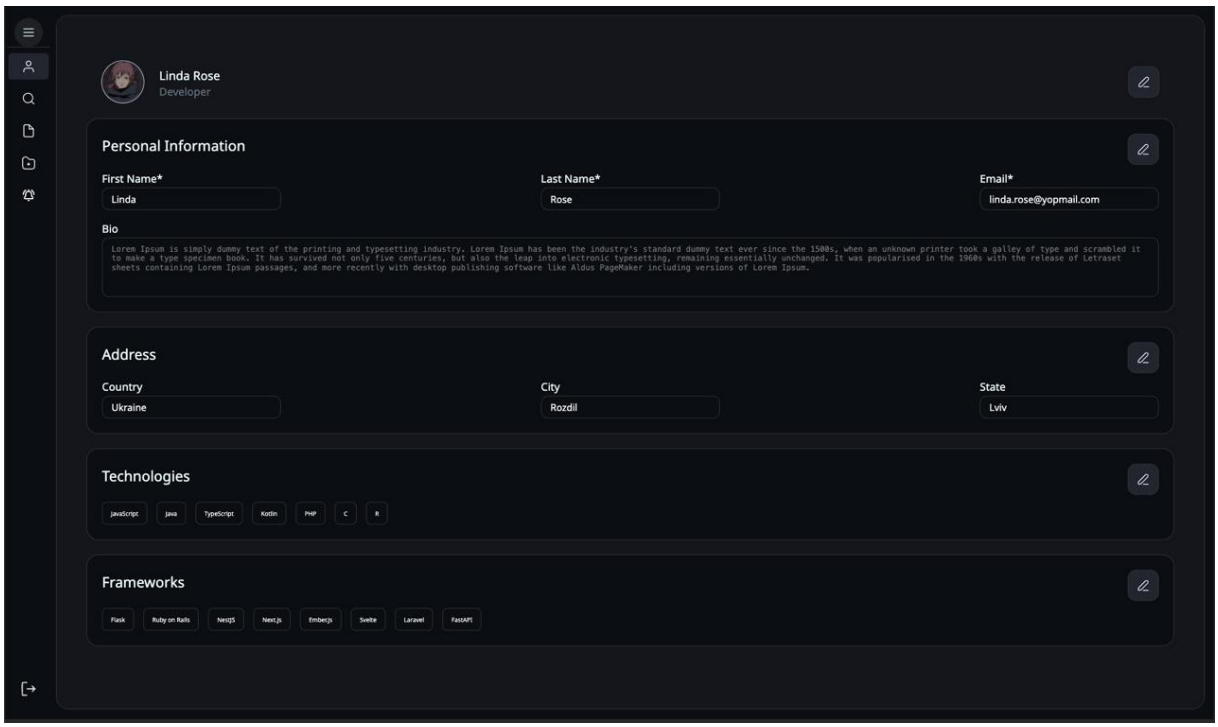


Рисунок 3.7 – Сторінка профілю розробника

Однією з ключових сторінок у платформі є огляд проекту (рис. 3.8), де відображаються загальні показники, поточний статус основних завдань і стислий звіт про хід виконання. Для детального менеджменту передбачені спеціальні розділи: дошка завдань (рис. 3.9), яка дозволяє переглядати та змінювати статус кожного завдання в режимі Kanban; сторінка запитів на приєднання до проекту (рис. 3.10), де проектний менеджер може схвалювати або відхиляти заявки розробників; а також сторінка команди проекту (рис. 3.11), на якій представлений повний список учасників з їхніми ролями та контактними даними. Така структура інтерфейсу забезпечує зручну навігацію між оглядом, контролем поточних завдань та управлінням ресурсами команди.

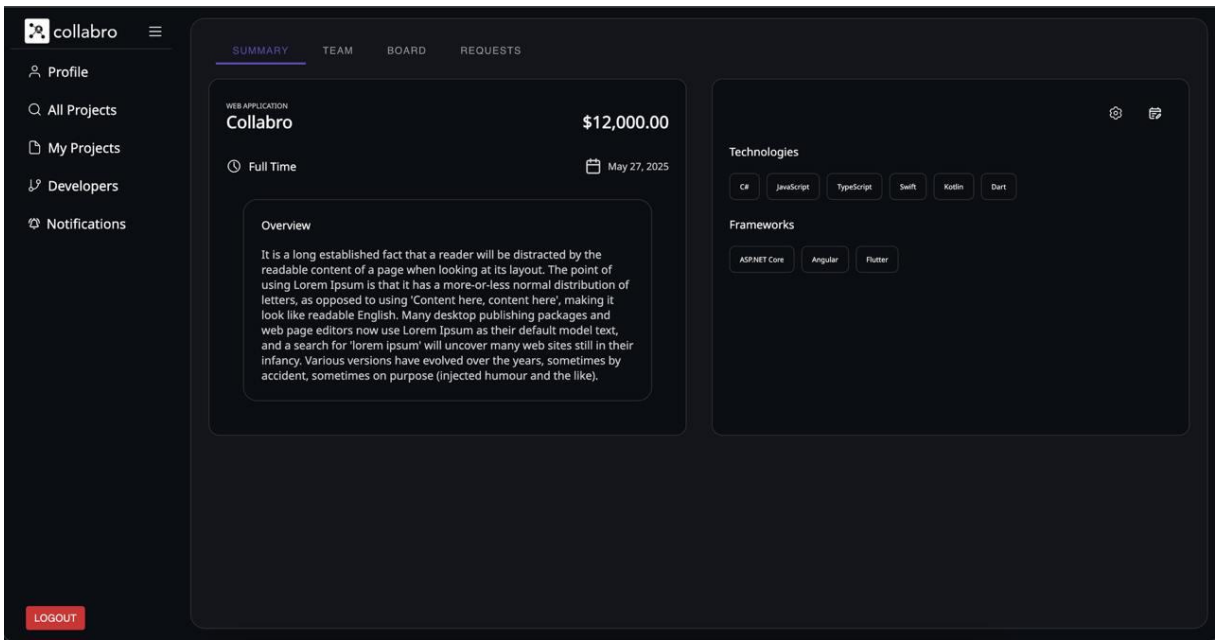


Рисунок 3.8 – Сторінка огляду проекту

У межах реалізації дошки завдань було впроваджено механізм контролю зміни статусів за допомогою state-машини. Це дозволило чітко обмежити допустимі переходи між етапами роботи, наприклад, заборонити перенесення задачі напряму з етапу "To Do" до "QA", минаючи розробку. Такий підхід гарантує логічну послідовність виконання завдань та зменшує ймовірність помилок у процесі управління проектом.

Окрім цього, було додано функціональність репорту дошки, яка дає змогу переглянути повний стан виконання завдань. У звіті відображається поточний статус кожної задачі та кількість розробників, які над нею працюють. Це значно покращує прозорість процесу та дозволяє менеджеру ефективніше керувати навантаженням команди.

