ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система управління ІТ-проєктами малого бізнесу»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 – Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

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

_____ Катерина ЩЕРБАКОВА

Виконав: здобувачка вищої освіти гр. ІСД-41
Катерина ЩЕРБАКОВА

Керівник: Каміла СТОРЧАК
науковий ступінь, вчене звання *доктор технічних наук, професор*

Рецензент: _____
науковий ступінь, вчене звання

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 – Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою інформаційних систем та технологій
Каміла СТОРЧАК
“____” _____2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Щербаковій Катерині Євгенівні

1.Тема кваліфікаційної роботи: Інформаційна система управління ІТ-проєктами малого бізнесу

керівник кваліфікаційної роботи: Каміла СТОРЧАК, доктор технічних наук, професор

затверджені наказом Державного університету інформаційно-комунікаційних технологій від “ 20 ” лютого 2026 р. № 51

2. Строк подання кваліфікаційної роботи «01» червня 2026 р.

3. Вихідні дані кваліфікаційної роботи:
1. Принципи побудови та функціонування інформаційних систем управління ІТ-проєктами.
2. Методи та підходи до управління ІТ-проєктами малого бізнесу.
3. Технології веб-розробки та клієнт-серверної взаємодії.
4. Науково-технічна література та сучасні програмні рішення у сфері управління проєктами.

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

1. Аналіз особливостей діяльності малого бізнесу у сфері ІТ та сучасних систем управління проєктами.
2. Проєктування інформаційної системи управління ІТ-проєктами малого бізнесу.
3. Розробка та реалізація клієнтської і серверної частин інформаційної системи.
4. Реалізація функціональних модулів системи та тестування програмного продукту.

5.Перелік ілюстраційного матеріалу: *презентація*

1. Титульний слайд.
2. Мета, об'єкт, предмет дослідження. Завдання роботи.
3. Порівняльний аналіз аналогів (таблиця: Trello / Jira / Asana / ClickUp — і чому не підходять для малого бізнесу).
4. Функціональні та нефункціональні вимоги до системи.
5. Архітектура системи та стек технологій.
6. Діаграма прецедентів (Use Case Diagram).
7. ER-діаграма бази даних.
8. Інтерфейс застосунку — авторизація та головна сторінка (скріни).
9. Інтерфейс застосунку — управління проєктами та завданнями (скріни).
10. Результати тестування.
11. Висновки.
12. Апробація.

6. Дата видачі завдання «21» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір технічної літератури	19.04-24.04.26	
2.	Аналіз особливостей діяльності малого бізнесу у сфері ІТ та сучасних систем управління проєктами	25.04-01.05.26	

3.	Розробка архітектури, бази даних та дизайну інтерфейсу системи	02.05-08.05.26	
4.	Розробка та реалізація клієнтської і серверної частин системи	09.05-20.05.26	
5.	Тестування програмного продукту та аналіз результатів роботи	21.05-24.05.26	
6.	Розробка демонстраційних матеріалів, доповідь.	24.05-27.05.26	
7.	Оформлення бакалаврської роботи	19.04-27.05.26	

Здобувачка вищої освіти _____ Катерина ЩЕРБАКОВА
(підпис)

Керівник
кваліфікаційної роботи _____ Каміла СТОРЧАК
(підпис)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра

Направляється здобувач Щербакова К.Є. до захисту кваліфікаційної роботи

(прізвище та ініціали)

за спеціальністю 126 Інформаційні системи та технології

(код, найменування спеціальності)

освітньо-професійної програми Інформаційні системи та технології

на тему: «Інформаційна система управління ІТ-проєктами малого бізнесу».

Кваліфікаційна робота і рецензія додаються.

Директор ННІІТ _____ Катерина НЕСТЕРЕНКО

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувачка Щербакова Катерина Євгенівна обрала тему кваліфікаційної роботи, метою якої була розробка та реалізація інформаційної системи управління ІТ-проєктами малого бізнесу, що забезпечує планування, контроль виконання завдань та координацію роботи команди. Під час виконання кваліфікаційної роботи Щербакова К.Є. показала добру теоретичну та практичну підготовку, вміння вирішувати самостійно питання та проводити дослідження. Робота виконана сумлінно та вчасно за планом.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувачки Щербакової К.Є. на оцінку «_____» та присвоїти їй кваліфікацію «бакалавр з інформаційних систем та технологій ».

Керівник кваліфікаційної роботи _____ Каміла СТОРЧАК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

"___" _____20__ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувачка Щербакова К.Є. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедри Інформаційних систем та технологій
(назва)

_____ Каміла СТОРЧАК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну роботу
на здобуття освітнього ступеня бакалавра

здобувачки вищої освіти **Щербакової Катерини Євгенівни**
на тему: «Інформаційна система управління ІТ-проєктами малого бізнесу».

Актуальність.

В умовах цифровізації бізнес-процесів ефективне управління ІТ-проєктами стає важливим чинником успішної діяльності підприємств. Особливо актуальною ця проблема є для малого бізнесу, який потребує доступних та зручних інструментів для планування роботи, контролю виконання завдань і координації команди. Актуальність роботи полягає у розробці інформаційної системи управління ІТ-проєктами малого бізнесу, що спрямована на підвищення ефективності управлінських процесів та автоматизацію проєктної діяльності.

Позитивні сторони.

1. Зміст роботи повністю відповідає завданню, а поставлені задачі виконані у повному обсязі.
2. У роботі проведено аналіз сучасних систем управління проєктами та обґрунтовано вибір архітектурного рішення.
3. Розроблений інтерфейс відзначається зручністю використання та сучасним дизайном.

Недоліки.

1. Недостатньо розглянуто питання інформаційної безпеки та захисту даних.
2. Доцільно було б розширити функціональні можливості системи шляхом інтеграції із зовнішніми сервісами.

Відзначені зауваження не впливають на загальну позитивну оцінку бакалаврської кваліфікаційної роботи.

Висновок: *кваліфікаційна робота на здобуття ступеня бакалавра заслуговує оцінку "_____", а здобувачка Щербакова Катерина Євгенівна заслуговує присвоєння кваліфікації: бакалавр з інформаційних систем та технологій.*

Рецензент: _____ _____
науковий ступінь, вчене звання *підпис* *Ім'я, ПРІЗВИЩЕ*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступня бакалавр: 66 стор., 17 рис., 10 табл., 23 джерел.

Мета роботи – розробка та реалізація інформаційної системи управління ІТ-проєктами малого бізнесу, що забезпечує ефективне планування, контроль виконання завдань, координацію роботи команди та оптимізацію використання ресурсів відповідно до сучасних вимог автоматизації бізнес-процесів.

Об'єкт дослідження – процес управління ІТ-проєктами на підприємствах малого бізнесу із застосуванням сучасних цифрових технологій.

Предмет дослідження – архітектура, функціональні можливості та принципи побудови інформаційної системи управління ІТ-проєктами малого бізнесу.

Короткий зміст роботи. У бакалаврській роботі проведено аналіз особливості управління ІТ-проєктами у сфері малого бізнесу, досліджено сучасні програмні рішення для планування та контролю проєктної діяльності, визначено їх переваги та недоліки. Сформовано вимоги до інформаційної системи та розроблено її концептуальну модель. Запропоновано архітектуру системи, структуру бази даних, функціональні модулі для управління завданнями, термінами виконання, командною взаємодією та формуванням звітності. Обґрунтовано вибір програмних засобів реалізації, наведено приклади інтерфейсу користувача та сценарії використання системи. Розроблено інформаційну систему управління ІТ-проєктами малого бізнесу.

КЛЮЧОВІ СЛОВА: інформаційна система, управління ІТ-проєктами, малий бізнес, веб-застосунок, автентифікація, авторизація, база даних, цифровізація, автоматизація процесів, клієнт-серверна архітектура, функціональні вимоги, тестування програмного забезпечення.

ЗМІСТ

ВСТУП

В умовах сучасної цифрової трансформації економіки важливу роль у підвищенні ефективності підприємств та їх конкурентоспроможності відіграють інформаційні технології. Особливого значення набуває впровадження сучасних інформаційних систем у діяльність підприємств малого бізнесу, які потребують ефективних інструментів управління ресурсами, персоналом та проєктною діяльністю. Малий бізнес є важливою складовою економіки більшості країн світу, оскільки забезпечує створення робочих місць, сприяє інноваціям та формує гнучке конкурентне середовище.

У сфері інформаційних технологій значна кількість малих підприємств здійснює діяльність у форматі проєктної роботи. Це можуть бути компанії з розробки програмного забезпечення, веб-студії, digital-агентства, консалтингові організації та стартапи. Для таких підприємств характерними є обмежені фінансові та кадрові ресурси, висока динамічність змін, необхідність швидкого реагування на потреби замовників та одночасне виконання кількох проєктів. У таких умовах ефективне управління ІТ-проєктами є одним із ключових чинників успішної діяльності підприємства.

Традиційні методи управління проєктами, що базуються на використанні електронних таблиць, паперової документації або розрізнених сервісів комунікації, не забезпечують належного рівня координації роботи команди, контролю строків виконання завдань та оперативного доступу до актуальної інформації. Це призводить до затримок виконання робіт, нераціонального використання ресурсів, втрати даних та зниження якості кінцевого результату. Саме тому зростає потреба у впровадженні спеціалізованих інформаційних систем управління проєктами.

На ринку існує значна кількість програмних продуктів для управління проєктами, зокрема Jira, Trello, Asana та Monday.com. Проте багато з них орієнтовані на великі компанії, мають складний функціонал, високі витрати на

використання або не повністю враховують потреби малого бізнесу. У зв’язку з цим актуальним є розроблення інформаційної системи управління ІТ-проєктами малого бізнесу, яка поєднуватиме необхідний функціонал, зручність використання та доступність впровадження.

Таким чином, тема бакалаврської роботи є актуальною, оскільки спрямована на вирішення практичного завдання підвищення ефективності управління ІТ-проєктами підприємств малого бізнесу шляхом створення сучасної інформаційної системи.

Метою бакалаврської роботи є розробка та реалізація інформаційної системи управління ІТ-проєктами малого бізнесу, що забезпечує планування, контроль виконання завдань, координацію роботи команди та підвищення ефективності управлінських процесів.

