МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБСЕРВІСУ ДЛЯ УПРАВЛІННЯ ПРОЄКТАМИ ФРІЛАНСЕРІВ ІЗ ВБУДОВАНОЮ СИСТЕМОЮ KANBAN ТА WIKI»**

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
О.О. Просвєтов
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент директор ТОВ «Голден-Тім» Калін М.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Просвєтову Олександру Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебсервісу для управління проєктами фрілансерів із вбудованою системою Kanban та Wiki

керівник роботи Лимаренко Юлія Олексіївна, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Проєктування архітектури вебсервісу.
3. Програмна реалізація вебсервісу.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	12.01.2025	
2.	Збір вихідних даних.	26.02.2026	
3.	Обробка методичних та теоретичних	06.03.2026	
	джерел.		
4.	Розробка першого та другого розділу.	25.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2026	

Студент _____ О.О. Просвєтов
(підпис) (ініціали та прізвище)

Керівник роботи _____ Ю.О. Лимаренко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебсервісу для управління проєктами фрілансерів із вбудованою системою Kanban та Wiki»: 52 с., 18 рис., 11 джерел.

УПРАВЛІННЯ ПРОЄКТАМИ, ФРИЛАНС, API, ASP.NET CORE, CI/CD, DEVOPS, DOCKER, GITOPS, KANBAN, KUBERNETES, POSTGRESQL, REACT, SPA, WIKI.

Об’єкт дослідження – процеси програмної інженерії під час розроблення та супроводу веборієнтованих систем управління проєктами.

Мета роботи – проєктування та розробка вебсервісу для управління фриланс-проєктами із інтегрованими підсистемами Kanban та Wiki, спрямованого на підвищення ефективності організації робочих процесів, координації командної взаємодії та централізованого ведення проєктної документації.

Методи дослідження – системний аналіз предметної області, методи порівняльного аналізу програмних рішень, методи об’єктно-орієнтованого проєктування, моделювання структури даних, клієнт-серверного проєктування вебзастосунків, а також методи контейнеризації та автоматизації процесів розгортання програмного забезпечення.

У межах кваліфікаційної роботи було розроблено вебсервіс для управління проєктами фрилансерів із інтегрованими підсистемами Kanban та Wiki. Для реалізації серверної частини застосовано платформу ASP.NET Core, клієнтську частину реалізовано за допомогою React і TypeScript, а для збереження даних використано реляційну систему управління базами даних PostgreSQL. Практична цінність роботи полягає у створенні єдиного інформаційного середовища для управління завданнями, взаємодії учасників проєкту та централізованого ведення документації.

SUMMARY

Bachelor's qualifying paper "Development of a Web Service for Freelancer Project Management with an Integrated Kanban and Wiki System": 52 pages, 18 figures, 11 references.

API, ASP.NET CORE, CI/CD, DEVOPS, DOCKER, GITOPS, KANBAN, KUBERNETES, POSTGRESQL, REACT, SPA, WIKI, FREELANCE, PROJECT MANAGMENT.

The object of the study is the processes of designing, developing, and maintaining web-based project management systems in the field of software engineering.

The subject of the study is methods and software tools for implementing a web service for freelance project management using SPA, REST API, Kanban, and Wiki subsystems.

The aim of the study is to reduce the time required for organizing and coordinating freelance team activities through the development of a web-based project management service with integrated Kanban and Wiki subsystems.

The methods of research include system analysis of the subject domain, object-oriented software design methods, database modeling, client-server interaction methods, as well as containerization and deployment automation methods for web applications.

Within the scope of the qualification work, a web service for freelancer project management with integrated Kanban and Wiki subsystems was developed. The server side of the application was implemented using ASP.NET Core, the client side was developed with React and TypeScript, and PostgreSQL was used as the relational database management system. The practical value of the work lies in the creation of a unified information environment for task management, team collaboration, and centralized project documentation.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз предметної області та постановка задачі 9
1.1 Аналіз специфіки управління проєктами у сфері фрилансу 9
1.2 Огляд та порівняльний аналіз наявних програмних рішень 11
1.3 Формування вимог до розроблюваного вебсервісу..... 12
2 Проєктування архітектури вебсервісу 16
2.1 Обґрунтування вибору технологій..... 16
2.1.1 Серверна частина 16
2.1.2 Клієнтська частина 18
2.1.3 Вибір інфраструктурного стека технологій 19
2.2 Архітектура серверної частини 20
2.3 Архітектура клієнтської частини..... 23
2.3.1 Структура React SPA 24
2.3.2 Kanban-дошка з Drag-and-Drop..... 27
2.4 Проєктування бази даних..... 28
3 Програмна реалізація вебсервісу..... 31
3.1 Ініціалізація середовища та підсистема автентифікації..... 31
3.2 Рольова модель управління проєктами та формування команд..... 35
3.3 Організація розгортання та DevOps-практики..... 43
Висновки 50
Перелік посилань..... 52

ВСТУП

Сучасна економіка стрімко змінюється: дедалі більше фахівців обирають фриланс замість традиційної найманої праці. Гіг-економіка давно вийшла за межі нішевого явища і охоплює спеціалістів із найрізноманітніших сфер – від IT та дизайну до маркетингу й консалтингу. Однак свобода самозайнятості має свою ціну. Фрилансер змушений одночасно виступати виконавцем, менеджером і бухгалтером: вести кілька проєктів, спілкуватися з різними замовниками, контролювати дедлайни та відстежувати фінанси. Впоратися з усім цим без належних інструментів украй складно – саме тому спеціалізоване програмне забезпечення стає для незалежних спеціалістів не розкішшю, а реальною необхідністю.

Існуючі системи управління проєктами умовно поділяються на два табори. З одного боку, існують потужні корпоративні рішення (наприклад, Jira), які є надлишково складними, важкими у налаштуванні та фінансово необґрунтованими для індивідуальних спеціалістів або невеликих тимчасових команд. З іншого боку, прості інструменти для управління завданнями (наприклад, Trello або Todoist) часто не мають вбудованих засобів для структурованого ведення проєктної документації, управління фінансами (бюджетами проєктів) та гнучкого розподілу ролей між замовником і виконавцем.

Через відсутність єдиної екосистеми фрилансери змушені використовувати цілий набір різних інструментів (окремо месенджери, таск-трекери та хмарні документи), що призводить до проблеми постійного перемикання контексту, втрати важливої інформації та зниження загальної продуктивності. Інтеграція візуальної системи управління завданнями (Kanban) та підсистеми управління знаннями (Wiki) в межах єдиного вебсервісу дозволяє розв’язати проблему фрагментації даних. Завдяки цьому створення спеціалізованої платформи, яка б поєднувала інтуїтивно зрозумілий

інтерфейс із достатньою функціональністю для потреб фриланс-діяльності, є актуальним науково-практичним завданням.

Об’єкт дослідження – процеси програмної інженерії під час розроблення та супроводу веборієнтованих систем управління проєктами.

Предметом дослідження є методи та програмні засоби реалізації вебсервісу для управління фриланс-проєктами із використанням технологій SPA, REST API, Kanban та Wiki-підсистем.

Мета роботи – розробити вебсервіс для незалежних спеціалістів, який поєднує Kanban-дошку для управління завданнями та Wiki-систему для ведення проєктної документації в єдиній зручній платформі.

Для досягнення поставленої мети було сформульовано та вирішено такі основні завдання:

– провести системний аналіз предметної області, дослідити існуючі програмні рішення на ринку та виявити їхні недоліки в контексті потреб фрилансерів;
– сформувати функціональні та нефункціональні вимоги до програмного комплексу;
– спроєктувати архітектуру системи на рівні клієнтської та серверної частин, а також розробити нормалізовану реляційну схему бази даних для ефективного зберігання інформації про користувачів, проєкти, завдання та документацію;
– реалізувати серверну частину вебсервісу (API) з використанням сучасних архітектурних патернів (Repository, Service Layer, Dependency Injection) та механізмів автентифікації;
– розробити інтерактивний клієнтський застосунок (SPA), що забезпечує підтримку drag-and-drop взаємодії для Kanban-дошки та зручний інтерфейс для роботи з Wiki-сторінками;
– спроєктувати інфраструктурний стек для автоматизованого розгортання та контейнеризації застосунку в хмарному середовищі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

Ефективність виконання завдань в умовах віддаленої роботи безпосередньо залежить від обраних інструментів тайм-менеджменту та способів збереження документації. У розділі виявляються ключові проблеми та критично аналізуються наявні рішення. Результатом розділу є формування чіткого переліку функціональних та нефункціональних вимог, які слугуватимуть фундаментальним базисом для проєктування архітектури та подальшої програмної реалізації вебсервісу.

1.1 Аналіз специфіки управління проєктами у сфері фрилансу

У сучасних умовах розвитку інформаційних технологій та поширення дистанційної роботи фриланс став однією з найбільш популярних форм професійної діяльності у сфері ІТ. Значна кількість спеціалістів у галузях розробки програмного забезпечення, вебдизайну, цифрового маркетингу та суміжних напрямів працює незалежно, співпрацюючи з кількома замовниками одночасно. Такий формат зайнятості забезпечує гнучкість, однак водночас створює додаткові труднощі, пов’язані з організацією робочого процесу.

На відміну від корпоративного середовища, де управління проєктами підтримується окремими спеціалістами та спеціалізованими інформаційними системами, фрилансери змушені самостійно координувати виконання завдань, вести комунікацію із замовниками, контролювати строки виконання та підтримувати проєктну документацію. Відсутність єдиного централізованого середовища призводить до фрагментації робочих процесів і ускладнює управління проєктами.

Однією з характерних особливостей діяльності незалежних спеціалістів є паралельне ведення декількох проєктів, кожен із яких має власні вимоги,

часові обмеження та специфіку взаємодії із замовником. Постійне перемикання між різними завданнями підвищує ризик втрати контексту, пропуску важливих дедлайнів та дублювання роботи.

Крім того, взаємодія із клієнтами та обмін інформацією часто здійснюються через різні канали – месенджери, електронну пошту, файлові сховища та сторонні сервіси управління завданнями. У результаті інформація розподіляється між кількома платформами, що ускладнює її пошук, знижує прозорість робочого процесу та збільшує часові витрати на організаційні дії.

Існуючі програмні рішення для управління проєктами переважно орієнтовані на корпоративний сегмент і часто містять надлишковий функціонал, який є складним або недоцільним для використання невеликими фриланс-командами. Це зумовлює потребу у створенні спеціалізованого вебсервісу, здатного поєднати інструменти управління завданнями та ведення документації в межах єдиного інформаційного простору.

Одним із найбільш придатних підходів до організації роботи незалежних спеціалістів є використання методології Kanban [1], яка базується на принципах візуального управління завданнями. Застосування Kanban-дошки дозволяє наочно відображати життєвий цикл задач, контролювати поточний стан виконання робіт, виявляти перевантаження та більш ефективно розподіляти робочий час.

Водночас лише трекінг завдань не забезпечує централізованого збереження проєктного контексту, технічних вимог і супровідної документації. Саме тому доцільним є поєднання Kanban-підходу з підсистемою Wiki, яка дозволяє накопичувати та структурувати інформацію в межах конкретного проєкту. Інтеграція цих інструментів в єдиному середовищі дає змогу зменшити витрати часу на пошук інформації, скоротити кількість перемикань між різними сервісами та підвищити зручність організації роботи фриланс-команд.

1.2 Огляд та порівняльний аналіз наявних програмних рішень

На сучасному ринку інформаційних технологій представлено широкий спектр систем управління проєктами (Project Management Systems) та платформ для спільної роботи. Проте абсолютна більшість із них проєктувалася з фокусом на корпоративний сектор, середні або великі команди розробників. Для обґрунтування доцільності створення спеціалізованого вебсервісу для індивідуальних спеціалістів доцільно провести аналіз найбільш популярних наявних рішень: Jira, Trello, Asana, ClickUp, Monday.com та Notion.

Jira є де-факто галузевим стандартом для команд розробки програмного забезпечення, що працюють за методологіями Agile (Scrum, Kanban). Система пропонує надзвичайно гнучкі налаштування робочих процесів (workflows), детальні системи звітності та управління правами доступу. Водночас для індивідуального спеціаліста цей інструмент характеризується надзвичайно високим порогом входження та перевантаженим користувацьким інтерфейсом, що створює значну функціональну надлишковість. Крім того, Jira не має вбудованої повноцінної системи управління знаннями (Wiki). Реалізація такого функціоналу вимагає інтеграції з іншим продуктом компанії (Confluence), що призводить до збільшення як фінансових витрат, так і когнітивного навантаження через необхідність постійного перемикання між вкладками.

Trello – один із найвідоміших інструментів, що популяризував методологію Kanban для широкого кола користувачів завдяки своєму інтуїтивно зрозумілому інтерфейсу та простоті використання. Однак суттєвим обмеженням системи в контексті комплексного управління проєктами є відсутність вбудованої Wiki. Можливості збереження документації тут зводяться лише до додавання описів у картки або прикріплення файлів. Відповідно, для ведення повноцінної бази знань проєкту користувач змушений застосовувати сторонні сервіси (наприклад, Google Docs), що неминуче призводить до децентралізації та фрагментації робочих даних.

ClickUp, Monday.com та Asana позиціонують себе як комплексні системи управління роботою (All-in-one workspaces). Вони підтримують безліч форматів відображення даних (списки, Kanban, діаграми Ганта, календарі) та дозволяють керувати портфелями проєктів. Проте головною проблемою цих платформ для фрилансера є явище «функціонального розбухання» (feature bloat). Наявність інструментів для управління завантаженістю команди (workload management), складних систем узгодження та масових розсилок виявляється абсолютно непотрібною для незалежного спеціаліста. Це лише суттєво ускладнює навігацію та уповільнює роботу із системою. Додатковою перешкодою стає цінова політика: повноцінний доступ до функціоналу вимагає придбання дорогих підписок, орієнтованих переважно на B2B-сегмент.

Notion є потужним інструментом для створення баз знань, ведення особистих записів та організації документів, який пропонує неперевершений досвід роботи з текстом (подібний до Wiki) та дозволяє створювати гнучкі реляційні бази даних. Хоча система і підтримує відображення записів у вигляді Kanban-дошок, її не можна вважати спеціалізованим таск-трекером. Notion бракує специфічної для Kanban ергономіки, зокрема швидкого перетягування об’єктів без затримок та наявності метрик швидкості виконання завдань. Крім того, при значному збільшенні обсягу проєктних даних інтерфейс має тенденцію ставати громіздким, що може призводити до зниження швидкодії клієнтського застосунку.

