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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для управління IT-проєктами за методологією Kanban з використанням механізмів гейміфікації»

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

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

_____ Денис МАЛІНСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44
Денис МАЛІНСЬКИЙ

Керівник: Богдан ХУДІК
доктор філософії (PhD)
Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

_____Малінському Денису Вікторовичу_____

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для управління IT-проєктами за методологією Kanban з використанням механізмів гейміфікації» керівник кваліфікаційної роботи доктор філософії (PhD) Богдан ХУДІК, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

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

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи керування IT-проєктами за методологією Kanban, опис існуючих рішень, проєктування рішення, процес розробки рішення, тестування розробленого рішення.

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

1. Огляд та аналіз існуючих рішень керування IT-проєктами за методологією Kanban.
2. Проєктування web-застосунку для керування IT-проєктами за методологією Kanban з використанням механізмів гейміфікації

3. Опис програмної реалізації web-застосунку для керування ІТ-проєктами за методологією Kanban з використанням механізмів гейміфікації
4. Тестування web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до додатку.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Структура бази даних.
6. Архітектура додатку.
7. Діаграма класів основної логіки.
8. Мапа веб-застосунку.
9. Екранні форми.
10. Апробація результатів дослідження.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих аналогів керування ІТ-проєктами за методологією Kanban.	14.03-20.03.2025	
4	Проєктування web-застосунку для керування ІТ-проєктами за методологією Kanban. з використанням механізмів гейміфікації	21.03-31.03.2025	
5	Програмна реалізація web-застосунку	01.04-21.04.2025	
6	Тестування web-застосунку	22.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Денис МАЛІНСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Богдан ХУДІК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 67 стор., 26 табл., 66 рис., 20 джерел.

Мета роботи – покращення процесу керування IT-проєктами за допомогою методології Kanban шляхом додавання механізмів гейміфікації у процес роботи.

Об'єкт дослідження – процес керування IT-проєктами за допомогою методології Kanban з механізмами гейміфікації.

Предмет дослідження – веб-застосунок для керування IT-проєктами за методологією Kanban з механізмами гейміфікації

Короткий зміст роботи: В дипломній роботі було проаналізовано існуючі рішення веб-застосунків для керування IT-проєктами за методологією Kanban. Проаналізовано існуючі інструменти та технології розробки та виявлено їх переваги. Розроблено web-застосунок для керування IT-проєктами за методологією Kanban з механізмами гейміфікації. Було реалізовано ключові функціональні можливості, такі як: автентифікація, керування просторами, керування дошками, керування статусами, керування задачами, керування профілем, керування статистикою, отримання досягнень, механіка рівня профілю. Проведення тестування за допомогою юніт-тестів та ручного тестування. В роботі використано середовище розробки Visual Studio для мови програмування C# та роботи із ASP.Net, а також середовище розробки Visual Studio Code для роботи із CSS, HTML, JavaScript.

Застосунок може використовуватися групою користувачів, які мають на меті зробити власний робочий процес більш зручним та менш рутинним.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, KANBAN, КЕРУВАННЯ IT-ПРОЄКТАМИ, C#, VISUAL STUDIO, ASP.NET, JAVASCRIPT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – application programming interface;

SQL – Structured Query Language.

DOM – Document Object Model

ВСТУП

В сучасному світі взаємодії в ІТ команді люди використовують планування роботи за допомогою різних підходів та з використанням допоміжних веб-застосунків. Більшість таких веб-застосунків є професійно спрямованими та витримують професійний тон, через що при багаторазовому повторному виконанні одних і тих самих дій у застосунку робота стає рутинною – що поступово призводить до зниження ефективності користувача.

Темою дипломної бакалаврської роботи є «Розробка веб-застосунку для управління ІТ-проєктами за методологією Kanban з використанням механізмів геймифікації».

Цей проєкт допоможе урізноманітнити рутині процеси організації роботи ІТ команди над проєктом за допомогою залучення елементів геймифікації, а саме досягнень, рівнів, анімацій і так далі.

Такий підхід може допомогти сучасному поколінню підходити до роботи із організації проєктів з більшим ентузіазмом та бажанням отримувати нові досягнення.

1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметною областю даної дипломної бакалаврської роботи є управління програмними проєктами.

Управління проєктами [1] це складний і багатогранний процес. Управління проєктами складається з декількох етапів, а саме:

- ініціація проєкту;
- планування;
- виконання;
- моніторинг та контроль;
- завершення.

Даний курсовий проєкт буде допомагати команді розробників на етапі виконання. Етап виконання полягає в наступному – реалізація плану, який було розробленого на етапі планування. Тобто, на даному періоді відбувається безпосереднє виконання завдань і заходів, що були передбачені планом проєкту. Основні кроки на цьому етапі:

- координація роботи команди;
- виконання завдань;
- забезпечення комунікації між учасниками проєкту.

В цілому на даному етапі необхідно вести облік поставлених задач та керувати виконанням поставлених задач кожного учасника команди. Для найбільш оптимальної роботи використовують різні методології та фреймворки. Однією із таких методологій є методологія Kanban.

Kanban [2] – гнучка методологія для невеликих проєктів, затрати часу на планування якого не є дуже великими. Постановка задач в методології Kanban виконується за допомогою візуалізації у вигляді дошки. На дошці

розміщуються списки усіх задач та їх виконавці. Задачі у методології Kanban має наступні фази:

- To Do (виконати);
- Doing (виконується);
- Done (виконано).

Фази виконання завдань можуть бути змінені відповідно до потреб команди розробників, яка працює із методологією, але ці 3 фази є фундаментальними у методології Kanban. Візуалізацію дошки Kanban зображено на рисунку 1.1.

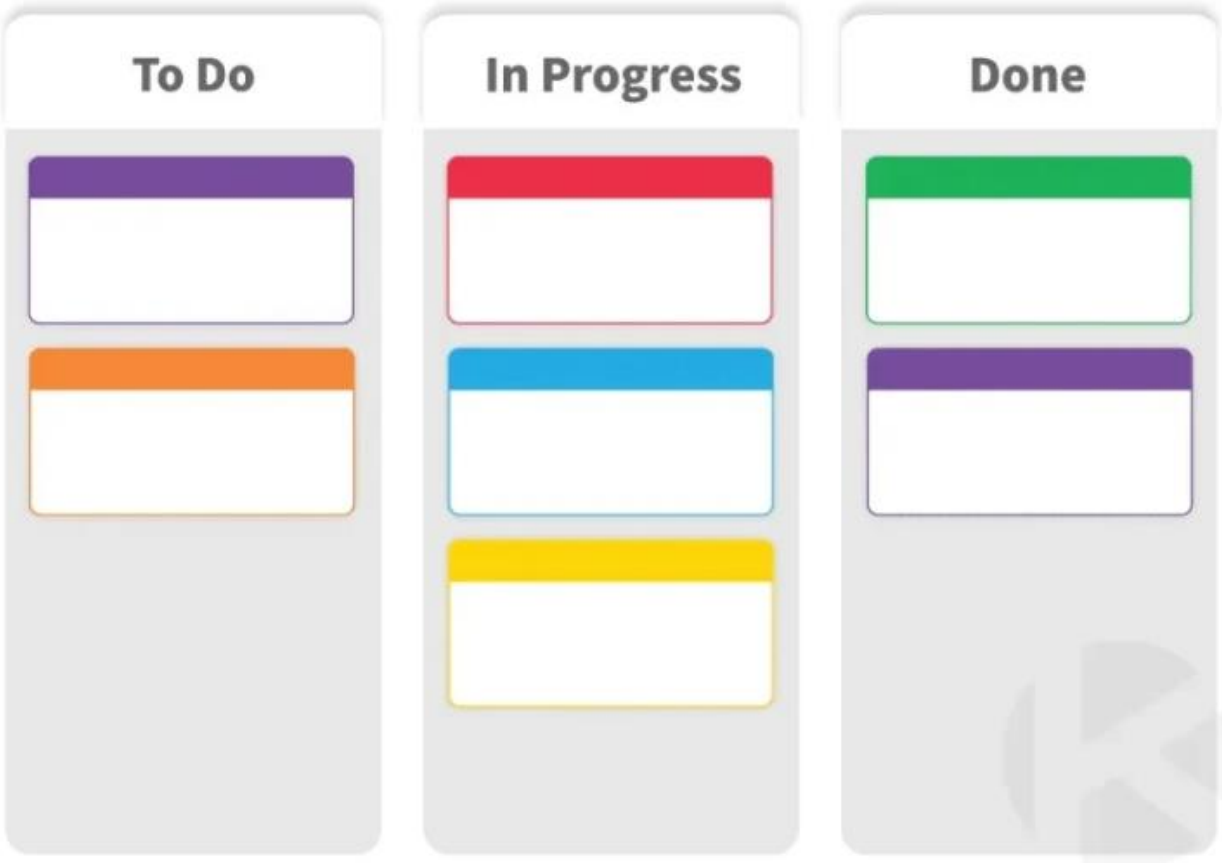


Рис. 1.1 Приклад Kanban дошки [3]

Розглянемо більш детально дошку, зображену на рисунку 1.1.

На дошці зображено 3 колонки, кожна колонка відповідає відповідній фазі виконання задачі. Колонок може бути безліч та вони можуть бути змінені відповідно до потреб роботи над проєктом.

Задачі на дошці зображені у вигляді карток, що знаходяться у колонках. Картка задачі містить назву задачі, опис задачі, оцінка задачі у вигляді Story Point та терміни виконання задачі. Також картки можуть містити додаткову інформацію по типу коментарів, прикріплених файлів, чеклістів та різні помітки до карток.

Розглянемо основні переваги та недоліки методології Kanban задля розуміння на чому необхідно концентруватися при майбутньому проєктуванні реалізації.

Переваги методології Kanban:

- гнучкість планування;
- велика активність участі команди у процесі;
- менше вузьких місць;
- наочність через доступність даних для усіх учасників.

Недоліки методології Kanban:

- методологія не підходить для команд понад більше п’яти людей;
- не призначена для довго строкового планування.

Тобто можна зробити висновок, що методологія Kanban призначена для невеликих проєктів з невеликою командою розробників та для команд, які часто можуть змінювати план роботи під час виконання задач.

1.2 Аналіз існуючих рішень

На ринку доволі багато популярних інструменті для організації роботи команд за допомогою методології Kanban. Із популярних рішень можна виділити наступні:

- Trello [4];
- Jira [5];
- MeisterTask [6].

Першим існуючим рішенням розглянемо веб-застосунок під назвою Trello.

Trello – це безкоштовний веб-застосунок, основний функціонал якого це створення Kanban дошок. Trello надає можливість створювати робочі області та дошки у них (див. рисунок 1.2).

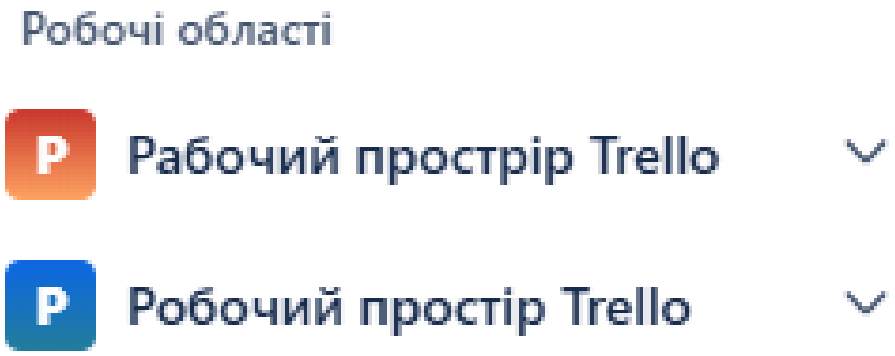


Рис. 1.2 Створені мною робочі області

При створенні дошки у Trello окрім звичайного створення також є можливість обрати готовий шаблон за яким буде згенеровано майбутню дошку. В Trello є багато категорій шаблонів дошок через те що веб-застосунок фокусується не тільки на методології Kanban у сфері IT розробок, а і у багатьох інших сферах в кожній якій є свої специфічні вимоги до Kanban дошки. Усі доступні категорії шаблонів зображено на рисунку 1.3.

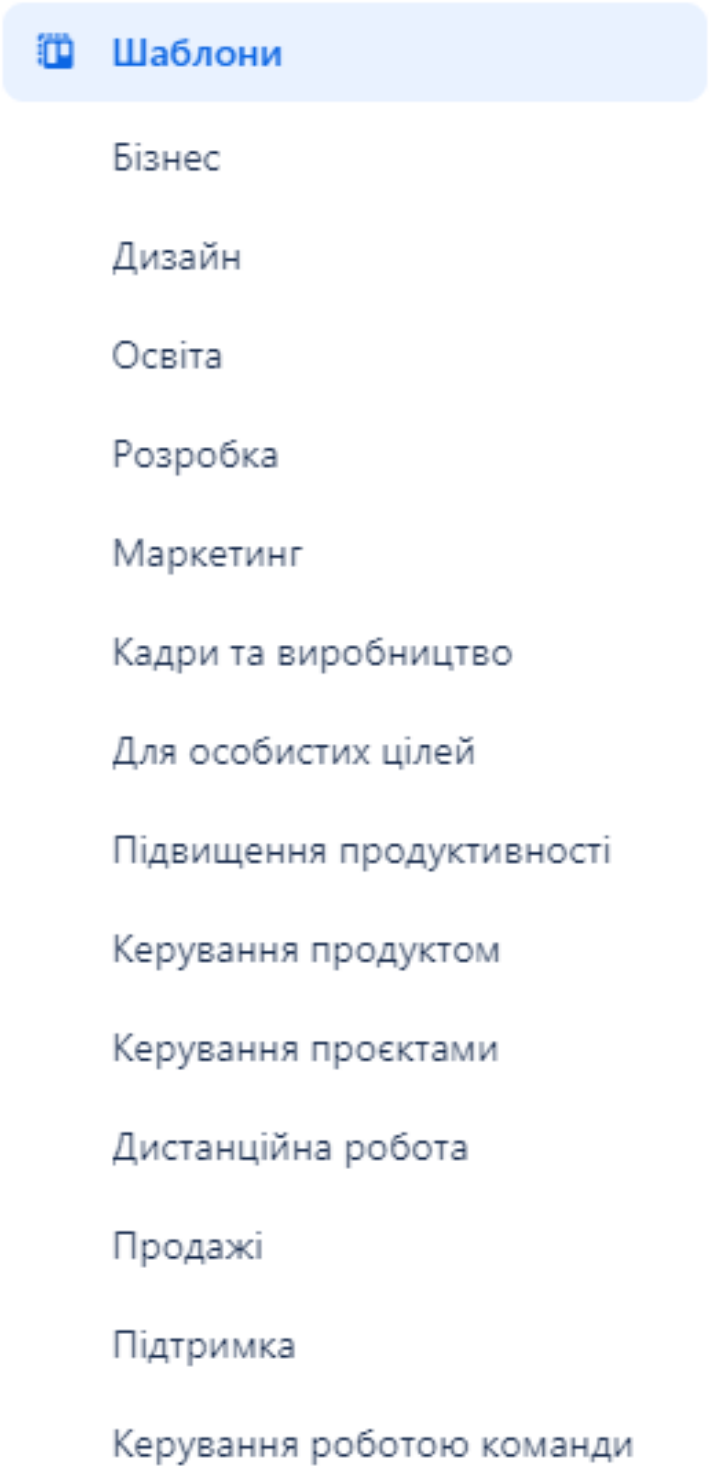


Рис. 1.3 Категорії шаблонів

Коли користувач обрав необхідну категорію шаблону, йому відображаються усі доступні шаблони за обраною категорією. Кожен шаблон має фотографію, посилання на автора, назву шаблону, опис шаблону та кількість використання шаблону. Приклад сторінки зображено на рисунку 1.4.

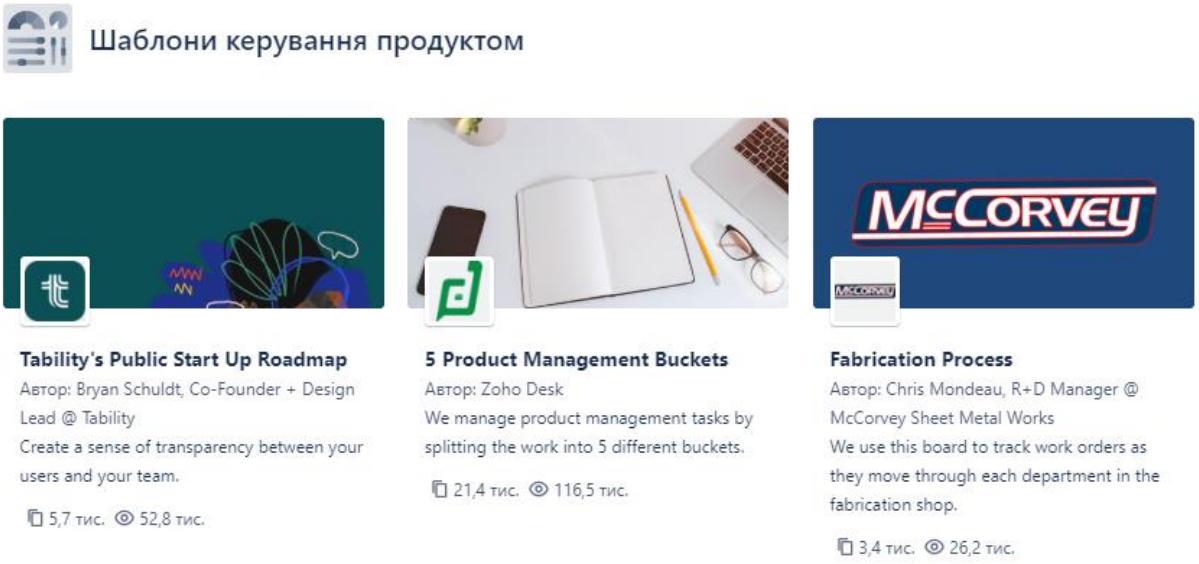


Рис. 1.4 Доступні шаблони дошок у категорії «Керування продуктом»

Якщо ж не обирати жодного шаблону, а створювати дошку з нуля під свої потреби, то необхідно лише обрати робочий простір та ввести назву дошки після чого вас буде автоматично перенесено на сторінку згенерованої дошки. Зображення сторінки новоствореної дошки наведено на рисунку 1.5.

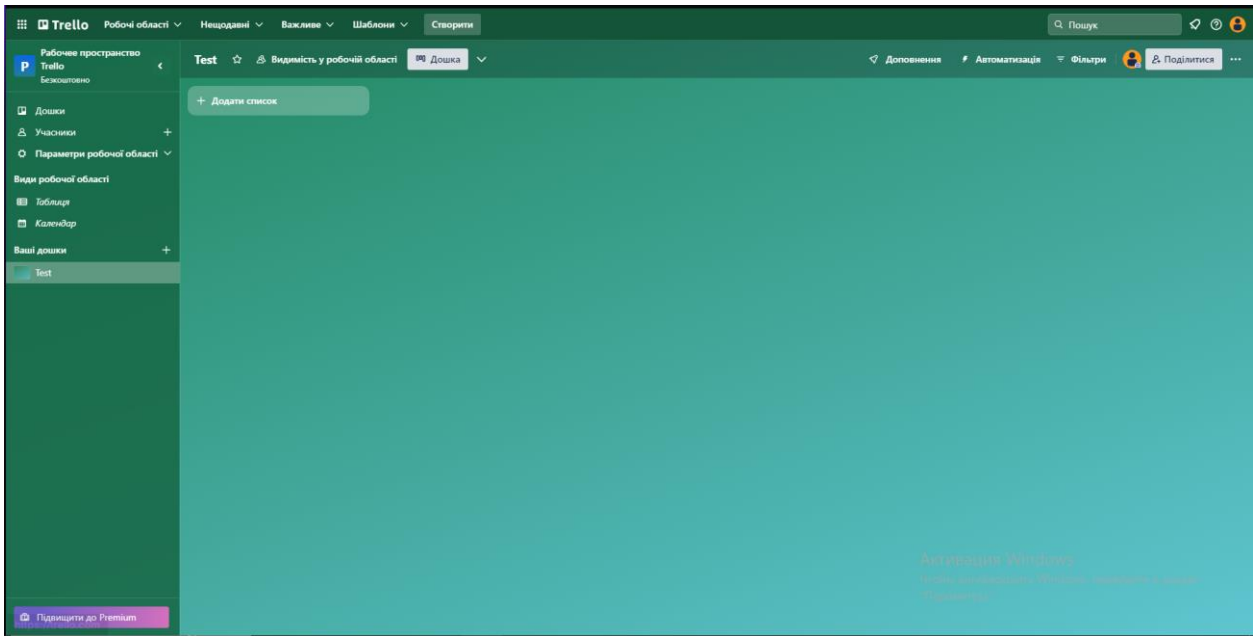


Рис. 1.5 Сторінка новоствореної дошки

На сторінці ми бачимо інструменти керування дошкою та панель перемикання між дошками у робочих областях.

По центру знаходиться сама дошка. Щоб створити нову фазу для майбутніх завдань на дошці необхідно натиснути на кнопку «Додати список». Після чого буде необхідно ввести назву створеної фази. Створення та результат створення зображено на рисунках 1.6 та 1.7.

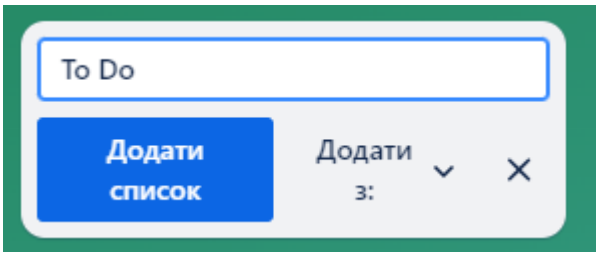


Рис. 1.6 Створення списку

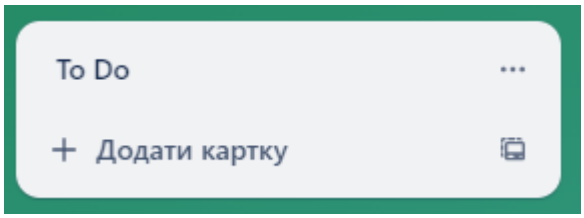


Рис. 1.7 Результат створення списку

Після створення фази у неї можна створити картку або задачу. Задача повинна містити опис задачі, призначеного виконавця та оцінку складності. Створена картка завдання зображена на рисунку 1.8.

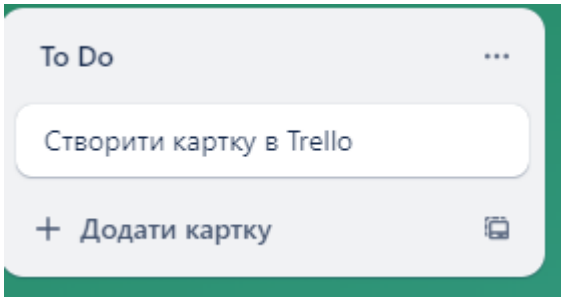


Рис. 1.8 Створена карточка завдання

Картка завдання може містити багато додаткової інформації та коментарі для зв’язку між учасниками команди. Додаткова інформація картки зображена на рисунку 1.9.