Для досягнення поставленої мети необхідно виконати такі завдання:

- дослідити особливості діяльності малого бізнесу у сфері інформаційних технологій;
- проаналізувати сучасні підходи до управління ІТ-проєктами;
- здійснити огляд існуючих програмних систем управління проєктами;
- визначити функціональні та технічні вимоги до інформаційної системи;
- розробити структуру та архітектуру системи;
- спроєктувати базу даних інформаційної системи;
- розробити інтерфейс користувацького інтерфейсу;
- оцінити практичну доцільність впровадження системи.

Об’єкт дослідження - процес управління ІТ-проєктами на підприємствах малого бізнесу.

Предмет дослідження - методи, моделі та програмні засоби побудови інформаційної системи управління ІТ-проєктами малого бізнесу.

У процесі виконання бакалаврської роботи використано методи аналізу та узагальнення наукових джерел, системного аналізу, моделювання інформаційних процесів, проєктування баз даних, порівняльного аналізу програмних продуктів, а також методи візуального проєктування інтерфейсів користувача.

Наукова новизна роботи полягає у розробленні концептуальної моделі інформаційної системи управління ІТ-проєктами, адаптованої до потреб підприємств малого бізнесу з урахуванням обмеженості ресурсів, необхідності гнучкого планування та оперативного контролю виконання завдань.

Практична значущість роботи полягає у можливості використання запропонованої інформаційної системи для автоматизації управління проєктною діяльністю малих ІТ-компаній, підвищення продуктивності праці команди та покращення якості управлінських рішень.

Основні положення та результати бакалаврської роботи можуть бути представлені на студентських науково-практичних конференціях та використані у подальших дослідженнях у сфері інформаційних систем управління проєктами.

Апробація результатів та публікації. Основні положення і результати бакалаврської роботи доповідались на науково-практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій:

1. Щербакова К. Є.— «Системний аналіз та порівняльна оцінка ефективності Agile та гібридних методологій управління ІТ-проєктами» // ІІІ Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». — Київ : ДУІКТ, 2025.
2. Щербакова К.Є. — «Аналітичні інструменти для підвищення ефективності малого бізнесу» // Всеукраїнська науково-технічна конференція «Технології

цифрового розвитку: програмні системи та децентралізація». — Київ : ДУІКТ, 2026.

1 АНАЛІЗ ОСОБЛИВОСТЕЙ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ МАЛОГО БІЗНЕСУ ТА СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Особливості діяльності малого бізнесу у сфері ІТ

За даними Європейської комісії, малі та середні підприємства становлять близько 99 % від загальної кількості компаній у країнах ЄС та забезпечують понад 50 % ВВП. Серед них значну частку займають підприємства у сфері інформаційних технологій – розробники програмного забезпечення, веб-студії, digital-агентства, ІТ-консалтингові фірми та технологічні стартапи. Попри різну спрямованість, усі вони працюють в умовах жорсткої конкуренції та постійної необхідності скорочувати час виходу продукту на ринок.

Характерною рисою малого ІТ-бізнесу є проєктна форма організації роботи. Більшість замовлень виконується у форматі окремих проєктів з фіксованими строками, бюджетом та командою виконавців. Нерідко кілька таких проєктів ведуться паралельно, що потребує чіткого розподілу задач між співробітниками та постійного контролю за їх виконанням. При цьому компанії, як правило, не мають окремого менеджера проєктів – цю роль часто поєднує з іншими обов'язками технічний керівник або власник бізнесу.

Відмінністю малих підприємств від корпоративного сектору є обмеженість ресурсів: невеликий штат, бюджет на програмне забезпечення та відсутність часу на складні організаційні процеси. За даними OECP, цифровізація операційної діяльності малих підприємств безпосередньо впливає на їх продуктивність та конкурентоспроможність. Водночас впровадження громіздких корпоративних систем для невеликих команд економічно недоцільне й організаційно надмірне. Це формує попит на прості та доступні цифрові інструменти, орієнтовані саме на потреби малого бізнесу.

Окремої уваги заслуговує поширення дистанційного та гібридного формату роботи. Члени команди можуть перебувати в різних містах або навіть країнах, тому інструменти для обміну інформацією, спільного доступу до задач і відстеження

прогресу набувають критичного значення. Відсутність спільного цифрового середовища в таких умовах призводить до дублювання роботи, порушення строків і втрати даних.

1.2 Особливості управління IT-проєктами

IT-проєкт – це обмежена в часі та ресурсах діяльність, результатом якої є програмний продукт, інформаційна система або цифровий сервіс. Відповідно до PMBOK Guide , управління проєктом охоплює планування, виконання, контроль і завершення робіт із метою досягнення визначеного результату в межах встановленого бюджету та строків.

На відміну від більшості інших галузей, IT-проєкти характеризуються підвищеним рівнем невизначеності. Вимоги замовника можуть змінюватися у процесі розробки, технічні рішення переглядаються після перших прототипів, а строки нерідко корегуються через технічні складнощі. Підходи до управління, що добре працюють у будівництві чи виробництві, у сфері IT часто виявляються негнучкими та неефективними.

Практика управління IT-проєктами зазнала суттєвих змін із поширенням гнучких методологій. Scrum, Kanban та інші Agile-підходи орієнтовані на ітераційну розробку, швидке отримання зворотного зв'язку та постійне вдосконалення продукту. Ці методології особливо ефективні для малих команд, де важливо швидко реагувати на зміни без зайвої бюрократії.

Ключовим чинником успіху проєкту є злагодженість команди. У типовому IT-проєкті беруть участь розробники, тестувальники, дизайнери та аналітики, і кожен з них повинен розуміти поточний стан загальної роботи. За відсутності централізованого інструменту відстеження задач координація часто відбувається через месенджери чи електронну пошту, що призводить до розпорошення інформації та складнощів із контролем строків. Саме тому наявність системи управління задачами є одним із базових організаційних інструментів для будь-якої IT-команди.

1.3 Аналіз сучасних систем управління проєктами

Ринок програмних засобів для управління проєктами є достатньо насиченим. Найбільш поширені рішення – Trello, Jira, Asana, Monday.com та ClickUp – мають широку аудиторію і закривають значну частину потреб командної роботи. Проте кожен із цих продуктів орієнтований на певний тип користувача і має характерні обмеження.

Trello побудований на принципі Kanban-дошок і підходить для невеликих команд з простими процесами. Інтерфейс мінімалістичний і не потребує навчання, однак функціонал обмежений: немає вбудованої звітності, залежностей між задачами та розгорнутого управління ресурсами. Для проєктів із розгалуженою структурою цього недостатньо.

Jira розроблена переважно для команд розробників і підтримує Scrum-методологію, баг-трекінг та аналітику спринтів. Платформа має широкі можливості налаштування, але вимагає часу на початкове конфігурування та певного рівня технічної підготовки адміністратора системи. Для невеликих компаній без виділеного IT-спеціаліста це може стати бар'єром для впровадження.

Asana зручна для планування командної роботи: підтримує відображення задач у вигляді списків, дошок і часових шкал. Разом з тим частина корисних функцій – зокрема, розширена звітність і автоматизація – доступна лише в платних тарифах, що суттєво обмежує безкоштовний варіант для малого бізнесу.

Monday.com орієнтована на автоматизацію бізнес-процесів і відрізняється гнучким налаштуванням робочих просторів . Однак вартість підписки є однією з найвищих серед конкурентів, а структура тарифів передбачає оплату за кожного користувача, що робить платформу економічно невигідною для невеликих команд.

ClickUp позиціонується як комплексний інструмент, що об'єднує задачі, документи, цілі та звітність в єдиному середовищі . Функціональність платформи є однією з найширших на ринку, проте великий обсяг налаштувань ускладнює початкове освоєння і може сповільнити адаптацію нових користувачів.

Порівняльна характеристика розглянутих систем наведена у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика сучасних систем управління проєктами

Система	Переваги	Недоліки	Доцільність для малого бізнесу
Trello	Простота використання, швидке впровадження	Обмежений функціонал	Висока
Jira	Потужні інструменти для IT-команд	Складність налаштування	Середня
Asana	Зручний інтерфейс, хороша організація роботи	Частина функцій платна	Висока
Monday.com	Автоматизація процесів, інтеграції	Висока вартість	Середня
ClickUp	Багатофункціональність	Перевантажений інтерфейс	Середня

1.4 Недоліки існуючих рішень для малого бізнесу

Незважаючи на широкий вибір інструментів, жоден із розглянутих продуктів не є оптимальним варіантом для невеликої IT-команди. Практичне застосування цих систем у контексті малого бізнесу виявляє кілька системних проблем.

Першою з них є вартість. Більшість платформ працює за передплатною моделлю з оплатою за кожного користувача. Безкоштовні тарифи суттєво обмежені за функціональністю: у Trello недоступна автоматизація, у Asana – звітність, у ClickUp – ряд інтеграцій. Для команди з 5–10 осіб витрати на корпоративні тарифи можуть становити від кількох сотень до кількох тисяч доларів на рік, що є відчутним навантаженням для малого підприємства.

Другою проблемою є надмірна складність інтерфейсу. Системи, орієнтовані на середній і великий бізнес, містять велику кількість налаштувань, що не лише ускладнює початкове освоєння, але й сповільнює щоденну роботу. Дослідження OECP щодо цифровізації малих підприємств підтверджує, що складність

інструментів є одним із ключових бар'єрів для їх впровадження .

Третій аспект стосується зберігання даних. Усі розглянуті системи є хмарними сервісами, тобто дані проєктів і клієнтів зберігаються на зовнішніх серверах. Для частини підприємств це породжує питання щодо конфіденційності комерційної інформації та залежності від стабільності постачальника послуги. Припинення роботи сервісу або зміна умов тарифікації можуть безпосередньо вплинути на доступ до робочих даних компанії.

Нарешті, більшість платформ не мають повноцінної локалізації для українського ринку. Інтерфейс та документація доступні переважно англійською, а технічна підтримка не завжди враховує специфіку роботи підприємств у вітчизняних правових та бізнес-умовах.

1.5 Постановка задачі розроблення інформаційної системи

Проведений аналіз підтверджує, що наявні рішення не повністю відповідають потребам малого IT-підприємства: вони або надто дорогі, або надмірно складні, або не враховують специфіку невеликих команд. Це обґрунтовує розроблення власної інформаційної системи, спрямованої на практичні задачі управління проєктами без зайвої функціональності та адміністративного навантаження.