1.3 Формування вимог до розроблюваного вебсервісу

На основі аналізу особливостей організації роботи фрилансерів, а також дослідження функціональних можливостей наявних програмних рішень було сформовано перелік вимог до розроблення вебсервісу управління проєктами. Сформовані вимоги визначають основний функціонал системи, її архітектурні особливості, вимоги до безпеки, продуктивності та зручності використання.

Розглянемо можливості, які належать до **функціональних вимог системи**.

Управління користувачами та профілями. Система повинна забезпечувати реєстрацію та автентифікацію користувачів із можливістю розмежування ролей між замовниками та виконавцями. Для кожного користувача має бути доступний персональний профіль, що дозволяє редагувати основну інформацію, контактні дані, професійні навички, портфоліо та інші параметри, необхідні для організації співпраці.

Управління проєктами. Користувачі із відповідними правами доступу повинні мати можливість створювати, редагувати та видаляти проєкти, визначати їх опис, бюджет та часові межі виконання. Також система повинна підтримувати механізм формування команди шляхом залучення інших користувачів до спільної роботи над проєктом.

Підсистема управління завданнями (Kanban). В межах кожного проєкту має бути реалізована інтерактивна дошка завдань, що дозволяє створювати окремі етапи виконання роботи, формувати картки завдань та змінювати їх поточний стан. Переміщення завдань між етапами повинно здійснюватися у візуальному режимі, забезпечуючи наочне відображення прогресу виконання проєкту.

Підсистема керування документацією (Wiki). Система повинна підтримувати створення та редагування внутрішньої документації проєкту, включаючи технічні специфікації, нотатки та інші матеріали. Інформаційні сторінки мають бути прив’язані до конкретного проєкту та підтримувати базові засоби форматування тексту для покращення структурованості інформації.

Для наочного представлення функціональних можливостей вебсервісу та сценаріїв взаємодії користувачів із системою побудовано діаграму прецедентів (див. рис. 1.1), яка відображає основні ролі користувачів та доступний їм функціонал.

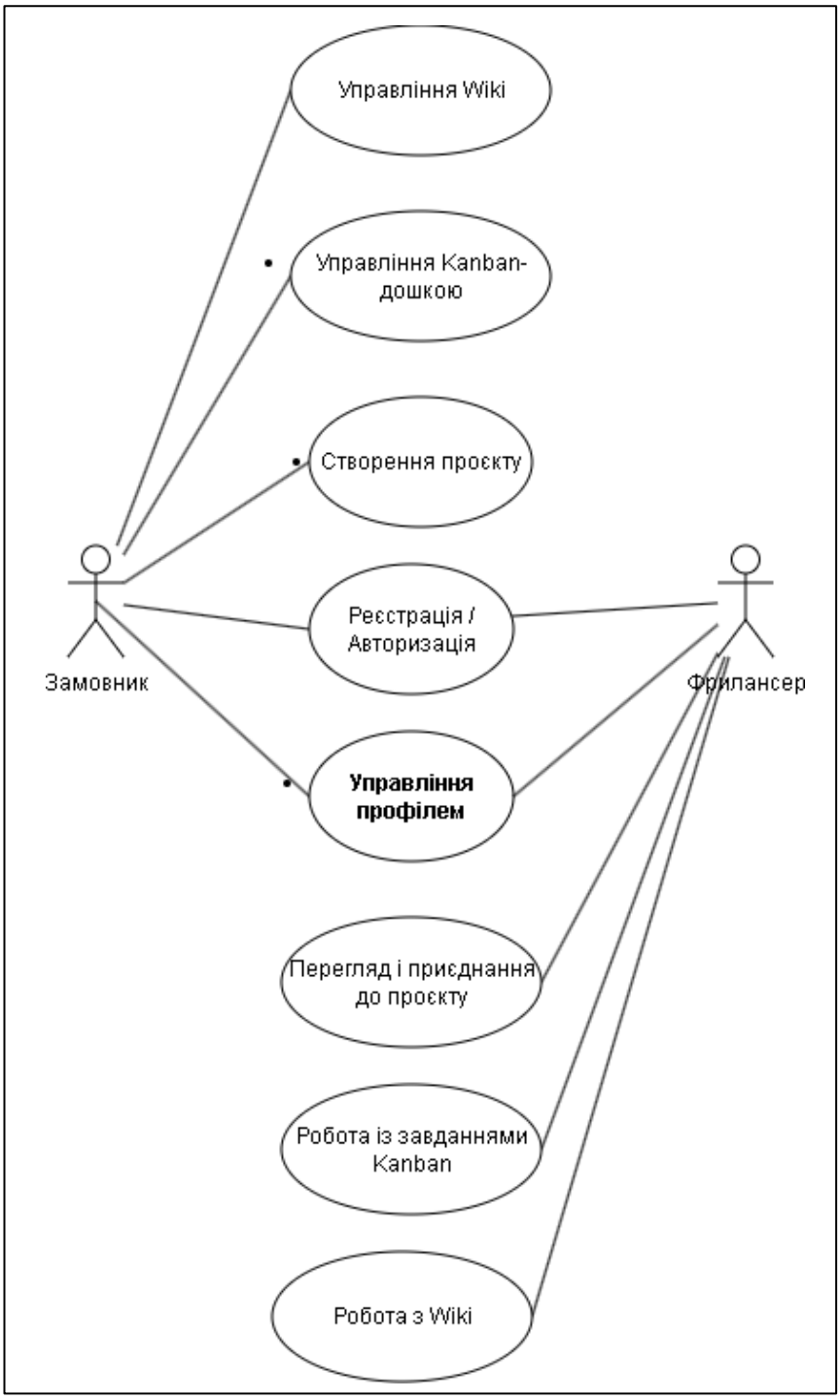


Рисунок 1.1 – Діаграма прецедентів вебсервісу управління проєктами фрилансерів

Окрім функціональних можливостей, сформовано перелік **нефункціональних вимог**, що визначають технічні характеристики та якісні параметри системи.

Архітектурні вимоги. Система повинна реалізовуватися відповідно до клієнт-серверної моделі взаємодії. Клієнтська частина має функціонувати у

вигляді односторінкового вебзастосунку (SPA), що забезпечує динамічне оновлення інтерфейсу без повторного завантаження сторінок. Серверна частина повинна надавати REST API для обміну даними у форматі JSON.

Технологічні вимоги. Для реалізації серверної частини повинна використовуватися платформа ASP.NET Core, клієнтської – React із використанням TypeScript, а для збереження даних – реляційна система управління базами даних PostgreSQL.

Вимоги до безпеки. Механізм автентифікації має базуватися на використанні JSON Web Token (JWT). Конфіденційні дані користувачів, зокрема паролі, повинні зберігатися у захищеному вигляді із застосуванням криптографічного хешування. Доступ до внутрішніх ресурсів системи має бути обмежений відповідно до ролей користувачів та визначених політик авторизації.

Вимоги до користувацького інтерфейсу. Інтерфейс вебсервісу повинен бути зрозумілим, адаптивним та забезпечувати зручну взаємодію із функціональними модулями системи. Особлива увага має приділятися простоті навігації та швидкому доступу до основних функцій управління проєктами.

Вимоги до розгортання та супроводу. Система повинна підтримувати контейнеризоване середовище виконання та можливість автоматизованого розгортання. Для цього передбачено використання Docker-контейнерів, оркестрації Kubernetes та механізмів автоматизованої інтеграції й доставки програмного забезпечення.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ВЕБСЕРВІСУ

На основі функціональних та нефункціональних вимог, визначених на етапі системного аналізу, здійснюється обґрунтування вибору технологічного стека, шаблонів проєктування та інструментальних засобів розробки. Детально описується логічна структура реляційної бази даних, процеси її нормалізації та організація зв’язків між ключовими сутностями предметної області. Окрема увага приділяється побудові багатошарової архітектури серверної частини (API) із застосуванням сучасних патернів, а також проєктуванню інтерактивного клієнтського застосунку (SPA) з акцентом на управління станом і реалізацію складних елементів користувацького інтерфейсу, зокрема Kanban-дошки та підсистеми Wiki.

2.1 Обґрунтування вибору технологій

Проєктування та реалізація вебсервісу для управління фриланс-проєктами з інтегрованими інструментами Kanban і Wiki вимагає використання технологічного стеку, здатного забезпечити гнучкість архітектури, підтримуваність програмного коду, продуктивність системи та можливість її подальшого масштабування.

2.1.1 Серверна частина

Для розробки серверної частини програмної системи обрано кросплатформне середовище ASP.NET Core [2] (на базі платформи .NET) – технологічне рішення з відкритим вихідним кодом від корпорації Microsoft.

Доцільність застосування цієї технології обґрунтовується низкою вагомих чинників:

- високими показниками продуктивності серед сучасних серверних технологій;
- інтегрованою підтримкою патерну впровадження залежностей (Dependency Injection), що забезпечує низький рівень зв’язності архітектурних компонентів;
- наявністю розвиненої екосистеми супутніх бібліотек та інструментальних засобів.

Взаємодію системи з базою даних реалізовано за допомогою технології об’єктно-реляційного відображення (ORM) Entity Framework Core. Зазначений інструмент підтримує парадигму проєктування «код насамперед» (Code-First), що дозволяє керувати схемою бази даних за допомогою механізму міграцій. Як систему керування базами даних (СКБД) застосовано PostgreSQL[3] – реляційну СУБД з відкритим вихідним кодом, яка відзначається високим рівнем надійності, оптимізованою обробкою складних запитів та повною відповідністю вимогам ACID щодо транзакцій. Безпосередня взаємодія із СКБД забезпечується через спеціалізований провайдер Npgsql.

Автентифікацію користувачів імплементовано з використанням технології маркерів доступу JSON Web Tokens (JWT) на базі відповідних модулів середовища ASP.NET. Для забезпечення цілісності та верифікації токени підписуються за допомогою криптографічного алгоритму HMAC-SHA256. Встановлений обмежений час дії маркера (дві години) дозволяє досягти оптимального балансу між вимогами інформаційної безпеки та ергономікою взаємодії користувача із системою.

Захист конфіденційних даних реалізовано за допомогою бібліотеки BCrypt.Net, що імплементує адаптивну криптографічну хеш-функцію bcrypt. Застосування механізму автоматичного додавання криптографічної «солі» суттєво підвищує стійкість системи до атак повного перебору (brute-force).

Трансформацію інформації між сутностями предметної області та об’єктами передачі даних (DTO) автоматизовано завдяки бібліотеці AutoMapper. Водночас перевірку коректності вхідної інформації покладено на

інструмент FluentValidation, використання якого уможливлює декларативний опис правил валідації для кожного окремого типу даних.

Для моніторингу функціонування системи налаштовано механізм структурного логування за допомогою компонента Serilog. Автоматизована генерація документації прикладного програмного інтерфейсу (API) здійснюється засобами бібліотеки Swashbuckle (Swagger).

2.1.2 Клієнтська частина

Клієнтську частину програмного комплексу спроєктовано за архітектурою односторінкового застосунку (SPA) із застосуванням бібліотеки React [4] та мови програмування TypeScript. Доцільність такого вибору обґрунтовується такими аспектами:

- компонентно-орієнтований підхід React сприяє ефективному повторному використанню елементів інтерфейсу користувача;
- наявність статичної типізації у TypeScript суттєво мінімізує ймовірність виникнення помилок на етапі компіляції коду;
- наявність розгалуженої екосистеми з широким спектром супутніх бібліотек та інструментальних засобів.

Процес збирання проєкту оптимізовано за допомогою інструменту Vite. Використання нативних ES-модулів на етапі розробки та інтеграція з Rollup для формування фінальних релізів значно прискорює загальний цикл створення програмного продукту.

Формування візуального стилю інтерфейсу реалізовано із застосуванням утилітарного CSS-фреймворку Tailwind CSS у комплексі з бібліотекою компонентів shadcn/ui, яка базується на примітивах Radix UI. Такий підхід гарантує високий рівень доступності (web accessibility) та гнучкість візуального налаштування без надмірного навантаження, притаманного монолітним UI-бібліотекам.

Керування серверним станом інтегровано засобами бібліотеки TanStack React Query, що забезпечує ефективні механізми кешування, фонового оновлення та синхронізації даних із сервером. Навігацію між представленнями (маршрутизацію) імплементовано з використанням оптимізованого роутера Wouter.

Обробку форм побудовано на базі рішення react-hook-form, а перевірку їх коректності – за допомогою інструменту Zod. Це створює надійний механізм типобезпечної валідації даних, цілком узгоджений зі схемами та системою типів TypeScript.

Інтерактивні можливості перетягування об’єктів (drag-and-drop), зокрема для Kanban-дошки, функціонують на основі бібліотеки dnd-kit. Це високопродуктивне та модульне рішення повноцінно підтримує взаємодію на сенсорних пристроях та відповідає сучасним стандартам доступності.

2.1.3 Вибір інфраструктурного стека технологій

Стек інфраструктурних технологій проєктувався з акцентом на комплексну автоматизацію, забезпечення безпеки та високу масштабованість розгортання:

- Docker [5] – для ізоляції компонентів системи шляхом контейнеризації з використанням багатоетапних (multi-stage) алгоритмів збирання;
- Kubernetes [6] – як основний засіб оркестрування контейнеризованих застосунків в межах кластерної інфраструктури;
- Helm [7] – у ролі менеджера пакетів для зручного управління конфігураціями Kubernetes;
- Argo CD [8] – як контролер методології GitOps, що забезпечує декларативний підхід до безперервного розгортання застосунків;
- GitHub Actions [9] – для побудови автоматизованих конвеєрів безперервної інтеграції та безперервного розгортання (CI/CD);

– Bitnami SealedSecrets – для захисту та безпечного зберігання конфіденційної інформації (секретів) в системах контролю версій Git;
– Traefik[10] – Ingress-контролер, що забезпечує ефективну маршрутизацію та балансування вхідного HTTP-трафіку.