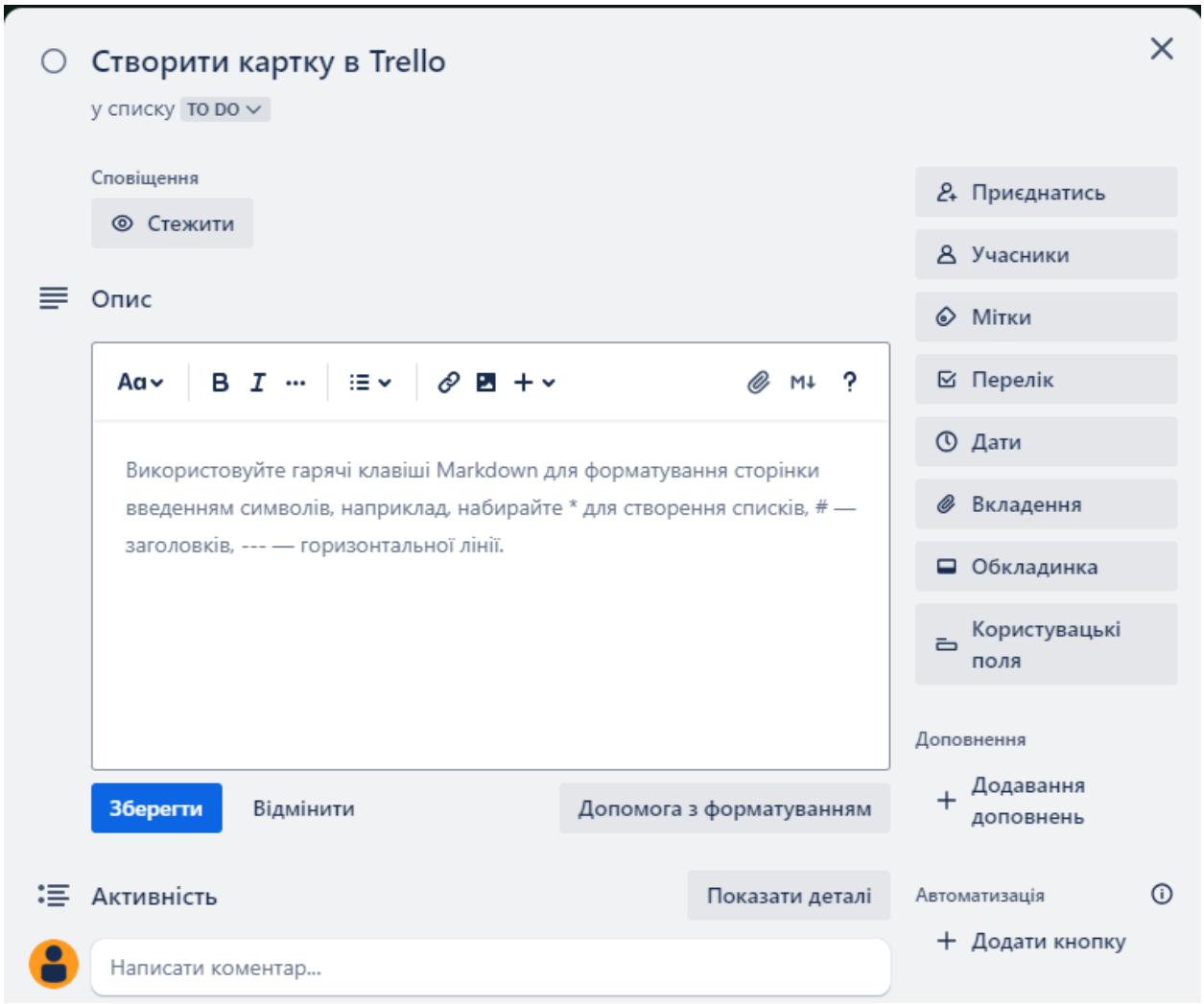


Рис. 1.9 Додаткова інформація картки завдання

Вище було розглянуто лише основний функціонал, необхідний для роботи дошки Kanban.Тепер розглянемо додатковий функціонал в веб-застосунку Trello:

- можливість інтеграції в дошку функцій із застосунків Google Drive, Slack та інші;

- можливість автоматизації процесів під час роботи із дошкою, шляхом створення кнопок, правил, генерації електронних звітів та інших функцій;
- фільтрування карточок завдань за різними видами фільтрів;
- можливість перегляду історії дій, що дозволяє контролювати процес роботи дошки.

Також, Trello має специфічні функції, які доступні лише користувачам із Premium-підпискою. Така підписка буде коштувати 10 доларів на місяць. Вона надає певні переваги.

Далі розглянемо веб-сервіс Jira.

Jira – безкоштовний веб-сервіс для керування виконання проєктів але, на відміну від Trello він призначений не тільки для роботи із методологію Kanban, а і з іншими методологіями.

Для створення дошки необхідно обрати відповідний шаблон серед існуючих. На сайті Jira він називається «Kanban». Також, Jira надає доволі детальний опис кожного шаблону(рис 1.11). Щоб переглянути шаблони потрібно спочатку обрати необхідний шаблон(рис 1.10).

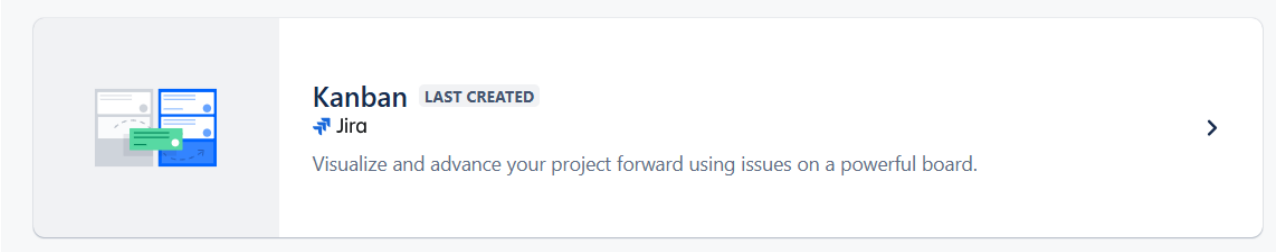


Рис. 1.10 Шаблон для створення дошки

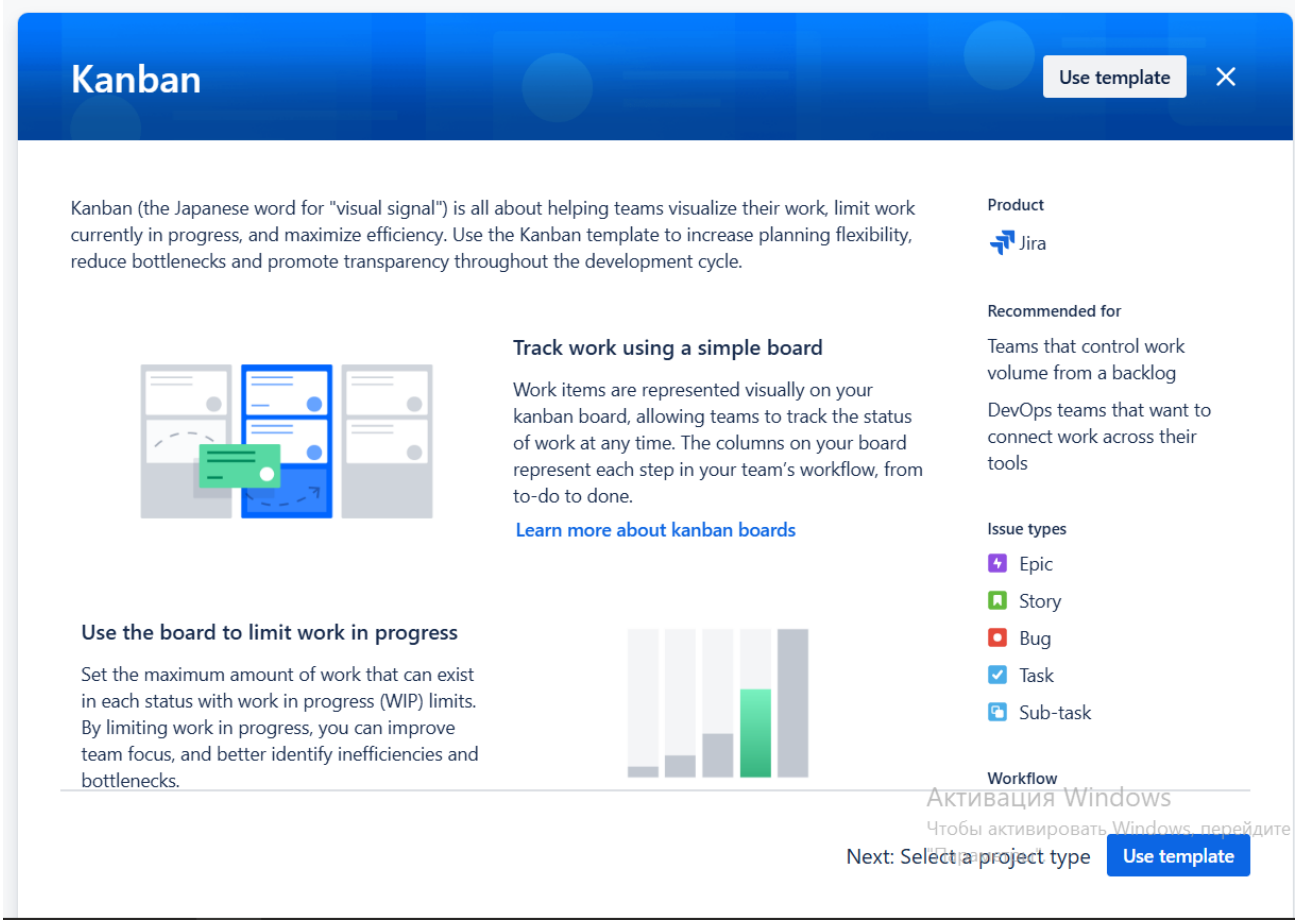


Рис 1.11 Опис шаблону Kanban

Наступним кроком користувач повинен обрати тип проєкту, над яким він працює (рисунок 1.12); Jira надає 2 типи проєктів:

- проєкт яким керує команда;
- проєкт яким керує компанія.

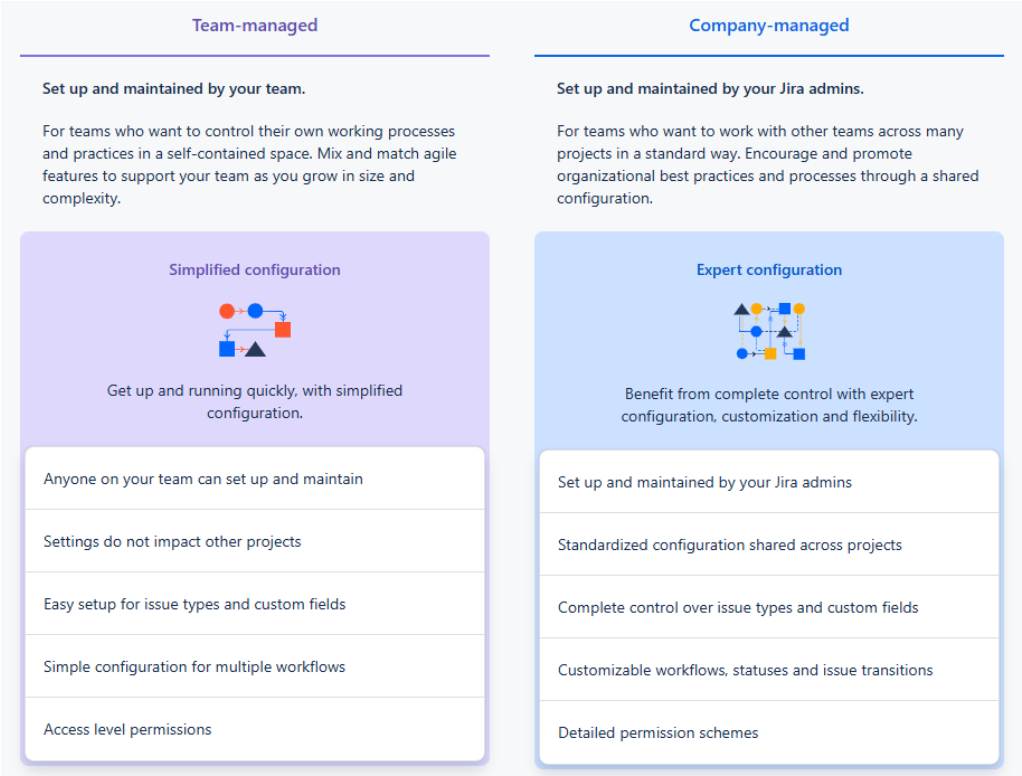


Рис. 1.12 Типи проєктів

Далі необхідно ввести назву проєкту, префікс проєкту та параметри доступу до проєкту. Також, можна переглянути обраний шаблон та тип проєкту (рисунок 1.13).

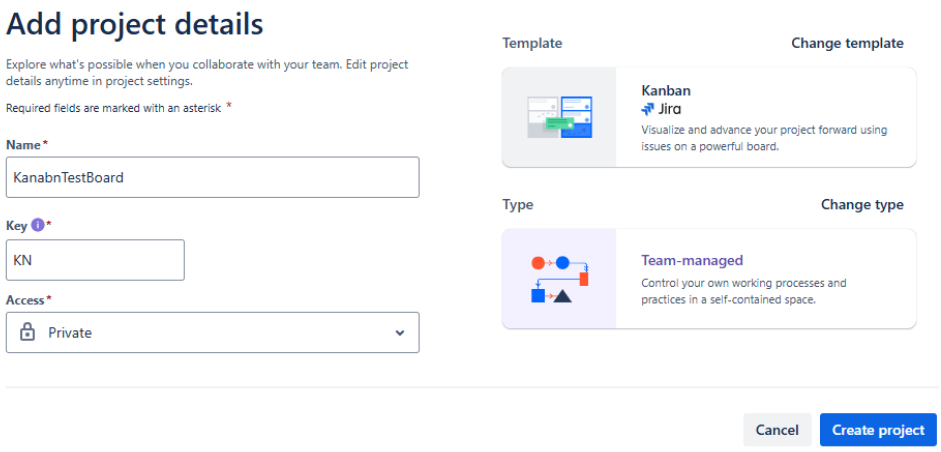


Рис. 1.13 Вікно базових налаштувань проєкту

Після створення проєкту Jira одразу пропонує додати членів вашої команди до створеної дошки (рисунок 1.14). Цей крок можна пропустити у випадку, якщо наразі немає потреби у додаванні членів команди до дошки. У випадку, якщо ви хочете одразу додати членів команди необхідно ввести їх email та обрати роль яку вони отримають при вході у дошку.

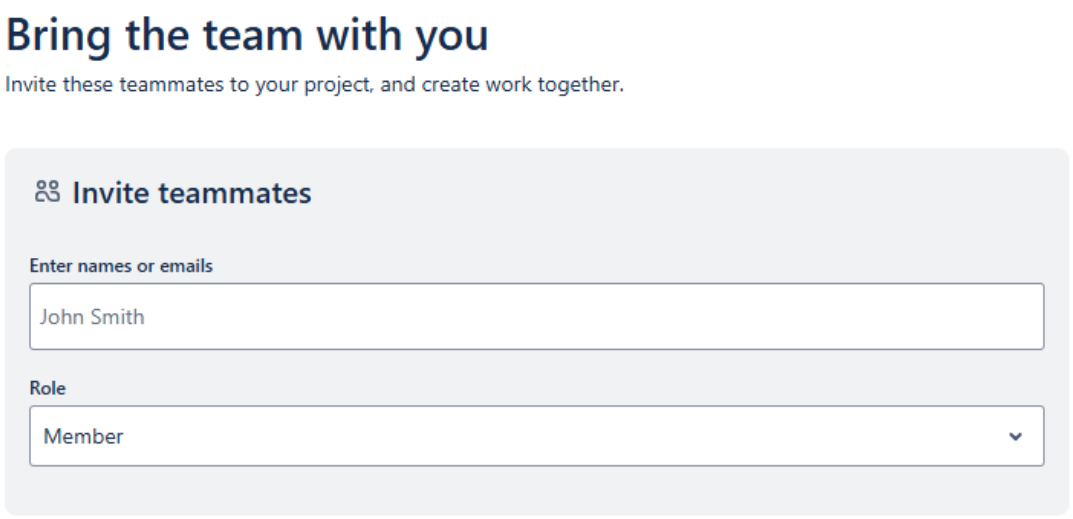


Рис. 1.14 Вікно для додавання користувачів

Після виконання усіх попередніх кроків користувач потрапляє на головну сторінку роботи із дошкою (рисунок 1.15).

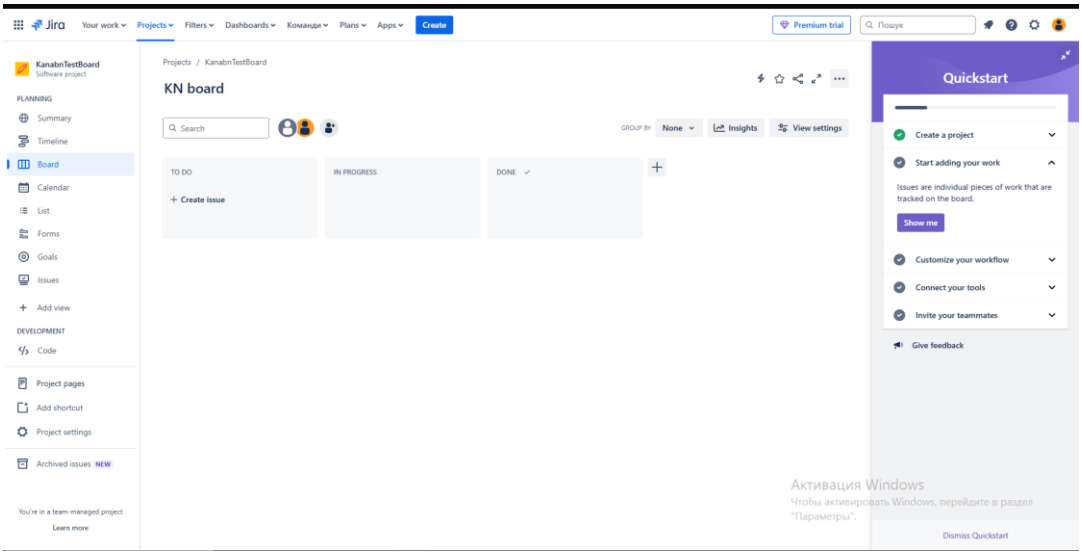


Рис. 1.15 Головна сторінка роботи із дошкою

На рисунку 1.16 розглянемо область роботи із дошкою.

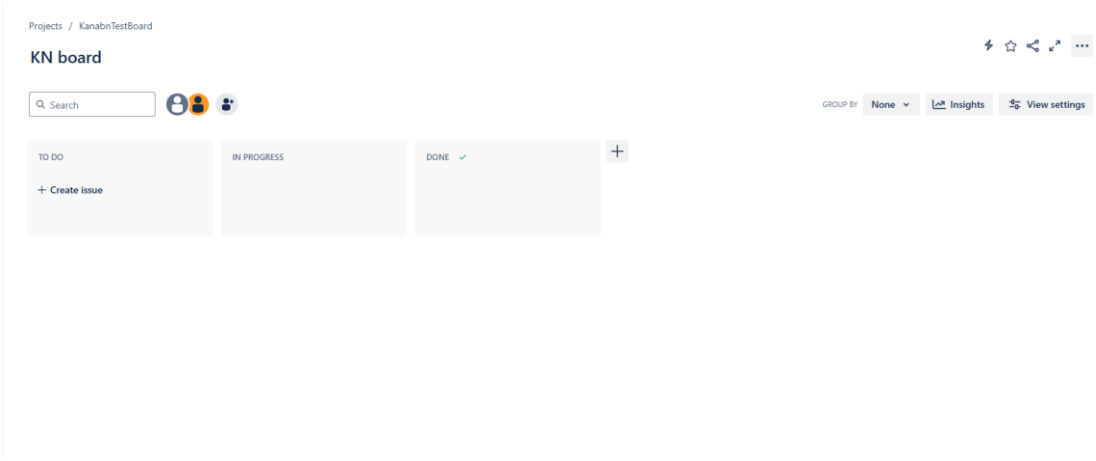


Рис. 1.16 Область роботи із дошкою

Зверху ми бачимо назву дошки та її префікс. Нижче можна побачити меню пошуку по дошці та учасників дошки. Також, слід зауважити, що шаблон у Jira автоматично створив 3 стовпчики для фаз виконання завдання на дошці.

Для створення задачі у дошці необхідно натиснути на «Create issue» у стовпчику в якому необхідно створити задачу. Після необхідно ввести характерний опис або назву задачі (рисунок 1.17).

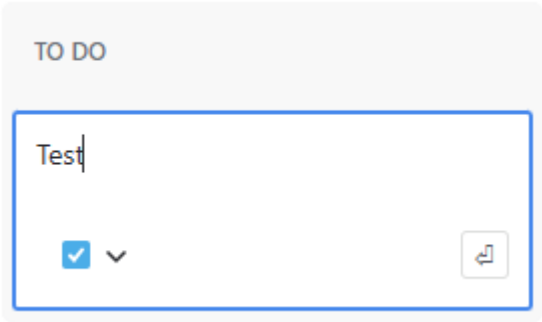


Рис. 1.17 Створення картки задачі

Після створення кратки можна переглянути більш детальну інформацію про неї (рисунок 1.18).