Метою розроблення є створення веб-орієнтованої інформаційної системи управління IT-проєктами малого бізнесу, яка дозволяє організувати командну роботу, відстежувати стан задач і зберігати актуальну інформацію про хід проєктів.

Система призначена для трьох категорій користувачів. Керівник проєкту створює та редагує проєкти, формує перелік задач, призначає виконавців і контролює строки. Виконавець переглядає призначені йому завдання, оновлює їх статус і додає коментарі. Адміністратор управляє обліковими записами користувачів та здійснює загальний нагляд за роботою системи.

Система повинна реалізовувати такі функції: реєстрацію та авторизацію користувачів із розмежуванням доступу за ролями; створення, редагування і

видалення проєктів; управління задачами в межах проєкту – з визначенням строків, пріоритетів і виконавців; зміну статусів задач відповідно до етапів виконання; додавання коментарів до задач; відображення зведеної статистики на головній сторінці системи.

До системи висуваються такі нефункціональні вимоги: простий і зрозумілий інтерфейс без потреби у навчанні; швидкодія при звичайному навантаженні команди; захист даних через хешування паролів і токенову авторизацію; доступність через браузер без встановлення додаткового програмного забезпечення; можливість подальшого розширення функціоналу.

Реалізація системи у вигляді веб-застосунку з REST API відповідає трирівневій клієнт-серверній архітектурі і дозволяє забезпечити доступ до системи з будь-якого пристрою. Визначені вимоги є основою для подальшого проєктування структури системи, вибору технологій і розроблення бази даних.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Визначення вимог до інформаційної системи

Розробка будь-якої інформаційної системи починається з чіткого визначення вимог, оскільки саме вони формують основу подальшого проєктування, реалізації та тестування програмного продукту. Згідно з рекомендаціями PMBOK Guide (7th Edition, 2021), вимоги до системи являють собою задокументовані потреби замовника та користувачів, які повинні бути реалізовані у кінцевому продукті. У контексті розробки інформаційної системи управління IT-проєктами для малого бізнесу вимоги поділяються на дві ключові групи – функціональні (що саме система має виконувати) та нефункціональні (як саме вона повинна працювати). Правильно сформульовані вимоги дозволяють уникнути надлишкової функціональності, знизити витрати на розробку та забезпечити відповідність системи реальним потребам невеликих IT-команд.

Функціональні вимоги описують конкретний набір операцій, які користувач повинен мати змогу виконувати у системі. Для досліджуваної предметної галузі визначено такі основні функціональні можливості:

- реєстрація нових користувачів та авторизація у системі за допомогою електронної пошти й пароля з підтримкою ролей (керівник проєкту, виконавець);
- створення проєктів із вказанням назви, опису, дати початку та запланованого завершення;
- управління завданнями у межах проєкту — додавання, редагування та видалення задач;
- призначення виконавців на конкретні завдання з можливістю перепризначення;
- зміна статусів завдань («Нове», «У роботі», «На перевірці», «Завершено»)

для відображення поточного етапу виконання;

- встановлення дедлайнів для завдань та автоматичне сповіщення про їх наближення;
- перегляд списку завдань і проєктів із можливістю фільтрації за виконавцем, статусом, пріоритетом та терміном виконання;
- формування базової звітності щодо стану проєкту, завантаженості працівників та обсягу виконаних робіт.

Нефункціональні вимоги визначають якісні характеристики системи, які безпосередньо впливають на досвід її використання. Для інформаційної системи малого ІТ-підприємства встановлено такі вимоги:

- зручність використання – інтуїтивно зрозумілий інтерфейс, що не потребує тривалого навчання, оскільки малі компанії зазвичай не мають ресурсів на тренінги персоналу;
- швидкодія – час відгуку основних операцій не повинен перевищувати 2–3 секунд за умов одночасної роботи до 50 користувачів;
- надійність – забезпечення стабільної роботи системи з мінімальним відсотком збоїв та можливістю автоматичного резервного копіювання даних;
- безпека – захист персональних даних користувачів і комерційної інформації за допомогою шифрування паролів (хешування), HTTPS-з’єднання та розмежування прав доступу за ролями;
- доступність – реалізація системи як веб-застосунку, що працює у сучасних браузерах без потреби встановлення додаткового програмного забезпечення на робочих станціях;
- масштабованість – архітектурна можливість збільшення кількості користувачів, проєктів і завдань без суттєвого зниження продуктивності, що особливо актуально для компаній, які активно розвиваються.

Окреслений перелік функціональних і нефункціональних вимог відображає реальні потреби малого IT-бізнесу та враховує обмеженість його технічних і фінансових ресурсів. На відміну від громіздких комерційних рішень, проєктована система зосереджується на ключовому функціоналі, необхідному для ефективного управління проєктами, що дозволяє забезпечити її практичну корисність та простоту впровадження. Сформовані вимоги слугуватимуть основою для подальшого вибору архітектурного рішення, технологій реалізації та проєктування структури бази даних, які будуть розглянуті у наступних підрозділах.

2.2 Моделювання системи

Прецедент являє собою певну послідовність дій, які виконує система у відповідь на запит зовнішнього користувача, унаслідок чого користувач отримує конкретний результат, що має для нього цінність. Окремими прецедентами можуть бути, наприклад, авторизація, створення задачі або формування звіту. Сукупність усіх прецедентів описує функціональну поведінку всієї системи. Перевага такого підходу в тому, що отримана модель є зрозумілою не лише розробникам, а й замовнику, який не має технічної освіти. У малому IT-бізнесі це особливо важливо, оскільки замовником системи та її основним користувачем нерідко є одна й та сама людина – наприклад, керівник компанії.

Побудова Use Case моделі починається з виділення акторів. Актор – це роль, у якій певний користувач або зовнішня система взаємодіє з продуктом. Одну й ту саму роль може виконувати кілька осіб, і навпаки – одна особа в різний час може виступати у різних ролях. На основі аналізу команди малого IT-підприємства, проведеного у першому розділі, для проєктованої системи виділено трьох акторів.

Керівник проєкту є основним користувачем системи. До його обов’язків належить планування роботи команди, тому в системі саме він створює нові проєкти, формує перелік задач, призначає виконавців, контролює терміни

виконання та отримує звіти про результати. У невеликих компаніях цю роль зазвичай виконує власник бізнесу або проєктний менеджер. Особливість його роботи у системі полягає в тому, що йому потрібен як докладний перегляд окремих задач, так і загальне уявлення про стан портфеля проєктів у цілому.

Виконавець – це член проєктної команди, який безпосередньо працює над поставленими задачами. Це можуть бути розробники, тестувальники, дизайнери або аналітики. На відміну від керівника, виконавець у системі працює переважно зі своїми власними задачами: переглядає їх перелік, вивчає опис, оновлює статус і фіксує результати. Доступ до зміни чужих задач чи редагування структури проєкту у виконавця обмежений, що відповідає принципу розмежування прав доступу за ролями.

Адміністратор у системі виконує переважно технічні функції. Він користується нею не щодня, а лише тоді, коли потрібно зареєструвати нового співробітника, заблокувати обліковий запис, скинути пароль або змінити роль користувача. Незважаючи на епізодичність використання, ця роль необхідна для забезпечення безпеки даних і коректної роботи системи у тривалій перспективі. У дуже малих компаніях обов’язки адміністратора може виконувати сам керівник, але у моделі ці ролі все одно розмежовано – щоб у майбутньому можна було розширити структуру користувачів без переробки системи.

Функціональність проєктованої системи описана через вісім ключових прецедентів. Далі наведено короткий опис кожного з них.

Вхід у систему. Прецедент є спільним для всіх акторів, оскільки без авторизації жодна інша функція системи недоступна. Користувач відкриває сторінку входу, вводить електронну пошту та пароль, після чого система перевіряє правильність даних. У разі успіху завантажується інтерфейс відповідно до ролі користувача, у разі помилки – відображається відповідне повідомлення. Для запобігання підбору пароля передбачено обмеження кількості невдалих спроб поспіль.

Створення проєкту. Прецедент доступний тільки керівникові. Він заповнює форму нового проєкту, в якій зазначає його назву, опис, заплановані дати початку

та завершення, а за потреби – додаткові категорії або теги. Після збереження проєкт з’являється у списку активних, прив’язується до автора й стає доступним для подальшого наповнення задачами. По суті, цей прецедент є відправною точкою всієї роботи команди над відповідним напрямом.

Створення задачі. Сценарій виконується керівником у межах раніше створеного проєкту. У формі нової задачі він вводить її назву, опис, обирає рівень пріоритету та встановлює дедлайн. З цим прецедентом тісно пов’язаний наступний – призначення виконавця, тому між ними у моделі встановлено відношення типу «include». Воно означає, що другий сценарій є обов’язковою частиною першого: задача без визначеного відповідального не може повноцінно потрапити в роботу.

Призначення виконавця. У межах цього прецеденту керівник обирає одного або кількох членів команди та пов’язує їх із конкретною задачею. Система автоматично сповіщає виконавця про нове призначення, а задача потрапляє до його особистого переліку. Якщо у процесі роботи з’являється потреба у перерозподілі навантаження, керівник може у будь-який момент змінити виконавця.

Зміна статусу задачі. Один із найчастіших сценаріїв, бо саме він відображає прогрес виконання робіт. Виконавець знаходить задачу у своєму списку та поступово переводить її зі стану «Нове» у «У роботі», далі – у «На перевірці», і, нарешті, у «Завершено». Кожна зміна фіксується з відміткою про час і автора, що дозволяє керівникові бачити динаміку роботи. Сам керівник також має змогу змінювати статус – наприклад, повертати задачу на доопрацювання після перевірки.

Перегляд задач. Прецедент доступний усім авторизованим користувачам, але обсяг видимих даних залежить від ролі. Керівник бачить усі задачі обраного проєкту, а виконавець – лише ті, де він безпосередньо задіяний. Для зручної навігації передбачено фільтри за статусом, виконавцем, проєктом, пріоритетом, датою створення або наближенням дедлайну. Така функціональність є корисною тоді, коли загальна кількість активних задач велика, а часу на ручний пошук обмаль.