Запропонований стек технологій формує надійний фундамент для побудови відмовостійкої хмарної архітектури. Детальний опис практичної реалізації цих інструментів, створення конвеєрів безперервної інтеграції (CI/CD) та процесів декларативного розгортання наведено у Розділі 3 даної роботи.

2.2 Архітектура серверної частини

Серверна частина вебсервісу реалізована відповідно до сучасних принципів побудови масштабованих та підтримуваних програмних систем. Для забезпечення низької зв’язності між компонентами, високого рівня тестованості та спрощення подальшого розширення функціоналу застосовано архітектурний підхід Clean Architecture.

Використання цього підходу дозволяє відокремити бізнес-логіку від деталей реалізації інфраструктури, таких як база даних, зовнішні сервіси або механізми взаємодії з користувачем. У результаті система стає менш залежною від конкретних технологій та більш придатною до супроводу й модифікації.

Проєкт складається з чотирьох шарів, реалізованих як окремі .NET-проєкти (див. рис. 2.1).

Core – центральний шар системи, що містить основні доменні сутності предметної області та базові перерахування. У цьому шарі визначено такі сутності, як User, Project, KanbanColumn, KanbanTask, WikiPage, FreelancerProfile, ClientProfile та ProjectMember, які відображають структуру

предметної області та взаємозв’язки між її компонентами. Крім того, шар містить перелічувані типи (UserRole, ProjectStatus, ProjectMemberRole), що використовуються для визначення ролей користувачів, статусів проєктів і механізмів доступу. Важливою особливістю шару є відсутність залежностей від зовнішніх бібліотек і технологій, що забезпечує незалежність бізнес-моделі від деталей реалізації інфраструктури.

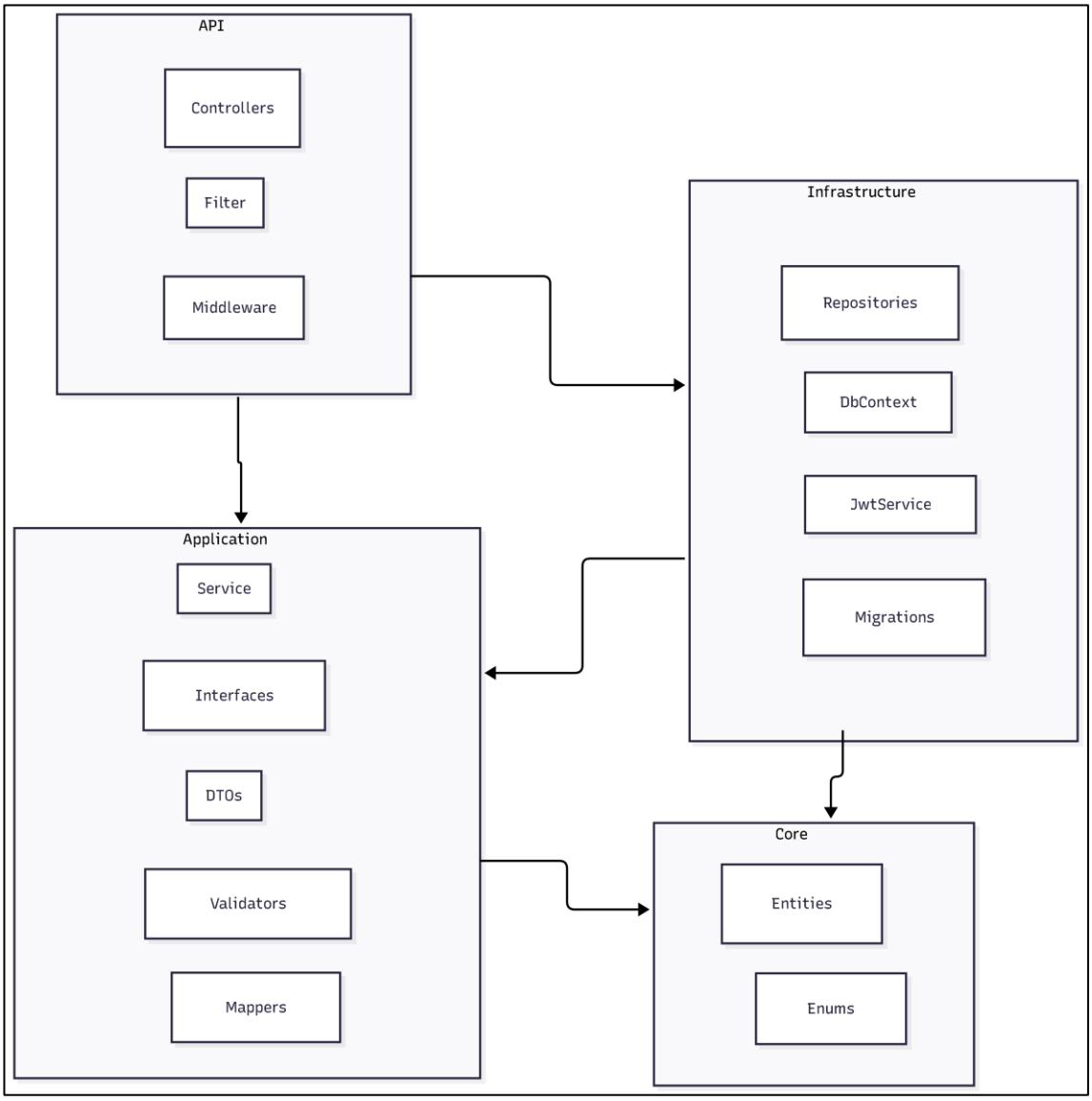


Рисунок 2.1 – Архітектура серверної частини
(відповідно до принципів Clean Architecture)

Application – шар прикладної логіки, що реалізує основні сценарії роботи системи та бізнес-правила. У його межах визначено інтерфейси сервісів і

репозиторіїв, моделі передачі даних (DTO), валідатори, а також профілі мапінгу об’єктів. Цей шар координує взаємодію між бізнес-сутностями та зовнішніми компонентами, залишаючись незалежним від конкретної реалізації доступу до даних. Залежності шару обмежуються лише компонентами рівня Core, що відповідає принципам Clean Architecture.

Infrastructure – інфраструктурний шар, призначений для реалізації механізмів взаємодії із зовнішніми ресурсами. Саме тут розміщено реалізації репозиторіїв на базі Entity Framework Core, контекст бази даних AppDbContext, механізм генерації JWT-токенів, а також інструменти для виконання міграцій бази даних. Основною функцією цього шару є ізоляція технічних деталей реалізації від бізнес-логіки застосунку.

API – рівень представлення, що забезпечує взаємодію клієнтської частини із сервером через REST API. Цей шар містить контролери ASP.NET Core, фільтри обробки запитів і валідації (FluentValidationFilter, TransformResponseFilter), конфігурацію middleware-компонентів та головну точку входу до застосунку – Program.cs. Основним завданням шару є прийом HTTP-запитів, їх обробка та передача відповідних команд до прикладного рівня.

Для забезпечення структурованості програмного коду, спрощення підтримки застосунку та підвищення гнучкості архітектури під час розроблення серверної частини використано низку поширених патернів проєктування. Їх застосування дозволило розмежувати відповідальність між компонентами системи, мінімізувати зв’язність між модулями та забезпечити можливість подальшого масштабування функціоналу.

Для ізоляції логіки доступу до даних від бізнес-логіки застосунку використано патерн Repository. У межах цього підходу взаємодія із базою даних здійснюється через набір спеціалізованих інтерфейсів, зокрема IUserRepository, IProjectRepository, IKanbanColumnRepository та інші. Безпосередня реалізація цих інтерфейсів знаходиться в інфраструктурному шарі та використовує можливості Entity Framework Core. Такий підхід

дозволяє зменшити залежність прикладної логіки від конкретної технології зберігання даних і спрощує внесення змін до системи в майбутньому.

Бізнес-логіку застосунку винесено до окремого сервісного шару, представленого класами AuthService, ProjectService, KanbanColumnService, WikiPageService та іншими. Сервіси відповідають за реалізацію основних сценаріїв роботи системи, координацію взаємодії між репозиторіями та перевірку бізнес-правил. Завдяки такому підходу контролери API залишаються максимально спрощеними та відповідають переважно за маршрутизацію HTTP-запитів і передачу команд до відповідних сервісів.

Для передачі інформації між архітектурними шарами та взаємодії клієнтської і серверної частин використано об’єкти передачі даних (Data Transfer Object, DTO). Їх застосування дозволяє відокремити внутрішню структуру доменних сутностей від зовнішнього представлення даних, а також передавати лише ту інформацію, яка необхідна для виконання конкретної операції. Це сприяє підвищенню рівня безпеки та забезпечує більш чіткий контракт взаємодії між компонентами системи.

Для реалізації принципу слабкої зв’язаності компонентів використано механізм Dependency Injection, який забезпечує централізоване управління залежностями застосунку. Реєстрація сервісів виконується під час запуску системи через спеціалізовані методи конфігурації, зокрема AddApplication() та AddInfrastructure(). Використання цього механізму спрощує супровід програмного коду, полегшує тестування окремих компонентів та підвищує гнучкість архітектури загалом.

2.3 Архітектура клієнтської частини

Клієнтська частина вебсервісу реалізована у вигляді односторінкового застосунку (Single Page Application, SPA) із використанням бібліотеки React та мови TypeScript. Такий підхід дозволяє забезпечити динамічне оновлення

інтерфейсу без повторного завантаження сторінок, підвищує швидкість взаємодії користувача із системою та сприяє створенню більш інтерактивного користувацького досвіду.

Під час проєктування клієнтської частини застосовано модульний підхід із розмежуванням відповідальності між окремими функціональними компонентами. Це дозволило ізолювати логіку роботи із серверними даними, механізми взаємодії із API та компоненти інтерфейсу, що позитивно вплинуло на структурованість програмного коду, спростило підтримку застосунку та створило основу для подальшого масштабування функціоналу.

2.3.1 Структура React SPA

Архітектуру клієнтського застосунку спроєктовано за функціональним та модульним принципами з дотриманням суворого розмежування зон відповідальності між різними архітектурними рівнями. Такий підхід забезпечує високий ступінь масштабованості системи, полегшує процес тестування ізольованих компонентів та суттєво оптимізує подальший супровід кодової бази.

Структурно клієнтську частину поділено на три ключові логічні шари, взаємодія між якими відбувається чітко в одному напрямку (див. рис. 2.2).

Клієнтська частина вебсервісу реалізована із використанням багаторівневого підходу, що забезпечує розділення відповідальності між окремими компонентами системи, підвищує зручність супроводу коду та спрощує масштабування функціоналу. У межах реалізованої архітектури можна виокремити інфраструктурний рівень, рівень управління даними та рівень представлення.

Інфраструктурний рівень (Infrastructure Layer) відповідає за організацію взаємодії між клієнтським застосунком та серверною частиною системи. Основним призначенням цього рівня є централізація логіки виконання HTTP-

запитів і приховування низькорівневих деталей мережевої взаємодії від інших компонентів застосунку.

У межах цього шару реалізовано єдиний HTTP-клієнт, що використовується для обміну даними із серверним API. До його функціональних обов’язків належить автоматичне додавання JWT-токенів до заголовків авторизації, що забезпечує автентифікацію користувача під час доступу до захищених ресурсів. Крім того, інфраструктурний рівень відповідає за обробку серверних відповідей та приведення отриманих даних до уніфікованого формату, придатного для подальшого використання в інших частинах застосунку.

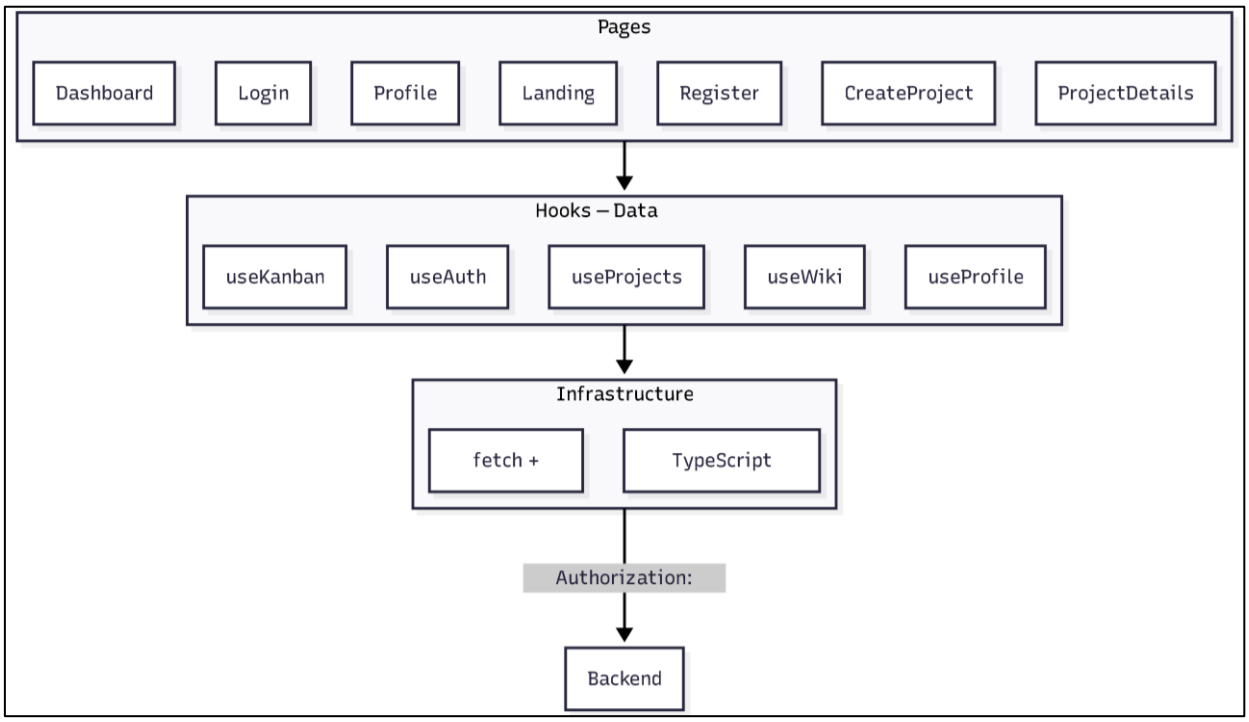


Рисунок 2.2 – Архітектура клієнтської частини вебзастосунку

Важливою особливістю реалізації є використання суворої типізації засобами TypeScript, що дозволяє здійснювати перевірку відповідності структури даних ще на етапі компіляції програми та зменшує ризик виникнення помилок під час виконання.