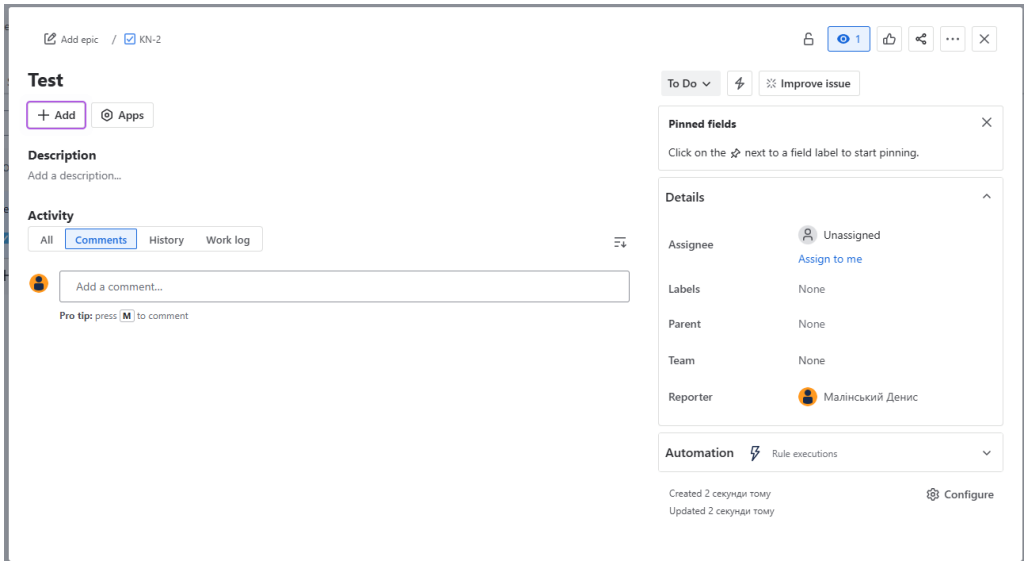


Рис. 1.18 Додаткова інформація картки задачі

В додатковій інформації можна побачити опис задачі, коментарі до задачі, людину яка створила задачу та виконавця. Також до задачі можна прикріпляти файли, посилання та підзадачі.

Після перегляду основного функціоналу звернемо увагу також на додатковий функціонал:

- можливість створити ціль для команди;
- функціонал створення форм;
- можливість інтеграції задач із Github, GitLab та Bitbucket;
- можливість перегляду статистики виконання задач у дошці.

Також, Jira має преміум-підписку в якій користувачеві надається доступ до штучного інтелекту, який допомагає під час роботи, необмежена кількість проєктів, які користувач може створити, та цілодобова підтримка команди розробників Jira.

Далі розглянемо сервіс MeisterTask.

MeisterTask – веб-сервіс створений для роботи із дошкою за методологією Kanabn. Він є подібним до Trello у плані функціональності. MeisterTask більше направлений на власне використання, а ніж на використання командами розробників.

Розглянемо головну сторінку сервісу. Головна сторінка сервісу зображена на рисунку 1.19.

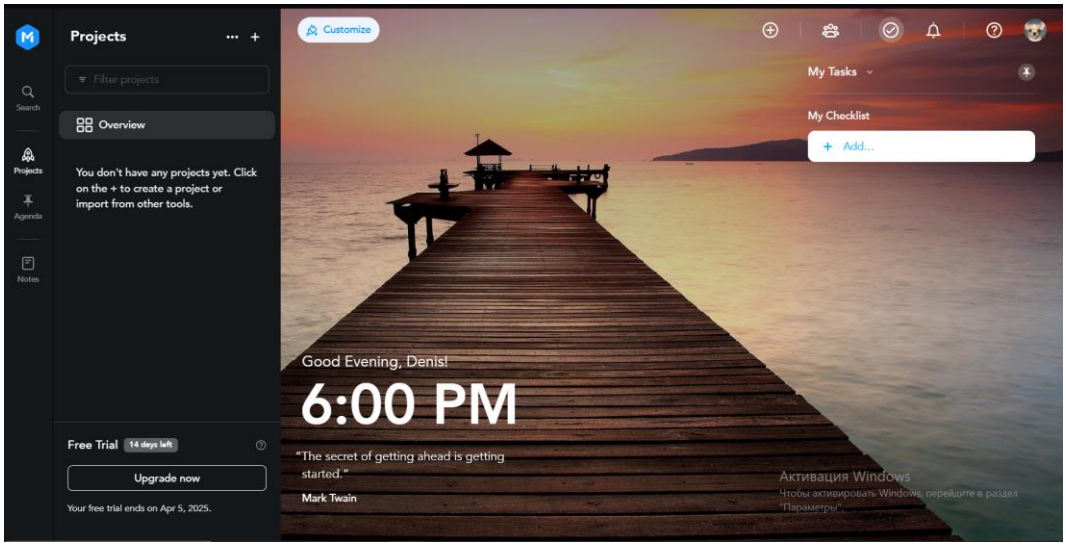


Рис. 1.19 Головна сторінка

На головній сторінці зліва зображено усі проєкти користувача, а справа чеклісти, які можуть допомагати при роботі одночасно над декількома проєктами.

Для створення проєкту необхідно лише натиснути на «Create Project» та ввести назву проєкту (рисунок 1.20).

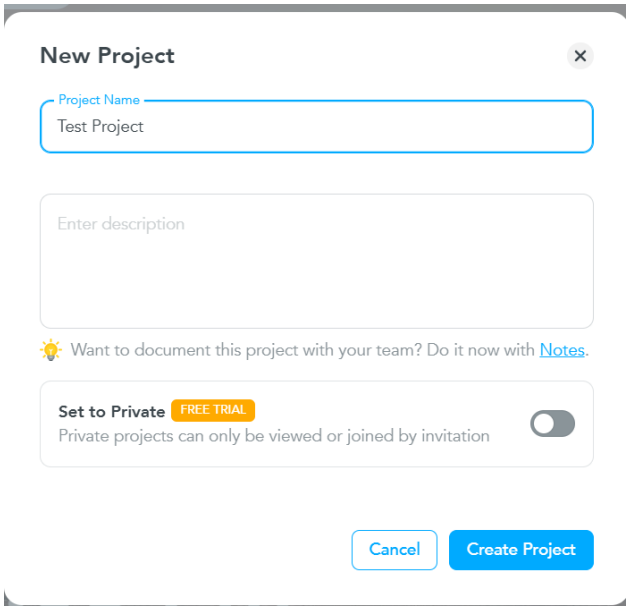


Рис. 1.20 Вікно створення проєкту

Після створення проєкту користувача автоматично перекидує на сторінку роботи із дошкою, де вже згенеровано 3 базові колонки для фаз виконання задач (рисунок 1.21).

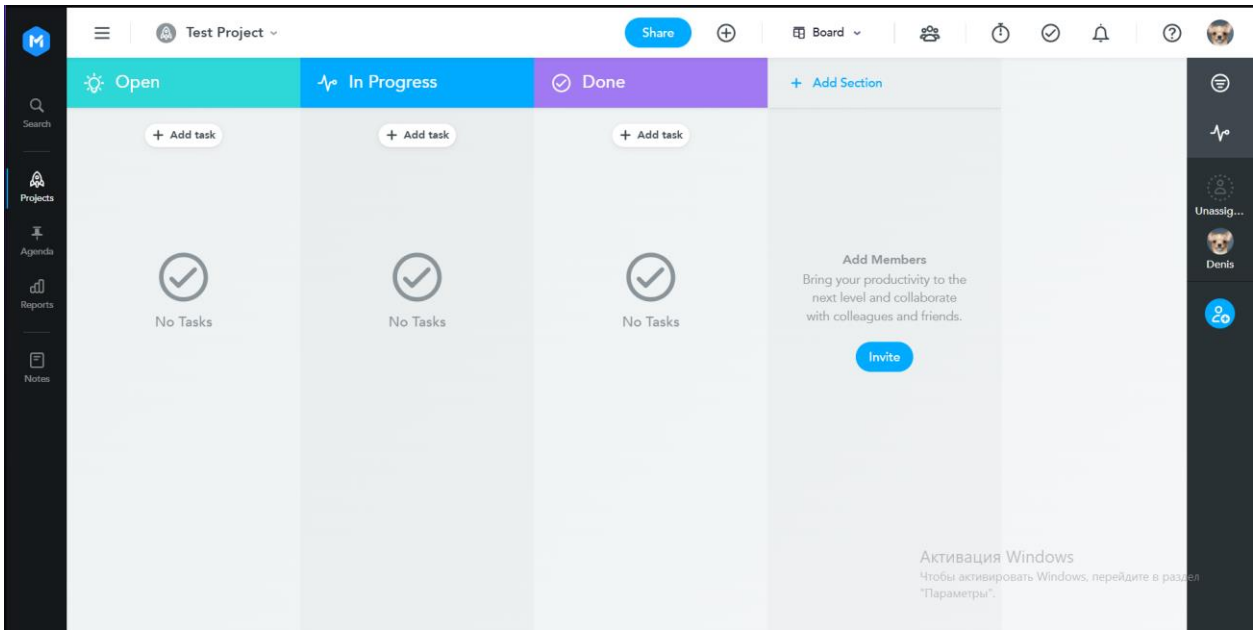


Рис. 1.21 Сторінка для роботи з дошкою

Створення задачі у сервісі MeisterTask виконується так само, як і в інших розглянутих сервісах – шляхом вводу назви задачі (рисунок 1.22).

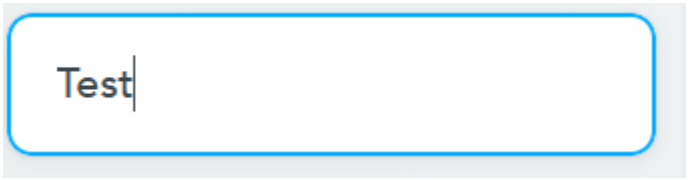


Рис. 1.22 Процес створення задачі

У більш детальній інформації створеної картки задачі, зображеної на рисунку 1.23, можна переглянути коментарі до задачі, опис задачі, чекліст задачі, вкладення до задачі, теги задачі, дати виконання задачі, зв’язки задачі із іншими задачами. Та також, є функція таймеру для фіксації часу виконання задачі користувачем.

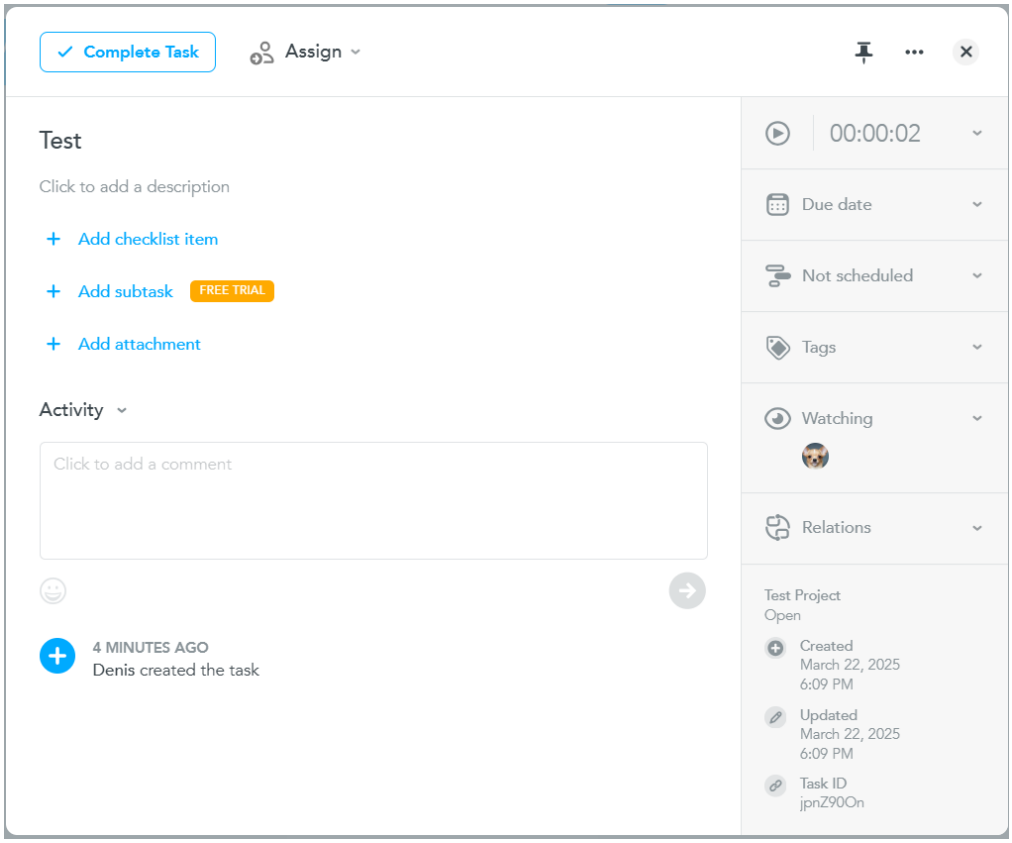


Рис. 1.23 Додаткова інформації задачі

Вище розглянуто основний функціонал – тепер розглянемо додатковий та специфічний функціонал відносно інших сервісів:

- наявність особистої дошки, де можна закріпляти задачі із робочих дошок, та також надавати їм індивідуальні статуси;
- особиста сторінка із звітами для усіх дошок, в яких приймає участь користувач;
- особиста сторінка для нотаток. Нотатки можна зберігати у різних файлах.

Також, MeisterTask має преміум-підписку яка знімає обмеження на кількість проєктів та нотаток. Також, підписка надає можливість користуватися штучним інтелектом, але в обмеженій за промптами кількостями.

1.3 Вимоги до веб-застосунку

Після проведення аналізу предметної області та аналізу існуючих рішень, можна виділити функціональні та нефункціональні вимоги до майбутнього веб-застосунка.

Функціональні вимоги наведено у таблиці 1.1.

Таблиця 1.1

Функціональні вимоги до веб-застосунку.

Вимога	Пояснення
Збереження інформації про створені простори та дошки у них	Веб-застосунок повинен мати функціонал збереження даних пов’язаних із просторами та дошками у них у базі даних
Робота із просторами	Веб-застосунок повинен мати функціонал створення, редагування, видалення та перегляду просторів
Робота із дошками	Веб-застосунок повинен мати функціонал створення, редагування, видалення та перегляду дошок, а також можливість редагувати, видаляти та створювати задачі та фази у дошці
Робота з учасниками дошок	Веб-застосунок повинен мати функціонал додавання учасників у дошки та функції для керування правами учасників дошки

Продовження таблиці 1.1

Функціональні вимоги до веб-застосунку.

Автентифікація	Веб-застосунок повинен мати функціонал авторизації та реєстрації за для можливості розподілення користувачів за обліковими записами
Механіка рівнів облікового запису	Веб-застосунок повинен мати функціонал прокачки рівня обліково запису за допомогою отримання досягнень під час роботи з веб-застосунком
Механіка досягнень	Веб-застосунок повинен мати функціонал отримання досягнень за виконання різноманітних дій під час роботи із веб- застосунком
Механіка рівнів для просторів	Веб-застосунок повинен мати функціонал прокачки рівня простору під час роботи із дошками
Механіка досягнень для просторів	Веб-застосунок повинен мати функціонал отримання досягнень під час роботи із дошками.

Нефункціональні вимоги наведено у таблиці 1.2.

Таблиця 1.2

Нефункціональні вимоги до веб-застосунку

Вимога	Пояснення
Зручний та інтуїтивно зрозумілий інтерфейс	Інтерфейс веб-застосунку повинен бути зрозумілим та не потребувати багато часу для розуміння, як користуватися веб-застосунком
Зрозумілі повідомлення про помилки	У разі виникнення помилок, повинна бути спеціальна сторінка, на яку буде переадресовано користувача та на якій буде зрозуміле пояснення помилки, яка виникла
Безпека	Уся приватна інформація про користувача повинна бути захищена. Пароль та логін користувача повинні бути захешовані у БД та захищенні для несанкціонованого доступу
Швидкодія	Реакція веб-застосунку на дію користувача не повинна перевищувати більше ніж 5 секунд
Доступність	Веб-застосунок повинен бути завжди доступним та працювати безперебійно
Тон і стиль геймифікації	Досягнення повинні бути оформлені у чіткому, але зрозумілому стилі

1.4 Вибір технологій та інструментів

Ключовим елементом дипломного проєкту є розроблений веб-застосунок для керування проєктами за допомогою методології Kanban з механізмами геймифікації. Провівши аналіз доступних безкоштовних технологій та інструментів для реалізації веб-застосунку, я обрав наступні інструменти та технології:

- C#;
- Visual Studio 2022;
- Asp.Net Core;
- HTML/CSS;
- JavaScript;
- PostgreSQL;
- Railway.app;
- Entity Framework Core;
- Razor Pages.

У якості основної мови програмування було обрано мову програмування C#.

C# [7] – об’єктно-орієнтована та багатопарадигмальна мова програмування розроблена компанією Microsoft та направлена на програмування широкого кола різновидностей програмних застосунків.

Саме ця мова програмування була обрана через такі її переваги:

- простота синтаксису (через що вивчення мови програмування займає менше часу);
- C# працює на платформі .Net, яка встигла себе зарекомендувати через свою надійність;
- велика спільнота розробників, через що багато проблем, які виникають під час розробки, на даний момент можна знайти на форумах;

- масштабованість та легкість обслуговування коду, написаного на мові C#.

У якості середовища розробки було обрано середовище розробки Visual Studio 2022.

Visual Studio [8] – потужне середовище розробки, яке допомагає реалізувати усі цикли розробки програмного забезпечення, яке було розроблено компанією Microsoft.

Саме це середовище розробки було обрано через такі його переваги:

- Visual Studio є «домашнім» середовищем розробки для мови програмування C# – у зв’язку з чим, має багато специфічних функцій саме під цю мову програмування;
- має зручні інструменти для пошуку помилок у коді та відладки коду;
- зручні готові шаблони застосунків на обраних мовах програмування;
- зручний інструмент для завантаження пакетів та бібліотек у проєкт.

У якості фреймворку для розробки веб-застосунку було обрано платформу ASP.Net Core.

ASP.Net Core [9] – гнучкий фреймворк для розробки веб-застосунків, який був розроблений компанією Microsoft. ASP.Net Core є новою еволюційною версією фреймворку ASP.Net.

Саме цей фремворк було обрано через такі його переваги:

- інтеграція із середовищем Visual Studio;
- гнучкість і простота в використанні;
- висока продуктивність фреймфорку.

У якості технологій для розробки фронтенд частини було обрано такі 3 технології:

- HTML;
- CSS;

- JavaScript.

HTML [10] – спеціальна мова розмітки, за допомогою якої будуть каркаси веб-сторінок.

HTML було обрано через такі його переваги:

- широка розповсюдженість та популярність
- сумісність із великою кількістю браузерів;
- проста інтеграція з мовами програмування;

CSS [11] – набір команд(селекторів), за допомогою яких можна стилізувати каркас веб-сторінки. CSS може стилізувати усі елементи веб-сторінки.

CSS було обрано через такі його переваги:

- широка розповсюдженість та популярність;
- гнучкість і масштабованість;
- підтримка адаптивного дизайну сторінок;
- легка інтеграція з технологіями.

JavaScript був обраний у проєкті у якості допоміжної мови програмування під час розробки фронтенду.

JavaScript [12] – високорівнева скриптова мова, яка використовується для створення інтерактивних застосунків та веб-сайтів. За допомогою JavaScript можна додавати анімації на веб-сторінці або додавати обробники подій на веб-сторінці.

JavaScript був обраний через такі його переваги:

- широка розповсюдженість та популярність;
- виконання у реальному часі в середовищі браузера;
- зручна інтеграція із CSS та HTML;
- вбудована підтримка DOM;
- велика кількість бібліотек.

У якості бази даних проєкту було обрано система управління базами даних (далі – СУБД) PostgreSQL.

PostgreSQL [13] – об’єктно-реляційна СУБД, яка базується на реляційній мові SQL. Тобто PostgreSQL поєднує в собі переваги як і реляційних баз даних, так і об’єктно орієнтованого підходу.

Саме цю СУБД було обрано через такі її переваги:

- цілісність даних через використання транзакцій з ACID властивостями;
- хороша продуктивність;
- гнучкість та масштабованість;
- можливість перегляду даних із БД у режимі реального часу.

У якості хоста для майбутнього проєкту було обрано Railway.app.

Railway.app [14] – сервіс для швидкого розгортання проєктів з зручними та інтуїтивно зрозумілими інструментами керування.

Railway.app було обрано у якості хоста через такі його переваги:

- наявність безкоштовного тарифу;
- простота в налаштуванні розгортання проєкту;
- інтеграція із GitHub та PostgreSQL;
- зручне масштабування ресурсів.

У якості фреймоворку для роботи із базою даних було обрано Microsoft Entity Framework Core.

Microsoft Entity Framework Core [15] – фреймворк, розроблений компанією Microsoft, який використовує об’єктно-орієнтовний підхід для роботи із даними із бази даних у застосунку.

Microsoft Entity Framework Core було обрано через такі переваги:

- зручність інтеграції у Visual Studio;
- підтримка C#;
- легка інтеграція та налаштування у середовищі проєкту;
- зручність роботи із даними за допомогою синтаксису C#, що не потребує зайвого часу на розробку запитів реляційною мовою.

У якості основної технології для розробки веб-сторінок було обрано Razor Pages.

Razor Pages [16] – сторінково-орієнтована технологія, яка використовується для створення кросбраузерних сторінок із чітким розділом задач на веб-сторінці.

Razor Pages було обрано через такі переваги:

- підтримка синтаксису C#;
- зручна інтеграція із контролерами;
- зручні інструменти для керування даними на сторінці.

1.5 Постановка задачі

Поставленою задачею було розробити web-застосунок для керування виконанням задач в проєкті за допомогою методології Kanban та Kanban дошки.

Веб-застосунок повинен зберігати інформацію про створені проєктних простори та створені у цих просторах Kanban дошок.

Веб-застосунок також зберігає усіх учасників дошки та доступні їм привілеї. Також, веб-застосунок повинен зберігати усю інформацію про задачі в дошках, а саме:

- дата початку та кінця виконання задачі;
- виконавець;
- опис задачі;
- її складність в Story point;
- прикріплені файли;
- коментарі до задачі.

Задля розподілу користувачів веб-застосунок повинен мати систему автентифікації.

Також, особливістю розроблюваного веб-застосунку є елементи геймифікації під час роботи із ним. Такими елементами геймифікації можуть бути:

- рівневі системи проєкту;
- досягнення, які можна отримати у рамках проєкту, або у рамках загальної роботи із веб-застосунком;
- рівні користувачів, які прокачуються від роботи із застосунком;
- різноманітні анімаційні ефекти, що Урізноманітнюють рутинні процеси.

Розроблений web-застосунок повинен бути доступний у мережі безперебійно та цілодобово.

Також, web-застосунок повинен мати надійні елементи безпеки задля підтримання високого рівня довіри серед користувачів.

2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Проєктування архітектури системи

Архітектура системи [17] – це план або структура, яка описує як саме система організована, які компоненти взаємодіють між собою та із зовнішніми системами.

Існують деякі шаблони архітектурних систем, а саме:

- монолітна архітектура;
- мікросервісна архітектура;
- клієнт-серверна архітектура;
- архітектура з використанням серверів обробки подій;
- сервіс-орієнтована архітектура.

Провівши аналіз перерахованих архітектур, зваживши усі плюси та мінуси, розглянувши доступні ресурси та технології, які будуть використовуватися для розробки системи, було обрано моноліту архітектуру у якості архітектури проєкту.

Монолітна архітектура [17] – це архітектура, коли система складається з єдиного блоку коду, де усі частини взаємодіють між собою напряму. Тобто, увесь функціонал та інтерфейс системи знаходяться в єдиному місці та розгортуються одночасно.