Формування звіту. Прецедент призначений переважно для керівника, який

періодично потребує узагальненої інформації про стан виконання робіт. Користувач обирає тип звіту (наприклад, «Прогрес проєкту», «Завантаженість команди», «Дотримання дедлайнів») та період, після чого система обробляє дані з бази й виводить результат у вигляді таблиці або діаграми. Готовий звіт можна зберегти у файл, щоб згодом надіслати замовнику або використати під час робочих обговорень.

Управління обліковими записами. Сценарій виконується тільки адміністратором. Він охоплює реєстрацію нових співробітників, призначення або зміну їхньої ролі, тимчасове блокування доступу та скидання паролів. Цей прецедент використовується рідко, але він є критично важливим з точки зору безпеки даних та дисципліни в команді.

Усі описані прецеденти поєднані з акторами лініями асоціації, які відображають доступний для кожної ролі набір операцій. Окремо вже згадано відношення «include» між створенням задачі та призначенням виконавця: воно показує, що другий сценарій логічно є частиною першого. У загальному вигляді розподіл взаємодії має наступний характер. Керівник проєкту керує системою «згори»: створює проєкти й задачі, призначає виконавців, контролює терміни та формує звіти. Виконавець працює переважно зі своїми задачами та оновлює їх стан. Адміністратор підтримує систему «зовні» – не втручається у бізнес-логіку, але забезпечує умови для коректної роботи інших користувачів.

Такий розподіл повноважень відповідає реальній організації роботи у малих IT-командах, де відсутні складні рівні підпорядкування, а кожна зайва дія у системі коштує часу. Спрощення взаємодії та зосередження на ключових сценаріях якраз і є однією з переваг проєктованого рішення порівняно з громіздкими комерційними платформами, що були розглянуті у першому розділі.

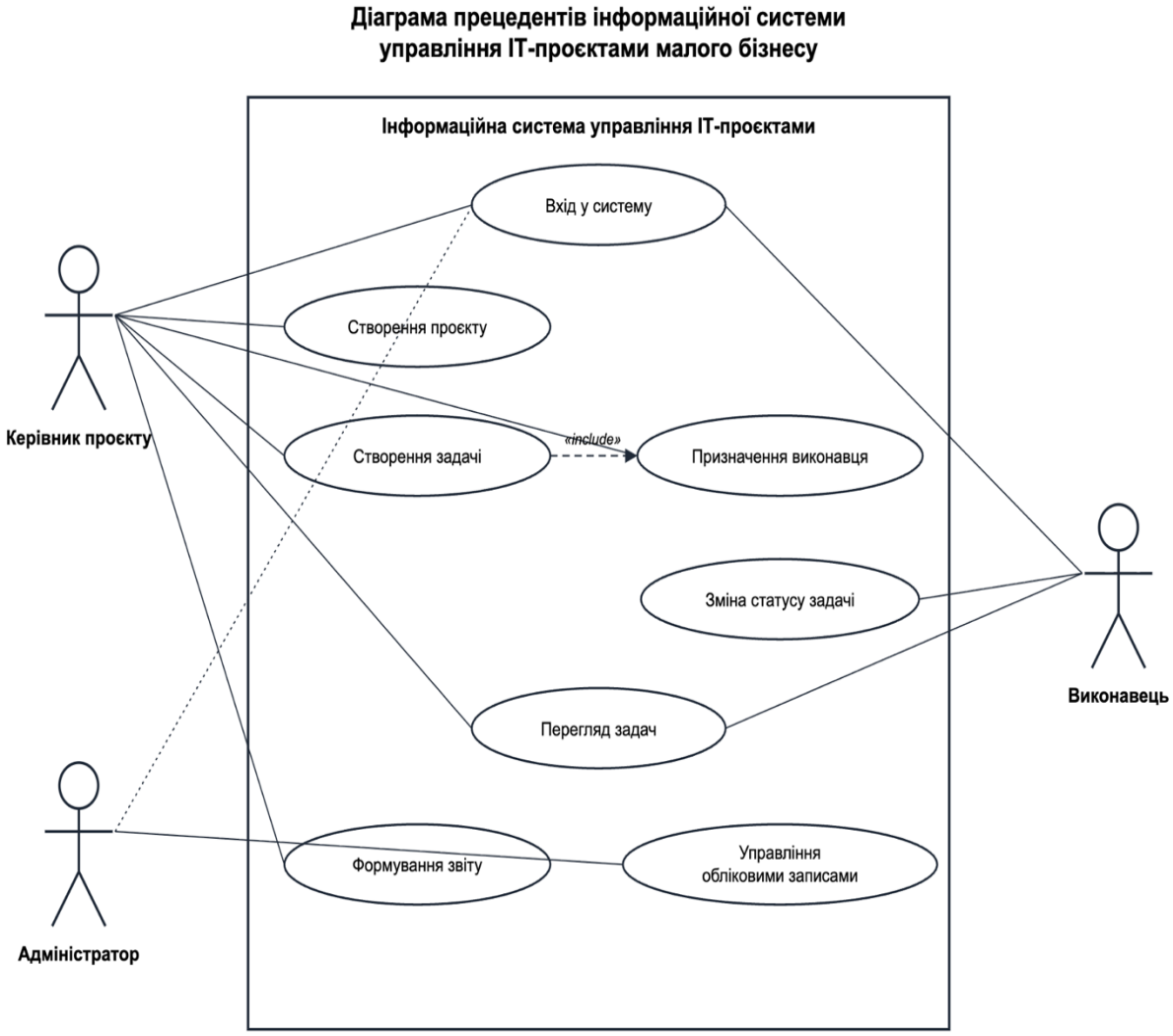


Рис. 2.1 – Діаграма прецедентів інформаційної системи управління IT-проєктами малого бізнесу

У межах цього підрозділу побудовано модель прецедентів інформаційної системи управління IT-проєктами малого бізнесу. Виділено трьох акторів – керівника проєкту, виконавця та адміністратора – і вісім прецедентів, що охоплюють життєвий цикл роботи з системою: від входу до формування звітів. Окрему увагу приділено зв’язку «include» між створенням задачі та призначенням виконавця, який відображає логічну цілісність цих сценаріїв. Побудована модель дозволяє перейти від загального переліку вимог до конкретного уявлення про поведінку системи й слугує основою для подальшого проєктування її архітектури та структури бази даних, чому буде присвячено наступні підрозділи.

2.3 Проєктування архітектури інформаційної системи

Після того як було визначено вимоги до системи та побудовано модель її поведінки з боку користувачів, наступним логічним кроком проєктування є вибір архітектури – тобто загальної структури майбутнього програмного продукту. Архітектура інформаційної системи описує її основні складові частини, способи їх взаємодії, розподіл функцій між компонентами та підходи до зберігання даних. По суті, це той «скелет», який визначає, як саме реалізовуватимуться раніше описані вимоги. Правильно обрана архітектура спрощує подальшу розробку, забезпечує можливість супроводу та доопрацювання системи у майбутньому, тоді як помилки на цьому етапі коштують найдорожче – виправити їх після написання значної частини коду буває вкрай складно.

В інженерії програмного забезпечення існує велика кількість архітектурних шаблонів – від простих монолітних рішень до складних мікро-сервісних систем. Вибір конкретного варіанту залежить від низки факторів: розміру проєкту, очікуваної кількості користувачів, специфіки задач, бюджету та кваліфікації команди. Для невеликого продукту, орієнтованого на потреби малого IT-бізнесу, доцільно обирати таку архітектуру, яка, з одного боку, забезпечуватиме достатню гнучкість та зрозумілий поділ компонентів, а з іншого – не вимагатиме значних витрат на розгортання та супровід. Виходячи з цих міркувань, для проєктованої інформаційної системи управління IT- проєктами обрано трирівневу клієнт-серверну архітектуру.

Її головна ідея полягає у поділі системи на три самостійні рівні: рівень представлення (клієнтська частина, або frontend), рівень бізнес-логіки (серверна частина, backend) і рівень зберігання даних (база даних). Кожен із рівнів відповідає за чітко окреслене коло задач, а взаємодія між ними здійснюється за допомогою стандартизованих протоколів. Такий поділ є логічним продовженням принципів, закладених у Use Case моделі, оскільки кожна група прецедентів реалізується у

відповідному шарі архітектури.

Клієнтська частина (frontend)

Клієнтська частина – це та частина системи, з якою користувач взаємодіє безпосередньо. Вона працює у браузері на комп'ютері або мобільному пристрої та відповідає за відображення інтерфейсу, прийом введених даних і передачу їх на сервер. На цьому рівні реалізуються форми реєстрації та входу, екран зі списком проєктів, інтерфейс для створення та редагування задач, фільтри для перегляду й візуалізація звітів у вигляді таблиць та простих графіків. Усе, що бачить користувач, – це результат роботи саме frontend-частини.

Важливо, що сам клієнт майже не зберігає бізнес-даних у себе: уся необхідна інформація приходить із сервера у момент її потреби. Це робить систему доступною з будь-якого пристрою, обладнаного браузером і підключенням до інтернету, без необхідності встановлення додаткового програмного забезпечення на робочих станціях співробітників. Такий підхід безпосередньо відповідає вимогам доступності, сформульованій у підрозділі 2.1.

Серверна частина (backend)

Серверна частина виконує роль «мозку» системи. Саме на ній зосереджена основна бізнес-логіка – правила, за якими обробляються запити користувачів. Коли користувач, наприклад, натискає кнопку «створити задачу», клієнтська частина формує запит до сервера, а вже сервер приймає рішення про те, чи правильно заповнені поля, чи має користувач відповідні права, і у разі позитивної відповіді записує нові дані у базу. Окрім цього, на серверному рівні реалізуються перевірка облікових записів під час входу, контроль доступу до різних розділів системи, обробка фільтрації та сортування задач, формування звітів та інші операції.

Зв’язок між клієнтською й серверною частинами організовано за принципами REST API – набору маршрутів, за якими frontend може звертатися до backend для виконання типових операцій: отримання списку задач, створення нової задачі, оновлення статусу, видалення проєкту тощо. Використання REST- підходу є стандартом для сучасних веб-додатків і дозволяє чітко розмежувати клієнта й сервер. У майбутньому це створює зручні умови для розширення системи – наприклад, для розробки мобільного клієнта, який зможе взаємодіяти з тим самим сервером без жодних змін у серверному коді.