Рівень управління даними (Hooks / Data Layer) виконує функцію проміжного шару між мережевою взаємодією та користувацьким

інтерфейсом. Основним завданням цього рівня є централізоване управління серверним станом застосунку, включаючи отримання, оновлення та кешування інформації.

Для реалізації зазначеного функціоналу використано механізм користувацьких React-хуків, які інкапсулюють логіку роботи із серверними запитами та дозволяють ізолювати асинхронні процеси від компонентів інтерфейсу. Завдяки цьому забезпечується ефективне кешування даних, автоматичне оновлення інформації у фоновому режимі, а також повторне отримання актуальних даних після внесення змін користувачем.

Такий підхід дозволяє зменшити кількість зайвих мережевих запитів, підвищити швидкодію інтерфейсу та забезпечити більш плавну взаємодію користувача із системою.

Крім того, винесення логіки роботи із серверним станом до окремого шару сприяє покращенню структурованості програмного коду та підвищує рівень повторного використання компонентів.

Рівень представлення (Pages Layer) відповідає за формування користувацького інтерфейсу та організацію взаємодії користувача із функціональними можливостями вебсервісу. Компоненти цього рівня пов’язані із конкретними сторінками або маршрутами застосунку та виконують роль координаторів між даними і візуальними елементами.

Сторінки отримують необхідну інформацію через рівень управління даними та передають її окремим компонентам інтерфейсу, відповідальним за відображення контенту. При цьому бізнес-логіка та логіка роботи з API залишаються винесеними за межі UI-компонентів, що дозволяє зберігати їх простими, незалежними та придатними для повторного використання в різних частинах системи.

Запропонована структура клієнтської частини забезпечує гнучкість подальшого розвитку вебсервісу, спрощує підтримку програмного коду та дозволяє ефективно масштабувати застосунок шляхом додавання нових функціональних модулів без істотного ускладнення архітектури.

2.3.2 Kanban-дошка з Drag-and-Drop

Інтерактивна підсистема управління завданнями у форматі Kanban-дошки імплементована з використанням високорівневої модульної бібліотеки dnd-kit. Цей інструментарій дозволив реалізувати складну ергономіку взаємодії користувача з елементами інтерфейсу та досягти високих показників продуктивності. Зокрема, імплементований механізм забезпечує комплексне просторове переміщення об’єктів. Він підтримує як вертикальне сортування сутностей у межах єдиної структурної одиниці для зміни пріоритетності завдань, так і горизонтальне крос-модульне переміщення між різними етапами життєвого циклу з метою зміни поточного статусу виконання.

Для підтримки якісного користувацького досвіду (UX) під час транзакції перетягування інтегровано оптимізований візуальний зворотний зв’язок. Це реалізовано шляхом застосування компонента DragOverlay, який динамічно генерує ізольовану візуальну репрезентацію – стилізований фантомний дублікат картки. Такий підхід абстрагує об’єкт від загального потоку документа (DOM), що гарантує плавність анімацій та відсутність артефактів відмальовування.

Важливим аспектом системи є евристичне визначення цільової зони. Обчислення оптимальної зони скидання (drop zone) базується на застосуванні математичного алгоритму closestCorners. Цей метод аналізує найближчі кутові координати перетягуваного об’єкта відносно потенційних контейнерів, що гарантує стабільну та передбачувану поведінку інтерфейсу навіть за умови складної геометрії сітки компонування.

З метою підвищення надійності роботи інтерфейсу забезпечено захист від хибних інтеракцій. Для нівелювання проблеми випадкових спрацьовувань, наприклад, під час звичайного кліку або спроби прокручування сторінки, впроваджено спеціалізований апаратний обробник PointerSensor. Його конфігурація налаштована на ініціацію процесу перетягування виключно після подолання курсором порогового значення зміщення у 5 пікселів, що повною мірою відповідає сучасним стандартам проєктування мікроінтеракцій.

На завершальному етапі операції, безпосередньо після спрацьовування події onDragEnd, клієнтський застосунок автоматично ініціює асинхронну синхронізацію оновленого стану з серверною частиною. Цей процес реалізується шляхом формування цілеспрямованого HTTP-запиту за допомогою методу PATCH до відповідного ендпоінту /api/kanbantasks/{id}/move. Тіло запиту інкапсулює суворо обмежений набір модифікованих параметрів: унікальний ідентифікатор нової батьківської колонки (columnId) та актуалізований порядковий індекс елемента (order).

Такий гранулярний підхід до передачі даних суттєво мінімізує обсяг мережевого трафіку та забезпечує сувору атомарність операцій оновлення у базі даних.

2.4 Проєктування бази даних

Логічна структура бази даних спроєктована з дотриманням принципів реляційної алгебри та нормалізована до рівня третьої нормальної форми (3NF). Це забезпечує оптимальний баланс між продуктивністю виконання SQL-запитів та мінімізацією ризиків виникнення аномалій модифікації, вставки чи видалення даних. Схема налічує вісім базових сутностей, об’єднаних розгалуженою системою реляційних зв’язків (див. рис. 2.3).

Логічна структура бази даних спроєктована з дотриманням принципів реляційної алгебри та нормалізована до рівня третьої нормальної форми (3NF). Це забезпечує оптимальний баланс між продуктивністю виконання SQL-запитів та мінімізацією ризиків виникнення аномалій модифікації, вставки чи видалення даних. Схема налічує вісім базових сутностей, об’єднаних розгалуженою системою реляційних зв’язків.

Центральною сутністю системи виступає User, яка інкапсулює базові автентифікаційні та авторизаційні дані. Для реалізації гнучкої рольової моделі застосовано патерн делегування: специфічна інформація користувачів

розділена на окремі сутності FreelancerProfile та ClientProfile, що пов’язані з основною таблицею відношенням «один-до-одного» (1:1). Такий підхід запобігає засміченню таблиці User надлишковими полями (NULL-значеннями) і дозволяє динамічно розширювати атрибути профілів залежно від типу облікового запису.

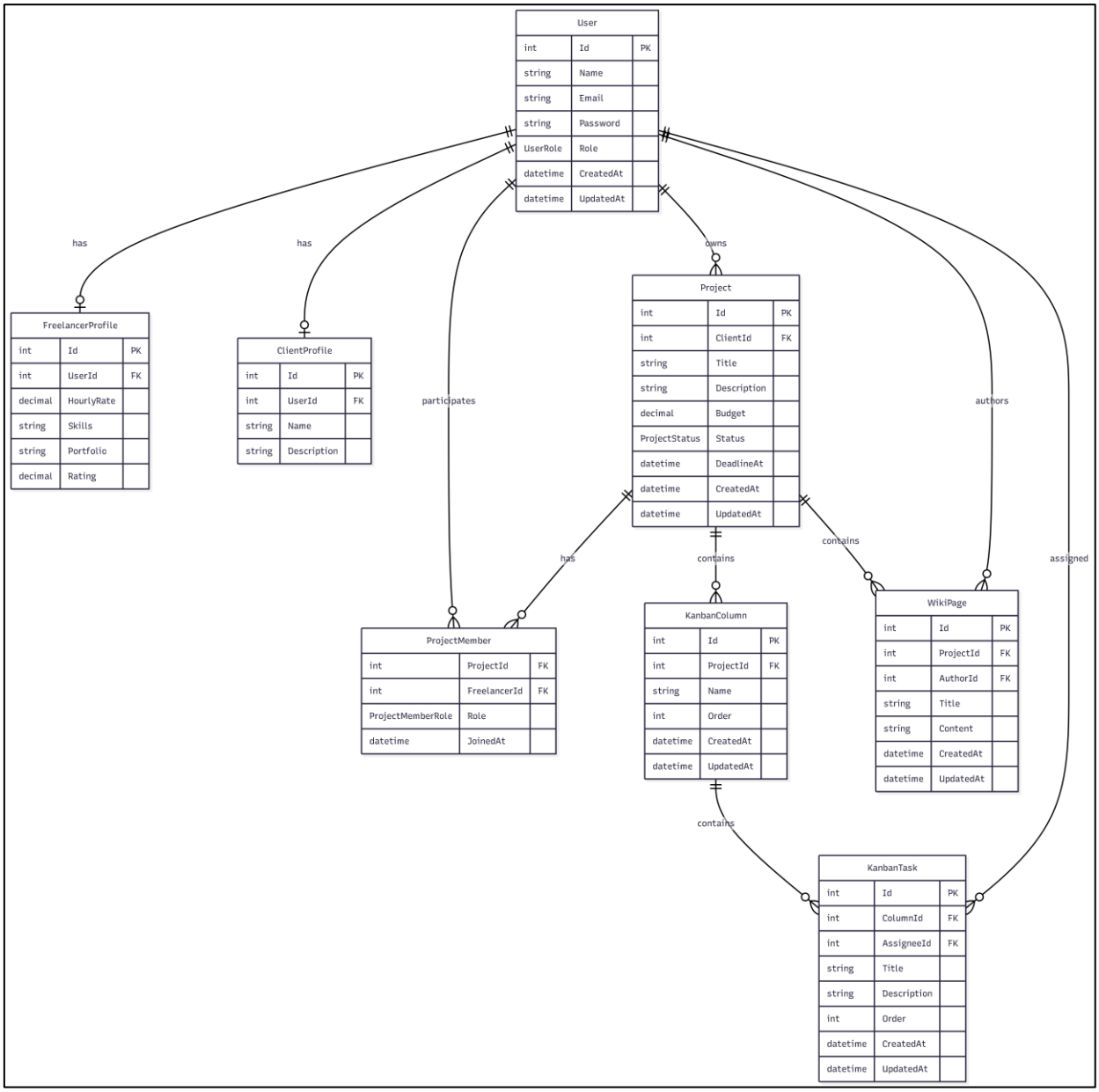


Рисунок 2.3 – ER діаграма схеми бази даних

Ієрархія предметної області базується на сутності Project. Зв’язок між замовником (Client) та проєктом реалізовано як «один-до-багатьох» (1:N). Для забезпечення колаборативної роботи імплементовано сутність ProjectMember,

яка виконує роль асоціативної таблиці для розв’язання зв’язку «багато-до-багатьох» (M:N) між проєктами та фрилансерами. Вона додатково зберігає контекстну інформацію про метадані участі, зокрема внутрішню роль у проєкті та дату приєднання.

Архітектура управління завданнями побудована за каскадним принципом: проєкт містить множину колонок (KanbanColumn), які, своєю чергою, агрегують конкретні завдання (KanbanTask). Це формує класичну структуру «один-до-багатьох» на кожному з рівнів. Кожне завдання може бути делеговане конкретному виконавцю через зв’язок з таблицею User. Додатково проєкт інтегрує підсистему ведення документації, що репрезентується сутністю WikiPage, яка зберігає контент та фіксує авторство сторінок.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСЕРВІСУ

На основі розробленої архітектури та обраного технологічного стека здійснюється безпосереднє написання програмного коду серверної та клієнтської частин. Детально розглядається імплементація ключових модулів бекенду, зокрема налаштування конвеєра обробки запитів, механізмів автентифікації та бізнес-логіки управління завданнями. Окрема увага приділяється створенню односторінкового клієнтського застосунку (SPA), організації взаємодії з API, управлінню станом та розробці інтерактивного користувацького інтерфейсу, що включає візуальну Kanban-дошку та текстовий редактор підсистеми Wiki.

3.1 Ініціалізація середовища та підсистема автентифікації

Практична демонстрація функціональних можливостей розробленого вебсервісу здійснювалася у попередньо підготовленому ізольованому середовищі виконання, яке забезпечує стабільність роботи системи, однакові умови тестування та відтворюваність результатів незалежно від конфігурації програмного або апаратного забезпечення. Архітектура програмного продукту побудована за принципом розподілу відповідальності між окремими компонентами, що дозволяє забезпечити масштабованість, спрощення підтримки та підвищення надійності функціонування системи.

Клієнтська частина вебсервісу реалізована у вигляді односторінкового застосунку (Single Page Application), розробленого із використанням бібліотеки React. Основною функцією клієнтської частини є забезпечення інтерактивної взаємодії користувача з платформою, відображення динамічного інтерфейсу, а також оперативне оновлення інформації без необхідності повного перезавантаження вебсторінки. Завдяки використанню реактивної моделі рендерингу користувач отримує можливість працювати із

сервісом у режимі, максимально наближеному до настільних застосунків, що позитивно впливає на загальну ергономіку системи.

Серверна частина вебсервісу реалізована на платформі ASP.NET API та відповідає за виконання бізнес-логіки, обробку HTTP-запитів, перевірку прав доступу, маршрутизацію, взаємодію із базою даних та централізоване управління інформаційними ресурсами. Для довготривалого збереження інформації використовується система управління базами даних PostgreSQL, яка забезпечує підтримку реляційних зв’язків між сутностями, високу продуктивність обробки запитів та цілісність збережених даних.

Запуск компонентів програмної системи виконується за допомогою засобів контейнеризації, що дозволяє створити стандартизоване середовище виконання незалежно від особливостей локального оточення. Такий підхід мінімізує ризики виникнення конфліктів між програмними залежностями, спрощує процес тестування та забезпечує передбачуваність функціонування системи під час демонстрації її можливостей.

Точкою входу до вебсервісу для неавторизованих користувачів є публічна сторінка, яка представлена на рисунку 3.1.