Переваги, через які було обрано саме монолітну архітектуру:

- простота розробки на початку проєкту;
- легке тестування програми;
- чітке уявлення взаємозв’язків в програмі;
- швидка розробка системи;
- легке розгортання у хмарні сервіси.

Але ця архітектура має також свої недоліки, на які необхідно звернути увагу:

- зі зростанням масштабу системи код буде ставати громіздким та незручним;
- зміна частини проєкту може впливати на іншу частину проєкту та викликати помилки;
- проблеми у масштабуванні окремих функцій системи.

Також, слід зауважити, що в архітектурі проєкту буде використовуватися не просто монолітна архітектура, а монолітна архітектура з використанням 3-рівневої архітектури. Але, все ж таки, основною архітектурою проєкту є монолітна архітектура через те, що всі елементи проєкту знаходяться в одному рішенні та працюють як єдиний процес.

3-рівнева архітектура [18] – це шаблон проєктування, що забезпечує структурований підхід до організації та розробки веб-застосунків. Архітектура складається з трьох шарів, а саме:

- шар представлення, що відповідає за взаємодію з користувачем;
- шар бізнес-логіки, що відповідає за реалізацію функціоналу;
- шар роботи із даними, що відповідає за роботу із базою даних.

3-рівнева архітектура має свої переваги:

- модульність завдяки розділу на 3 шари;
- масштабованість шарів через їх розділ;
- гнучкість через розділ шарів.

Для більш зрозумілої взаємодії між шарами було побудовано діаграму, яка зображена на рисунку 2.1.

Рис. 2.1 Діаграма взаємодії шарів

Використання 3-рівневої архітектури пом’якшує недоліки монолітної архітектури; крім того, використання такої архітектури допоможе у майбутній підтримці проєкту або переносу на іншу архітектуру проєкта.

2.2 Проєктування бази даних

Наступним етапом проєктування системи є етап проєктування бази даних.

Проєктування бази даних складалося з двох частин:

- проєктування ER-діаграми;
- проєктування таблиць бази даних.

Першим етапом було проєктування ER-діаграми. Діаграма зображена на рисунку 2.2.

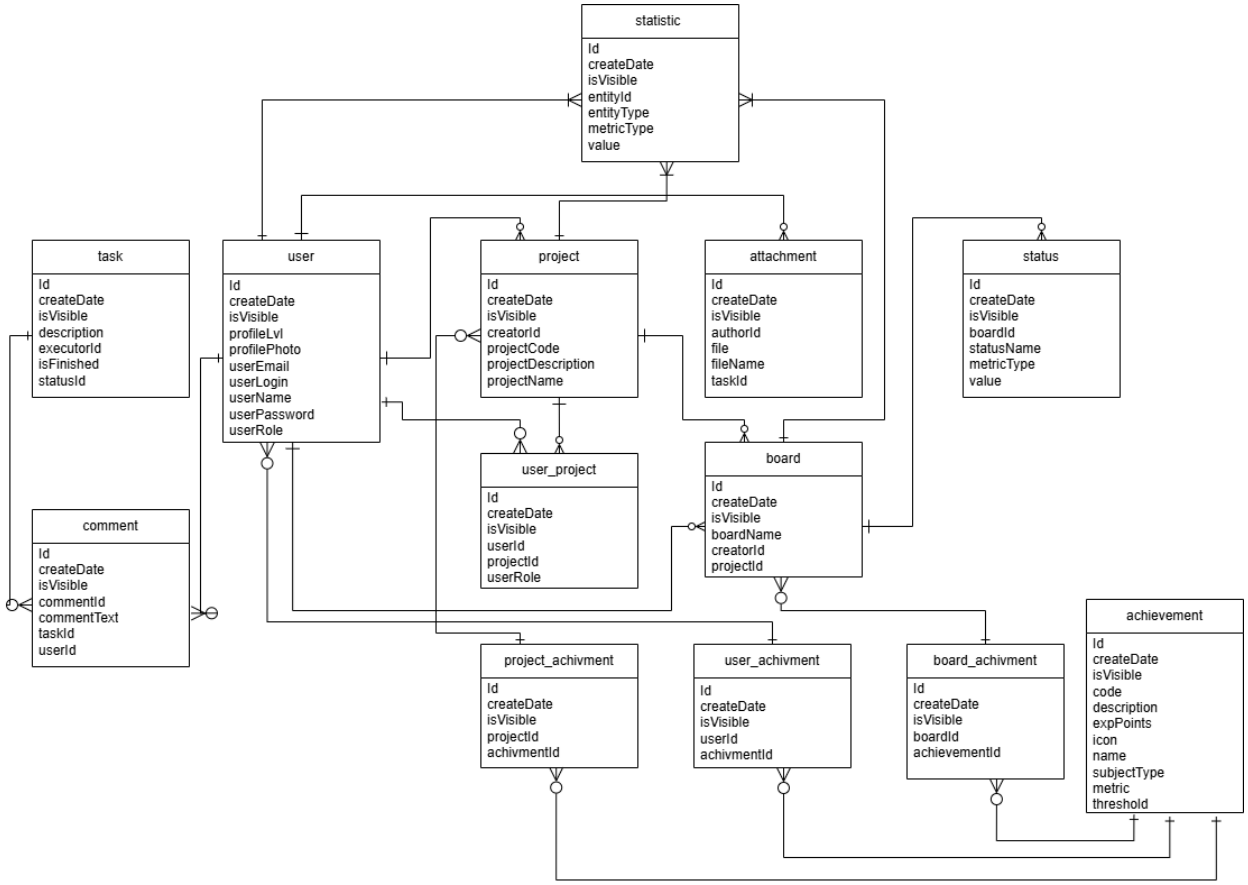


Рис. 2.2 ER-діаграма

Далі було проведено проєктування кожної таблиці.

Таблиця 2.1

Таблиця User.

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
profileLvl	INTEGER	Not Null
profilePhoto	BLOB	
userEmail	VARCHAR(50)	Not Null
userLogin	VARCHAR(50)	Not Null
userName	VARCHAR(50)	Not Null
userPassword	VARCHAR(250)	Not Null
userRole	VARCHAR(50)	Not Null

Таблиця 2.2

Таблиця attachment

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
authorId	INTEGER	FK, Not Null
file	BLOB	Not Null
fileName	VARCHAR(250)	Not Null
taskId	INTEGER	FK, Not Null

Таблиця 2.3

Таблиця Project

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
creatoId	INTEGER	FK, Not Null
projectCode	VARCHAR(50)	Not Null
projectDescription	VARCHAR(250)	Not Null
projectName	VARCHAR(50)	Not Null

Таблиця 2.4

Таблиця task

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
taskName	VARCHAR(50)	Not Null
taskDescription	VARCHAR(250)	Not Null
executorId	INTEGER	FK, NOT NULL
isFinished	BOOLEAN	Not Null
statusId	INTEGER	FK, NOT NULL

Таблиця 2.5

Таблиця status

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
statusName	VARCHAR(50)	Not Null
statusMetric	VARCHAR(50)	Not Null
boardId	INTEGER	FK, NOT NULL
value	INTEGER	NOT NULL

Таблиця 2.6

Таблиця statistik

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
entityId	INTEGER	FK, NOT NULL
entityMetric	VARCHAR(50)	Not Null
entityType	VARCHAR(50)	NOT NULL
value	INTEGER	NOT NULL

Таблиця 2.7

Таблиця comment

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
commentId	INTEGER	FK, NOT NULL
commentText	VARCHAR(250)	Not Null
taskId	INTEGER	FK, NOT NULL
userId	INTEGER	FK, NOT NULL

Таблиця 2.8

Таблиця user_project

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
userId	INTEGER	FK, NOT NULL
projectId	INTEGER	FK, NOT NULL
userRole	VARCHAR(50)	NOT NULL

Таблиця 2.9

Таблиця board

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
creatorId	INTEGER	FK, NOT NULL
projectId	INTEGER	FK, NOT NULL
boardName	VARCHAR(50)	NOT NULL

Таблиця 2.10

Таблиця project_achivment

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
achivmentId	INTEGER	FK, NOT NULL
projectId	INTEGER	FK, NOT NULL

Таблиця 2.11

Таблиця user_achivment

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
achivmentId	INTEGER	FK, NOT NULL
userId	INTEGER	FK, NOT NULL

Таблиця 2.12

Таблиця board_achivment

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
achivmentId	INTEGER	FK, NOT NULL
boardId	INTEGER	FK, NOT NULL

Таблиця 2.13

Таблиця achivment

Назва поля	Тип поля	Модифікатори
Id	INTEGER	PK, Auto increment, Not Null
createDate	DATETIME	Not Null
isVisible	BOOLEAN	Not Null
code	VARCHAR(50)	NOT NULL
name	VARCHAR(50)	NOT NULL
description	INTEGER	NOT NULL
expPoints	INTEGER	NOT NULL
icon	VARCHAR(50)	NOT NULL
subjectType	VARCHAR(50)	NOT NULL
metric	VARCHAR(50)	NOT NULL
threshold	INTEGER	NOT NULL

2.3 Моделювання вимог

На основі отриманих функціональних вимог у 1.3 було побудовано діаграму варіантів використання системи. Для більшої зручності діаграму було представлено у двох різних виглядах – а саме, для двох різних акторів:

- адміністратор;
- користувач.

Для актора «адміністратор» було визначено 6 прецедентів.

Для актора «користувач» було визначено 16 прецедентів.

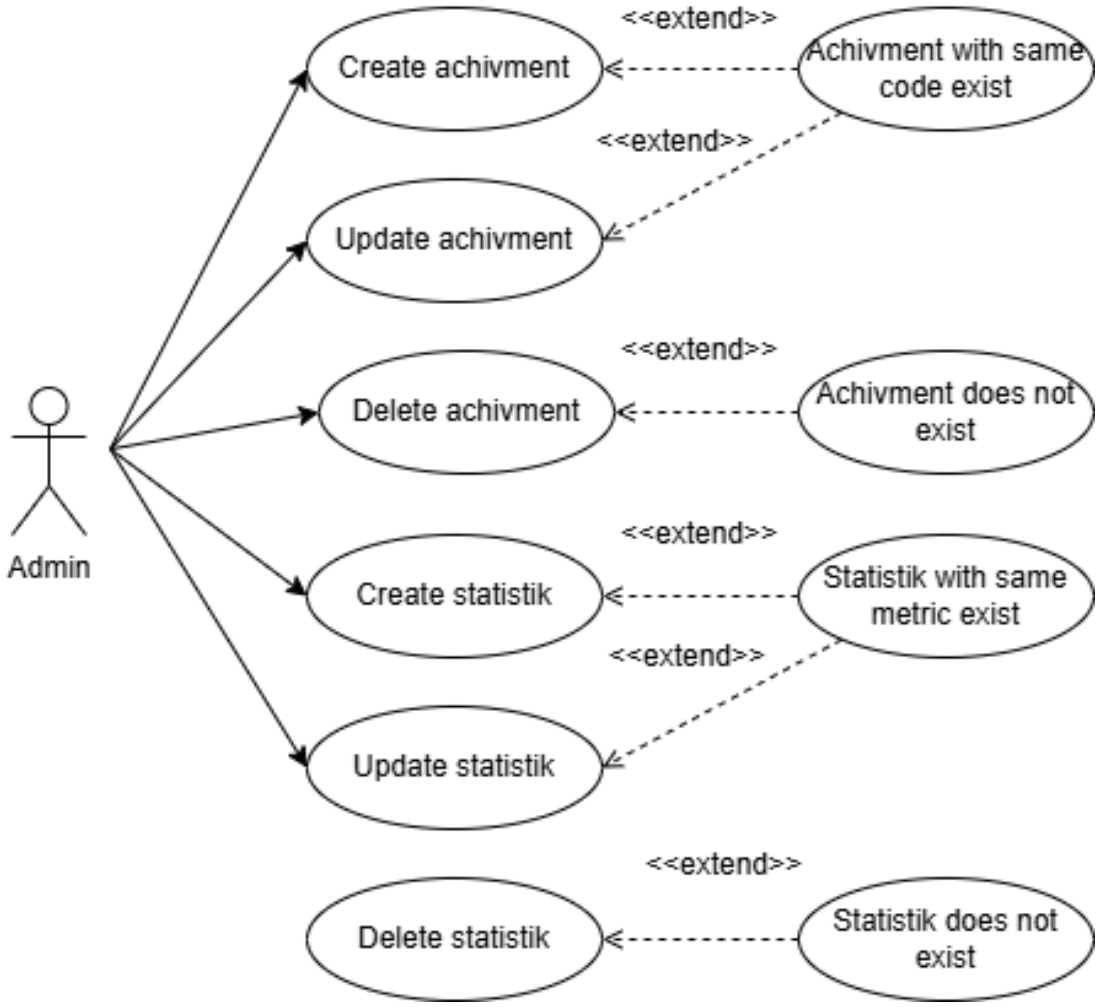


Рис. 2.3 Діаграма варіантів використання (частина 1)

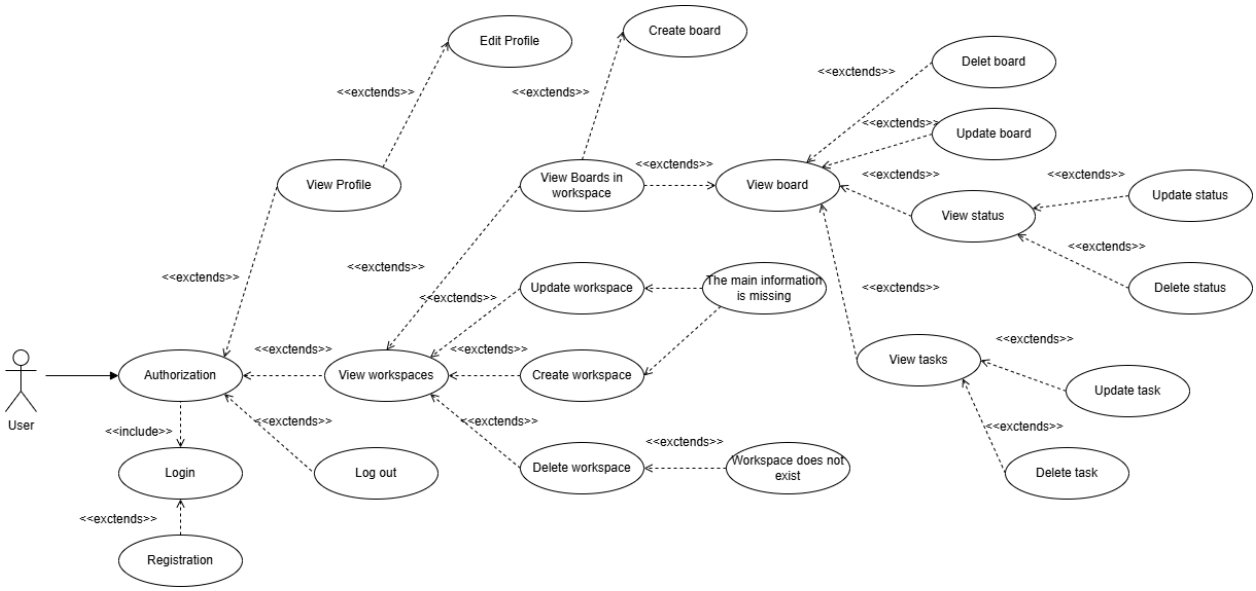


Рис. 2.4 Діаграма варіантів використання (частина 2)

Наступним етапом було проведено опис кожного прецеденту для кожного актора.

Спочатку було описано варіанти використання для актора «Admin».

Таблиця 2.14.

Варіант використання «Створення досягнення»

Короткий опис	Створення досягнення для користувача/дошки/простору.
Дійові особи	Адміністратор
Передумова	Адміністратор має обліковий запис
Основний потік подій	1. Адміністратор входить в панель адміністратора 2. Адміністратор натискає на кнопку створення досягнення 3. Адміністратор додає усю необхідну інформацію 4. Адміністратор зберігає створене досягнення
Альтернативний потік подій	Помилка бази даних Помилка валідації
Постумова	POST-1. Було створено досягнення

Таблиця 2.15.

Варіант використання «Оновлення досягнення»

Короткий опис	Оновлення існуючого досягнення
Дійові особи	Адміністратор
Передумова	Досягнення існує у системі
Основний потік подій	1. Адміністратор входить в панель адміністратора 2. Адміністратор гукає необхідне досягнення 3. Адміністратор вносить зміни у запис 4. Адміністратор зберігає внесені зміни
Альтернативний потік подій	Помилка бази даних Помилка валідації
Постумова	PUT-1. Було додано зміни у досягнення

Таблиця 2.16.

Варіант використання «Видалення досягнення»

Короткий опис	Видалення існуючого досягнення.
Дійові особи	Адміністратор
Передумова	Досягнення існує у системі
Основний потік подій	1. Адміністратор входить в панель адміністратора 2. Адміністратор гукає необхідне досягнення 3. Адміністратор видаляє знайдене досягнення
Альтернативний потік подій	Помилка бази даних
Постумова	DELETE-1. Було видалено запис досягнення

Таблиця 2.17.

Варіант використання «Створення статистики»

Короткий опис	Додавання статистики у систему.
Дійові особи	Адміністратор
Передумова	Адміністратор має обліковий запис
Основний потік подій	1. Адміністратор входить в панель адміністратора 2. Адміністратор натискає на кнопку додавання статистики 3. Адміністратор заповняє усі необхідні дані 4. Адміністратор зберігає додану статистику

Продовження таблиці 2.17

Варіант використання «Створення статистики»

Короткий опис	Додавання статистики у систему.
Альтернативний потік подій	Помилка бази даних Помилка валідації
Постумова	POST-2. Було додано статистику

Таблиця 2.18.

Варіант використання «Оновлення статистики»

Короткий опис	Оновлення статистики у системі.
Дійові особи	Адміністратор
Передумова	Статистика має запис у системі
Основний потік подій	1. Адміністратор входить в панель адміністратора 2. Адміністратор шукає необхідний запис 3. Адміністратор вносить зміни у запис 4. Адміністратор зберігає оновлені дані статистики
Альтернативний потік подій	Помилка бази даних Помилка валідації
Постумова	PUT-2. Було додано статистику

Таблиця 2.19.

Варіант використання «Видалення статистики»

Короткий опис	Видалення статистики у системі.
Дійові особи	Адміністратор
Передумова	Статистика має запис у системі
Основний потік подій	1. Адміністратор входить в панель адміністратора 2. Адміністратор шукає необхідний запис 3. Адміністратор видаляє знайдений запис
Альтернативний потік подій	Помилка бази даних
Постумова	DELETE-2. Було видалено статистику

Тепер опишемо найголовніші варіанти використання для актора «User».

Таблиця 2.20.

Варіант використання «Вхід в ситему»

Короткий опис	Авторизація користувача
Дійові особи	Користувач
Передумова	Користувач має обліковий запис
Основний потік подій	1. Користувач переходить на початкову сторінку веб-сервісу 2. Користувач заповнює дані для авторизації 3. Користувач підтверджує введені дані
Альтернативний потік подій	Помилка серверу Помилка бази даних Не вірно введені дані
Постумова	GET-1. Повідомлення про вдалий вхід

Таблиця 2.21.

Варіант використання «Реєстрація в системі»

Короткий опис	Реєстрація користувача
Дійові особи	Користувач
Передумова	Користувач не має обліковий запис
Основний потік подій	1. Користувач переходить на сторінку реєстрації 2. Користувач заповнює дані для реєстрації 3. Користувач підтверджує введені дані
Альтернативний потік подій	Помилка серверу Помилка бази даних Не вірно введені дані
Постумова	POST-3. Обліковий запис було створено

Таблиця 2.22.

Варіант використання «Редагування профілю»

Короткий опис	Редагування профілю користувача
Дійові особи	Користувач
Передумова	Користувач має обліковий запис
Основний потік подій	1. Користувач авторизується у системі 2. Користувач переходить у профіль 3. Користувач вводить нові дані 4. Користувач зберігає внесені дані

Продовження таблиці 2.22

Варіант використання «Редагування профілю»

Альтернативний потік подій	Помилка серверу Помилка бази даних Не вірно введені дані для входу Помилка валідації
Постумова	PUT-3. Дані було успішно змінено

Таблиця 2.23.

Варіант використання «Створення простору»

Короткий опис	Створення простору.
Дійові особи	Користувач
Передумова	Користувач має обліковий запис
Основний потік подій	1. Користувач авторизується у системі 2. Користувач натискає на кнопку створення простору 3. Користувач вводить необхідні дані 4. Користувач зберігає внесені дані
Альтернативний потік подій	Помилка серверу Помилка бази даних Помилка валідації
Постумова	POST-4. Простір було успішно створено

Таблиця 2.24.

Варіант використання «Створення дошки»

Короткий опис	Створення дошки.
Дійові особи	Користувач
Передумова	Користувач має обліковий запис та простір
Основний потік подій	1. Користувач авторизується у системі 2. Користувач переходить у простір 3. Користувач натискає на кнопку створення дошки 4. Користувач вводить необхідні дані 4. Користувач зберігає внесені дані
Альтернативний потік подій	Помилка серверу Помилка бази даних Помилка валідації
Постумова	POST-5. Дошку було успішно створено

Таблиця 2.25.

Варіант використання «Редагування дошки»

Короткий опис	Редагування дошки.
Дійові особи	Користувач
Передумова	Користувач має дошку
Основний потік подій	1. Користувач авторизується у системі 2. Користувач переходить у простір 3. Користувач переходить на сторінку дошки 4. Користувач натискає на кнопку редагування дошки 4. Користувач вводить необхідні дані 4. Користувач зберігає внесені дані
Альтернативний потік подій	Помилка серверу Помилка бази даних Помилка валідації
Постумова	PUT-4. Дані було збережено

На основі аналізу діаграми варіантів використання системи було побудовано діаграму класів основної логіки веб-застосунку. Діаграма класів зображена на рисунку 2.5.