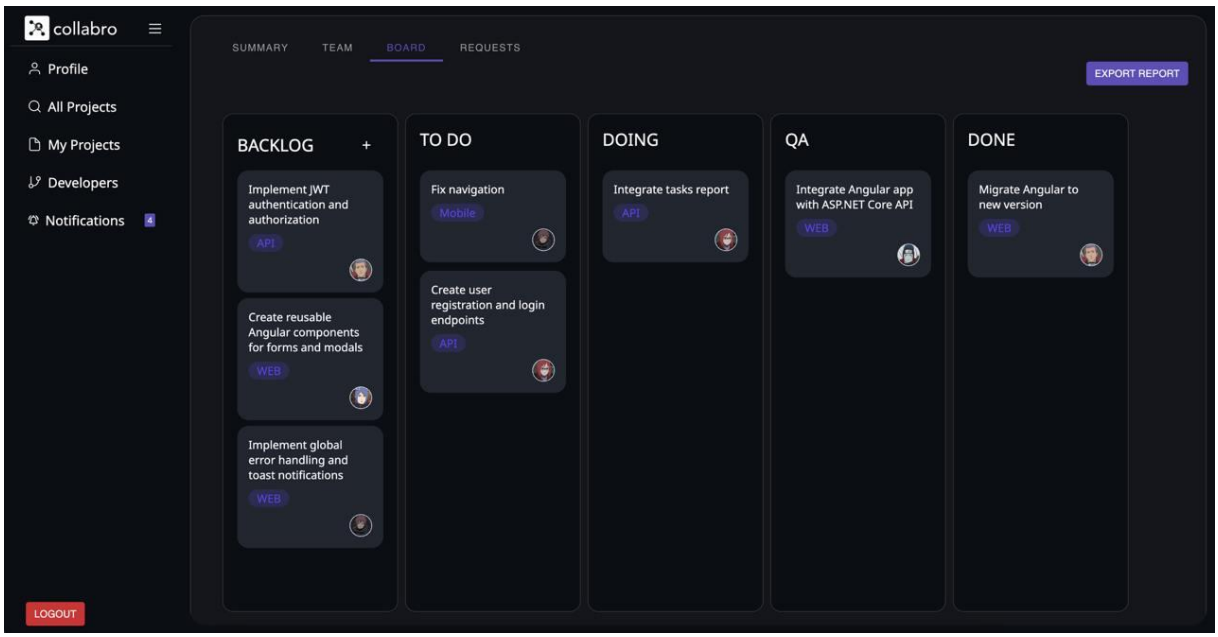


Рисунок 3.9 – Сторінка дошки проекту

На сторінці запитів на приєднання до проекту проектний менеджер має змогу не тільки переглянути саму заявку, а й ознайомитися з детальним профілем розробника, у якому містяться його персональні дані, портфоліо, перелік технологій та фреймворків. Крім того, поряд із профілем доступні фідбеки від інших менеджерів про співпрацю з цим спеціалістом, що допомагає скласти об’єктивну картину його професійних якостей. На основі цієї інформації менеджер може прийняти або відхилити запит на участь у проекті, забезпечуючи команді найкращий склад.

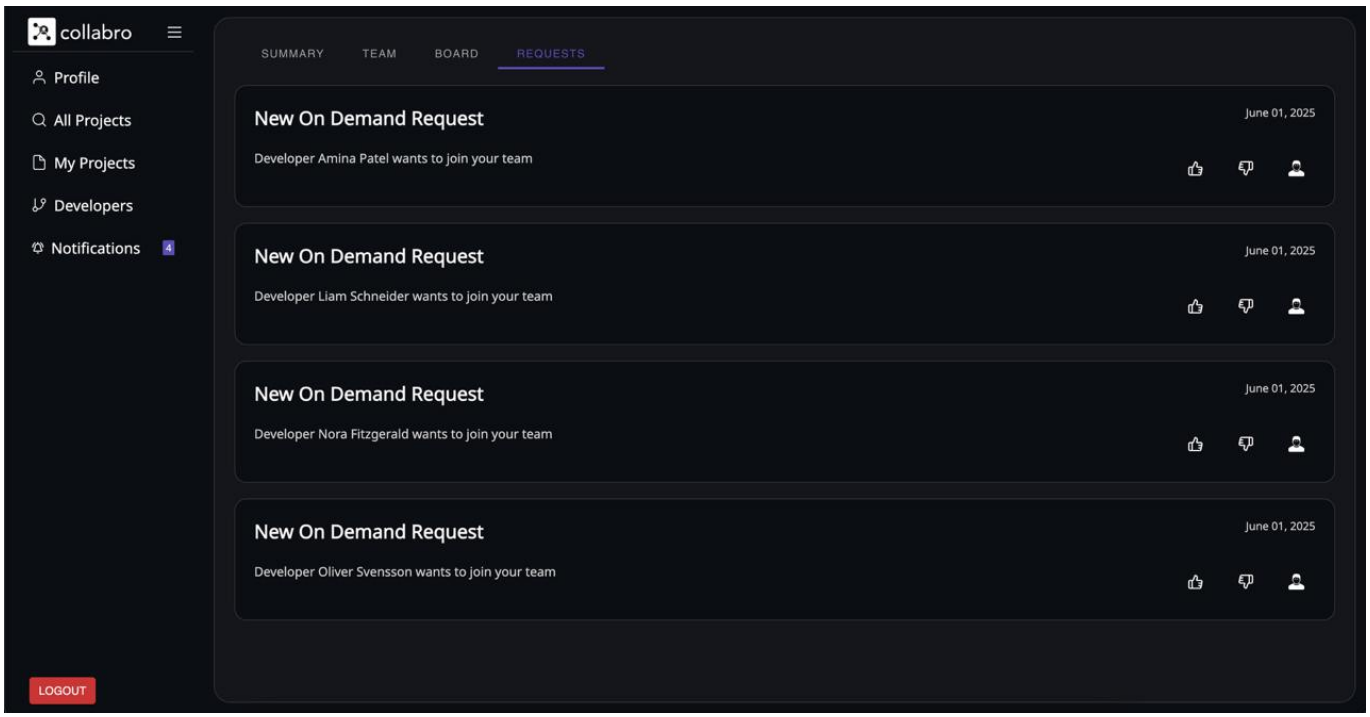


Рисунок 3.10 – Сторінка запитів на приєднання до проекту

На сторінці зі списком розробників, які беруть участь у проекті, представлено стислу інформацію про кожного учасника. Це дає змогу швидко ознайомитися з ключовими відомостями без необхідності переходу на повний профіль. У разі потреби проектний менеджер може переглянути детальний профіль розробника для оцінки його навичок. Крім того, реалізована можливість виключення розробника з команди, якщо цього вимагає ситуація в межах керування складом проекту.