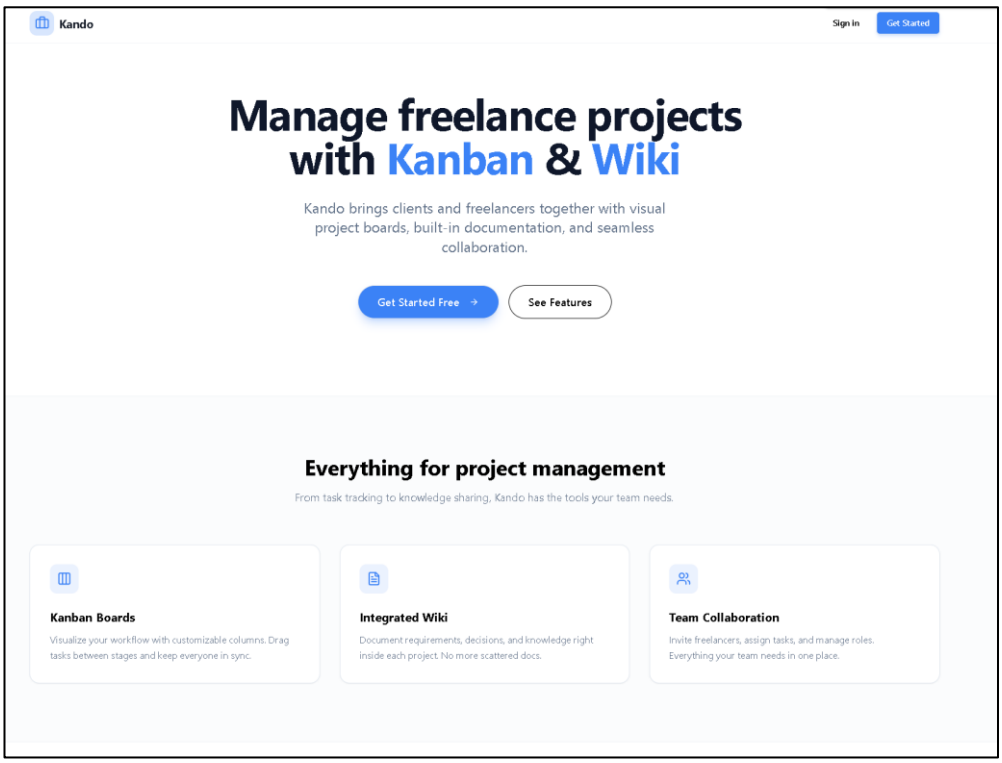


Рисунок 3.1 – Лендінг-сторінка застосунку

Її основним призначенням є ознайомлення потенційних користувачів із можливостями платформи, ключовими перевагами використання системи та принципами організації співпраці між замовниками й виконавцями. На сторінці відображається загальна інформація щодо функціоналу сервісу, зокрема можливостей керування проєктами, використання Kanban-дошки для організації робочого процесу та Wiki-модуля для централізованого ведення документації.

Для отримання доступу до повного функціоналу платформи користувачеві необхідно пройти процедуру реєстрації, яка реалізована через відповідну форму введення даних (див. рис. 3.2).

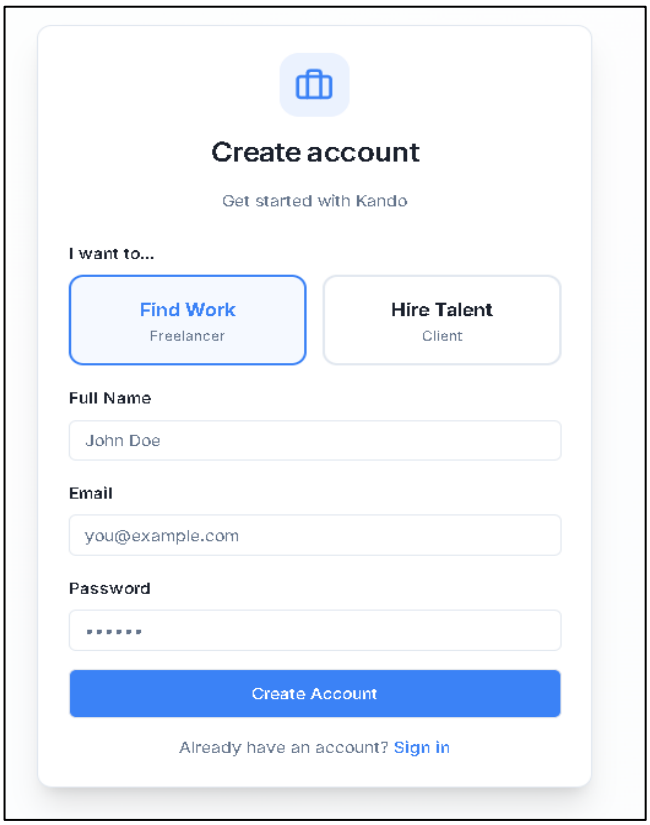


Рисунок 3.2 – Форма реєстрації

Під час створення нового облікового запису користувач повинен вказати базову інформацію, зокрема ім’я, адресу електронної пошти та пароль. Окрім цього, важливим елементом реєстрації є вибір ролі користувача у системі. Реалізовано дві основні ролі: «Замовник» (Client), що передбачає створення

проєктів і керування ними, а також «Виконавець» (Freelancer), основною функцією якого є участь у виконанні робочих завдань.

Для забезпечення коректності введених даних та мінімізації кількості помилок реалізовано механізм дворівневої валідації. На клієнтському рівні здійснюється автоматична перевірка введених значень із негайним відображенням повідомлень про помилки, що покращує користувацький досвід і дозволяє швидко виправити некоректно введену інформацію. Одночасно із цим серверна частина виконує додаткову перевірку отриманих даних, що унеможливлює обхід встановлених обмежень шляхом маніпуляції HTTP-запитами або зміни структури переданих параметрів.

Після успішного завершення процедури реєстрації сервер автоматично генерує JWT-токен, який використовується для подальшої автентифікації користувача та підтвердження його прав доступу. Отриманий токен зберігається у локальному сховищі браузера та автоматично використовується під час взаємодії із захищеними ресурсами системи. Такий підхід дозволяє підтримувати активну користувацьку сесію без необхідності повторної авторизації при кожному переході між сторінками.

Процес входу до системи реалізований через окрему форму авторизації (див. рис. 3.3), де користувач вводить свої облікові дані.

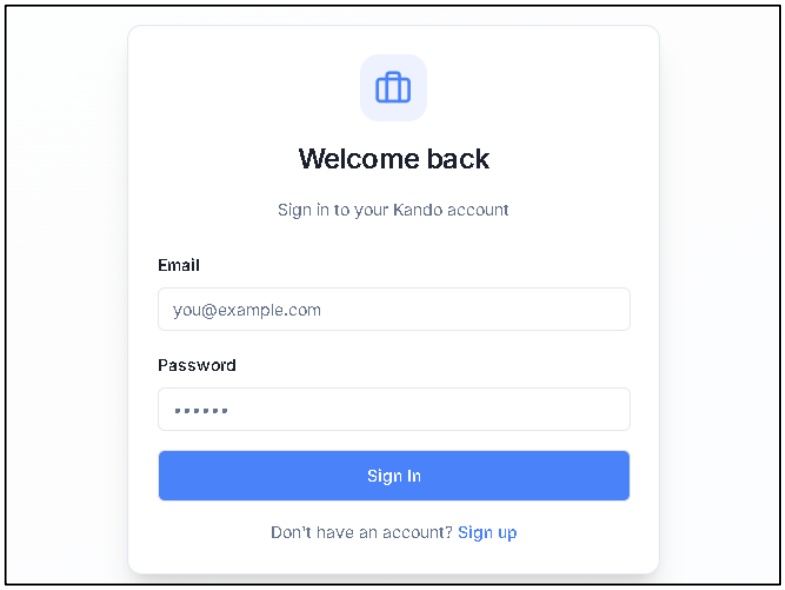


Рисунок 3.3 – Форма входу в обліковий запис

Після успішної перевірки введеної інформації сервер підтверджує особу користувача та надає доступ до внутрішнього функціоналу вебсервісу. У разі введення некоректних облікових даних система відображає відповідне повідомлення про помилку, що дозволяє уникнути несанкціонованого доступу.

Окрему увагу під час розроблення системи було приділено механізму захисту маршрутів та забезпеченню інформаційної безпеки. Будь-яка спроба переходу до внутрішніх сторінок вебсервісу без наявності валідного JWT-токена автоматично перехоплюється механізмом маршрутизації клієнтського застосунку. У такому випадку користувач перенаправляється на сторінку авторизації, що виключає можливість несанкціонованого перегляду конфіденційних даних.

Додатково реалізовано серверну перевірку доступу, відповідно до якої кожен HTTP-запит до захищених API-ресурсів проходить процедуру перевірки валідності токена автентифікації. Це дозволяє гарантувати, що доступ до функціоналу системи можуть отримати виключно авторизовані користувачі, права яких відповідають визначеним умовам доступу. Такий підхід підвищує рівень безпеки вебсервісу та забезпечує захист персональних і проєктних даних від несанкціонованого використання.

3.2 Рольова модель управління проєктами та формування команд

Після успішного проходження процедури авторизації користувач отримує доступ до головної панелі керування (Dashboard), яка виступає центральним елементом взаємодії із системою та забезпечує доступ до основного функціоналу вебсервісу. Інтерфейс панелі керування динамічно адаптується відповідно до ролі користувача, що дозволяє реалізувати принцип персоналізації функціональних можливостей та оптимізувати користувацький досвід (див. рис. 3.4).

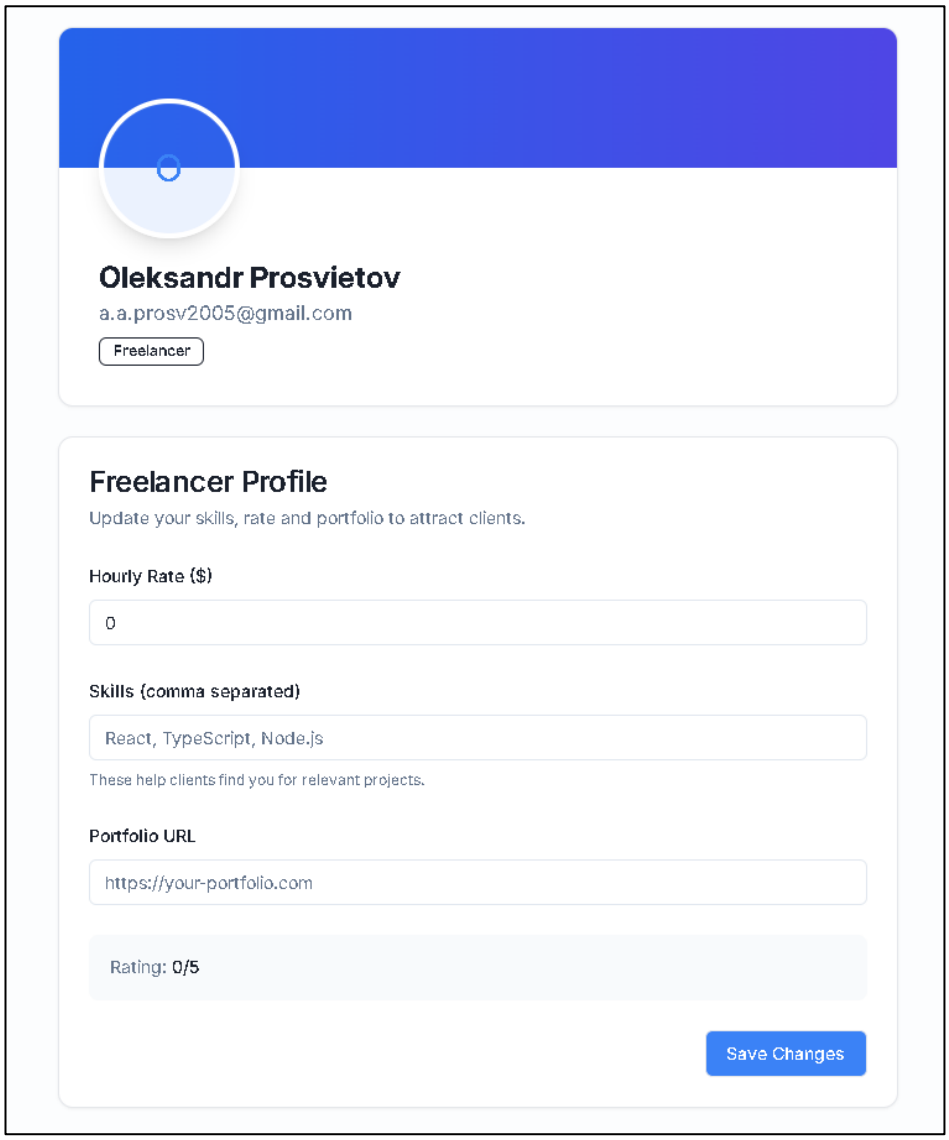


Рисунок 3.4 – Головна панель керування користувача

Однією з ключових особливостей реалізованої системи є використання рольової моделі доступу, яка передбачає розмежування функціональних можливостей між двома основними категоріями користувачів – замовниками (Client) та виконавцями (Freelancer). Такий підхід дозволяє не лише забезпечити логічне структурування функціоналу, але й підвищити рівень безпеки шляхом обмеження доступу до окремих ресурсів залежно від типу облікового запису.

Для покращення навігації та швидкого доступу до необхідної інформації всі списки проєктів оснащені механізмом миттєвого пошуку, який забезпечує інтерактивну фільтрацію елементів у режимі реального часу без необхідності оновлення сторінки. Завдяки використанню реактивних механізмів

клієнтського застосунку користувач може швидко знаходити необхідні проєкти, що позитивно впливає на загальну продуктивність роботи із системою.

Для користувачів із роллю замовника головна панель орієнтована насамперед на створення, організацію та контроль виконання проєктів. У відповідному розділі відображається перелік проєктів, створених конкретним користувачем, а також надається можливість ініціювати створення нових робочих просторів (див. рис. 3.5).