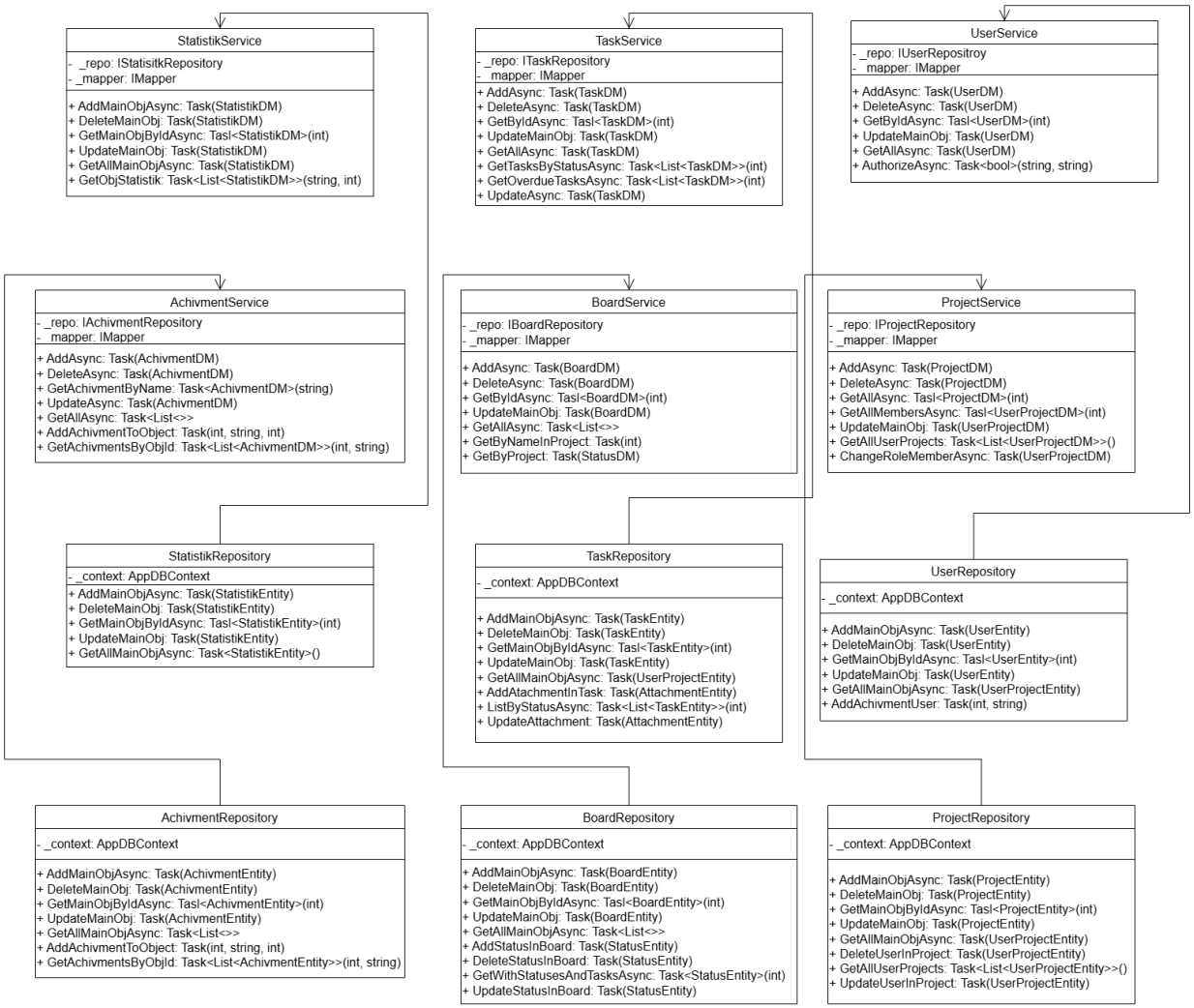


Рис. 2.5 Діаграма класів основної логіки

Тобто, виконання основної логіки застосунку припадає на сервіси та репозиторії.

Класи Repository містять в собі поле типу AppDBContext, яке відповідає за зв’язок із базою даних, та містять у собі поля для реалізації базових CRUD операцій над даними у базі даних.

Класи Services містять у собі поле класу Repository, який відповідає за роботу із даними, які необхідні сервісу, та поле типу IMapper, яке зберігає в собі клас, який виконує конвертацію даних для кожного рівня, щоб рівень міг працювати із даними.

2.4 Проєктування інтерфейсу користувача

На основі варіантів використання було спроєктовано мапу веб-сайту для повноцінної роботи веб-застосунку. Мапа веб-застосунку зображена на рисунку 2.6.

Рис. 2.6 Мапа веб-застосунку

Для більш глибшого розуміння, виконаємо опис кожної сторінки на мапі веб-сайту:

- сторінка Автентифікації повинна є початковою сторінкою, яка повинна містити опис проєкту та форми для авторизації та реєстрації користувача;
- після автентифікації, користувач потрапляє на сторінку перегляду просторів, на якій він має можливість переглядати усі свої створені простори, або перейти на форму створення просторів для створення нового простору. Також, він має можливість перейти на сторінку перегляду профілю;
- сторінка профілю містить інформацію про користувача, яку він ввів, а також інформацію про його статистику у веб-застосунку та отримані ним досягнення під час роботи. Також, він має можливість за допомогою форми редагування профілю додати зміни в персональну інформацію;
- сторінка дошки є однією з основних сторінок, на якій користувач буде проводити найбільшу частку свого часу під час роботи із веб-застосунком. Ця сторінка повинна містити увесь необхідний функціонал для

керуванням виконання задач у проєкті та містити форми для редагування, створення та видалення інформації;

На основі проведеного аналізу інтерфейсу користувача було створено прототипи інтерфейсу користувача.

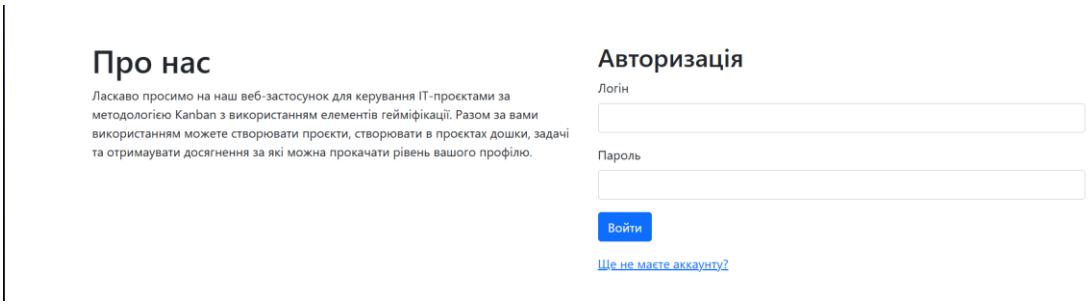


Рис. 2.7 Початкова сторінка

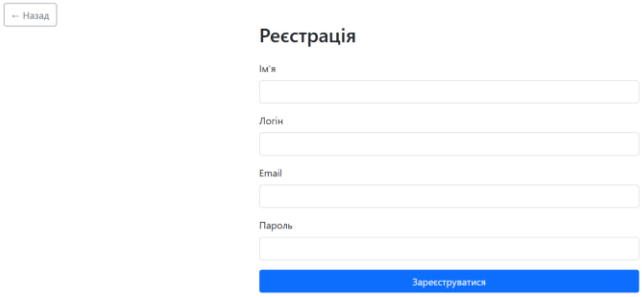


Рис. 2.8 Сторінка реєстрації

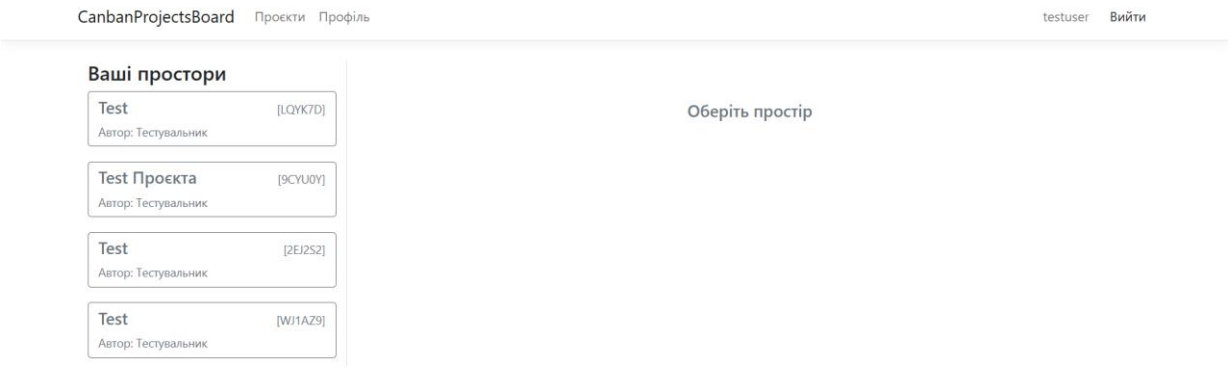


Рис. 2.9 Сторінка просторів

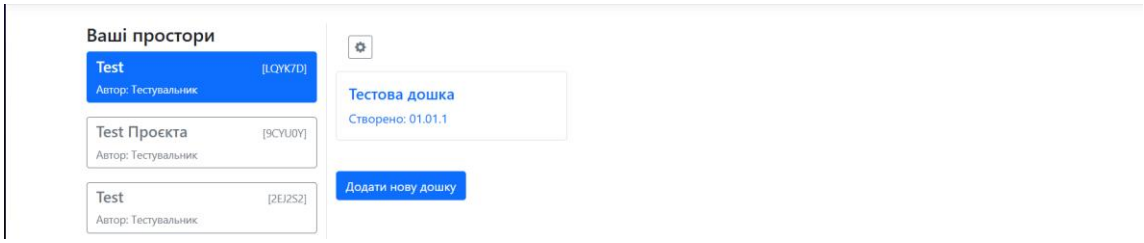


Рис. 2.10 Сторінка перегляду дошок у просторі

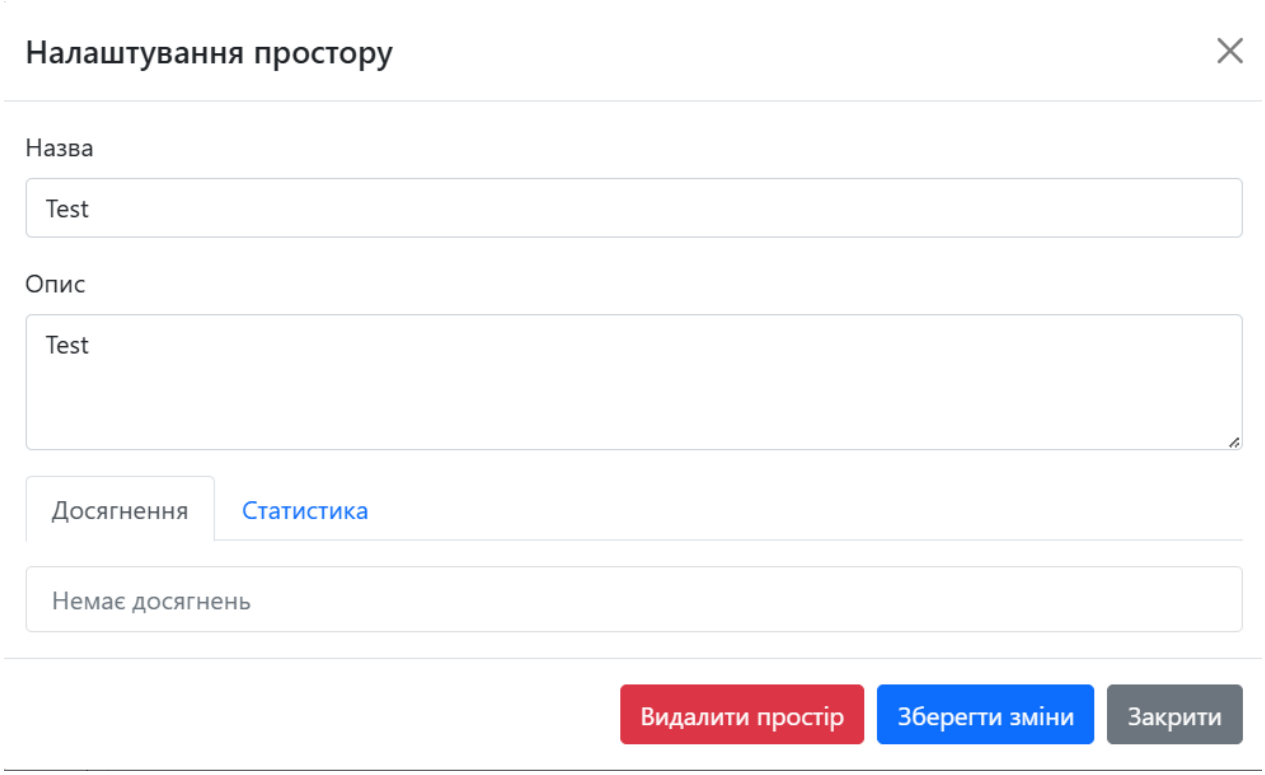


Рис. 2.11 Форма налаштувань простору

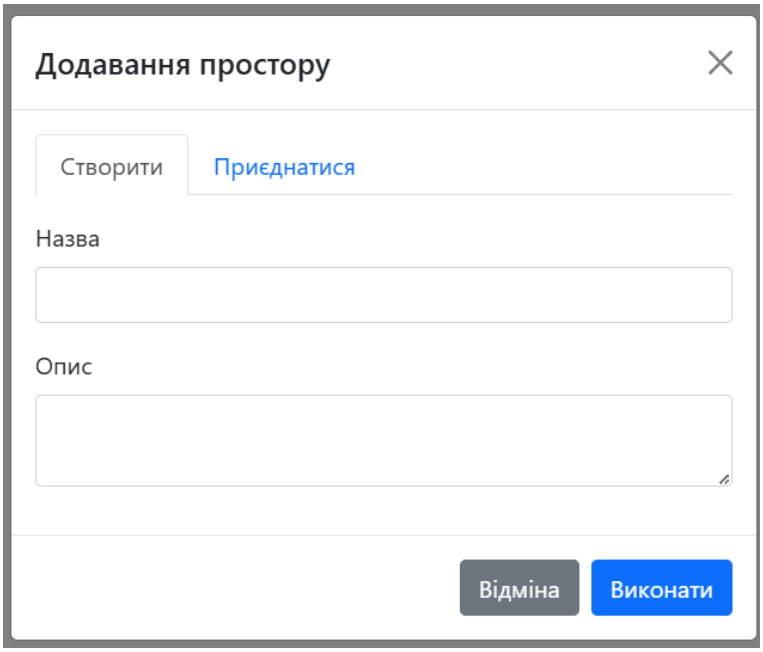


Рис. 2.12 Форма створення простору

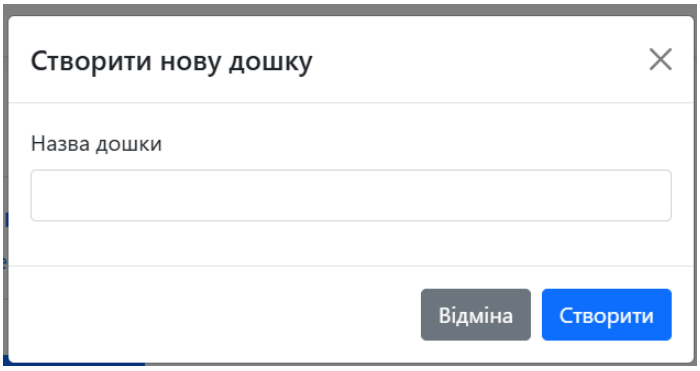


Рис. 2.13 Форма створення дошки

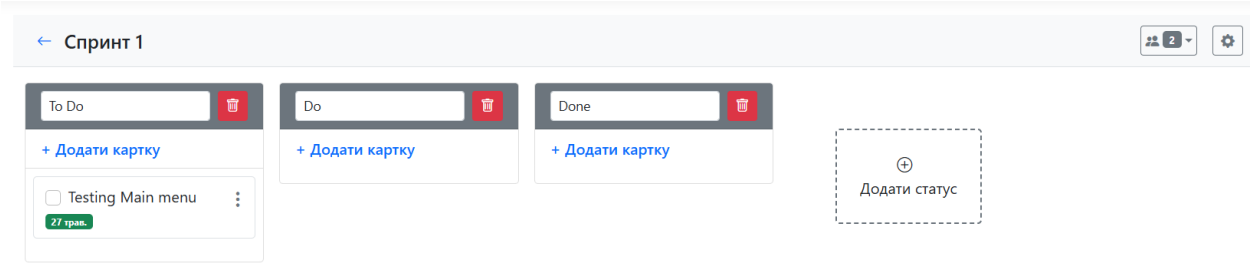


Рис. 2.14 Сторінка дошки

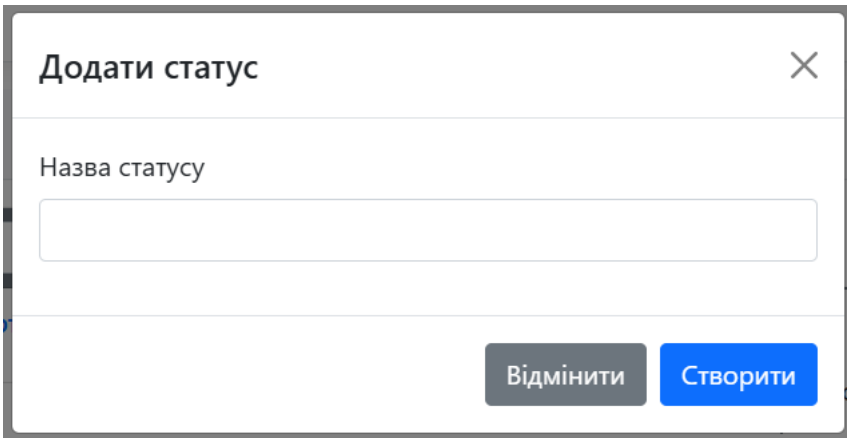


Рис. 2.15 Форма створення статусу

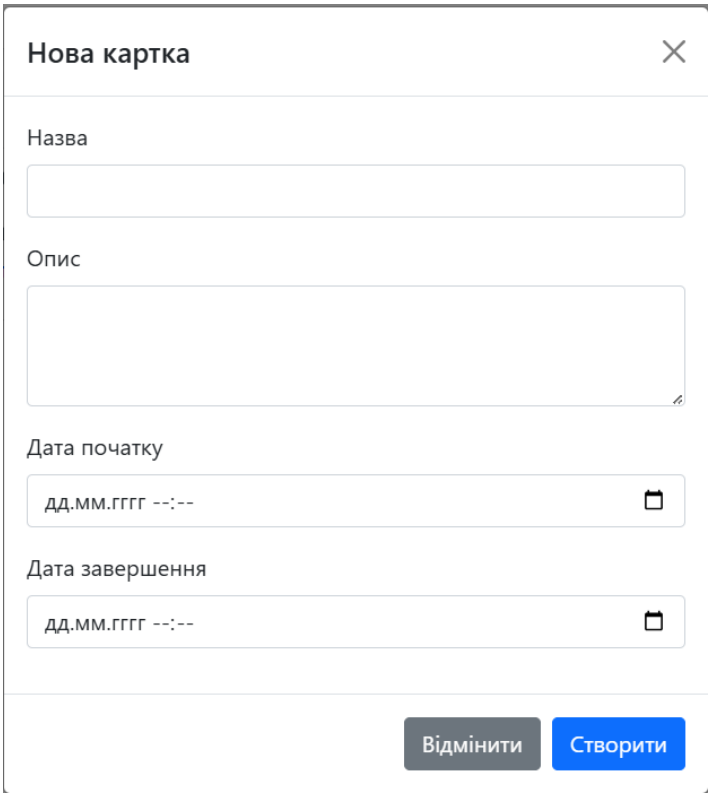


Рис 2.16 Форма створення картки задачі

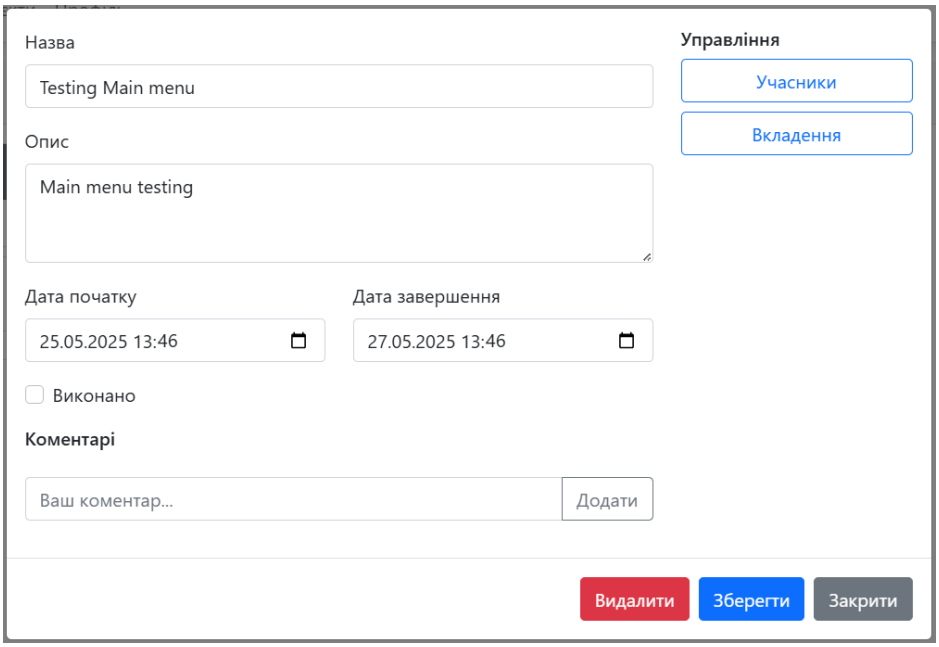


Рис 2.17 Форма налаштування картки задачі

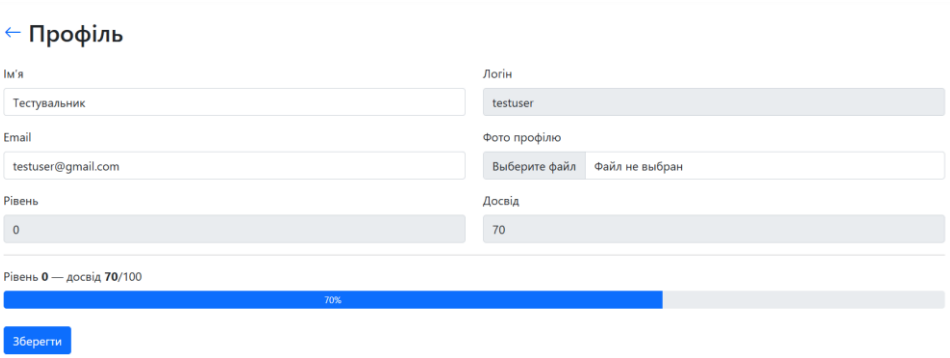


Рис. 2.18 Сторінка профілю (частина 1)

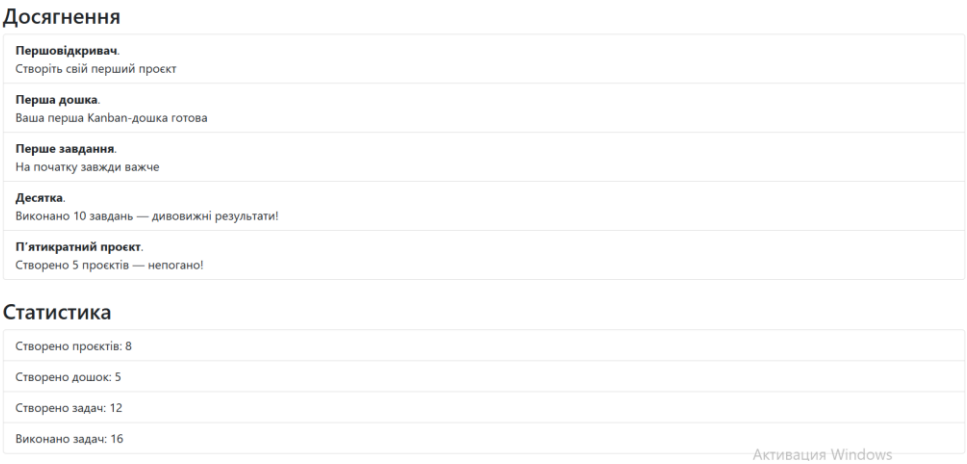


Рис. 2.19 Сторінка профілю (частина 2)

3. РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ

3.1 Основні компоненти системи

Спочатку було розроблено компоненти та розділено їх по окремим проєктам згідно до спроєктованої раніше архітектури. Структуру основних компонентів зображено на рисунку 3.1 та вигляд розроблених компонентів у проєкті зображено на рисунку 3.2.