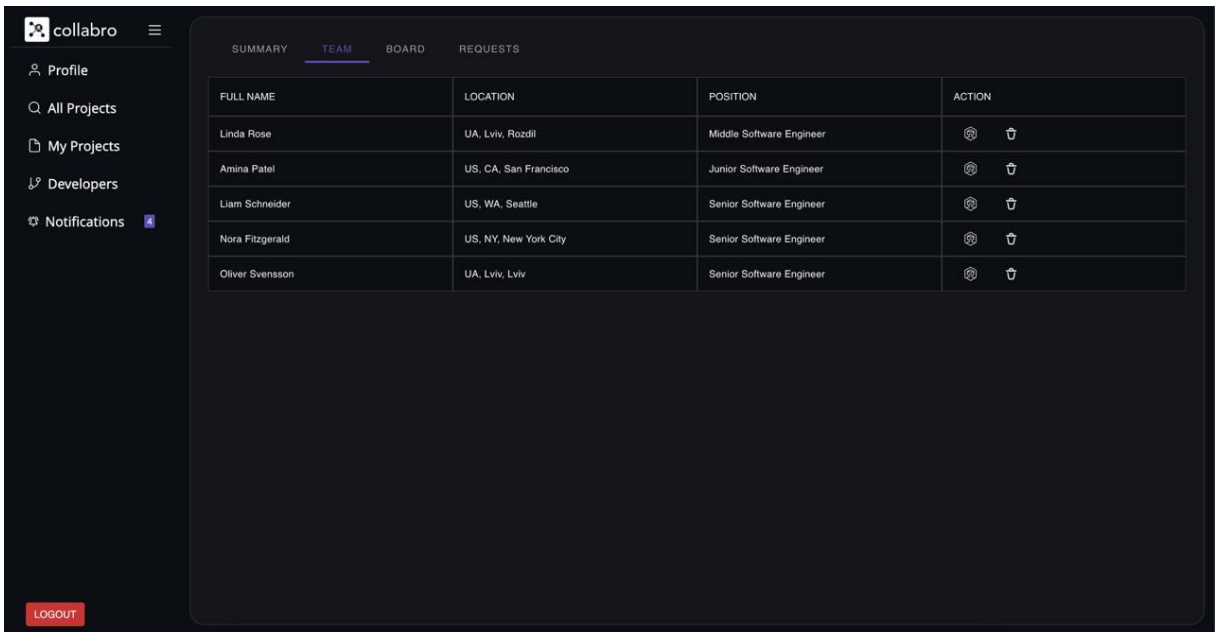


Рисунок 3.11 – Сторінка команди проекту

У платформі реалізовано окрему сторінку «Мої проекти» (рис. 3.12), призначену як для розробників, так і для проектних менеджерів. Розробник бачить тут лише ті проекти, до яких його було долучено, тоді як менеджер отримує перелік власних проектів. Для зручності навігації та швидкого пошуку потрібного проекту доступна система фільтрації: можна відсортувати проекти за їхнім статусом (наприклад, активні, завершені чи в очікуванні), обрати проекти за використаними технологіями або звузити список відповідно до типу оплати (фіксована ставка, погодинна оплата тощо). Такий підхід дозволяє кожному користувачу оперативно знаходити актуальні проекти та зосереджуватися на тих завданнях, які відповідають його ролі та інтересам.

Додатково реалізована автоматична пагінація на основі прокрутки сторінки, що означає поступове підвантаження проектів у процесі перегляду. Це рішення забезпечує оптимізацію продуктивності інтерфейсу, зменшує навантаження на браузер користувача та пришвидшує початкове завантаження сторінки, особливо при великій кількості проектів.

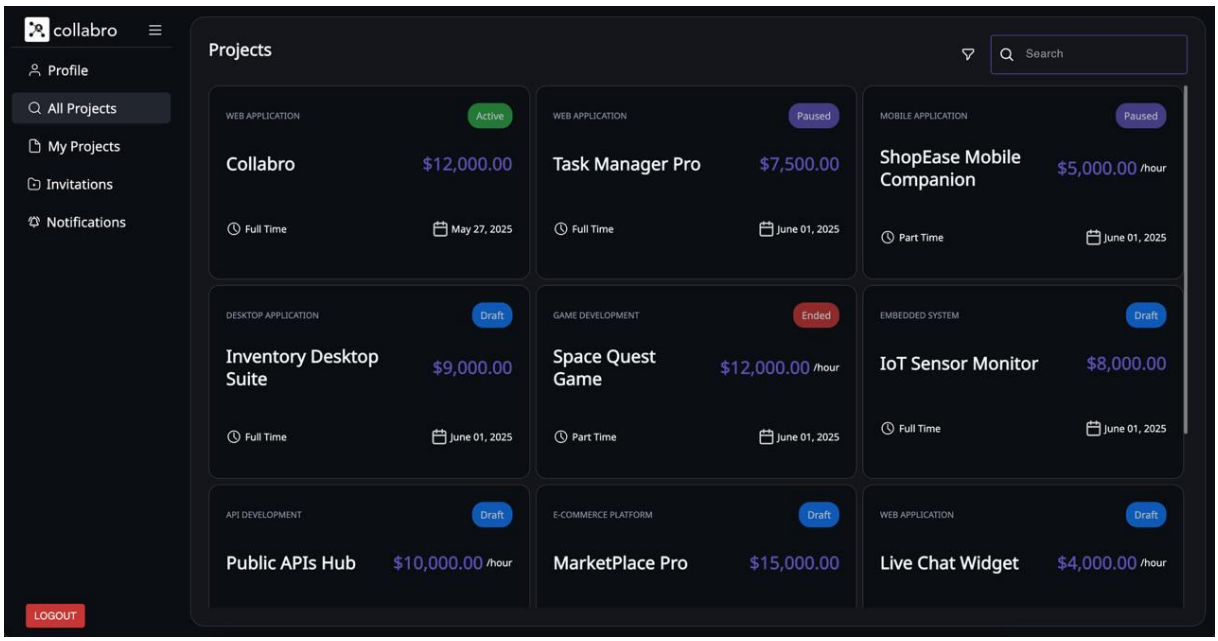


Рисунок 3.12 – Сторінка «Мої проекти»

Окрема увага була приділена механізму фідбеків для розробників, що дозволяє проектним менеджерам надавати зворотний зв’язок щодо виконаних завдань безпосередньо в інтерфейсі платформи (рис. 3.13). Важливою особливістю є те, що менеджер може змінити свій фідбек лише протягом 24 годин після його створення, що гарантує актуальність і відповідальність оцінок та уникає хаотичних правок у подальшому.

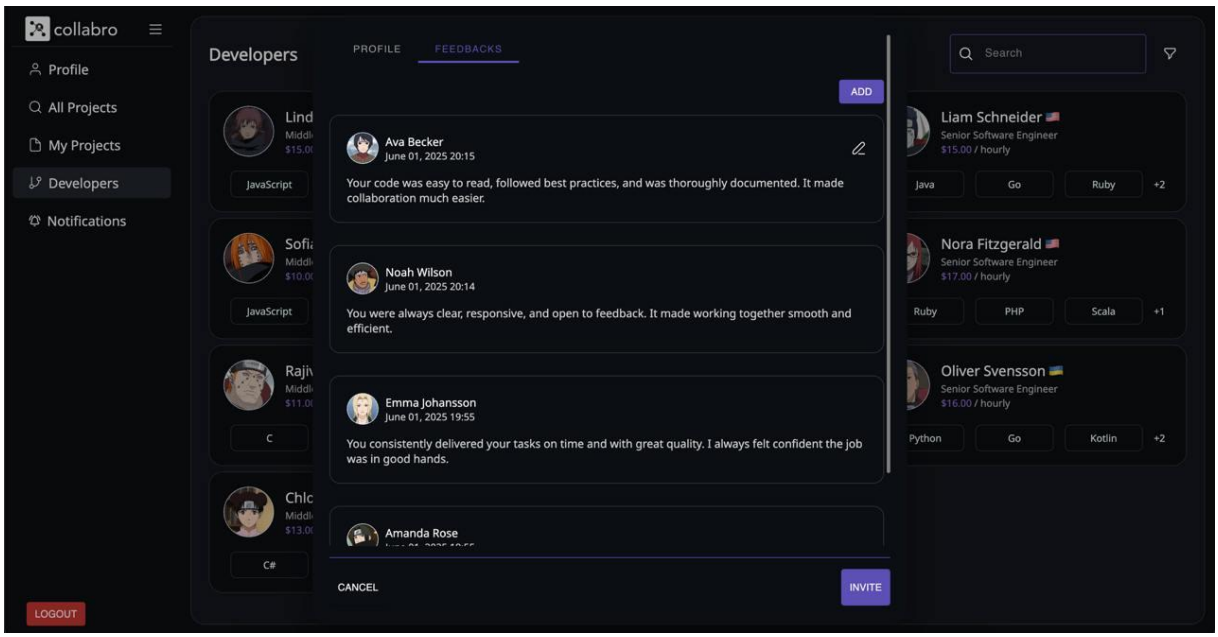


Рисунок 3.13 – Діалог списку фідбеків розробника

На додаток до звичних ролей у системі передбачено окрему панель адміністратора, в якій зібрано ключові показники роботи платформи. Адмін має доступ до сторінки статистики (рис. 3.14), де відображається загальна кількість зареєстрованих розробників, менеджерів та проектів, а також розподіл проектів за їхніми типами та статусами. Для аналізу динаміки розвитку створено графік, що демонструє кількість проектів, ініційованих у кожному місяці, що дозволяє відстежувати активність користувачів та планувати подальшу стратегію розвитку платформи.