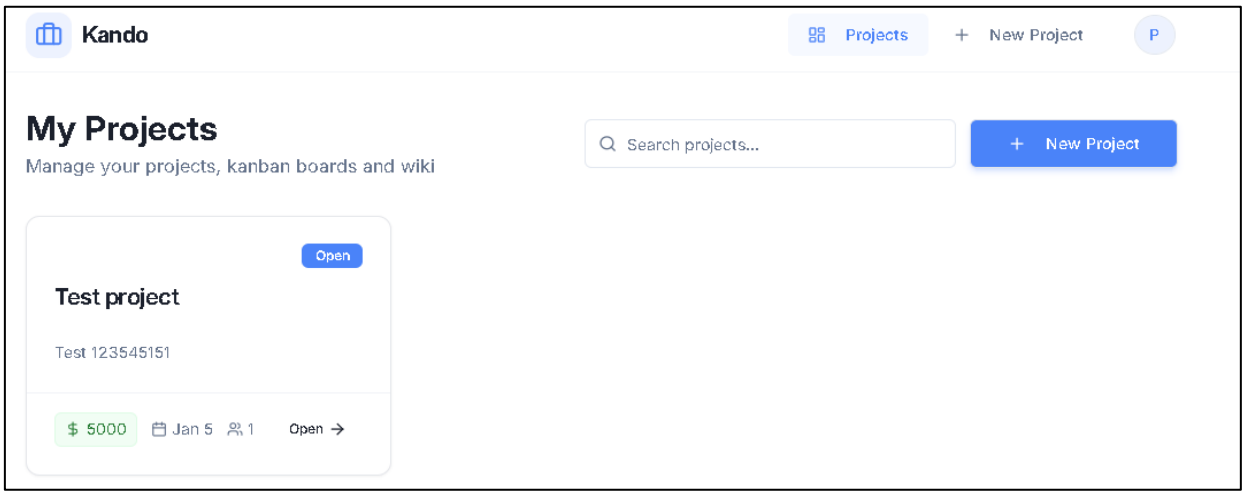


Рисунок 3.5 – Відображення списку проєктів замовника

Процес створення нового проєкту реалізовано через спеціалізовану форму введення даних, яка містить набір обов'язкових параметрів для опису майбутньої роботи (рис. 3.6). Під час створення проєкту користувач повинен вказати назву, детальний опис, виділений бюджет та кінцевий термін виконання. Така структура дозволяє ще на етапі ініціалізації сформувати чітке уявлення щодо очікуваного результату та умов співпраці.

Після успішного створення проєкту замовник отримує доступ до його детальної сторінки, яка виступає централізованим середовищем керування всіма аспектами робочого процесу (див. рис. 3.7). Інтерфейс сторінки структуровано у вигляді окремих функціональних вкладок, серед яких представлено загальний огляд проєкту, Kanban-дошку, Wiki-модуль та систему управління учасниками.

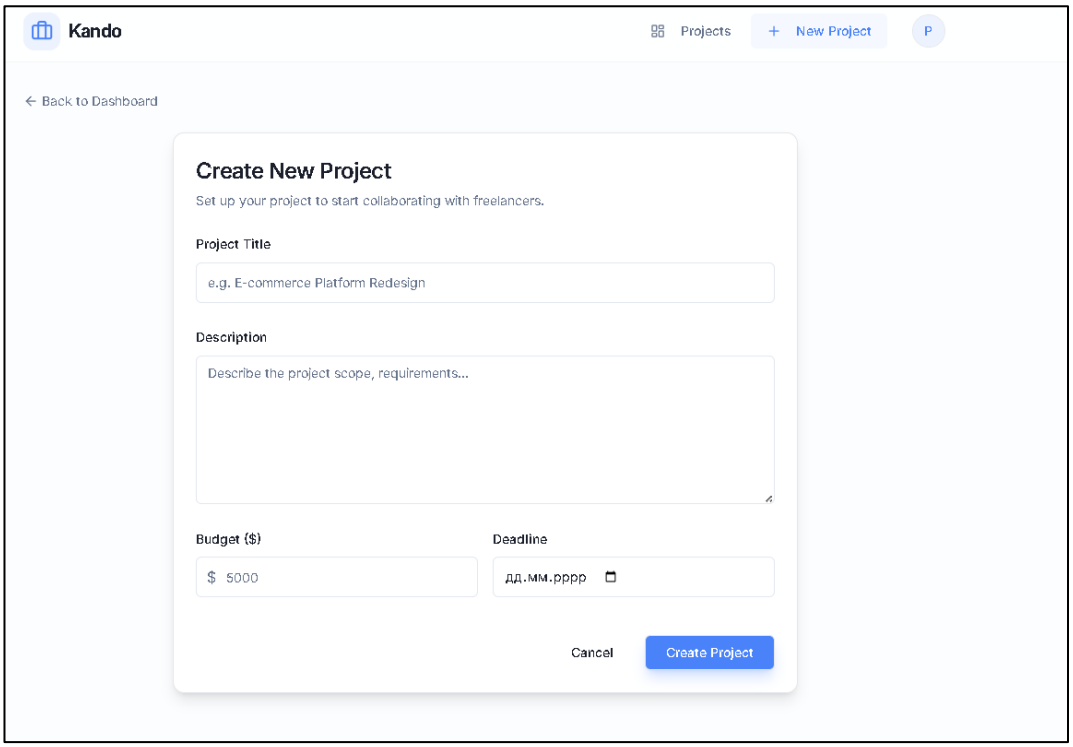


Рисунок 3.6 – Форма створення нового проєкту

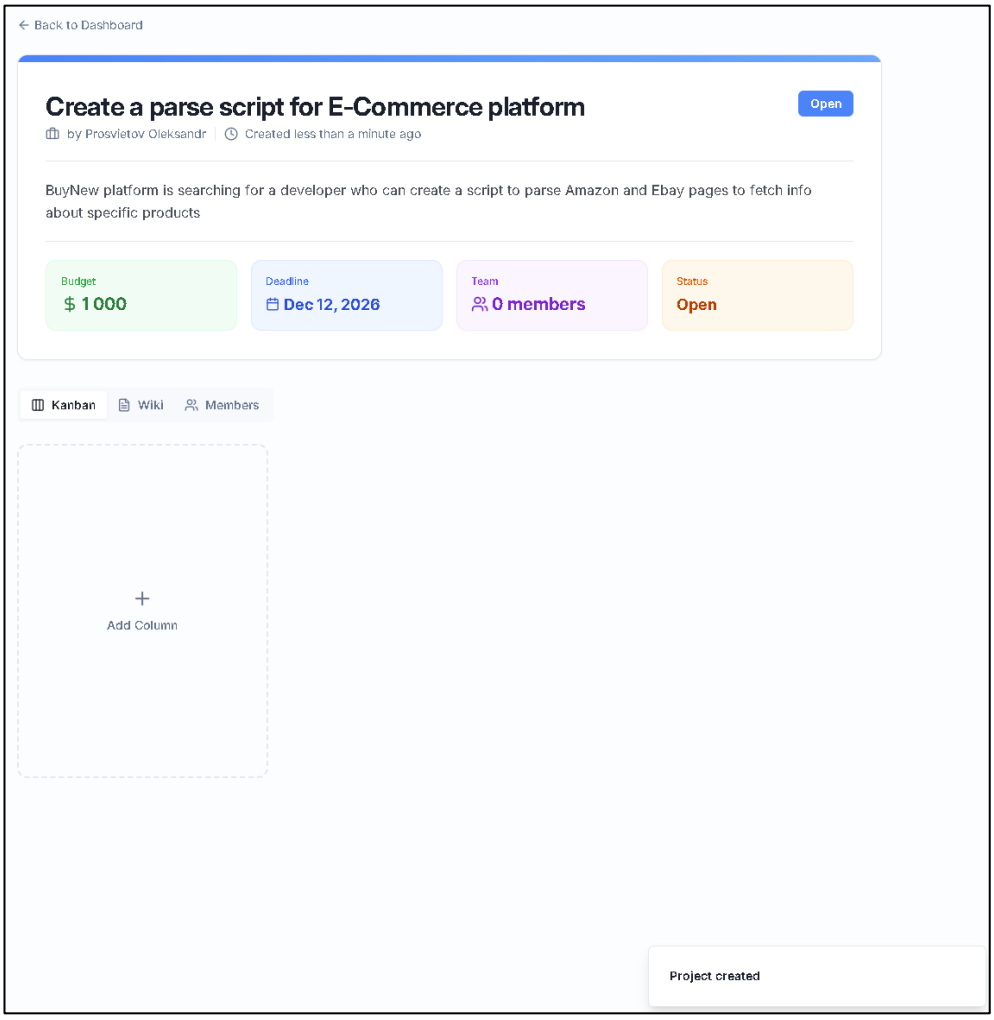


Рисунок 3.7 – Сторінка управління проєктом

У блоці загальної інформації користувачеві доступні ключові параметри проєкту, включаючи опис, бюджет, терміни виконання, а також інформацію щодо активності команди. Такий підхід дозволяє забезпечити централізоване відображення найважливішої інформації та підвищити ефективність управління процесом реалізації завдань.

Окреме місце у функціоналі замовника займає механізм формування команди виконавців. Для цього реалізовано спеціалізований модуль керування учасниками, який дозволяє здійснювати пошук зареєстрованих фрилансерів за ключовими ідентифікаційними параметрами, зокрема ім’ям користувача або адресою електронної пошти (див. рис. 3.8).

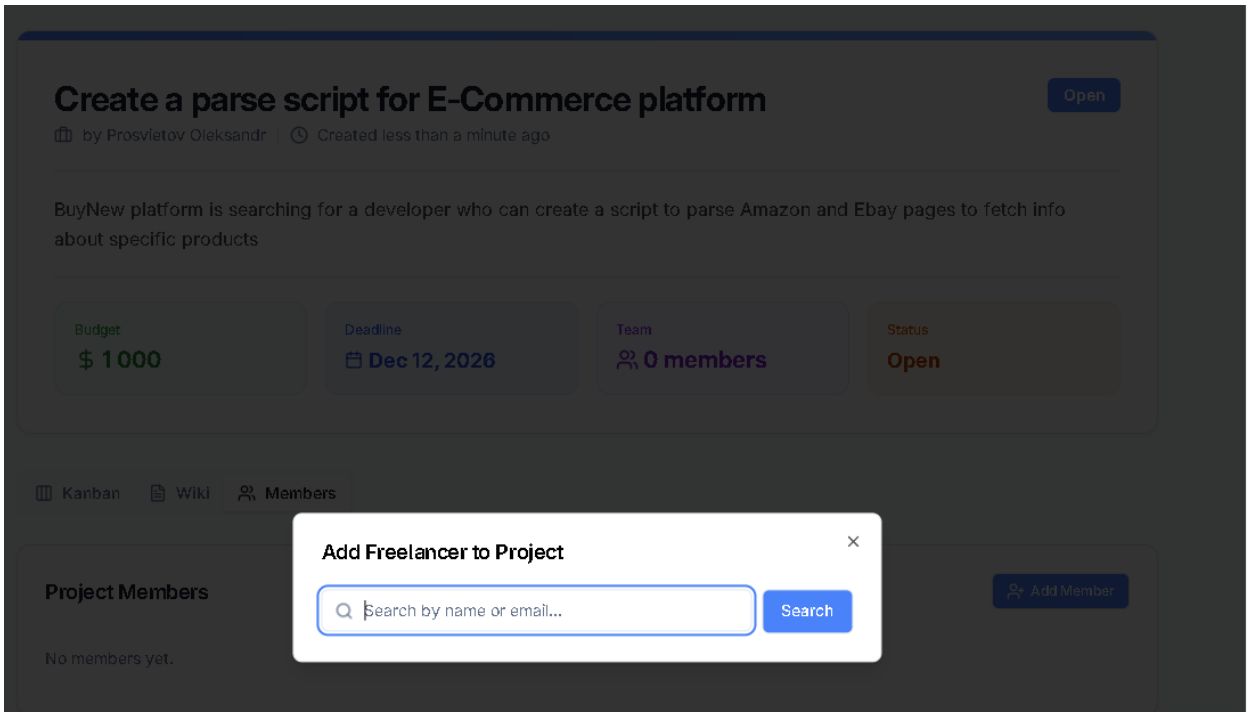


Рисунок 3.8 – Інтерфейс формування команди проєкту

Після знаходження відповідного користувача замовник може безпосередньо додати його до робочої групи проєкту. У результаті цього виконавець отримує доступ до внутрішніх ресурсів проєкту та можливість брати участь у виконанні поставлених завдань. Такий механізм значно спрощує процес організації співпраці та дозволяє оперативно формувати команди відповідно до вимог конкретного проєкту.

Важливо зазначити, що реалізовані механізми обмеження доступу дублюються не лише на рівні графічного інтерфейсу, але й на серверній стороні. Кожен запит до захищених ресурсів проходить обов’язкову перевірку валідності токена автентифікації, ролі користувача та його належності до конкретного проєкту. Це дозволяє запобігти спробам несанкціонованого отримання доступу до конфіденційної інформації шляхом прямої взаємодії з API.

Таким чином, реалізована рольова модель забезпечує чітке розмежування функціональних можливостей між замовниками та виконавцями, підтримує ефективний процес формування команд і створює безпечне середовище для організації спільної роботи над фриланс-проєктами.

Після отримання доступу до проєкту ключовими інструментами організації спільної роботи стають інтегровані підсистеми Kanban та Wiki, які забезпечують структуроване управління робочими процесами та централізоване збереження документації у межах єдиного інформаційного простору.

Одним із центральних компонентів вебсервісу є Kanban-підсистема, призначена для організації процесу виконання завдань та візуального контролю за станом проєкту. Дошка реалізована у вигляді інтерактивного робочого середовища, структурованого за допомогою окремих колонок, кожна з яких відповідає певному етапу життєвого циклу задачі, наприклад створенню, активному виконанню або завершенню роботи.

Реалізована структура дозволяє учасникам команди швидко оцінювати поточний стан проєкту, визначати рівень навантаження та контролювати прогрес виконання завдань. Замовник має можливість керувати структурою Kanban-дошки, створюючи нові колонки, редагуючи їх назви або видаляючи непотрібні елементи відповідно до особливостей організації конкретного проєкту.

Учасники команди можуть створювати нові картки завдань, редагувати їх вміст, змінювати опис, а також переміщувати задачі між колонками за

допомогою механізму Drag-and-Drop. Використання інтерактивного способу взаємодії значно покращує зручність користування системою, оскільки дозволяє виконувати керування задачами без використання додаткових меню чи складних механізмів навігації (див. рис. 3.9).