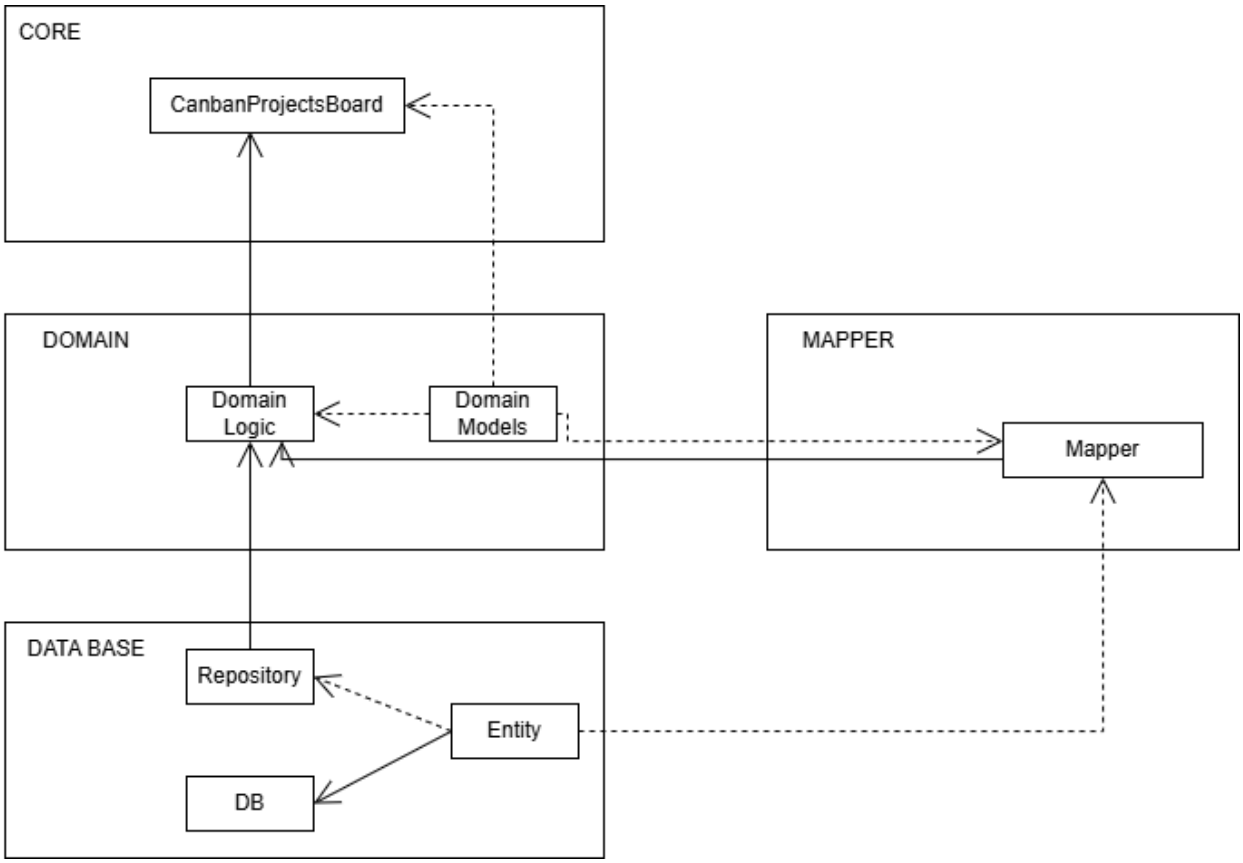


Рис. 3.1 Діаграма компонентів програми

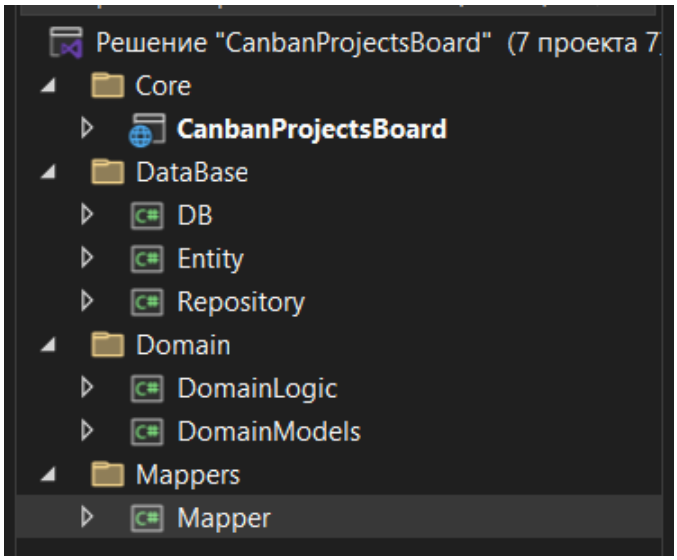


Рис. 3.2 Структура проєкту

Розглянемо більш детально кожну групу компонентів:

- Core це ядро застосунку, яке складається з єдиного компоненту, а саме ASP.Net Core проєкт, який компілює усі інші групи компонентів;
- Database це група, яка відповідає за підключення до бази даних та внесення чи отримання даних із неї. Ця група складається з трьох проєктів, які є бібліотекою класів;
- Domain це група, яка відповідає за реалізацію бізнес логіки веб-застосунку. Група складається з двох проєктів, які є бібліотекою класів;
- Mappers це група, яка відповідає за мапінг одних видів даних до інших. Це дозволяє створювати пайплайни між шарами без помилок у різновидності об’єктів класів. Група складається лише з одного проєкту, який є бібліотекою класів.

Наступним етапом було створення класів моделей та інтерфейсів для основних виконавчих класів для груп компонентів Domain та Database. На рисунках 3.3, 3.4, 3.5 та 3.6 зображено структуру файлів розроблених інтерфейсів та класів моделей.

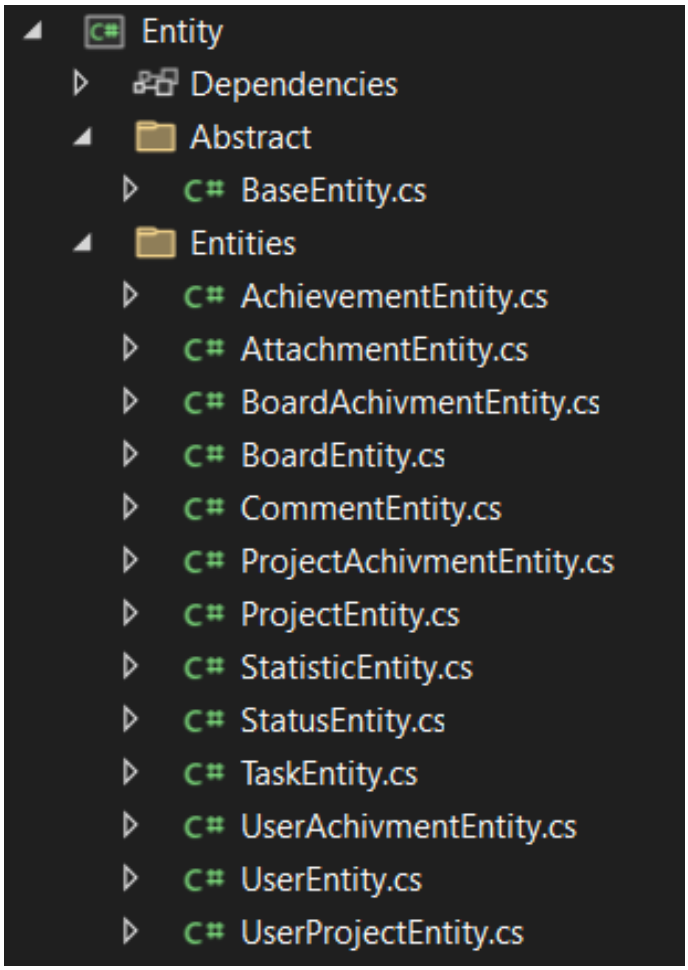


Рис. 3.3 Розроблені моделі даних для Database

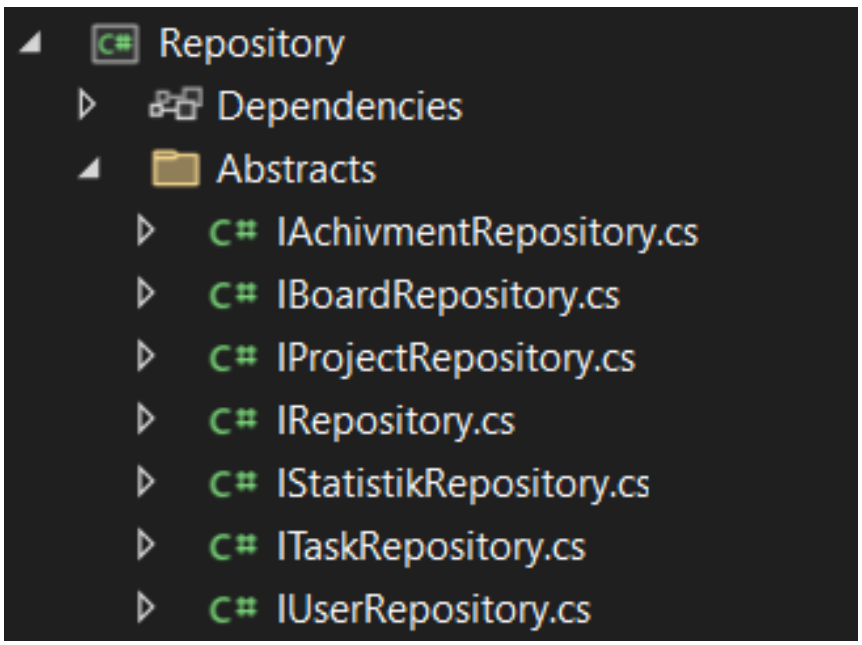


Рис. 3.4 Розроблені інтерфейси для виконавчих класів в Database

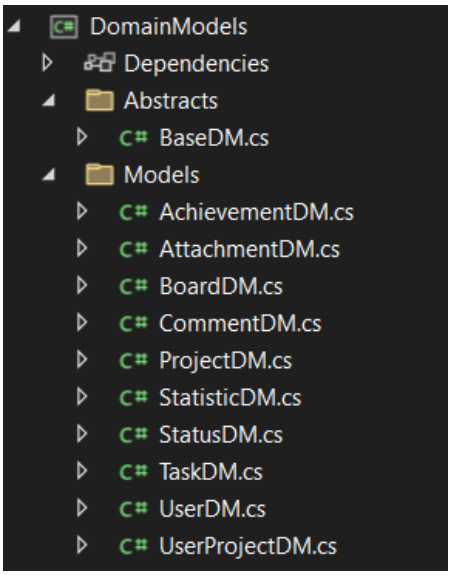


Рис. 3.5 Розроблені класи моделі даних для Domain

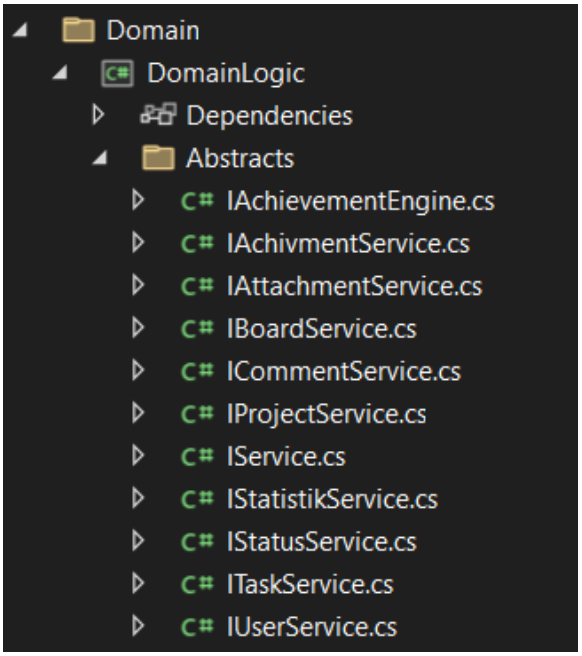


Рис. 3.6 Розроблені інтерфейси для виконавчих класів в Domain

Кількість виконавчих класів у Database та Domain відрізняються по тій причині що прості CRUD операції, які реалізовує група компонентів Database, можна прив’язати до одного з 6 класів репозиторіїв задля того, щоб зменшити кількість класів. В свою чергу вже в Domain необхідно виконувати розподілення у велику кількість сервісів функціоналу, щоб класи не були занадто великими та важкочитабельними.

Наступним етапом було створення класів для реалізації інтерфейсів, розроблених на попередньому етапі, створення класів контролерів та реєстрація класів у DI контейнери.

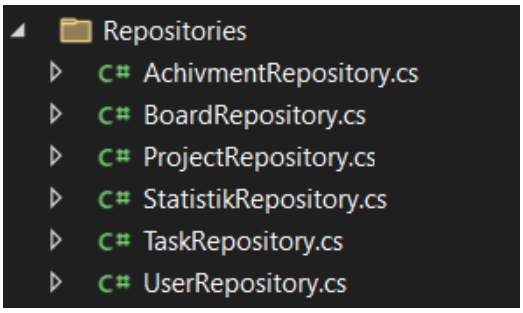


Рис. 3.7 Розроблені класи репозиторіїв

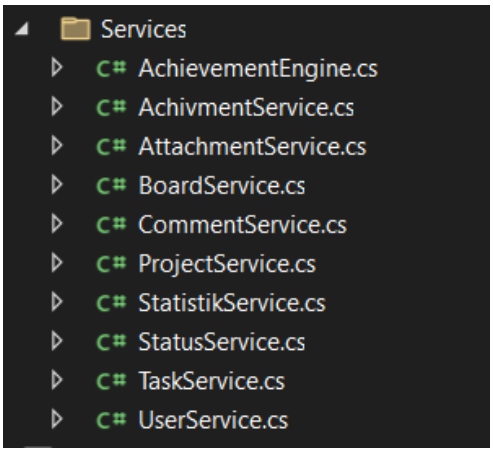


Рис. 3.8 Розроблені класи сервісів

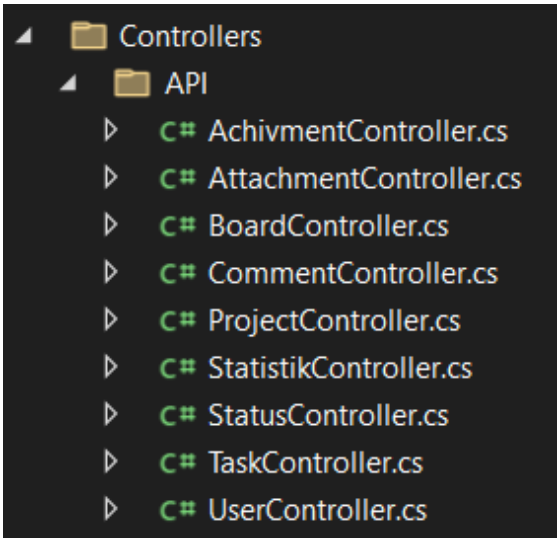


Рис. 3.9 Розроблені класи контролерів

```
builder.Services.AddScoped<IAchivmentRepository, AchivmentRepository>();
builder.Services.AddScoped<IBoardRepository, BoardRepository>();
builder.Services.AddScoped<IProjectRepository, ProjectRepository>();
builder.Services.AddScoped<ITaskRepository, TaskRepository>();
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IStatistikRepository, StatistikRepository>();


builder.Services.AddScoped<IAchievementEngine, AchievementEngine>();
builder.Services.AddScoped<IAchivmentService, AchivmentService>();
builder.Services.AddScoped<IAttachmentService, AttachmentService>();
builder.Services.AddScoped<IBoardService, BoardService>();
builder.Services.AddScoped<ICommentService, CommentService>();
builder.Services.AddScoped<IStatistikService, StatistikService>();
builder.Services.AddScoped<IProjectService, ProjectService>();
builder.Services.AddScoped<IStatusService, StatusService>();
builder.Services.AddScoped<ITaskService, TaskService>();
builder.Services.AddScoped<IUserService, UserService>();
```

Рис. 3.10 Реєстрація класів у DI контейнери

Наступним етапом було налаштування API-контроллерів та підключення swagger до системи задля майбутнього тестування.

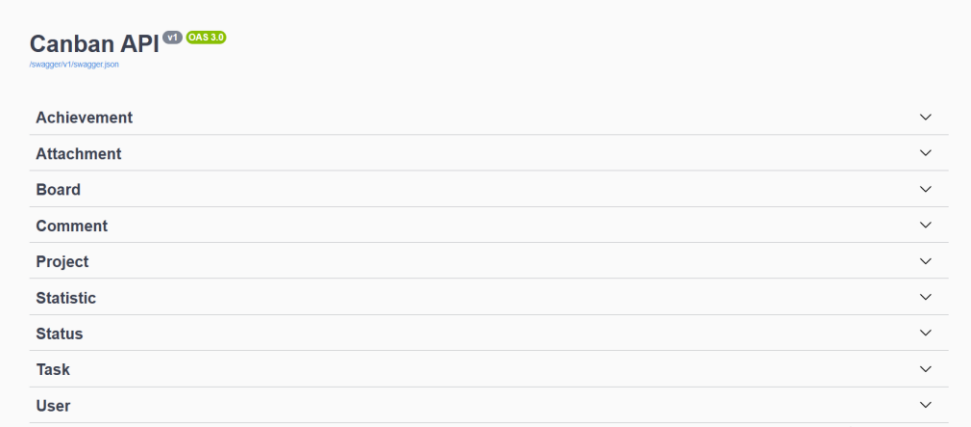


Рис. 3.11 Підключений swagger до системи

Наступним і заключним етапом у розробці основних компонентів є створення класів мапперів для того, щоб під час реалізації функціоналу не було проблем між різними типами класів моделей на різних рівнях архітектури.

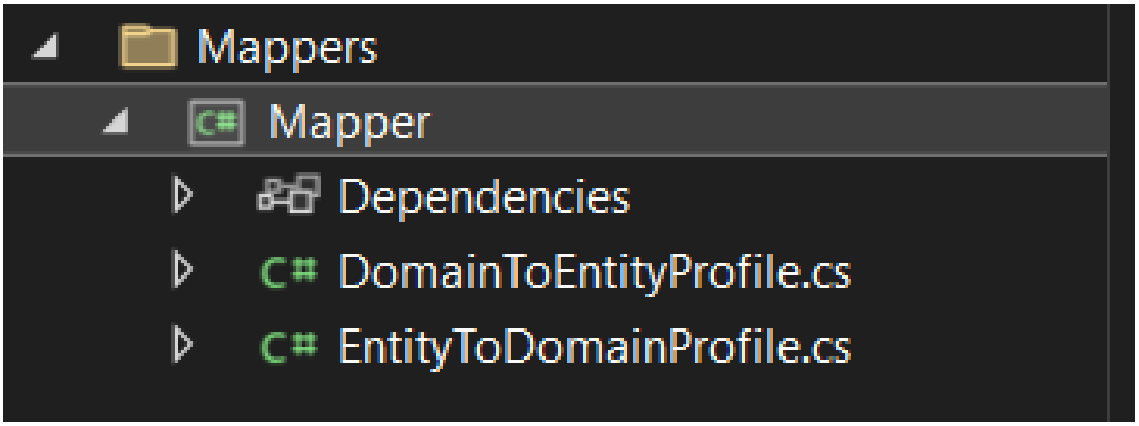


Рис. 3.12 Створені профілі маперів

```
46      // 4. AutoMapper
47      builder.Services.AddAutoMapper(typeof(EntityToDomainProfile), typeof(DomainToEntityProfile));
48
```

Рис. 3.13 Підключення розроблених профілів у AutoMapper

3.2 Реалізація функціоналу

В цьому підрозділі продемонстровано розробку функціоналу веб-застосунку на основі раніше розроблених основних компонентів.

Першим етапом було підключено Microsoft Entity Framework Core та створення класу контексту бази даних.