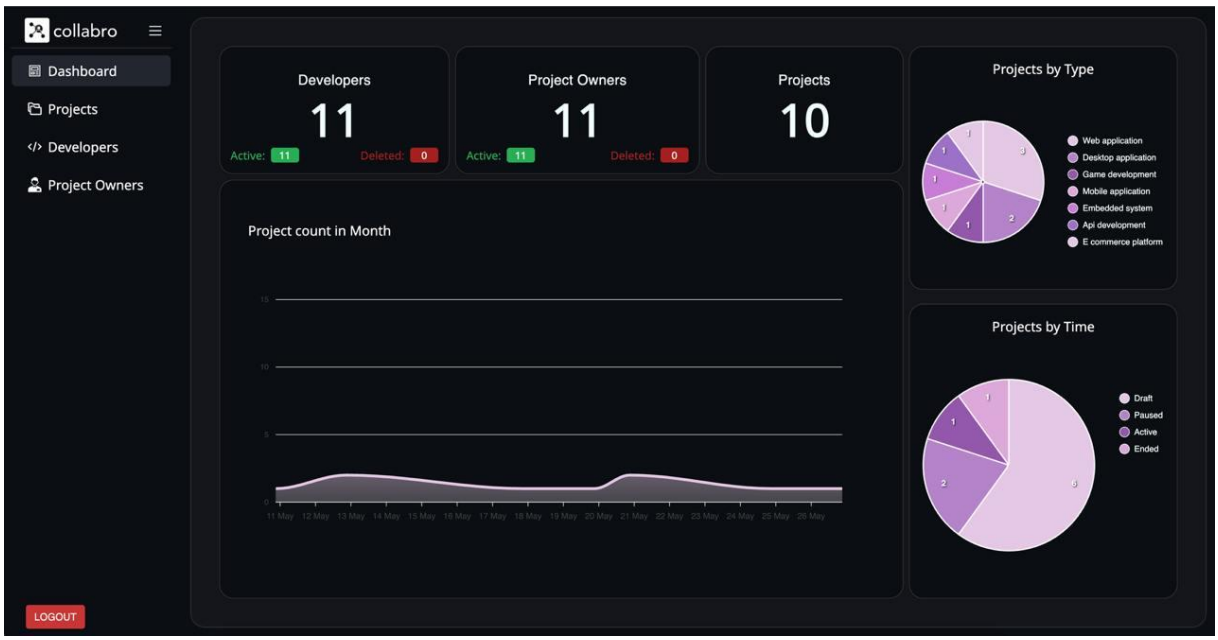


Рисунок 3.14 – Сторінка статистики Адміна

Окрім загальної статистики, адмін має доступ до сторінки зі списком усіх проектів (рис. 3.15), де представлено ключову інформацію про кожен із них: назву, власника, статус, використовувані технології та дати початку й завершення. Це дає змогу швидко перевіряти, що під час розробки вся діяльність відповідає внутрішнім правилам та політикам платформи, і вчасно виявляти будь-які невідповідності чи ризики неправомірного використання.

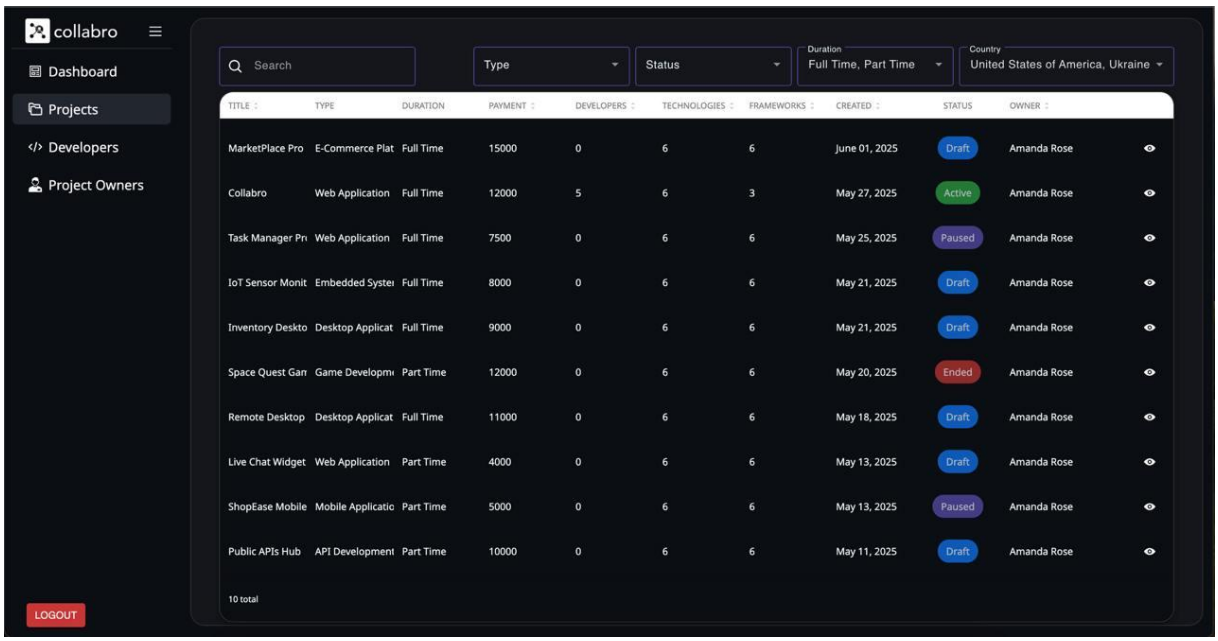


Рисунок 3.15 – Сторінка зі списком усіх проектів

На сторінці списку користувачів адміністратор може переглянути повний перелік розробників (рис. 3.16) та проектних менеджерів (рис. 3.17) разом із їхніми основними даними, а також зі списками отриманих і написаних фідбеків. Для кожного профілю відображаються контактні відомості, кількість проектів у роботі та історія взаємодії з іншими користувачами. У разі потреби адміністратор має можливість здійснити програмне видалення будь-якого облікового запису, що призведе до деактивації користувача без фізичного видалення його даних з бази, зберігаючи історію дій для подальшого аудиту.