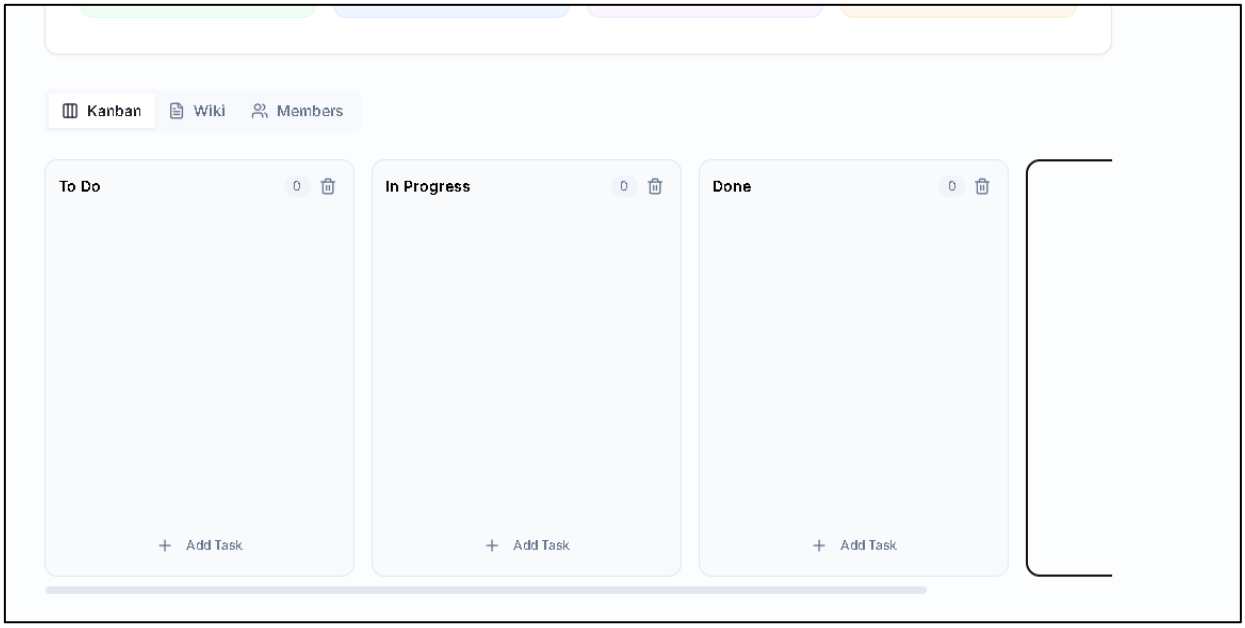


Рисунок 3.9 – Інтерфейс Kanban-дошки проєкту

Процес переміщення задач супроводжується плавним візуальним відображенням потенційної позиції картки, що покращує користувацький досвід та забезпечує зрозумілу взаємодію із системою. Після завершення переміщення клієнтський застосунок автоматично відправляє оновлені дані на сервер, завдяки чому забезпечується синхронізація змін та консистентність інформації без необхідності повторного завантаження сторінки.

Для централізованого накопичення та збереження інформації в межах проєкту реалізовано Wiki-підсистему, яка використовується для ведення технічної документації, опису функціональних вимог, специфікацій, нотаток та інших матеріалів, необхідних у процесі командної роботи. Наявність єдиного простору для документування дозволяє уникнути фрагментації інформації та спрощує доступ до важливих матеріалів для всіх учасників команди.

Wiki-модуль реалізовано у вигляді окремого розділу проєкту, де користувачі можуть створювати нові сторінки, редагувати вже наявні матеріали та використовувати базові інструменти форматування тексту. Завдяки цьому забезпечується структуроване представлення інформації та підтримується можливість централізованого зберігання знань у межах конкретного проєкту (див. рис. 3.10).

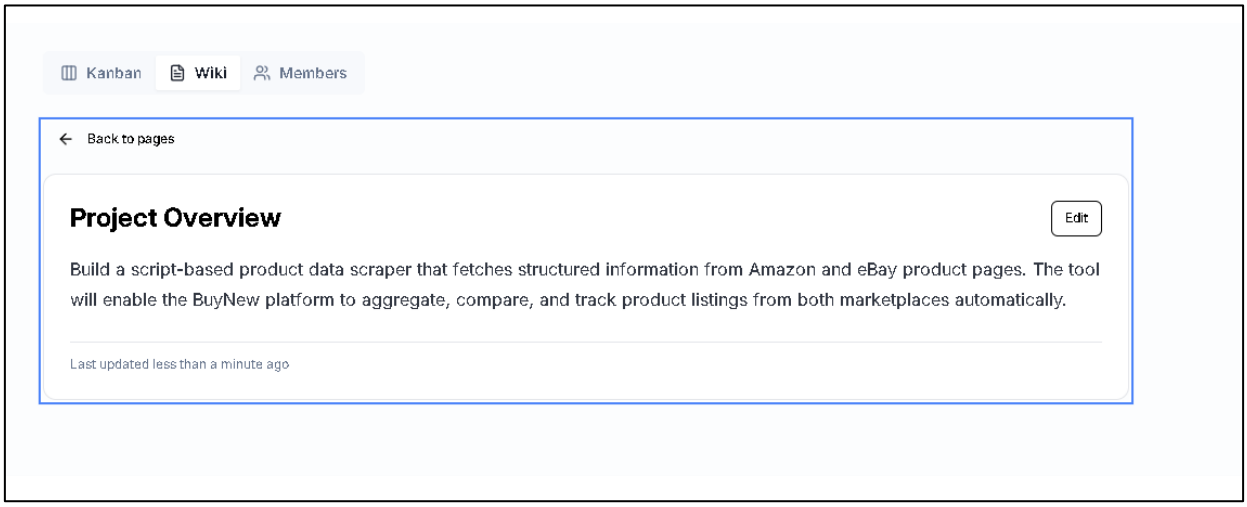


Рисунок 3.10 – Інтерфейс Wiki-підсистеми проєкту

З метою запобігання випадковому видаленню критично важливої інформації у системі реалізовано механізм розмежування прав доступу до Wiki-модуля. Створення та редагування сторінок доступне всім учасникам команди, тоді як право остаточного видалення матеріалів делеговано виключно власнику проєкту. Такий підхід дозволяє підтримувати цілісність документації та мінімізує ризик втрати важливої інформації.

Інтеграція підсистем Kanban та Wiki у межах одного вебсервісу дозволяє забезпечити комплексний підхід до організації роботи над фриланс-проєктами.

Поєднання інструментів управління завданнями та централізованого документування сприяє підвищенню ефективності командної взаємодії, покращує прозорість виконання робіт та забезпечує зручне середовище для координації діяльності всіх учасників проєкту.

3.3 Організація розгортання та DevOps-практики

Під час розроблення сучасних програмних систем важливим аспектом є не лише реалізація функціональних можливостей застосунку, але й забезпечення стабільного, передбачуваного та автоматизованого процесу його розгортання. З метою підвищення надійності вебсервісу, спрощення процесів оновлення та мінімізації впливу людського фактора в межах розробленої системи реалізовано набір DevOps-практик, що охоплюють контейнеризацію компонентів, автоматизацію збірки та перевірки програмного забезпечення, оркестрацію контейнерів і централізоване керування конфігураціями.

Однією з ключових технологій, використаних у процесі розроблення, стала контейнеризація, що дозволяє стандартизувати середовище виконання та усунути проблему відмінностей між конфігураціями різних пристроїв. Такий підхід забезпечує однакові умови запуску програмного забезпечення незалежно від операційної системи або локального оточення, що особливо важливо під час командної розробки та тестування.

Для ізоляції компонентів системи використано Docker-контейнери, у межах яких окремо розгортаються серверна частина застосунку, клієнтський інтерфейс та система управління базами даних PostgreSQL. З метою підвищення продуктивності та зменшення розміру фінальних контейнерних образів реалізовано підхід багатоетапної збірки (multi-stage build). Для серверної частини процес побудови контейнера поділено на декілька логічних етапів: початкову стадію компіляції вихідного коду, етап оптимізованої публікації застосунку та формування фінального середовища виконання. Завдяки цьому фінальний контейнер містить виключно необхідні для запуску компоненти без додаткових інструментів розробки, що позитивно впливає на безпеку та швидкість запуску.

Аналогічний підхід застосовано і для клієнтської частини системи. React-застосунок компілюється у середовищі Node.js, після чого сформовані статичні файли передаються до вебсервера Nginx, відповідального за їх

подальше обслуговування. Така архітектура дозволяє мінімізувати використання ресурсів і забезпечити стабільну роботу користувацького інтерфейсу.

Для локальної організації інфраструктури використано Docker Compose, що забезпечує централізоване керування взаємодією між контейнерами та дозволяє швидко ініціалізувати повноцінне середовище виконання за допомогою єдиної конфігурації (див. рис. 3.11). Використання такого підходу дозволяє уникнути ситуацій, коли програмне забезпечення працює лише у середовищі розробника, але демонструє помилки після перенесення на іншу систему.

```yaml
services:
  asp.net-demo:
    image: asp.net-demo
    build:
      context: .
      dockerfile: API/Dockerfile
    restart: unless-stopped
    environment:
      ConnectionStrings__DefaultConnection: "Host=db;Port=5432;Database=${DB_NAME:-test_db};Username=${DB_USER:-test_user};Password=${DB_PASSWORD:-test_pass}"
      Jwt__Key: ${JWT_KEY:-your_secret_key_that_should_be_long}
      Jwt__Issuer: ${JWT_ISSUER:-yourapp}
      Jwt__Audience: ${JWT_AUDIENCE:-yourapp_users}
    ports:
      - "8080:8080"
    networks:
      - db_net

  db:
    image: postgres:17-alpine
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${DB_USER:-test_user}
      POSTGRES_PASSWORD: ${DB_PASSWORD:-test_pass}
      POSTGRES_DB: ${DB_NAME:-test_db}
    ports:
      - "5432:5432"
    volumes:
      - db_data:/var/lib/postgresql/data
    networks:
      - db_net

volumes:
  db_data:

networks:
  db_net:
```

Рисунок 3.11 – Контейнеризована інфраструктура вебсервісу

У межах локальної інфраструктури окремо розгортаються API-застосунок, клієнтський інтерфейс, база даних PostgreSQL та допоміжні компоненти, необхідні для коректного функціонування сервісу. Для забезпечення збереження даних між перезапусками контейнерів реалізовано механізм іменованих томів (volumes), що дозволяє підтримувати сталість

стану системи та запобігати втраті інформації під час оновлення середовища виконання.

З метою автоматизації процесу перевірки програмного забезпечення та пришвидшення випуску нових версій реалізовано конвеєр безперервної інтеграції та доставки (CI/CD), побудований на базі GitHub Actions (рис. 3.12). Реалізований конвеєр автоматично запускається при внесенні змін до основної гілки репозиторію та виконує комплекс перевірок, спрямованих на забезпечення стабільності та безпеки програмного продукту.

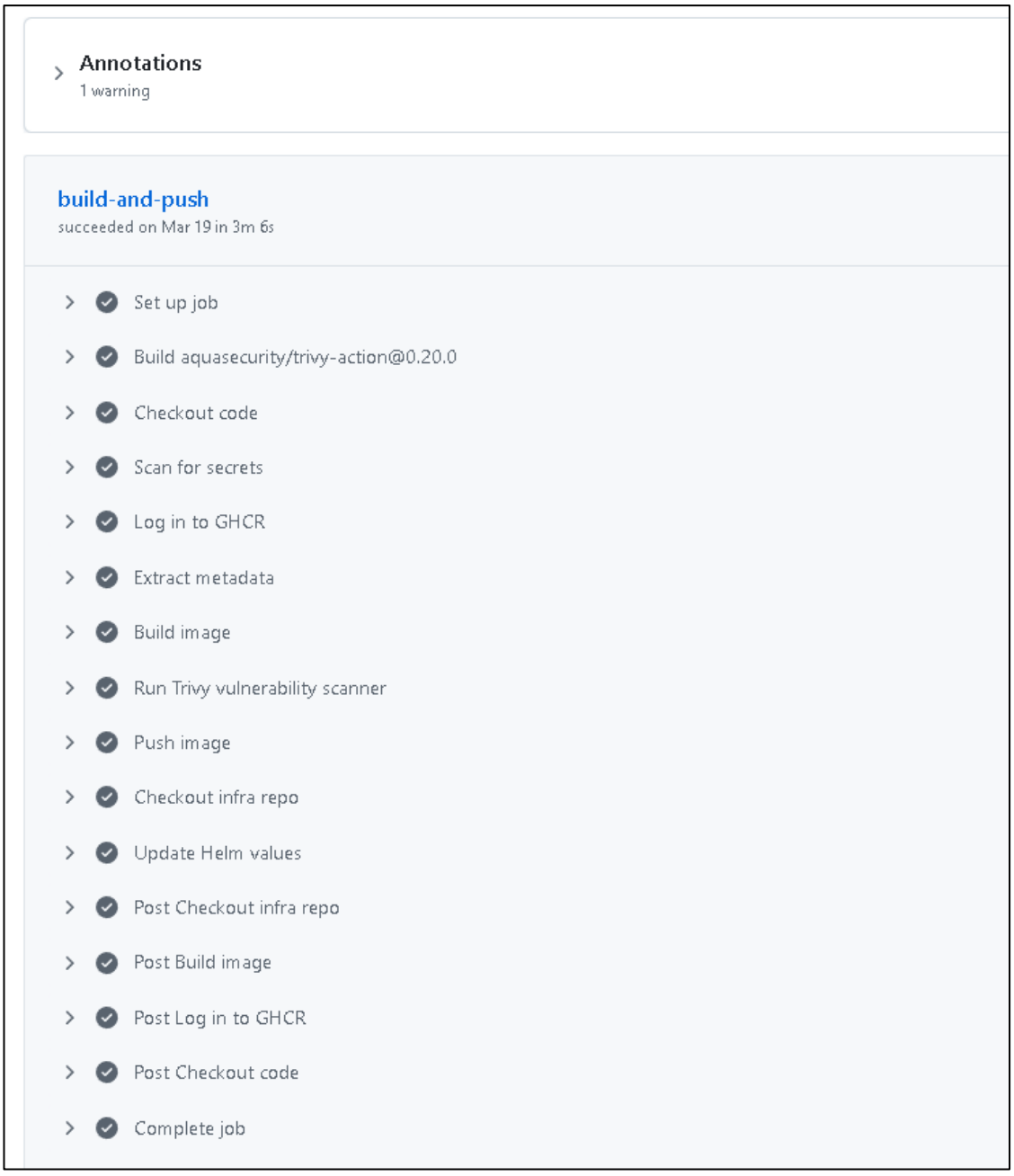


Рисунок 3.12 – Автоматизований конвеєр CI/CD

Одним із початкових етапів перевірки є автоматичний аналіз вихідного коду на предмет випадкового потрапляння конфіденційних даних, таких як API-ключі, токени або облікові дані доступу. Після успішного завершення перевірки запускається автоматизована збірка контейнерних образів, де кожна версія застосунку отримує унікальний ідентифікатор, прив'язаний до конкретного коміту репозиторію. Це забезпечує можливість чіткої ідентифікації версії програмного забезпечення та спрощує процес відстеження змін.