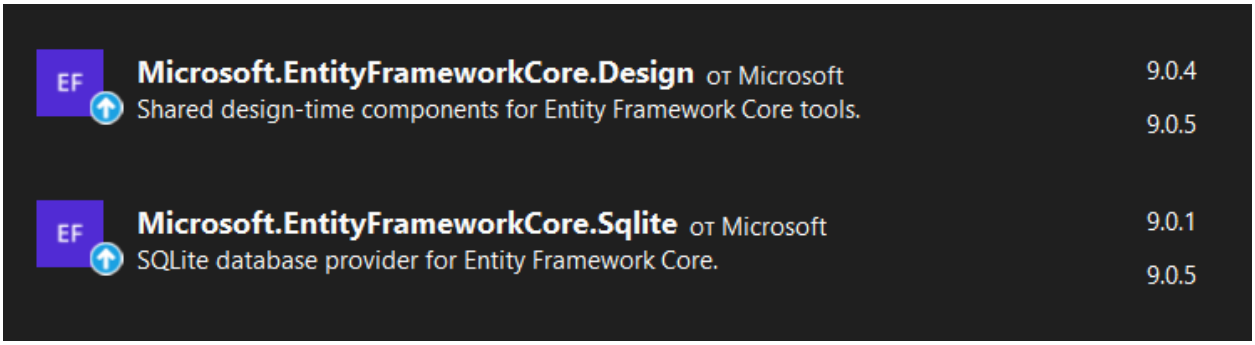


Рис. 3.14 Підключення Microsoft Entity Framework Core

```
public class AppDBContext : DbContext
{
    public DbSet<AchievementEntity> achievements {  get; set; }
    public DbSet<UserEntity> users {  get; set; }
    public DbSet<UserAchivmentEntity> userAchivments { get; set; }
    public DbSet<ProjectEntity> projects { get; set; }
    public DbSet<BoardEntity> boards { get; set; }
    public DbSet<BoardAchivmentEntity> boardAchivments { get; set; }
    public DbSet<ProjectAchivmentEntity> projectAchivments { get; set; }
    public DbSet<UserProjectEntity> usersProjects { get; set; }
    public DbSet<StatusEntity> statuses { get; set; }
    public DbSet<TaskEntity> tasks { get; set; }
    public DbSet<CommentEntity> comments { get; set; }
    public DbSet<AttachmentEntity> attachments { get; set; }
    public DbSet<StatisticEntity> statistics { get; set; }

    public AppDBContext(DbContextOptions<AppDBContext> options) : base(options)
    {
    }
}
```

Рис. 3.15 Створений клас контекст

Наступним етапом було реалізовано функціонал інтерфейсів у класах-репозиторіях та у класах-сервісах.

```
Ссылок: 2
public class StatistikRepository : IStatistikRepository
{
    private readonly AppDBContext _context;

    Ссылок: 0
    public StatistikRepository(AppDBContext context)...

    Ссылок: 8
    public async Task AddMainObjAsync(StatisticEntity entity)...

    Ссылок: 8
    public async Task DeleteMainObj(StatisticEntity entity)
    {
        entity.isVisible = false;
        await UpdateMainObj(entity);
    }

    Ссылок: 12
    public async Task<IEnumerable<StatisticEntity>> GetAllMainObjAsync()
    {
        var list = await _context.statistics.ToListAsync();
        return list;
    }

    Ссылок: 2
    public async Task<StatisticEntity?> GetMainObjByIdAsync(int id)
    {
        var list = await GetAllMainObjAsync();
        return list.FirstOrDefault(x => x.Id == id);
    }

    Ссылок: 10
    public async Task UpdateMainObj(StatisticEntity entity)
    {
        _context.statistics.Update(entity);
        await _context.SaveChangesAsync();
    }
}
```

Рис. 3.16 Один із розроблених класів-репозиторіїв

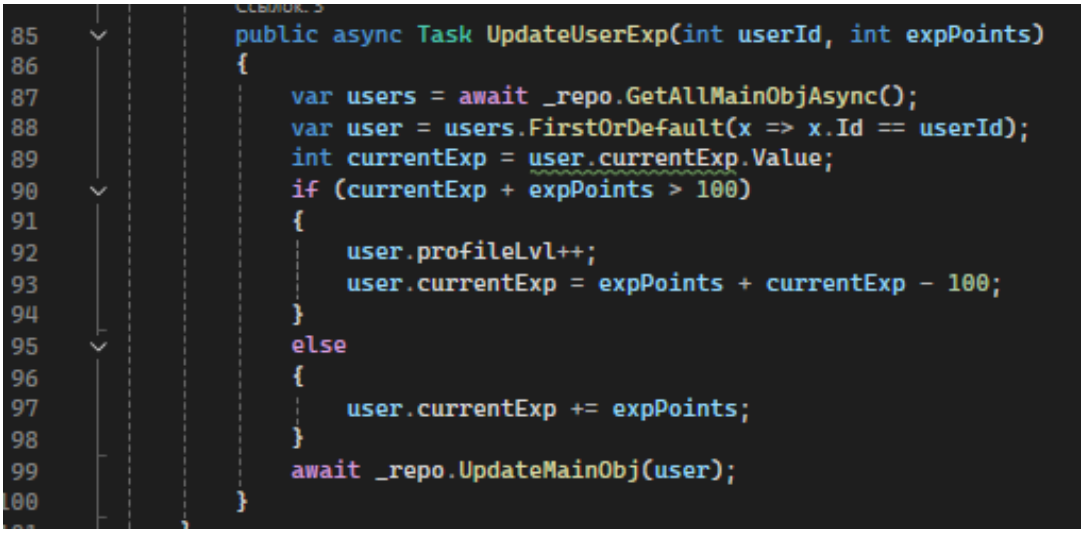
```
public class StatistikService : IStatistikService
{
    private readonly IStatistikRepository _repo;
    private readonly IMapper _mapper;

    Ссылок: 0
    public StatistikService(IStatistikRepository repo, IMapper mapper)...
    Ссылок: 10
    public async Task AddAsync(StatisticDM model)...
    Ссылок: 7
    public async Task DeleteAsync(StatisticDM model)
    {
        var mappedEntity = _mapper.Map<StatisticEntity>(model);
        await _repo.DeleteMainObj(mappedEntity);
    }
    Ссылок: 12
    public async Task<IEnumerable<StatisticDM>> GetAllAsync()
    {
        var list = await _repo.GetAllMainObjAsync();
        var mappedList = _mapper.Map<IEnumerable<StatisticDM>>(list);
        return mappedList;
    }
    Ссылок: 17
    public async Task<StatisticDM?> GetByIdAsync(int id)
    {
        var list = await GetAllAsync();
        var statistik = list.FirstOrDefault(x => x.Id == id);
        return statistik;
    }
    Ссылок: 2
    public async Task<IEnumerable<StatisticDM>> GetObjStatistik(string entityType, int entityId)
    {
        var list = await GetAllAsync();
        var objStatistik = list.Where(x => x.EntityType == entityType && x.EntityId == entityId);
        return objStatistik;
    }
    Ссылок: 10
    public async Task UpdateAsync(StatisticDM model)
    {
        var mappedEntity = _mapper.Map<StatisticEntity>(model);
        await _repo.UpdateMainObj(mappedEntity);
    }
}
```

Рис. 3.17 Один із розроблених класів-сервісів

Наступним кроком було реалізовано функціонал додавання досягнень користувача та оновлення досвіду профіля користувача. Функціонал було реалізовано наступним чином: оновлення досвіду відбувається за допомогою UserServices, а функціонал перевірки та надання нових досягнень реалізовано у класі AchivmentEngine.

```
public class AchievementEngine : IAchievementEngine
{
    private readonly IAchivmentService _achvSvc;
    private readonly IUserService _userService;

    Ссылок: 0
    public AchievementEngine(
        IAchivmentService achvSvc,
        IUserService userService)
    {
        _achvSvc = achvSvc;
        _userService = userService;
    }

    Ссылок: 2
    public async Task<IEnumerable<AchievementDM>> HandleMetricUpdated(
        int userId,
        string metricKey,
        int newValue)
    {
        var defs = await _achvSvc.GetByMetricAsync(metricKey);

        var awarded = await _achvSvc.GetAchivmentsByObjId(userId, "User");
        var awardedCodes = awarded.Select(a => a.code).ToHashSet();

        var toAward = defs
            .Where(d => d.threshold <= newValue
                    && !awardedCodes.Contains(d.code))
            .ToList();

        var result = new List<AchievementDM>();

        foreach (var def in toAward)
        {
            await _achvSvc.AddAchivmentToObject(userId, "User", (int)def.Id);

            await _userService.UpdateUserExp(userId, (int)def.expPoints);

            result.Add(def);
        }

        return result;
    }
}
```

Рис. 3.18 Клас оновлення досягнень



```
public async Task UpdateUserExp(int userId, int expPoints)
{
    var users = await _repo.GetAllMainObjAsync();
    var user = users.FirstOrDefault(x => x.Id == userId);
    int currentExp = user.currentExp.Value;
    if (currentExp + expPoints > 100)
    {
        user.profileLvl++;
        user.currentExp = expPoints + currentExp - 100;
    }
    else
    {
        user.currentExp += expPoints;
    }
    await _repo.UpdateMainObj(user);
}
```

Рис. 3.19 Метод оновлення досвіду користувача

Наступним етапом є підключення автентифікації користувачів у системі та хешування паролю користувача для подальшого зберігання у базі даних.

```
81
82      [HttpPost("authorize")]
        Ссылок: 0
83      public async Task<IActionResult> AuthorizeAsync(
84          [FromQuery] string login,
85          [FromQuery] string password)
86      {
87          var user = await _service.GetByLoginAsync(login);
88          if (user == null)
89              return Unauthorized("Invalid login or password");
90
91          if (!BCrypt.Net.BCrypt.Verify(password, user.userPassword!))
92              return Unauthorized("Invalid login or password");
93
94          var claims = new List<Claim>
95          {
96              new Claim(ClaimTypes.Name,   user.userLogin ?? ""),
97              new Claim("UserId",          (user.Id ?? 0).ToString()),
98              new Claim("UserName",        user.userName  ?? ""),
99              new Claim(ClaimTypes.Email,  user.userEmail ?? ""),
100             new Claim("profileLvl",      (user.profileLvl ?? 0).ToString()),
101             new Claim("currentExp",      (user.currentExp ?? 0).ToString())
102         };
103         var principal = new ClaimsPrincipal(new ClaimsIdentity(claims, "MyCookieAuth"));
104         await HttpContext.SignInAsync("MyCookieAuth", principal);
105
106         return Ok(user);
107     }
```

Рис. 3.20 Метод авторизації користувача

```
21      [HttpPost]
        Ссылок: 0
22      public async Task<ActionResult<UserDM>> AddAsync([FromBody] UserDM dto)
23      {
24          await _service.AddAsync(dto);
25
26          var created = await _service.GetByLoginAsync(dto.userLogin!);
27
28          var claims = new List<Claim>
29          {
30              new Claim(ClaimTypes.Name,   created.userLogin ?? ""),
31              new Claim("UserId",          (created.Id ?? 0).ToString()),
32              new Claim("UserName",        created.userName  ?? ""),
33              new Claim(ClaimTypes.Email,  created.userEmail ?? ""),
34              new Claim("profileLvl",      (created.profileLvl ?? 0).ToString()),
35              new Claim("currentExp",      (created.currentExp ?? 0).ToString())
36          };
37          var identity = new ClaimsIdentity(claims, "MyCookieAuth");
38          var principal = new ClaimsPrincipal(identity);
39
40          await HttpContext.SignInAsync("MyCookieAuth", principal);
41
42          return Ok(created);
43      }
44
```

Рис. 3.21 Метод реєстрації користувача

```
21      public async Task AddAsync(UserDM model)
22      {
23          var hashed = BCrypt.Net.BCrypt.HashPassword(model.userPassword ?? "");
24          model.userPassword = hashed;
25
26          var mappedUser = _mapper.Map<UserEntity>(model);
27          await _repo.AddMainObjAsync(mappedUser);
28      }
```

Рис. 3.22 Метод для збереження користувача

Фінальним кроком було створення RazorPages для інтерфейсу користувача.

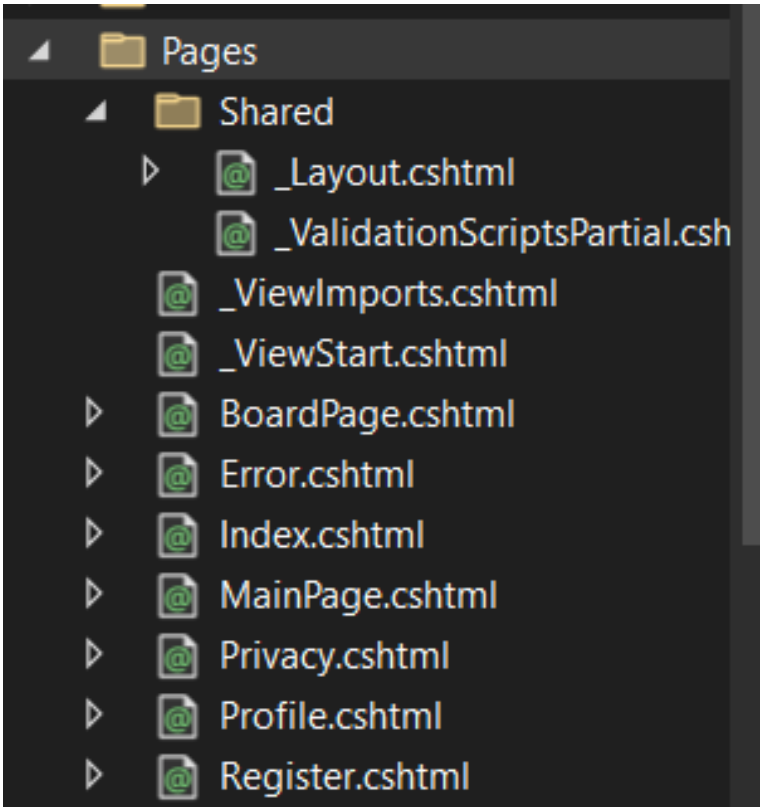


Рис. 3.23 Розроблені сторінки web-застосунку

4. ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

4.1 Обґрунтування способу тестування

Для виконання тестування розробленого веб-застосунку було прийнято рішення провести 2 види перевірок:

- ручне тестування;
- модульне тестування.

Вибір саме цих двох типів перевірок був здійснений через специфіку веб-застосунку та специфіку технологій, які використовувалися під час розробки веб-застосунку. Ручне та модульне тестування дозволяє виконати комплексну перевірку інтерфейсу користувача та функціональної реалізації веб-застосунку.

Ручне тестування виконується за попередньо створеними тестовими сценаріями. Ручне тестування направлено на тестування інтерфейсу користувача та в цілому аспектів, які неможливо автоматизувати. Також, інтерфейс користувача найкраще може оцінити лише жива людина. Тобто, ручне тестування буде покривати повністю тестування інтерфейсу користувача та частково покривати тестування функціоналу, а саме функції, які важко автоматизувати, чи для яких оптимальним є тестування саме живою людиною. [19]

Модульне тестування виконується за попередньо створеними автоматизованими сценаріями виконання, а саме Unit-тестами. Unit-тести допомагають швидко та без участі користувача виконати перевірку тестових сценаріїв, порівняти їх з очікуваними результатами та видати результати тестування. Автоматизовані тести мінімізують вплив людського фактору та є дуже ефективними для повторюваних задач – наприклад, при регресійному тестуванні після внесення змін у код. [20]

4.2 Тестові сценарії

В цьому розділі підрозділі описано тестові перевірки відповідно до раніше визначених типів тестування.

Тестові сценарії зображені у таблиці 4.1.

Таблиця 4.1

Основні тестові сценарії системи

№	Тип тесту	Тестовий сценарій	Очікуваний результат
1	Функціональний	**Авторизація:** 1. Користувач переходить на початкову сторінку; 2. Користувач заповнює поля для авторизації; 3. Користувач натикає на кнопку «Увійти».	Користувач переходить на сторінку його просторів
2	Функціональний	**Реєстрація:** 1. Користувач переходить на початкову сторінку; 2. Користувач натискає кнопку «Ще не маю акаунту»; 3. Користувач заповнює усі дані для реєстрації; 4. Користувач натискає на кнопку «Зареєструвати».	Користувач переходить на сторінку просторів та бачить свій логін у правому верхньому куті сторінки
3	Функціональний	**Створення нової задачі:** 1. Користувач переходить на дошку; 2. Користувач натискає у статусі кнопку «Додати задачу»; 3. Користувач повинен ввести усі необхідні дані; 4. Користувач натискає на кнопку «Додати».	Нова задача з’явилася у статусі

Продовження таблиці 4.1

Основні тестові сценарії системи

4	Функціональний	**Створення статусу:** 1.Користувач переходить на дошку; 2. Далі натискає на кнопку «Додати статус»; 3. Користувач вводить необхідні дані; 4. Та натискає на кнопку «Додати».	На дошці з’явився новий запис статусу
5	Модульний	**Видання користувачам досягнень:** Створити 5 користувачів та створити кожному від 3 до 18 проєктів	Користувач повинен отримувати досягнення при створені першої дошки, 5-ї дошки та 15-ї дошки
6	Модульний	**Створення дошок в просторі:** Створити у двох просторах по п’ять дошок	Успішно створено дошки у просторах
7	UI/UX	**Інтуїтивність інтерфейсу:** Новий користувач системи повинен виконати базові задачі, а саме створити обліковий запис, додати простір, додати дошку, статус та створити декілька задач	Користувач без зайвих питань та великих пауз на обдумування дії виконав базові задачі
8	UI/UX	**Відповідність стилів дизайну:** Перевірка стилів дизайну на підтримку та роботу без помилок у різних браузерах	Дизайн відповідає задумці та працює без проблем в тогп 10 найпопулярніших браузерах

Продовження таблиці 4.1

Основні тестові сценарії системи

9	Безпековий	**Вхід із неправильним паролем:** Користувач вводить правильний логін, але вводить не правильний пароль	Система сповіщає користувача про те що він ввів не вірні дані для входу
10	Безпековий	**Спроба звернутися до http API без авторизації:** Користувач не виконуючи вхід в систему намагається звертатися за адресами для роботи веб-застосунку	Користувача автоматично переадресовує на сторінку авторизації

4.3 Результати тестування

Функціональне тестування показало, що у системі відсутні критичні дефекти та дефекти високого рівня. Система працює правильно та містить в собі елементи для обробки помилок або інших випадків, які знаходяться поза сценарієм виконання задачі.

UI/UX тестування показало, що стилі працюють без помилок в інших браузерах, а також те, що інтерфейс є інтуїтивно зрозумілим для нового користувача.

Модульне тестування показало, що система без помилок створює нові записи у базі даних та відображає їх без помилок. Також, модульне тестування показало, що механізм видачі досягнень користувачам працює без проблем.

ВИСНОВКИ

Під час виконання дипломного проєкту було проведено аналіз предметної області дипломного проєкту, спроєктовано веб-застосунок для реалізації поставленої задачі дипломного проєкту, розроблено веб-застосунок та проведено тестування розробленого веб-застосунку.

Під час виконання першого розділу, було проведено аналіз предметної області, розглянуто та проаналізовано існуючі рішення на ринку, визначено та сформовано вимоги до веб-застосунку та виконано постановку задачі. Також – визначено технології, які будуть використовуватися для реалізації веб-застосунку.

Під час виконання другого розділу, було спроєктовано архітектуру системи, спроєктовано базу даних, а саме спроєктовано логічну схему у вигляді ER-діаграми та спроєктовано фізичні таблиці у базі даних, виконано моделювання вимог (а саме, побудова діаграми використання та опис варіантів використання системи), після чого на основі діаграми варіантів використання було побудовано діаграму класів основної бізнес-логіки системи. На основі усіх даних проєктування було спроєктовано інтерфейс користувача, а саме мапу сайту та інтерфейс форм та сторінок веб-застосунку.

Під час виконання третього розділу спочатку було розроблено основні компоненти веб-застосунку (згідно із спроєктованими діаграмами у другому розділі), реалізовано основний функціонал веб-застосунку та розроблено веб-сторінки для роботи користувача із системою.

Під час виконання четвертого розділу було проведено тестування розробленого веб-застосунку. Тестування не виявило критичних та помилок високого рівня, але було виявлено декілька помилок низького рівня (в підсумку, їх було усунено).

Фінальним результатом дипломної роботи стала готова MVP-модель веб-застосунку для управління IT проєктами за методологією Kanban з використанням механізмів гейміфікації. Проєкт готовий для демонстрації першим потенційним клієнтам з ціллю отримання відгуків для подальшого внесення фінальних штрихів у веб-застосунок та виходу його на ринок.

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

1. Малінський Д.В., Худік Б.О. Вплив механізмів гейміфікації на мотивацію. VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». 15.04.25, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025, (подано до друку)
2. Малінський Д.В. Захист інформації у web-застосунку управління IT-проєктами за методологією Kanban з використанням механізмів гейміфікації. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 297-298

ПЕРЕЛІК ПОСИЛАНЬ

1. Головченко Ю. Управління IT проєктами – як це відбувається на практиці. Lemon School. 29.07.2024. URL: (дата звернення: 01.05.2025).

2. Agile. What is Kanban?. Agile Aliance. Agile Glossary Kanban. URL: https://www.agilealliance.org/glossary/kanban/ (дата звернення: 02.05.2025).

3. Валерій Б. Kanban — планування, що можна використовувати в побуті. *Cases Media*. *CodeIT*. 21.09.2023. URL: https://cases.media/en/article/kanban-planuvannya-sho-mozhna-vikoristovuvati-v-pobuti (дата звернення: 02.05.2025).

4. Trello. Веб-застосунок для керування проєктами за методологією Kanban. URL: https://www.trello.com (дата звернення: 03.05.2025).

5. – Jira. Веб-застосунок для керування різними процесами у проєкті. URL: https://www.atlassian.com/software/jira (дата звернення: 03.05.2025).

6. – MeisterTask. Веб-застосунок для керування повсякденними задачами за допомогою Kanban. URL: https://www.meistertask.com (дата звернення: 03.05.2025).

7. Microsoft. Microsoft Learn. A tour of the C# language. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview (дата звернення: 04.05.2025).

8. Microsoft. Microsoft Learn. What is Visual Studio? URL: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022 (дата звернення: 04.05.2025).

9. Microsoft. Microsoft Learn. What is ASP.NET Core? URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core (дата звернення: 04.05.2025).

10. Микола Л. Що таке HTML та як за допомогою нього увійти в IT. *MC.Today*. 29.09.2022. URL: https://mc.today/uk/shho-take-html-ta-yak-za-dopomogoyu-nogo-uvijti-do-it/ (дата звернення: 04.05.2025).

11. Anthony C. What Is CSS?. *Builtin*. 29.12.2022. URL: https://builtin.com/software-engineering-perspectives/css (дата звернення: 04.05.2025).

12. Brian O. JavaScript Demystified: Why You Should Learn This Essential Language. *Code institute*. 05.03.2020. URL: https://codeinstitute.net/de/blog/what-is-javascript-and-why-should-i-learn-it/ (дата звернення: 04.05.2025).

13. PostgreSQL. About. URL: https://www.postgresql.org/about/ (дата звернення: 04.05.2025).

14. Serge A. What is Railway App?. *Medium*. 25.08.2024. URL: https://medium.com/@sangeloz69/what-is-railway-app-5b1bd17aa8f0 (дата звернення: 04.05.2025).

15. Microsoft. Microsoft Learn. Entity Framework Core. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 04.05.2025).

16. Kirtesh S. Introduction To ASP.NET Core Razor Pages. URL: https://www.c-sharpcorner.com/article/introduction-to-asp-net-core-razor-pages/ (дата звернення: 04.05.2025).

17. Товма О. Основні типи архітектури програмного забезпечення. URL: https://www.artofba.com/uk/post/main-types-of-software-architecture (дата звернення: 10.05.2025).

18. Лукас Б. N Tier (Багаторівневий), 3-рівневий, 2-рівневий Architектура з ПРИКЛАДОМ. URL: https://www.guru99.com/uk/n-tier-architecture-system-concepts-tips.html (дата звернення 10.05.2025)

19. Malyy A. Quality Assurance: Automated vs Manual Testing vs Unit Testing. URL: https://triare.net/insights/automated-vs-manual-testing/ (дата звернення 20.05.2025)

20. Microsoft. Microsoft Learn. Unit testing best practices for .NET. URL: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices (дата звернення 20.05.2025)

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для управління IT-проєктами за методологією Kanban з використанням механізмів гейміфікації

Виконав студент 4 курсу
групи ПД-44
Малінський Денис Вікторович
Керівник роботи
доктор філософії (PhD), доцент кафедри ІПЗ, Худік Богдан Олександрович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – покращення процесу керування IT-проєктами за допомогою методології Kanban шляхом додавання механізмів гейміфікації у процес роботи.