Рівень зберігання даних

База даних є третім рівнем архітектури й відповідає за тривале зберігання інформації. У ній зберігаються облікові записи користувачів, перелік усіх проєктів, задачі з усіма їх параметрами (назва, опис, дедлайн, статус, пріоритет, виконавець), історія змін статусів та інші дані, які повинні бути доступними між сеансами роботи з системою. База даних безпосередньо не «спілкується» з користувачем – усі звернення до неї виконує серверна частина, яка виступає у ролі посередника. Завдяки такому розмежуванню вдається уникнути прямого доступу клієнта до сховища, що значно підвищує загальний рівень безпеки системи. Користувач не може ані змінити структуру таблиць або колекцій, ані обійти правила доступу, ані виконати запити в обхід бізнес-логіки. Це особливо важливо для систем, які працюють із комерційною інформацією компаній, де навіть випадкове розголошення даних може мати суттєві наслідки для бізнесу.

Взаємодія між компонентами

Взаємодія між компонентами у трирівневій моделі має чітку послідовність. Усе починається з дії користувача в інтерфейсі: він натискає кнопку, заповнює форму або переходить за посиланням. Клієнтська частина обробляє цю дію і формує HTTP-запит до сервера. Запит надходить на відповідний маршрут REST

API, де серверна частина його приймає, перевіряє автентифікацію та права користувача, виконує необхідну логіку й, за потреби, звертається до бази даних – для зчитування або зміни інформації. База повертає результат серверу, сервер формує відповідь у форматі JSON та надсилає її клієнту, а той уже відображає її користувачеві у зрозумілому вигляді. Таким чином, повний цикл «користувач → frontend → backend → база даних → backend → frontend → користувач» проходить за частки секунди й залишається непомітним для самої людини, що працює із системою. Графічно ця схема представлена на рис. 2.2.

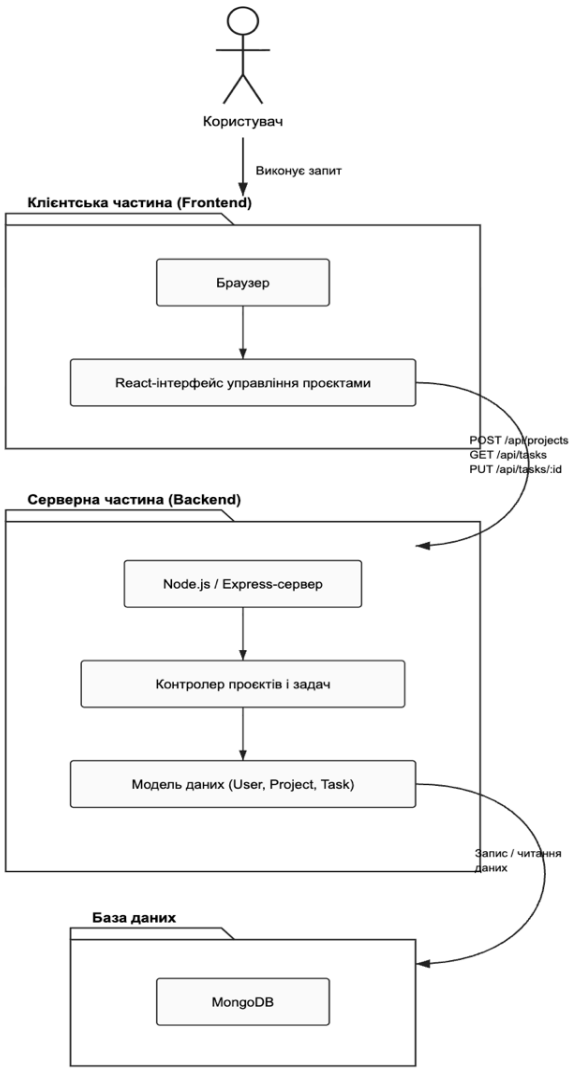


Рис. 2.2 – Архітектура інформаційної системи управління ІТ-проєктами малого бізнесу

Обґрунтування вибору архітектури

Доцільність застосування саме трирівневої клієнт-серверної архітектури тісно пов’язана з особливостями малого бізнесу, описаними у першому розділі. Невеликі ІТ-компанії, як правило, мають обмежений бюджет, малу кількість штатних розробників і потребують швидкого впровадження інформаційних систем. У такому контексті обрана архітектура має одразу декілька суттєвих переваг.

По-перше, реалізація продукту як веб-системи звільняє користувачів від необхідності встановлювати застосунки на кожен робочий пристрій – достатньо мати браузер та доступ до мережі. Це відповідає вимогам доступності й істотно знижує витрати на технічну підтримку. По-друге, чіткий поділ на три рівні дозволяє паралельно вести розробку різними фахівцями: один займається інтерфейсом, інший – серверною логікою, а робота з даними виноситься в окремий шар. Завдяки цьому скорочується загальний час реалізації проєкту, що особливо важливо для невеликих команд із обмеженими ресурсами. По-третє, така архітектура добре масштабується: коли кількість користувачів зросте, сервер можна перенести на більш потужну машину, винести базу даних на окремий вузол або додати кешування – і це не потребуватиме переробки усього продукту.

Не менш важливим є те, що трирівнева клієнт-серверна архітектура є типовою для сучасної веб-розробки. Існує величезна кількість готових інструментів, бібліотек і документації, що значно спрощує реалізацію та зменшує її вартість. Для замовника з категорії малого бізнесу це означає коротші терміни розробки й нижчий поріг входу при подальшому супроводі системи власними силами або силами фрилансерів.

2.4 Проєктування бази даних інформаційної системи

В архітектурі, описаній у попередньому підрозділі, окремий рівень відведено для зберігання даних. Він є фундаментом усієї системи: саме у базі даних накопичується інформація про проєкти, задачі, виконавців і результати їхньої роботи, причому ця інформація має бути доступною не лише під час однієї сесії, а й через місяці та роки після створення. Якщо клієнтську й серверну частини за потреби можна переписати, оновити чи замінити іншою технологією, то втрата даних або їх пошкодження здатні зупинити роботу компанії повністю.

Саме тому етап проєктування бази даних є одним з найвідповідальніших у процесі створення інформаційної системи. На цьому етапі визначається, які саме сутності будуть зберігатися, які поля кожна з них матиме, як ці сутності пов’язані між собою та яких обмежень слід дотримуватись для забезпечення цілісності інформації. Помилка, допущена у структурі бази, у подальшому призводить до серйозних проблем: дублювання записів, неузгодженості, ускладнення запитів та зниження продуктивності. Тому до проєктування бази даних доцільно підходити максимально системно, спираючись на вимоги та сценарії, описані у підрозділах 2.1 та 2.2.

Для проєктованої системи обрано реляційну модель бази даних. У цій моделі дані подаються у вигляді таблиць, кожна з яких відповідає окремій сутності реального світу – користувачеві, проєктові, задачі тощо. Рядки таблиці є окремими записами (екземплярами сутності), а стовпці – атрибутами, що описують характеристики цих записів. Для однозначної ідентифікації кожного запису використовується первинний ключ (Primary Key), а для встановлення зв’язків між таблицями – зовнішні ключі (Foreign Key), які посилаються на первинні ключі інших таблиць.

Вибір саме реляційної моделі обумовлений декількома міркуваннями. По-перше, дані, з якими працює інформаційна система управління проєктами, мають чітку структуру: задача завжди належить певному проєктові, проєкт – певному користувачеві, коментар – певній задачі. Така структурованість дуже добре лягає у табличну форму. По-друге, реляційні бази даних забезпечують високу цілісність інформації за рахунок системи ключів та обмежень, що практично виключає появу «осиротілих» записів або суперечливих значень. По- третє, мова SQL, якою працюють усі реляційні СУБД, є зрозумілою і добре документованою, що скорочує час як на розробку, так і на подальший супровід системи невеликою командою. Для практичної реалізації може бути використано PostgreSQL, MySQL або інший доступний реляційний рушій – на функціональну структуру даних це не вплине.

Основні сутності бази даних

Аналіз вимог та сценаріїв роботи дозволив виділити п’ять основних сутностей, які повністю покривають предметну область системи управління IT-проєктами малого бізнесу.

Користувач (User) представляє у системі реальну людину – працівника компанії, який має обліковий запис. У цій сутності зберігається базова інформація про користувача (ім’я, електронна пошта, пароль у зашифрованому вигляді) та його роль (керівник проєкту, виконавець або адміністратор). Без сутності користувача неможливо реалізувати ані авторизацію, ані призначення задач, ані фіксацію авторства коментарів.

Проєкт (Project) – це окремий напрям роботи команди, у межах якого формується перелік завдань. Сутність зберігає назву проєкту, опис, заплановані дати початку й завершення, а також посилання на користувача, що є його ініціатором (керівником). Один користувач може створити кілька проєктів, тоді як кожен проєкт має лише одного відповідального.

Завдання (Task) є основною робочою одиницею у системі. Саме навколо задач будується вся щоденна діяльність команди. Сутність містить назву задачі, її опис, пріоритет, дедлайн, а також посилання на проєкт, у межах якого вона виконується, виконавця та поточний статус. Будь-яка задача неможлива без прив’язки до конкретного проєкту, а статус і виконавець можуть змінюватися протягом її життєвого циклу.

Статус (Status) є допоміжною сутністю, що описує поточний стан задачі (наприклад, «Нове», «У роботі», «На перевірці», «Завершено»). Винесення статусу в окрему таблицю замість того, щоб зберігати його як текстове поле всередині задачі, дозволяє у майбутньому без значних змін додавати нові стани або змінювати назви існуючих, не торкаючись основної логіки системи.

Коментар (Comment) призначений для організації обговорень навколо конкретної задачі. У ньому зберігається текст повідомлення, посилання на задачу, до якої він належить, автор коментаря та дата створення. Це дозволяє вести історію обговорень безпосередньо у системі, не вдаючись до сторонніх месенджерів, що відповідає прагненню зосередити робочу комунікацію в одному місці.

Узагальнений перелік сутностей наведено у таблиці 2.1.