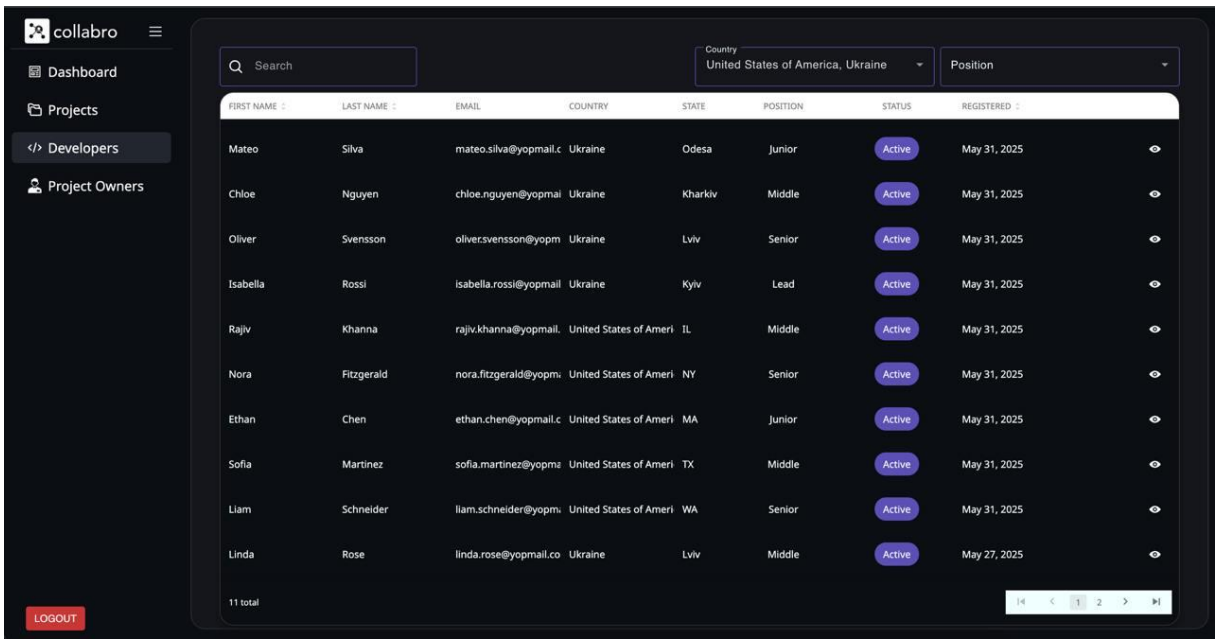


Рисунок 3.16 – Сторінка з переліком розробників

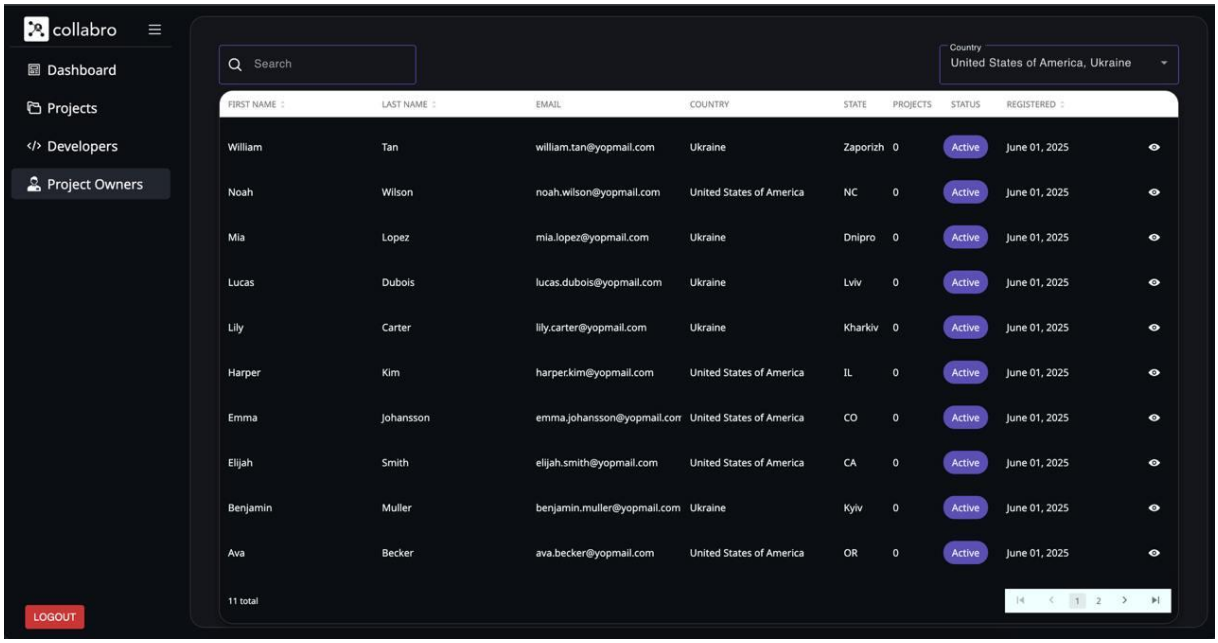


Рисунок 3.17 – Сторінка з переліком проектних менеджерів

ВИСНОВКИ

У процесі створення платформи для онлайн-співпраці в командних IT-проектах було здійснено всебічне дослідження сучасних підходів до організації розробки, менеджменту проектів і використання вебтехнологій у цих процесах. Аналіз існуючих рішень дав змогу виявити потребу у зручному, адаптивному середовищі, яке б поєднувало всі ключові аспекти управління проектами, роботи розробників і комунікації між учасниками команди. Як наслідок було реалізовано комплексну систему, яка дозволяє ефективно керувати проектами від етапу створення до їхнього завершення.

Архітектурно застосунок побудовано за слоїстою моделлю, що забезпечило чітке розділення відповідальностей, зручність у масштабуванні, тестуванні та підтримці. Значну увагу було приділено безпеці – інтеграція Microsoft Identity дала змогу реалізувати надійний механізм авторизації та аутентифікації з підтримкою токенів, ролей і прав доступу. У поєднанні з REST API та використанням сучасних бібліотек і фреймворків це надало системі високої гнучкості та розширюваності.

З боку клієнтської частини платформа реалізована на Angular, що дозволило забезпечити адаптивність інтерфейсу до користувачів та операційних систем. Особлива увага приділена функціоналу з боку адміністратора, який має доступ до статистики, керування користувачами та моніторингу проектної діяльності.

У технічній реалізації були використані мови програмування C# та TypeScript, фреймворки ASP.NET Core і Angular, бібліотеки для створення токенів, побудови UI, обробки стану та тестування (NUnit). Також реалізовано автоматичне тестування окремих модулів.

У підсумку, розроблена система демонструє повноцінне рішення для організації командної роботи над IT-проектами. Вона є прикладом застосування сучасних технологій, архітектурних підходів і практик у створенні масштабованого, зручного та функціонального вебзастосунку. Така платформа здатна значно підвищити ефективність команди, спростити комунікацію між учасниками та забезпечити прозорість усіх етапів розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