Об'єкт дослідження – процес керування IT-проєктами за допомогою методології Kanban з механізмами гейміфікації.

Предмет дослідження – веб-застосунок для керування IT-проєктами за методологією Kanban з механізмами гейміфікації.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз та визначити недоліки та переваги існуючих рішень для керування IT проектами за методологією Kanban.
2. Визначити та сформулювати функціональні та нефункціональні вимоги до web-застосунку.
3. Провести огляд засобів для розробки та визначити технології та інструменти, необхідні для розробки web-застосунку для управління IT-проєктами за методологією Kanban.
4. Спроектувати майбутній web-застосунок для керування проєктами за методологією Kanban з використанням елементів гейміфікації.
5. Розробити серверну та клієнтську частини web-застосунку.
6. Провести тестування розробленого web-застосунку.

3

АНАЛІЗ АНАЛОГІВ

Характеристика	Trello	Jiro	MeisterTask	MyCanban
Приєднання учасників	+	+	+	+
Додавання/налаштування статусів	+	+	+	+
Робота з картками та файлами карток	+	+	+	+
Додавання дедлайнів до задач	+	+	+	+
Необмежена кількість просторів	- (потрібна підписка)	- (потрібна підписка)	- (потрібна підписка)	+
Механіка досягнень	-	-	-	+
Механіка рівню профіля	-	-	-	+

4

ВИМОГИ ДО WEB-ЗАСТОСУНКУ

Функціональні вимоги:

1. Зберігати інформацію про створені простори та дошки у них.
2. Робота із просторами.
3. Робота із дошками.
4. Робота з учасниками дошок.
5. Автентифікація.
6. Механіка рівнів облікового запису.
7. Нараховувати досягнення користувачу за виконання певних дій у застосунку.
8. Підвищувати рівень простору за активність у ньому (кількість задач, коментарів тощо).
9. Нараховувати досягнення простору за досягнення встановлених цілей.

Нефункціональні вимоги:

1. Повідомлення про помилку повинні з'являтися не довше, а наж 5 секунд та містити текст помилки до 100 символів
2. Безпека – 100% запитів до серверу повинні здійснюватися через протокол HTTPS
3. Швидкодія – реакція web-застосунок на дію користувача не повинна перевищувати 5 секунд
4. Веб-застосунок повинен бути доступним у 95%
5. Тон і стиль геймифікації повинен відповідати діловому стилю

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

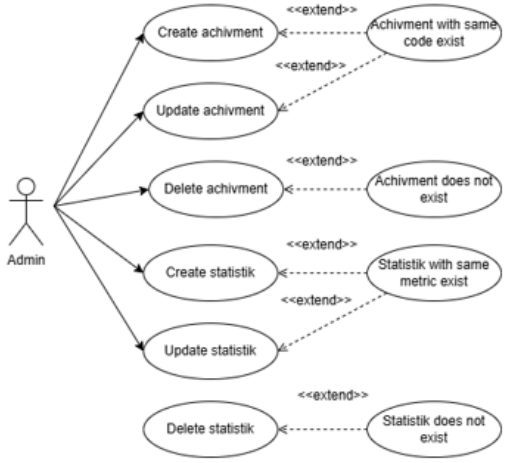
Visual Studio 2022

PostgreSQL

RailwayApp

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

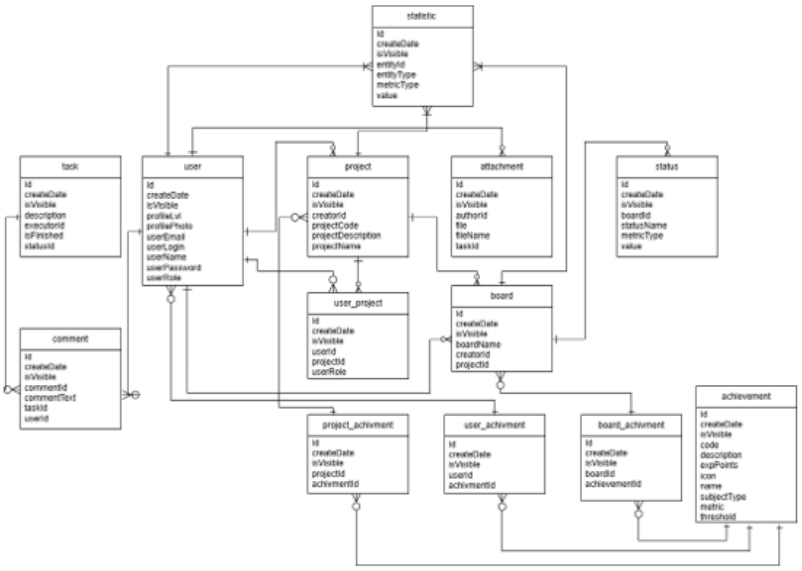


7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

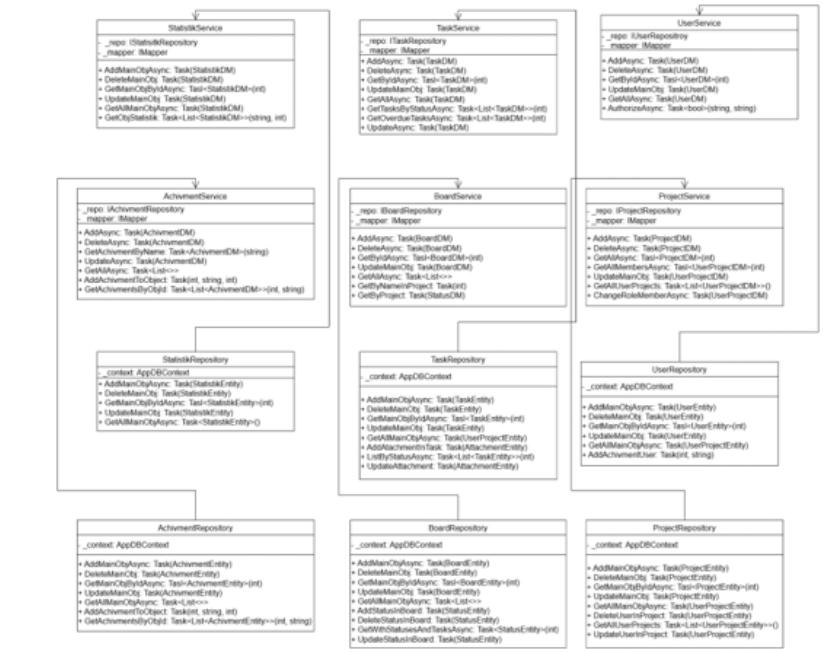
8

СТРУКТУРА БАЗИ ДАННИХ



АРХІТЕКТУРА ДОДАТКУ

ДІАГРАМА КЛАСІВ ОСНОВНОЇ ЛОГІКИ

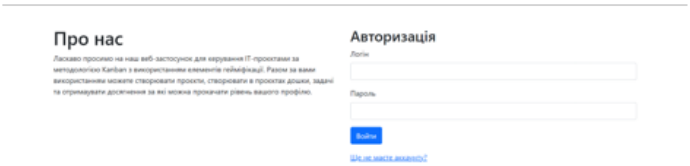


11

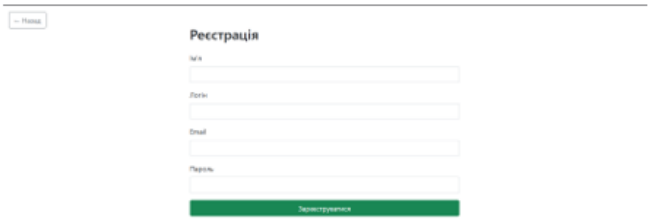
МАПА WEB-ЗАСТОСУНКУ

12

ЕКРАННІ ФОРМИ



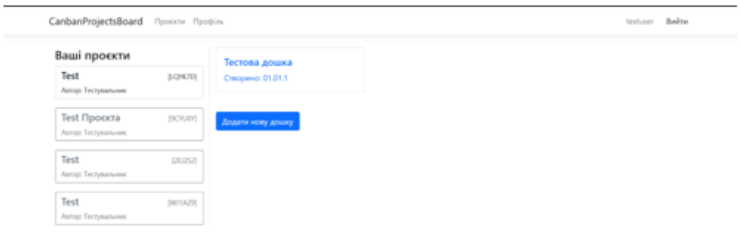
Форма авторизації



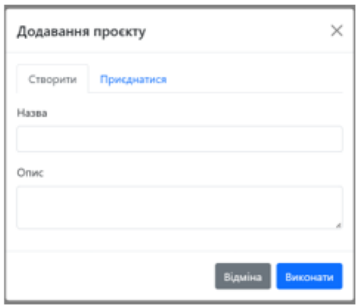
Форма реєстрації

13

ЕКРАННІ ФОРМИ



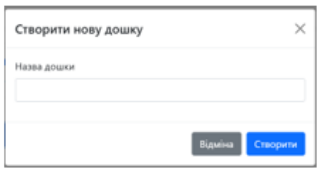
Сторінка проєктів



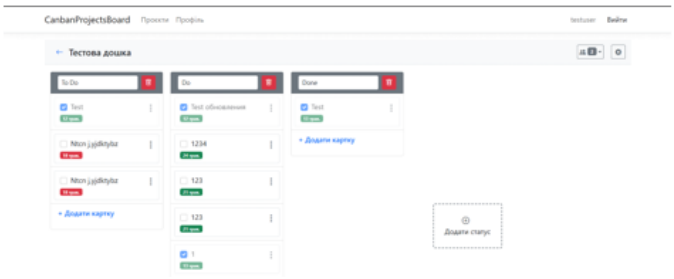
Форма створення проєкту

14

ЕКРАННІ ФОРМИ



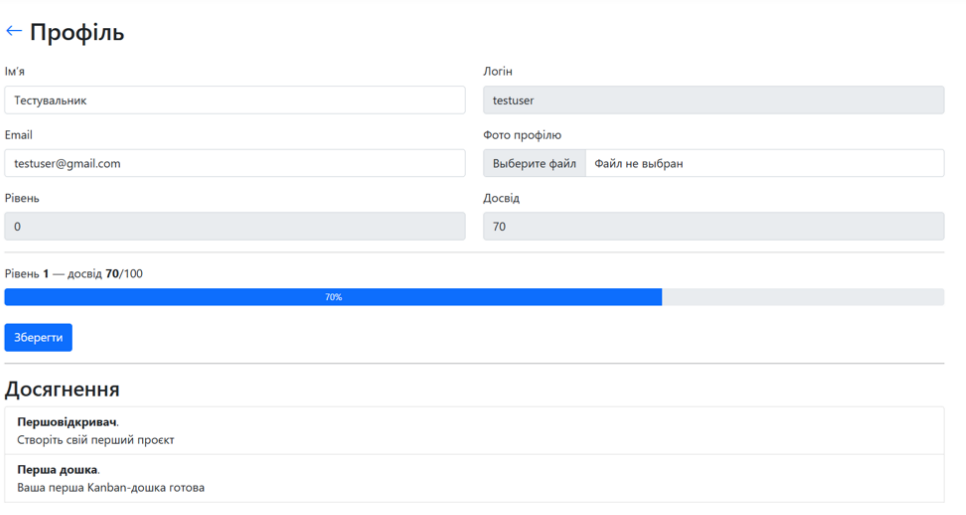
Форма створення дошки



Сторінка дошки

15

ЕКРАННІ ФОРМИ



Сторінка профілю користувача

16

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Малінський Д.В., Худік Б.О. Вплив механізмів гейміфікації на мотивацію. VI Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025, (подано до друку)

2. Малінський Д.В. Захист інформації у web-застосунку управління IT-проєктами за методологією Kanban з використанням механізмів гейміфікації. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 297-298

17

ВИСНОВКИ

1. Проведено аналіз, завдяки якому було визначено недоліки та переваги існуючих рішень для керування IT проєктами за методологією Kanban. Це дозволило глибше зрозуміти предметну область та визначити ключові сильні сторони, на основі яких слід будувати web-застосунок.
2. Визначено та сформовано детальні вимоги до web-застосунку, а саме: функціональні вимоги (робота з просторами, робота з досягненнями) та нефункціональні вимоги (швидкодія, доступність).
3. Проведено огляд засобів для розробки та визначено технології та інструменти, необхідні для розробки web-застосунку для управління IT-проєктами за методологією Kanban. У якості основного фреймворку для серверної частини обрано ASP.Net Core, у якості основної мови програмування C# та у якості СУБД PostgreSQL.
4. Спроєктовано web-застосунок для керування проєктами за методологією Kanban з використанням елементів гейміфікації, а саме: структуру базу даних, діаграму класів та діаграму варіантів використання, що дали більш чітке та структуроване бачення процесу розробки web-застосунку.
5. Розроблено серверну та клієнтську частини web-застосунку, в яких реалізовано базовий функціонал керування проєктами за методологією Kanban та додано функціонал досягнень, статистики та рівню профіля користувача.
6. Проведено тестування розробленого web-застосунку, під час якого було виявлено декілька незначних дефектів, які були одразу виправлені.

18

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

```
using AutoMapper;
using DomainLogic.Abstracts;
using DomainModels.Models;
using Entity.Entities;
using Repository.Abstracts;


namespace DomainLogic.Services
{
   public class UserService : IUserService
   {
      private readonly IUserRepository _repo;
      private readonly IMapper _mapper;

      public UserService(IUserRepository repo,
IMapper mapper)
      {
         _repo = repo;
         _mapper = mapper;
      }

      public async Task AddAsync(UserDM model)
      {
         var hashed =
BCrypt.Net.BCrypt.HashPassword(model.userPassw
ord ?? "");
         model.userPassword = hashed;

         var mappedUser =
_mapper.Map<UserEntity>(model);
         await
_repo.AddMainObjAsync(mappedUser);
      }


      public async Task<bool> AuthorizeAsync(string
login, string password)
      {
         var users = await
_repo.GetAllMainObjAsync();
         var user = users.FirstOrDefault(x =>
x.userLogin == login && x.userPassword ==
password);
         if (user == null)
         {
            return false;
         }
         return true;
      }

      public async Task DeleteAsync(UserDM model)
      {
         var mappedUser =
_mapper.Map<UserEntity>(model);
         await _repo.DeleteMainObj(mappedUser);
      }

      public async Task<IEnumerable<UserDM>>
GetAllAsync()
      {
         var list = await _repo.GetAllMainObjAsync();
         var mappedList =
_mapper.Map<IEnumerable<UserDM>>(list);
         return mappedList;
      }

      public async Task<UserDM?>
GetByIdAsync(int id)
      {
         var list = await GetAllAsync();
         var user = list.FirstOrDefault(x => x.Id == id);
         return user;
      }

      public async Task<UserDM>
GetByLoginAsync(string login)
      {
         var list = await GetAllAsync();
         var user = list.FirstOrDefault(x => x.userLogin
== login);
         return user;
      }

      public async Task<UserDM>
GetUserByLoginPassword(string login, string
password)
      {
         var list = await GetAllAsync();
         var user = list.FirstOrDefault(x => x.userLogin
== login && x.userPassword == password);
         return user;
      }

      public async Task UpdateAsync(UserDM model)
      {
         if
(!string.IsNullOrEmpty(model.userPassword))
         {
            model.userPassword =
BCrypt.Net.BCrypt.HashPassword(model.userPassw
ord);
         }
         var mappedUser =
_mapper.Map<UserEntity>(model);
         await _repo.UpdateMainObj(mappedUser);
      }
      public async Task UpdateUserExp(int userId, int
expPoints)
      {
         var users = await
_repo.GetAllMainObjAsync();
         var user = users.FirstOrDefault(x => x.Id ==
userId);
         int currentExp = user.currentExp.Value;
```

```
        if (currentExp + expPoints > 100)
        {
            user.profileLvl++;
            user.currentExp = expPoints + currentExp -
100;
        }
        else
        {
            user.currentExp += expPoints;
        }
        await _repo.UpdateMainObj(user);
    }
  }
}

using DomainLogic.Abstracts;
using DomainLogic.Services;
using DomainModels.Models;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Security.Claims;
using System.Threading.Tasks;


namespace CanbanProjectsBoard.Controllers.API
{
  [ApiController]
  [Route("api/users")]
  public class UserController : ControllerBase
  {
    private readonly IUserService _service;
    public UserController(IUserService service) =>
_service = service;

    // POST api/users
    [HttpPost]
    public async Task<ActionResult<UserDM>>
AddAsync([FromBody] UserDM dto)
    {
      await _service.AddAsync(dto);

      var created = await
_service.GetByLoginAsync(dto.userLogin!);

      var claims = new List<Claim>
      {
        new Claim(ClaimTypes.Name,
created.userLogin ?? ""),
        new Claim("UserId", (created.Id ??
0).ToString()),
        new Claim("UserName",
created.userName ?? ""),
        new Claim(ClaimTypes.Email,
created.userEmail ?? ""),
        new Claim("profileLvl",
(created.profileLvl ?? 0).ToString()),
        new Claim("currentExp",
(created.currentExp ?? 0).ToString())
      };
      var identity = new ClaimsIdentity(claims,
"MyCookieAuth");
      var principal = new ClaimsPrincipal(identity);

      await
HttpContext.SignInAsync("MyCookieAuth",
principal);

      return Ok(created);
    }

    // GET api/users
    [HttpGet]
    public async
Task<ActionResult<IEnumerable<UserDM>>>
GetAllAsync()
      => Ok(await _service.GetAllAsync());

    // GET api/users/{id}
    [HttpGet("{id:int}")]
    public async Task<ActionResult<UserDM>>
GetByIdAsync(int id)
    {
      var dm = await _service.GetByIdAsync(id);
      return dm is null ? NotFound() : Ok(dm);
    }

    // PUT api/users/{id}
    [HttpPut("{id:int}")]
    public async Task<ActionResult>
UpdateAsync(int id, [FromBody] UserDM dto)
    {
      if (id != dto.Id) return BadRequest();
      await _service.UpdateAsync(dto);
      return NoContent();
    }

    // DELETE api/users/{id}
    [HttpDelete("{id:int}")]
    public async Task<ActionResult>
DeleteAsync(int id)
    {
      var user = await _service.GetByIdAsync(id);
      await _service.DeleteAsync(user);
      return NoContent();
    }

    [HttpGet("register")]
    public IActionResult Register()
    {
      return Redirect("/Register");
    }

    [HttpPost("authorize")]
    public async Task<IActionResult>
AuthorizeAsync(
      [FromQuery] string login,
      [FromQuery] string password)
    {
      var user = await
_service.GetByLoginAsync(login);
      if (user == null)
        return Unauthorized("Invalid login or
password");
```

```
        if (!BCrypt.Net.BCrypt.Verify(password, user.userPassword!))
            return Unauthorized("Invalid login or password");

        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.Name, user.userLogin ?? ""),
            new Claim("UserId", (user.Id ?? 0).ToString()),
            new Claim("UserName", user.userName ?? ""),
            new Claim(ClaimTypes.Email, user.userEmail ?? ""),
            new Claim("profileLvl", (user.profileLvl ?? 0).ToString()),
            new Claim("currentExp", (user.currentExp ?? 0).ToString())
        };
        var principal = new ClaimsPrincipal(new ClaimsIdentity(claims, "MyCookieAuth"));
        await HttpContext.SignInAsync("MyCookieAuth", principal);

        return Ok(user);
      }

      [HttpPost("experience")]
      public async Task<IActionResult> AddExperienceAsync([FromBody] int exp)
      {
        var userIdClaim = User.FindFirst("UserId")
                  ?? throw new UnauthorizedAccessException("UserId claim missing");
        var userId = int.Parse(userIdClaim.Value);

        // Вызываем ваш сервис
        await _service.UpdateUserExp(userId, exp);

        return Ok();
      }

      [HttpPost("logout")]
      public async Task<IActionResult> Logout()
      {
        await HttpContext.SignOutAsync("MyCookieAuth");
        return Redirect("/");
      }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DomainLogic.Abstracts;
using DomainModels.Models;

namespace DomainLogic.Services
{
  public class AchievementEngine : IAchievementEngine
  {
    private readonly IAchivmentService _achvSvc;
    private readonly IUserService _userService;

    public AchievementEngine(
      IAchivmentService achvSvc,
      IUserService userService)
    {
      _achvSvc = achvSvc;
      _userService = userService;
    }

    public async Task<IEnumerable<AchievementDM>> HandleMetricUpdated(
        int userId,
        string metricKey,
        int newValue)
    {
      var defs = await _achvSvc.GetByMetricAsync(metricKey);

        var awarded = await _achvSvc.GetAchivmentsByObjId(userId, "User");
        var awardedCodes = awarded.Select(a => a.code).ToHashSet();

        var toAward = defs
          .Where(d => d.threshold <= newValue
                &&
!awardedCodes.Contains(d.code))
          .ToList();

        var result = new List<AchievementDM>();

        foreach (var def in toAward)
        {
          await _achvSvc.AddAchivmentToObject(userId, "User", (int)def.Id);

          await _userService.UpdateUserExp(userId, (int)def.expPoints);

          result.Add(def);
        }
        return result;
    }
  }
}
```

```
using DB.DB_SQLight;
using Entity.Entities;
using Microsoft.EntityFrameworkCore;
using Repository.Abstracts;

namespace Repository.Repositories
```

```
{
   public class UserRepository : IUserRepository
   {
      private readonly AppDBContext _context;

      public UserRepository(AppDBContext context)
      {
         _context = context;
      }

      public async Task AddAchivmentUSer(int
userId, object code)
      {
         var achivment = await
_context.achievements.FirstOrDefaultAsync(x =>
x.code == code);
         var achivmentId = achivment.Id;
         _context.userAchivments.Add(new
UserAchivmentEntity
         {
            userId = userId,
            achievementId = achivmentId,
            isVisible = true,
            createDate = DateTime.Now
         });
         await _context.SaveChangesAsync();
      }

      public async Task
AddMainObjAsync(UserEntity entity)
      {
         entity.currentExp = 0;
         entity.profileLvl = 1;
         entity.isVisible = true;
         entity.createDate = DateTime.Now;
         _context.Add(entity);
         await _context.SaveChangesAsync();
      }

      public async Task DeleteMainObj(UserEntity
entity)
      {
         entity.isVisible = false;
         await UpdateMainObj(entity);
      }

      public async Task<IEnumerable<UserEntity>>
GetAllMainObjAsync()
      {
         var list = await _context.users.ToListAsync();
         return list;
      }

      public async Task<UserEntity?>
GetMainObjByIdAsync(int id)
      {
         var user = await
_context.users.FirstOrDefaultAsync(x => x.Id == id);
         return user;
      }

      public async Task UpdateMainObj(UserEntity
entity)
      {
         _context.Update(entity);
         await _context.SaveChangesAsync();
      }
   }
}
```