Таблиця 2.1 – Основні сутності бази даних

Сутність	Опис	Призначення
Користувач (User)	Запис про працівника, який має обліковий запис у системі	Авторизація, призначення задач, фіксація авторства дій
Проєкт (Project)	Запис про окремий проєкт компанії	Об’єднання задач у спільний напрям роботи
Завдання (Task)	Окрема робоча задача у межах проєкту	Розподіл роботи, контроль виконання, фіксація прогресу

Статус (Status)	Стан, у якому перебуває задача	Відображення прогресу та фільтрація задач за станом
Коментар (Comment)	Повідомлення, прив’язане до конкретної задачі	Організація обговорень і ведення історії взаємодії

На основі описаних сутностей побудовано конкретні таблиці бази даних. Для кожної визначено перелік полів, їхні типи та коротке пояснення призначення. У стовпці «Тип даних» використано спрощені позначення (integer, varchar, text, date), які надалі під час реалізації можна замінити на конкретні типи обраної СУБД.

Таблиця користувачів зберігає інформацію про всі облікові записи системи.

Таблиця 2.2 – Структура таблиці «Користувач»

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор користувача (первинний ключ)
name	varchar	Ім’я та прізвище користувача
email	varchar	Електронна адреса (використовується для входу, унікальна)
password	varchar	Хеш пароля користувача
role	varchar	Роль у системі (керівник, виконавець, адміністратор)
created_at	date	Дата створення облікового запису

Таблиця проєктів містить дані про всі активні та завершені проєкти компанії.

Таблиця 2.3 – Структура таблиці «Проєкт»

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор проєкту (первинний ключ)

name	varchar	Назва проєкту
description	text	Опис проєкту
start_date	date	Запланована дата початку проєкту
end_date	date	Запланована дата завершення проєкту
user_id	integer	Зовнішній ключ на користувача-керівника

Центральним об’єктом моделі є таблиця завдань, що містить відомості про окремі задачі у межах проєктів.

Таблиця 2.4 – Структура таблиці «Завдання»

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор завдання (первинний ключ)
project_id	integer	Зовнішній ключ на таблицю проєктів
user_id	integer	Зовнішній ключ на виконавця завдання
status_id	integer	Зовнішній ключ на таблицю статусів
title	varchar	Коротка назва завдання
description	text	Детальний опис завдання
priority	varchar	Пріоритет завдання (низький, середній, високий)
deadline	date	Кінцевий термін виконання
created_at	date	Дата створення завдання

Таблиця коментарів забезпечує організацію обговорень навколо задач.

Таблиця 2.5 – Структура таблиці «Коментар»

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор коментаря (первинний ключ)
task_id	integer	Зовнішній ключ на завдання, до якого належить коментар

user_id	integer	Зовнішній ключ на автора коментаря
text	text	Текст повідомлення
created_at	date	Дата та час створення коментаря

Таблиця статусів містить перелік можливих станів задачі.

Таблиця 2.6 – Структура таблиці «Статус»

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор статусу (первинний ключ)
name	varchar	Назва статусу (Нове, У роботі, На перевірці, Завершено)

Окрім самих таблиць, важливою частиною проєктування є визначення зв’язків між ними. У реляційній моделі це робиться за допомогою зовнішніх ключів, які встановлюють логічні відповідності між записами різних таблиць. Для проєктованої бази характерні зв’язки типу «один-до-багатьох» (1:N), оскільки кожен ключовий об’єкт системи може бути пов’язаний з кількома підпорядкованими. Зв’язок Користувач – Проєкт має тип 1:N: один користувач (керівник) може створити багато проєктів, але кожен проєкт прив’язаний лише до одного відповідального користувача. Реалізується це через поле user_id у таблиці проєктів. Зв’язок Проєкт – Завдання також є 1:N: один проєкт містить багато завдань, тоді як кожне завдання належить тільки до одного проєкту. Це відповідає реальній організації роботи, коли діяльність компанії групується у вигляді портфеля проєктів. Зв’язок Користувач – Завдання має тип 1:N з боку виконавця: один користувач може мати призначеними багато завдань, але у поточній моделі за кожне завдання відповідає лише одна людина. У майбутньому, за потреби, цю модель можна розширити до зв’язку «багато-до-багатьох» через окрему таблицю, але для типового малого бізнесу варіант «один виконавець на задачу» є достатнім.

Зв’язок Статус – Завдання реалізує тип 1:N: один статус може використовуватись у багатьох задачах, тоді як кожна задача у конкретний момент

часу перебуває лише в одному стані. Завдяки винесенню статусів в окрему таблицю змінити їх перелік стає простіше – достатньо додати або відредагувати запис у довіднику.

Зв’язок Завдання – Коментар є 1:N: під кожною задачею може накопичуватися довільна кількість коментарів, але кожен коментар належить точно до однієї задачі. Окремо існує зв’язок Користувач – Коментар (також 1:N), який дозволяє визначити автора кожного повідомлення.

Для наочного представлення описаної моделі побудовано діаграму «сутність – зв’язок» (Entity-Relationship Diagram, ER-діаграма), що подана на рис. 2.3 Центральною сутністю на діаграмі виступає Проєкт, що відповідає його ключовій ролі у предметній області: саме навколо проєктів організована робота користувачів і формується перелік задач. Знизу від проєкту розташовано сутність Завдання, праворуч від неї – Статус, який задає поточний стан задачі, а ліворуч – Користувач, що виступає одночасно як ініціатор проєктів та виконавець завдань. Під сутністю Завдання розміщено Коментар, який групує обговорення під конкретною задачею.

Лінії між сутностями позначають зв’язки, а підписи «1» та «N» на їх кінцях вказують на кратність – скільки об’єктів однієї таблиці може бути пов’язано з об’єктами іншої. Така діаграма дозволяє у компактній формі побачити логіку всієї бази даних і слугує орієнтиром під час подальшого створення таблиць та написання SQL-запитів.

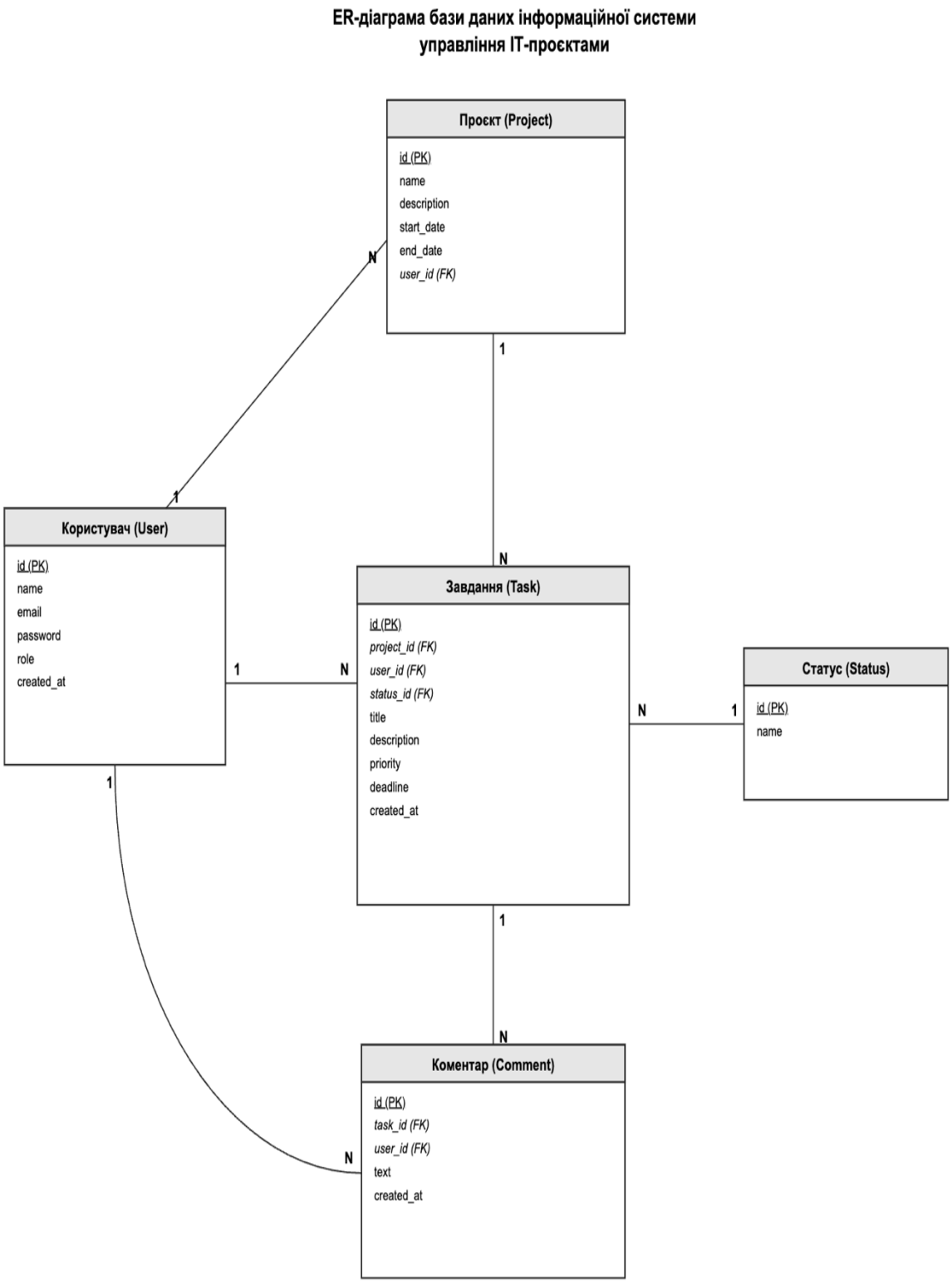


Рис. 2.3 – ER-діаграма

У підрозділі побудовано структуру реляційної бази даних для інформаційної системи управління ІТ-проєктами малого бізнесу. Виділено п’ять основних сутностей – користувач, проєкт, завдання, статус і коментар – і для кожної з них наведено перелік полів та їх призначення. Описано зв’язки між таблицями типу «один-до-багатьох», що формують цілісну модель предметної області, а також побудовано ER-діаграму, що дає наочне уявлення про структуру даних. На цьому проєктний етап роботи можна вважати завершеним: визначено вимоги, поведінкову модель, архітектуру та структуру бази даних. Це створює необхідне підґрунтя для переходу до програмної реалізації системи у наступному розділі роботи.