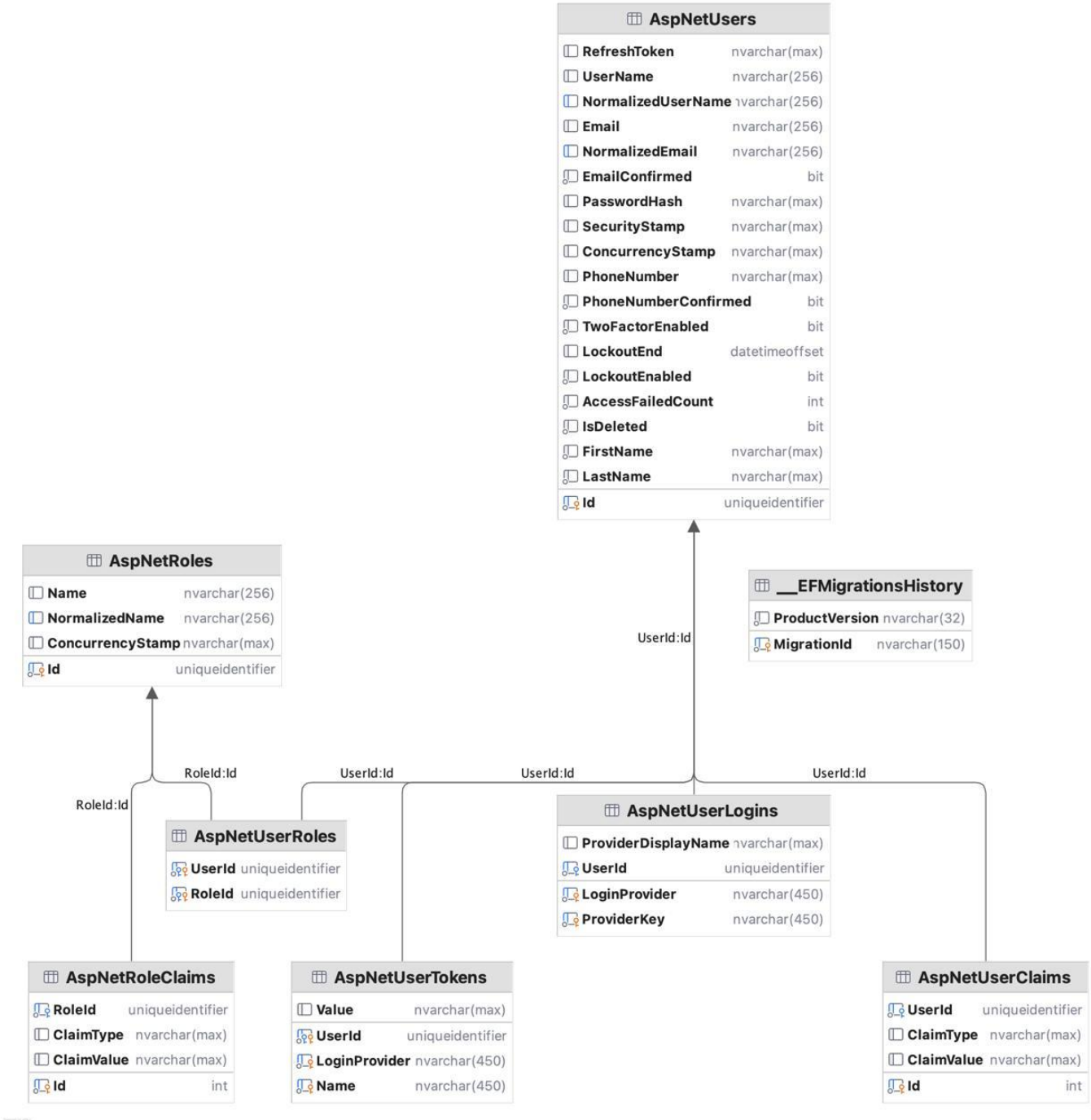
1. Comparing team collaboration software [Електронний ресурс] // Eficode. – 2024. – Режим доступу до ресурсу: https://www.eficode.com/blog/comparing-team-collaboration-software.
2. The best free project management software in 2025 [Електронний ресурс] // Zapier. – 2024. – Режим доступу до ресурсу: https://zapier.com/blog/free-project-management-software.
3. Types of Databases [Електронний ресурс] // DataCamp. – 2024. – Режим доступу до ресурсу: https://www.datacamp.com/blog/types-of-databases-overview.
4. Understanding The Different Types Of Databases & When To Use Them [Електронний ресурс] // Rivery. – 2025. – Режим доступу до ресурсу: https://rivery.io/data-learning-center/database-types-guide.
5. Sequence Diagrams - Unified Modeling Language (UML) [Електронний ресурс] // GeeksForGeeks. – 2025. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams.
6. ASP.NET Core Basics: Organizing Projects with Architectural Patterns [Електронний ресурс] // Telerik. – 2024. – Режим доступу до ресурсу: https://www.telerik.com/blogs/asp-net-core-basics-organizing-projects-architectural-patterns.
7. WebSockets explained: What they are and how they work [Електронний ресурс] // Ably. – 2025. – Режим доступу до ресурсу: https://ably.com/topic/websockets.
8. Using Entity Framework Core Code First Approach [Електронний ресурс] // C# Corner. – 2020. – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/using-entity-framework-core.
9. Building Authentication Service using Microsoft ASP.NET Core Identity and .NET 8 [Електронний ресурс] // Medium. – 2024. – Режим доступу до ресурсу:

https://medium.com/aeturnuminc/building-authentication-service-using-microsoft-asp-net-core-identity-and-net-8-3fea53cfe0f8.

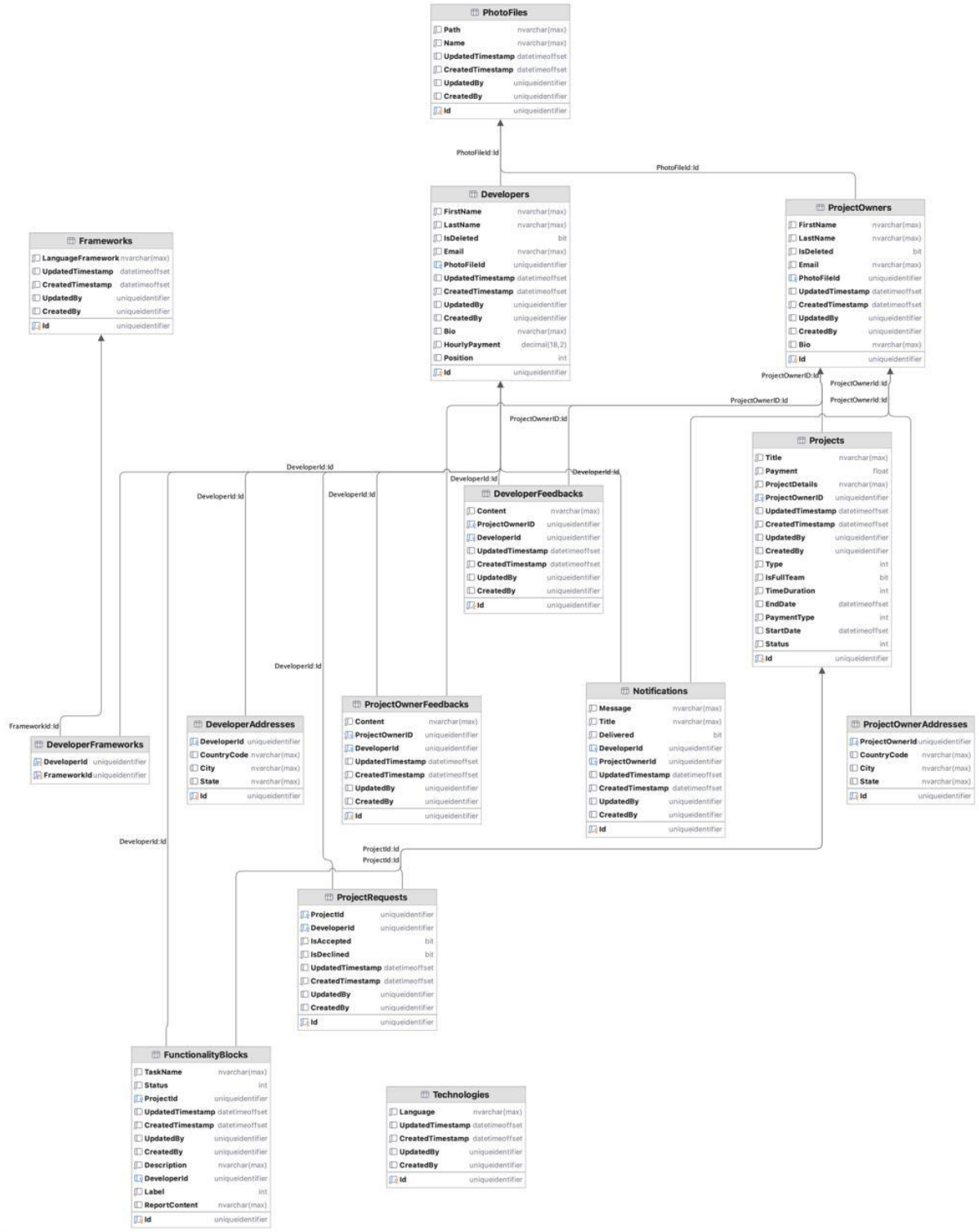
10. Angular 18 / .NET 8 SignalR Usage [Електронний ресурс] // Medium. – 2024. – Режим доступу до ресурсу: https://medium.com/@ferhatblnk/angular-18-net-8-signalr-usage-6d0186906946.
11. Angular Material UI component library [Електронний ресурс] // Angular Material. – 2024. – Режим доступу до ресурсу: https://material.angular.dev.
12. Using ApexCharts in Angular [Електронний ресурс] // ApexCharts. – 2024. – Режим доступу до ресурсу: https://apexcharts.com/docs/angular-charts.
13. Angular Documentation [Електронний ресурс] // Angular. – 2024. – Режим доступу до ресурсу: https://angular.dev.
14. HTTP Interceptors in Angular [Електронний ресурс] // Medium. – 2023. – Режим доступу до ресурсу: https://medium.com/@jaydeepvpatil225/http-interceptors-in-angular-6e9891ae0538.
15. Creating Background Routines with Hangfire in ASP.NET Core [Електронний ресурс] // Telerik. – 2024. – Режим доступу до ресурсу: https://www.telerik.com/blogs/creating-background-routines-hangfire-aspnet-core.
16. Designing Web APIs: Building APIs That Developers Love 1st Edition // Amir ShevatBrenda JinSaurabh Sahni – O'Reilly – 2018, – 163 с.
17. Restful Web API Patterns and Practices Cookbook. Connecting and Orchestrating Microservices and Distributed Data. 1st Edition // Mike Amundsen – O'Reilly – 2022, – 4 с.