Додатково реалізовано автоматичне сканування контейнерних образів на наявність відомих вразливостей у системних пакетах та підключених бібліотеках. Лише після успішного проходження усіх перевірок сформовані образи публікуються у контейнерному реєстрі та можуть бути використані для подальшого розгортання системи.

Для управління контейнерною інфраструктурою використано платформу Kubernetes, яка забезпечує автоматизацію запуску, масштабування та підтримки працездатності компонентів системи (див. рис. 3.13).

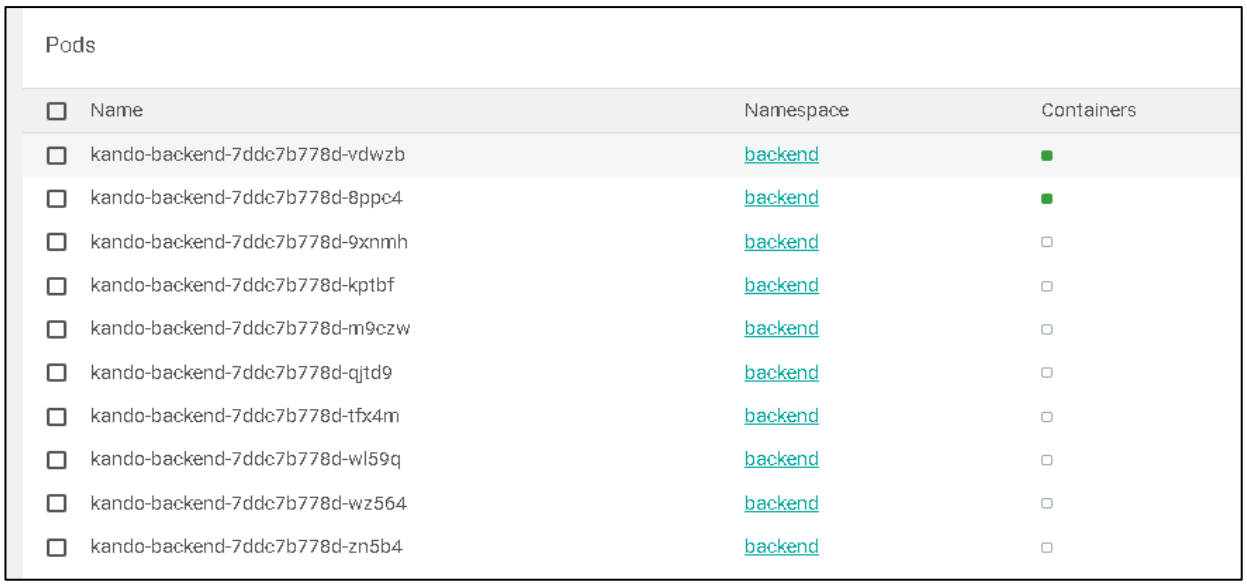


Рисунок 3.13 – Оркестрація контейнерів у Kubernetes-середовищі

Використання оркестратора дозволяє централізовано керувати життєвим циклом контейнерів, підтримувати стабільність середовища та спрощувати процес оновлення застосунку.

Для спрощення роботи з конфігураційними файлами Kubernetes використано пакетний менеджер Helm, що дозволяє централізовано описувати параметри розгортання та уникати дублювання конфігураційного коду. У межах Helm-конфігурацій визначаються параметри запуску контейнерів, обмеження використання ресурсів, правила мережевої взаємодії та стратегії оновлення застосунку.

Маршрутизація зовнішнього трафіку реалізована за допомогою Ingress-контролера Traefik, який виконує роль проміжного рівня між користувачем і внутрішніми сервісами системи. Використання цього механізму дозволяє централізовано керувати маршрутизацією запитів, балансуванням навантаження та конфігурацією доступу до окремих компонентів вебсервісу.

Для забезпечення стабільної роботи застосунку під час змін навантаження реалізовано механізм горизонтального автоматичного масштабування (Horizontal Pod Autoscaler, HPA [11]). Відповідно до поточного рівня використання ресурсів система автоматично регулює кількість активних екземплярів застосунку, що дозволяє підтримувати продуктивність у періоди підвищеної активності та уникати надмірного споживання ресурсів під час низького навантаження.

Окрему увагу приділено механізмам контролю працездатності контейнерів. Для цього реалізовано систему перевірок стану застосунку, яка включає перевірку успішності запуску, постійний моніторинг працездатності та оцінювання готовності контейнера до обслуговування користувацьких запитів. У випадку виявлення помилок Kubernetes автоматично перезапускає проблемні компоненти, що дозволяє підтримувати безперервність роботи системи без втручання адміністратора.

Для уникнення конфліктів під час оновлення структури бази даних процес застосування міграцій винесено в окремий механізм виконання, який гарантує завершення структурних змін до моменту запуску оновленої версії застосунку. Це дозволяє забезпечити цілісність даних та уникнути ризиків, пов’язаних із несумісністю версій.

Для автоматизації процесу розгортання та підтримання узгодженого стану інфраструктури в межах проєкту реалізовано GitOps-підхід із використанням платформи Argo CD (рис. 3.14). У запропонованій архітектурі Git-репозиторій використовується як централізоване джерело конфігурації Kubernetes-інфраструктури, тоді як Argo CD виконує постійний моніторинг змін та автоматичну синхронізацію фактичного стану кластера із заданими параметрами розгортання.

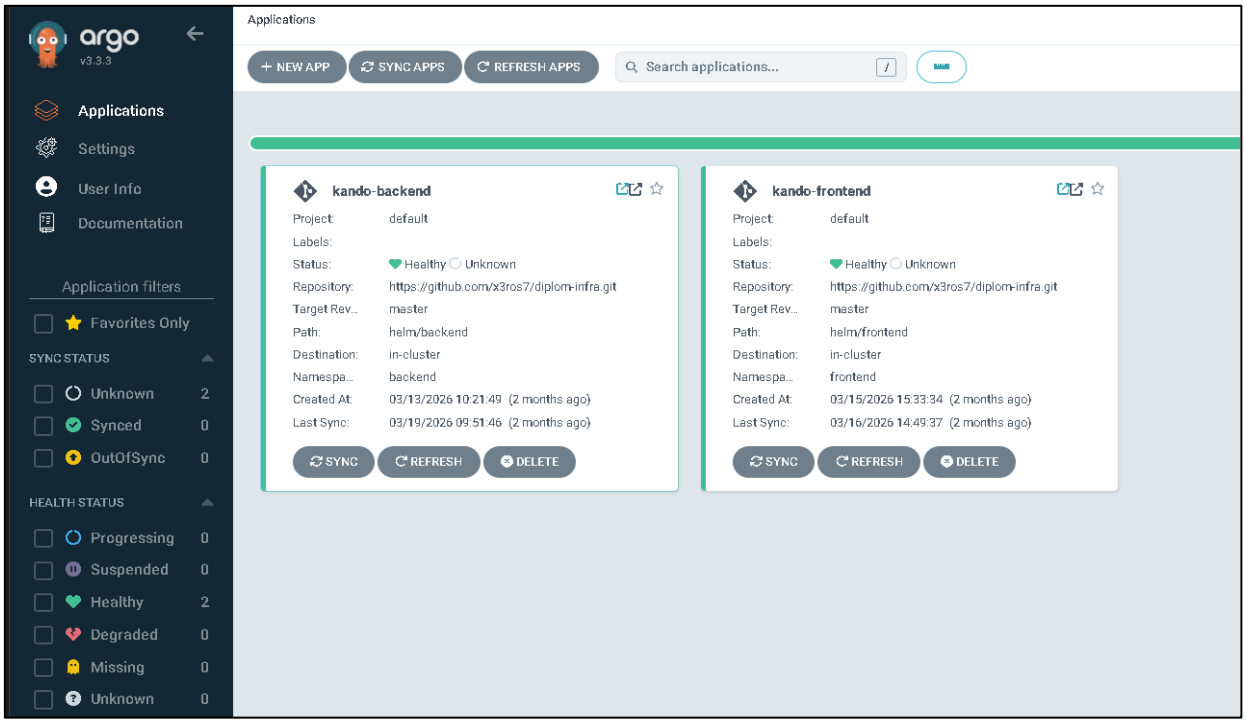


Рисунок 3.14 – Інтерфейс керування розгортанням застосунків в Argo CD

У межах реалізованої конфігурації окремо керуються клієнтська та серверна частини вебсервісу, що дозволяє незалежно оновлювати компоненти системи, зменшуючи ризик порушення працездатності всієї платформи. Конфігурація розгортання базується на Helm-чартах, які визначають параметри запуску контейнерів, маршрутизацію мережевих запитів та особливості конфігурації середовища.

Використання Argo CD дозволило автоматизувати процес оновлення застосунку, спростити контроль версій інфраструктурних конфігурацій та забезпечити відтворюваність середовища розгортання. Крім того, система

автоматично виявляє розбіжності між фактичним і визначеним станом інфраструктури, що сприяє підтримці стабільності роботи вебсервісу.

Таким чином, використання сучасних DevOps-практик дозволило забезпечити стандартизоване середовище виконання, автоматизувати процес перевірки та розгортання застосунку, підвищити рівень безпеки, а також створити гнучку інфраструктуру, придатну до масштабування та подальшого розвитку вебсервісу.

ВИСНОВКИ

У кваліфікаційній роботі виконано проєктування та розроблення вебсервісу для управління фриланс-проєктами, спрямованого на підвищення ефективності організації спільної роботи між замовниками та виконавцями. У процесі виконання роботи реалізовано комплексне програмне рішення, що поєднує засоби управління завданнями, командної взаємодії та ведення проєктної документації в межах єдиного інформаційного середовища.

За результатами виконання кваліфікаційної роботи було проведено аналіз предметної області та існуючих програмних рішень у сфері управління проєктами для незалежних спеціалістів. У ході дослідження визначено основні проблеми організації діяльності фрилансерів, що пов’язані з використанням розрізнених цифрових інструментів для управління завданнями, комунікації та ведення документації. Проведений аналіз сучасних аналогів дозволив виявити їхні функціональні переваги та обмеження, а також сформувати перелік вимог до розроблення спеціалізованого вебсервісу.

У межах роботи спроєктовано архітектуру бази даних та загальну структуру взаємодії компонентів системи. Розроблено реляційну модель даних, яка забезпечує підтримку процесів управління користувачами, проєктами, ролями, завданнями та документацією. Запропонована структура сприяє забезпеченню цілісності даних, підтримує масштабованість програмного продукту та дозволяє реалізувати гнучке розмежування прав доступу між учасниками системи.

Реалізовано серверну частину вебсервісу, побудовану на основі багатошарової архітектури із використанням сучасних підходів до розроблення програмного забезпечення. Забезпечено механізми автентифікації та авторизації користувачів, реалізовано захищену взаємодію між клієнтською та серверною частинами застосунку, а також впроваджено бізнес-логіку, необхідну для управління проєктами, ролями та доступом до функціональних модулів системи.

Також розроблено клієнтський вебзастосунок із адаптивним користувацьким інтерфейсом та підтримкою рольової моделі взаємодії. У складі вебсервісу реалізовано інтегровану Kanban-дошку для управління завданнями із підтримкою інтерактивного переміщення карток, а також Wiki-підсистему для централізованого збереження технічної документації, робочих нотаток та проєктних матеріалів.

Окрему увагу приділено автоматизації процесів розгортання та супроводу програмного забезпечення. Для цього застосовано контейнеризацію компонентів вебсервісу, засоби автоматизованої збірки та перевірки програмного забезпечення, а також механізми оркестрації контейнерів на основі Kubernetes. Додатково реалізовано GitOps-підхід до управління інфраструктурними конфігураціями та впроваджено механізми безпечного зберігання конфіденційних даних.

У результаті виконання роботи створено повноцінний вебсервіс для управління проєктами фрилансерів, що об’єднує засоби організації завдань і ведення документації в єдиному інформаційному середовищі. Розроблене програмне рішення дозволяє зменшити часові витрати на координацію робочих процесів, підвищити рівень структурованості проєктної інформації та спростити взаємодію між учасниками проєкту. Отримані результати підтверджують досягнення поставленої мети кваліфікаційної роботи.

Практичне значення одержаних результатів полягає у створенні повноцінного програмного продукту, придатного до практичного використання для організації фриланс-проєктів. Запропоноване рішення дозволяє централізувати процеси управління завданнями та ведення документації, підвищити прозорість взаємодії між учасниками команди та скоротити часові витрати на організацію робочого процесу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Kanban Methodology. *Atlassian*. URL: https://www.atlassian.com/agile/kanban (дата звернення: 03.03.2026).

2. ASP.NET Core Documentation. *Microsoft Learn*. URL: https://learn.microsoft.com/aspnet/core (дата звернення: 03.03.2026).

3. PostgreSQL Documentation. *PostgreSQL Global Development Group*. URL: https://www.postgresql.org/docs (дата звернення: 03.03.2026).

4. React Team. React Documentation. URL: https://react.dev (дата звернення: 03.03.2026).

5. Docker Inc. *Docker Documentation*. URL: https://docs.docker.com (дата звернення: 10.03.2026).

6. Kubernetes Authors. *Kubernetes Documentation.* URL: https://kubernetes.io/docs (дата звернення: 20.03.2026).

7. Helm Project. Helm Documentation. URL: https://helm.sh/docs/ (date of access: 26.04.2026).

8. Argo Project. *Argo CD Documentation*. URL: https://argo-cd.readthedocs.io (дата звернення: 05.04.2026).

9. GitHub Actions Documentation. URL: https://docs.github.com/actions (дата звернення: 05.04.2026).

10. Traefik Proxy Documentation. *Traefik Labs Docs*. URL: https://doc.traefik.io/traefik/ (дата звернення: 12.04.2026).

11. Kubernetes Authors. Horizontal Pod Autoscaling. URL: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale (дата звернення: 10.04.2026).