2.5 Проєктування інтерфейсу користувача

Інтерфейс користувача є важливою складовою інформаційної системи, оскільки саме через нього здійснюється взаємодія користувача з функціоналом системи. Від зручності та зрозумілості інтерфейсу залежить ефективність роботи користувачів, швидкість виконання завдань та загальне сприйняття програмного продукту. Під час проєктування інтерфейсу було враховано вимоги, визначені у підрозділі 2.1, зокрема простоту використання та доступність.

Дизайн інтерфейсу розроблено у середовищі Figma, що дозволяє створювати сучасні прототипи веб-застосунків та забезпечує зручність подальшого доопрацювання. Основною концепцією дизайну є мінімалізм, чітка структура та використання біло-синьої кольорової гами. Основний фон виконано у світлих тонах, тоді як синій колір використовується для акцентування кнопок, активних елементів та навігації.

На рисунку 2.4 представлено сторінку входу до системи. Вона містить форму авторизації з полями для введення електронної пошти та пароля, а також кнопку входу. Інтерфейс є максимально простим та зрозумілим для користувача.

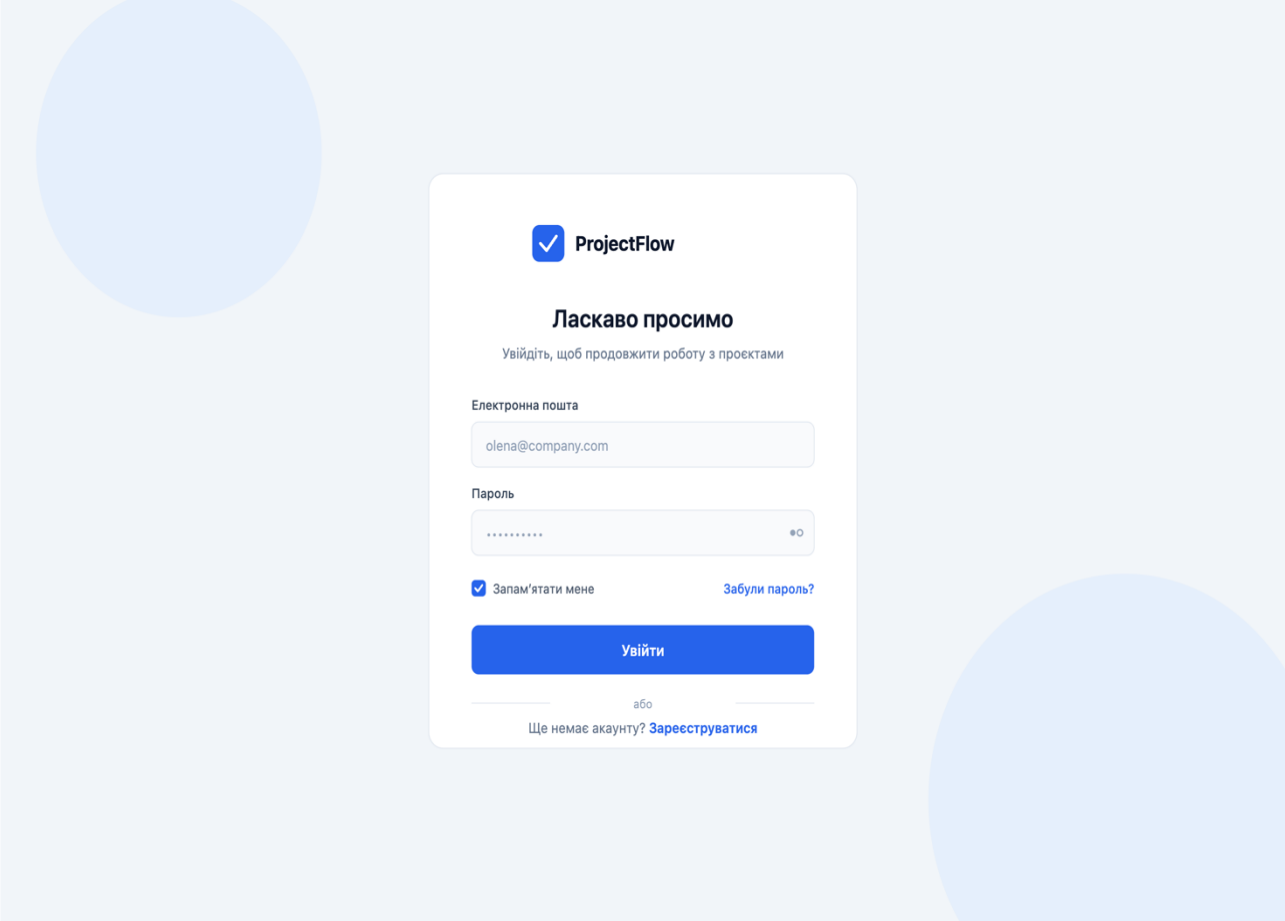


Рис. 2.4 – Макет сторінки входу в систему

Головна сторінка (рис. 2.5) реалізована у вигляді інформаційної панелі (dashboard), яка відображає ключові показники: кількість проєктів, активні завдання, завершені та прострочені задачі. Також представлено блок останньої активності та список найближчих дедлайнів, що дозволяє користувачу швидко оцінити поточний стан роботи.

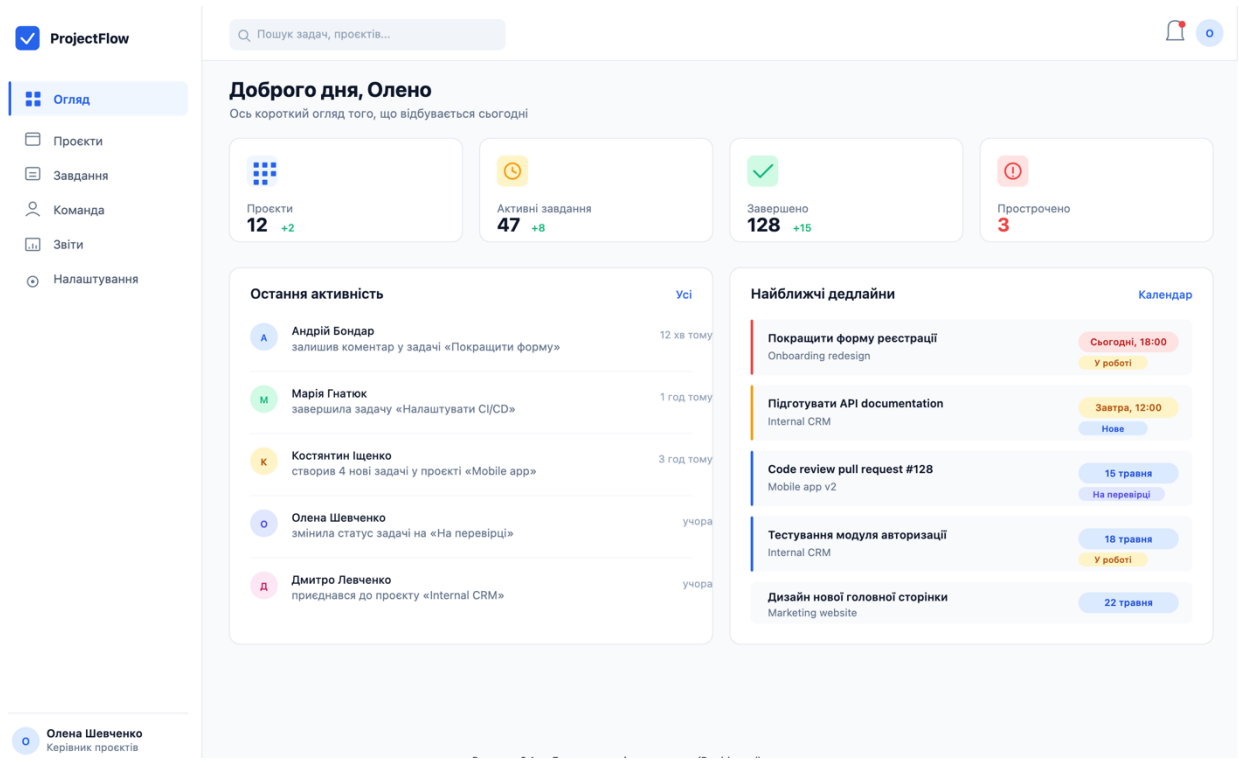


Рис. 2.5 – Макет головної сторінки системи

Сторінка проєктів (рис. 2.6) містить список усіх проєктів у вигляді карток із короткою інформацією: назва, опис, статус та прогрес виконання. Передбачено можливість створення нового проєкту та пошуку серед існуючих.

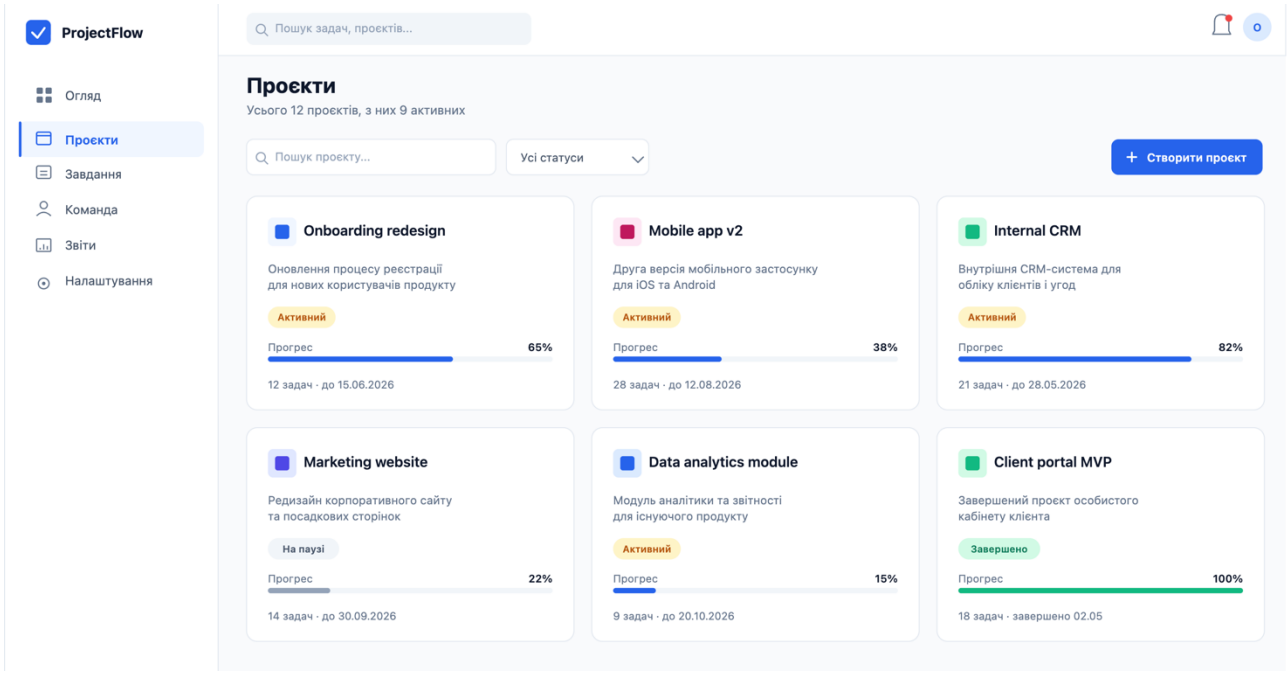


Рис. 2.6 – Макет сторінки проєктів

На сторінці завдань (рис. 2.7) реалізовано зручне відображення задач із можливістю фільтрації за статусом, виконавцем та термінами виконання. Завдання згруповані за статусами, що спрощує контроль процесу виконання.