ДОДАТОК А

Діаграма зв’язків сутностей *Collabro-Identity*:

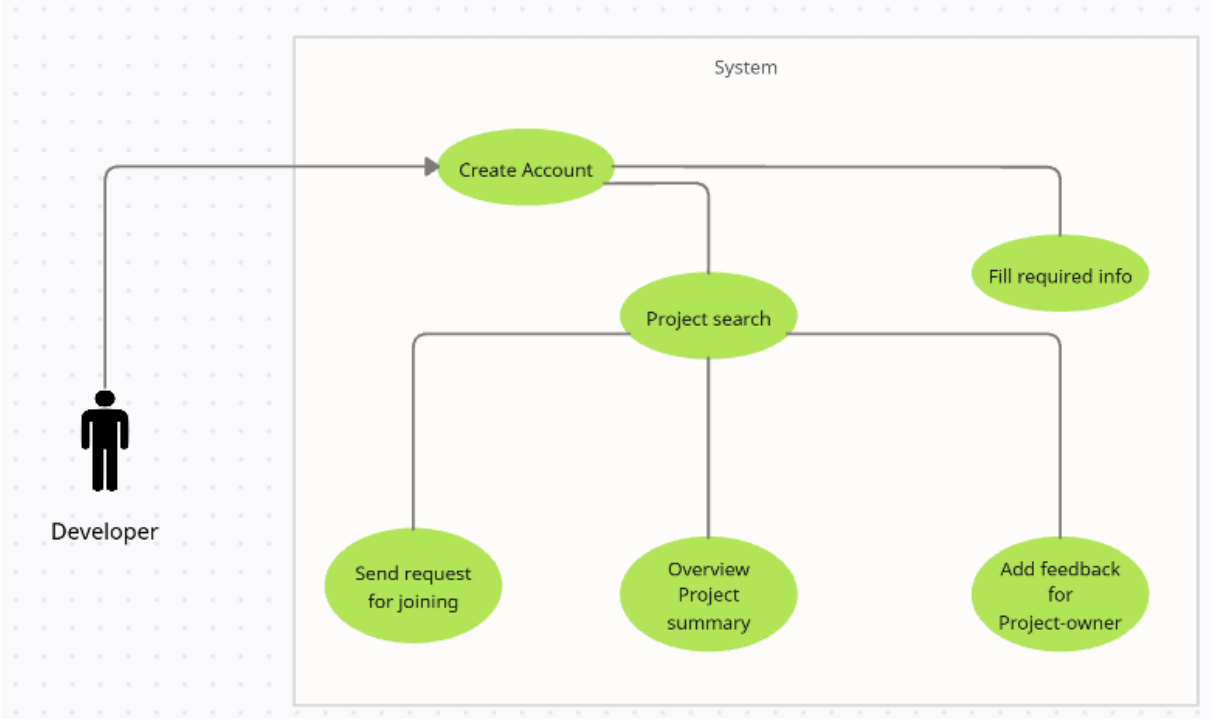


Діаграма зв’язків сутностей *Collabro-API*:

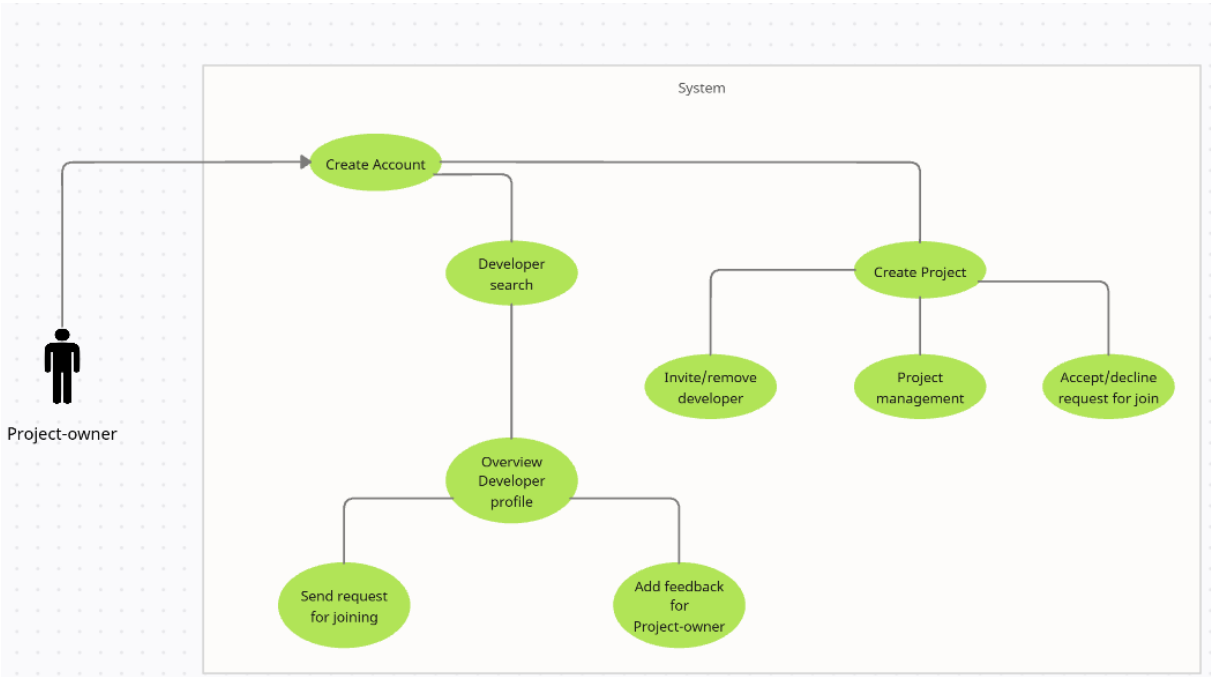


ДОДАТОК Б

Use case діаграма ролі *Розробник*:



Use case діаграма ролі *Проектний менеджер*:



ДОДАТОК В

Таблиця сутності *Notifications*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	Message	nvarchar(MAX)	☐	☐	
=	Title	nvarchar(MAX)	☐	☐	
=	Delivered	bit	☐	☐	
=	DeveloperId	uniqueidentifier	☐	☑	
=	ProjectOwnerId	uniqueidentifier	☐	☑	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	

Таблиця сутності *Notifications*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	ProjectOwnerId	uniqueidentifier	☐	☐	
=	CountryCode	nvarchar(MAX)	☐	☑	
=	City	nvarchar(MAX)	☐	☑	
=	State	nvarchar(MAX)	☐	☑	

Таблиця сутності *Notifications*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	DeveloperId	uniqueidentifier	☐	☐	
=	CountryCode	nvarchar(MAX)	☐	☑	
=	City	nvarchar(MAX)	☐	☑	
=	State	nvarchar(MAX)	☐	☑	

Таблиця сутності *Notifications*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	ProjectID	uniqueidentifier	☑	☐	
=	TechnologyID	uniqueidentifier	☑	☐	

Таблиця сутності *Notifications*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	LanguageFramework	nvarchar(MAX)	☐	☐	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	

Таблиця сутності *Technologies*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	Language	nvarchar(MAX)	☐	☐	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	

Таблиця сутності *PhotoFiles*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	Path	nvarchar(MAX)	☐	☐	
=	Name	nvarchar(MAX)	☐	☐	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	

Таблиця сутності *DeveloperFeedbacks*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	Content	nvarchar(MAX)	☐	☐	
=	ProjectOwnerID	uniqueidentifier	☐	☐	
=	DeveloperId	uniqueidentifier	☐	☐	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	

Таблиця сутності *ProjectOwnerFeedbacks*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	Id	uniqueidentifier	☑	☐	
=	Content	nvarchar(MAX)	☐	☐	
=	ProjectOwnerID	uniqueidentifier	☐	☐	
=	DeveloperId	uniqueidentifier	☐	☐	
=	UpdatedTimestamp	datetimeoffset(7)	☐	☑	
=	CreatedTimestamp	datetimeoffset(7)	☐	☐	
=	UpdatedBy	uniqueidentifier	☐	☑	
=	CreatedBy	uniqueidentifier	☐	☑	

Проміжна таблиця сутності *DeveloperFrameworks*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	DeveloperId	uniqueidentifier	☑	☐	
=	FrameworkId	uniqueidentifier	☑	☐	

Проміжна таблиця сутності *DeveloperTechnologies*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	DeveloperID	uniqueidentifier	☑	☐	
=	TechnologyID	uniqueidentifier	☑	☐	

Проміжна таблиця сутності *ProjectDevelopers*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	ProjectID	uniqueidentifier	☑	☐	
=	DeveloperID	uniqueidentifier	☑	☐	

Проміжна таблиця сутності *ProjectFrameworks*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	ProjectId	uniqueidentifier	☑	☐	
=	FrameworkId	uniqueidentifier	☑	☐	

Проміжна таблиця сутності *ProjectTechnologies*:

Move	Name	Type	Primary Key	Allow Nulls	Default Value
=	ProjectID	uniqueidentifier	☑	☐	
=	TechnologyID	uniqueidentifier	☑	☐	

ДОДАТОК Г

Сервіс *UserService*, що відповідає за обробку процесів авторизації та реєстрації користувачів у системі:

```
public class UserService(
    UserManager<User> userManager,
    RoleManager<IdentityRole<Guid>> roleManager,
    ApplicationDbContext context,
    ITokenGenerator tokenGenerator)
    : IUserService
{
    private readonly RoleManager<IdentityRole<Guid>> _roleManager = roleManager;


    public async Task<bool> AddUser(SignUpDTO userDTO)
    {
        var user = new User()
        {
            UserName = userDTO.Name,
            Email = userDTO.Email,
            FirstName = userDTO.FirstName,
            LastName = userDTO.LastName
        };
        var createdResult = await userManager.CreateAsync(user, userDTO.Password);
        if (createdResult.Succeeded)
        {
            var createdUser = await userManager.FindByEmailAsync(user.Email);
            var addRoleResult = await AddUserRole(createdUser, userDTO.RoleName);

            var isSent = await SendEmailVerification(createdUser);
            if (!isSent)
                await userManager.DeleteAsync(createdUser);
            return true;
        }
```

```
    else
    {
        throw new CustomApiException()
        {
            StatusCode = StatusCodes.Status500InternalServerError,
            Title = "Error creating user",
            Detail = "User doesn't created"
        };
    }
}

private async Task<string> AddUserRole(User user, string userRole)
{
    var result = await userManager.AddToRoleAsync(user, userRole);

    if (result.Succeeded)
    {
        var role = await userManager.GetRolesAsync(user);
        return role.First();
    }
    else
    {
        throw new CustomApiException()
        {
            StatusCode = StatusCodes.Status500InternalServerError,
            Title = "Error adding role",
            Detail = "Role doesn't added"
        };
    }
}

private Task<string> CallBackUrl(User user, string code)
{
    var ngrok = ConstantLink.ngrok;
    var callbackUrl = ngrok + "/api/User/EmailVerification" + $"?userId={user.Id}&code={code}";
```

```
    return Task.FromResult(callbackUrl);

}

public async Task<bool> CheckUserPassword(string email, string password)
{
    var userByEmail = await userManager.FindByEmailAsync(email);
    return await userManager.CheckPasswordAsync(userByEmail, password);
}

public async Task<bool> DeleteUser(string email)
{
    var user = await userManager.FindByEmailAsync(email);
    var result = await userManager.DeleteAsync(user);
    return result.Succeeded;
}

public Task<bool> IsUserExists(string email)
    => context.Users.AnyAsync(u => u.Email == email);

public async Task<UserDTO> GetUserById(string id)
{
    var user = await userManager.FindByIdAsync(id);

    return new UserDTO()
    {
        Id = user.Id,
        Email = user.Email,
    };
}

private async Task<bool> SendEmailVerification(User user)
{
    var code = await userManager.GenerateEmailConfirmationTokenAsync(user);
```

```
    var callBackUrl = await CallBackUrl(user, code);

    try
    {
        var email = new MimeMessage();
        email.From.Add(MailboxAddress.Parse("collabro.pcp@gmail.com"));
        email.To.Add(MailboxAddress.Parse(user.Email));
        email.Subject = "Email verification";
        email.Body = new TextPart(TextFormat.Html)
        {
            Text = EmailContentHelper.EmailVerification(user.UserName, callBackUrl)
        };

        using var smtp = new SmtpClient();
        await smtp.ConnectAsync("smtp.gmail.com", 587, SecureSocketOptions.StartTls);
        await smtp.AuthenticateAsync("collabro.pcp@gmail.com", "nqaw axfe aumh ifpr");

        await smtp.SendAsync(email);
        await smtp.DisconnectAsync(true);
        return true;

    }
    catch (Exception)
    {
        return false;
    }
}

public async Task<bool> SendPasswordResetCode(UserDTO userDTO)
{
    var user = new User()
    {
        Email = userDTO.Email,
    };
    var resetCode = await userManager.GeneratePasswordResetTokenAsync(user);
```

```
        var ngrok = ConstantLink.ngrok;
        var callbackUrl = ngrok + "/api/User/VerifyPasswordResetCode" +
$"?userId={userDTO.Id}&code={resetCode}";

        try
        {
            var email = new MimeMessage();
            email.From.Add(MailboxAddress.Parse("collabro.pcp@gmail.com"));
            email.To.Add(MailboxAddress.Parse(userDTO.Email));
            email.Subject = "Reset Password";
            email.Body = new TextPart(TextFormat.Html)
            {
                Text = EmailContentHelper.PasswordResetCode(callbackUrl)
            };

            using var smtp = new SmtpClient();
            await smtp.ConnectAsync("smtp.gmail.com", 587, SecureSocketOptions.StartTls);
            await smtp.AuthenticateAsync("collabro.pcp@gmail.com", "nqaw axfe aumh ifpr");

            await smtp.SendAsync(email);
            await smtp.DisconnectAsync(true);
            return true;

        }
        catch (Exception)
        {
            return false;
        }
    }

    public async Task<bool> VerifyEmail(UserDTO user, string code)
    {
        var userVerify = await userManager.FindByEmailAsync(user.Email);
        var result = await userManager.ConfirmEmailAsync(userVerify, code);
```

```
        return result.Succeeded;
    }

    public async Task<bool> VerifyPasswordResetCode(UserDTO userDTO, string password, string
newPassword)
    {

        var user = await userManager.FindByEmailAsync(userDTO.Email);
        var result = await userManager.ResetPasswordAsync(user, password, newPassword);

        return result.Succeeded;
    }

    public async Task<UserDTO> GetUserByEmail(string email)
    {
        var user = await context.Users
            .Where(i => i.Email == email)
            .Select(u => new UserDTO()
            {
                Id = u.Id,
                Email = u.Email,
                UserName = u.UserName,
            })
            .FirstOrDefaultAsync();
        return user;
    }

    public async Task<AuthenticationResponse> GenerateTokens(string email)
    {
        var user = await userManager.FindByEmailAsync(email);
        return await tokenGenerator.GenerateTokens(user);
    }
}
```