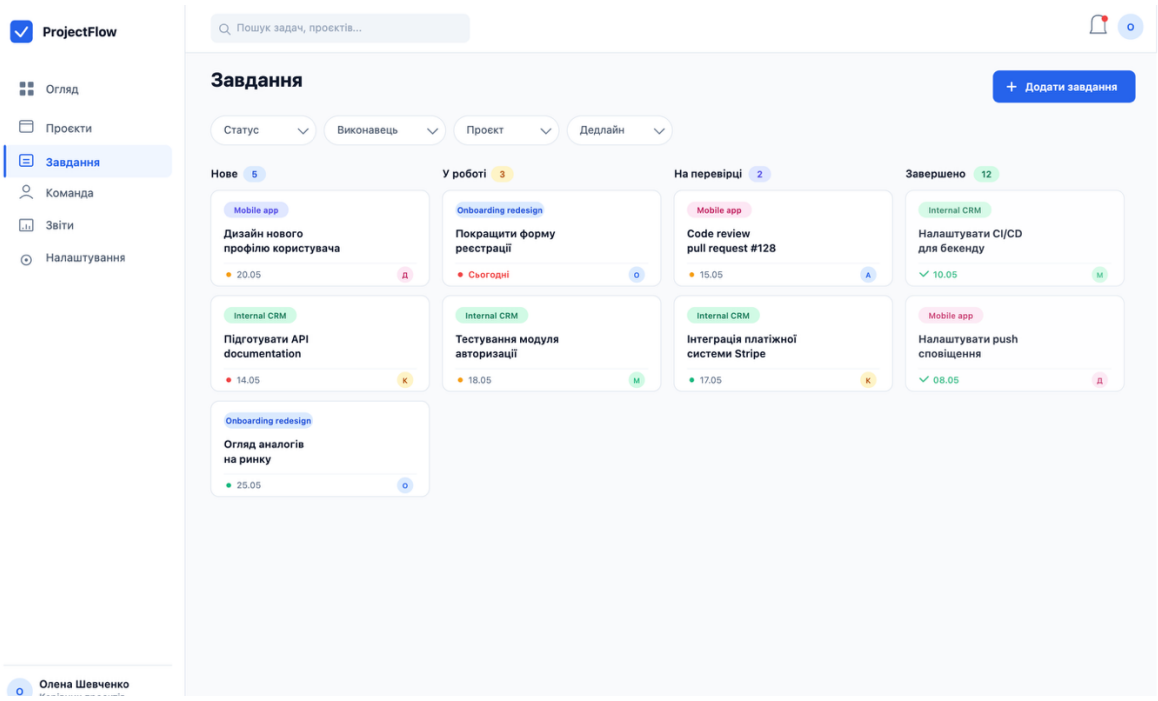


Рис. 2.7 – Макет сторінки завдань

Детальна сторінка завдання (рис. 2.8) містить повну інформацію про задачу: опис, виконавця, дедлайн, статус, а також блок коментарів і історію змін. Це забезпечує ефективну комунікацію між учасниками проєкту.

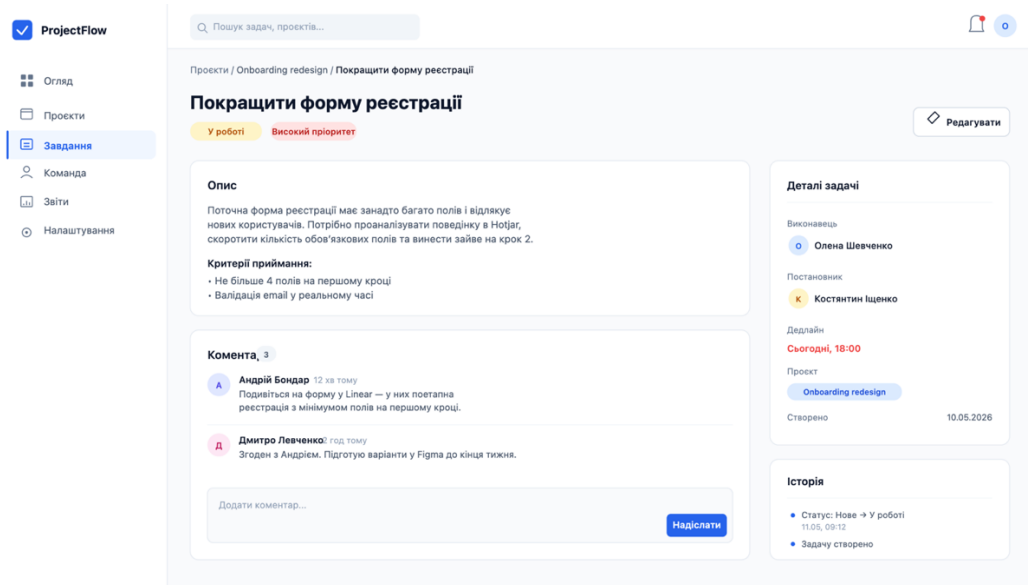


Рис. 2.8 – Макет детальної сторінки завдань

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Вибір технологій розробки

Після того як було сформульовано вимоги, побудовано модель прецедентів, обрано трирівневу архітектуру, спроєктовано базу даних і підготовлено макети інтерфейсу, наступним кроком є вибір технологій, за допомогою яких буде реалізовано інформаційну систему. Цей етап є важливою складовою процесу розробки, оскільки саме від правильно підібраного технологічного стеку залежить ефективність створення програмного продукту, зручність подальшого супроводу та можливість розширення функціоналу у майбутньому.

Під час вибору технологій враховувалися особливості розроблюваної системи, її архітектура, функціональні вимоги та специфіка веб-орієнтованих інформаційних систем. Основна увага приділялася сучасним технологіям, які широко використовуються у веб-розробці, мають активну підтримку спільноти, достатню кількість документації та дозволяють реалізувати клієнтську й серверну частини системи у межах єдиного технологічного підходу.

Розроблювана інформаційна система повинна забезпечувати авторизацію користувачів, управління проєктами та завданнями, збереження даних у базі, взаємодію між клієнтською і серверною частинами, а також зручний сучасний інтерфейс користувача. Саме тому для реалізації було обрано набір технологій, який поєднує простоту використання, гнучкість та можливість подальшого розвитку системи.

Критерії вибору технологій

Під час вибору технологічного стека враховувалося декілька ключових критеріїв. Першим критерієм була відповідність функціональним та нефункціональним вимогам системи, визначеним у попередніх підрозділах. Обрані

технології повинні забезпечувати реалізацію клієнтської та серверної частин, роботу з базою даних, підтримку авторизації користувачів і взаємодію між компонентами системи.

Другим критерієм стала відповідність трирівневій клієнт-серверній архітектурі, оскільки кожна технологія повинна органічно виконувати роль на відповідному рівні системи: frontend, backend або база даних. Також важливим фактором була гнучкість технологій та можливість подальшого розширення функціоналу системи без необхідності суттєвої зміни її архітектури.

Окрему увагу було приділено поширеності технологій у сучасній веб-розробці. Використання популярних інструментів забезпечує доступність документації, велику кількість навчальних матеріалів, бібліотек та готових рішень, що спрощує процес розробки та супроводу системи. Крім того, сучасні веб-технології дозволяють створити зручний користувацький інтерфейс і забезпечити стабільну взаємодію між клієнтською та серверною частинами.

Важливим критерієм також стала уніфікація технологічного стеку. Для реалізації системи було обрано підхід, заснований на використанні JavaScript як основної мови програмування як для клієнтської, так і для серверної частини. Такий підхід забезпечує цілісність процесу розробки, спрощує взаємодію між компонентами системи та дозволяє ефективно організувати структуру програмного продукту.

Опис обраних технологій

React – це JavaScript-бібліотека для побудови інтерфейсів користувача, розроблена компанією Meta (раніше Facebook). У проєкті вона відповідає за весь рівень представлення: рендеринг сторінок, обробку дій користувача, відображення карток проєктів, задач, форм та інших візуальних елементів. Вибір React обумовлений його модульністю – інтерфейс складається з невеликих повторюваних компонентів (Button, Card, TaskItem тощо), які зручно описувати

один раз і використовувати по всьому застосунку. До того ж React має величезну спільноту і дуже багату екосистему готових компонентів, що для MVP є істотною перевагою.

Node.js – це серверне середовище виконання JavaScript, побудоване на двигуні V8. Воно дозволяє запускати JavaScript-код поза браузером – на сервері, де він виконує функції бізнес-логіки. Перевага Node.js полягає у тому, що розробник може використовувати одну й ту саму мову програмування і для клієнта, і для сервера, що знижує поріг входу та спрощує розробку. Для невеликої системи з обмеженою кількістю одночасних користувачів Node.js забезпечує цілком достатню продуктивність.

Express.js – мінімалістичний веб-фреймворк, що працює поверх Node.js. У системі він використовується для реалізації REST API: визначення маршрутів (наприклад, POST /api/projects або GET /api/tasks/:id), обробки HTTP-запитів і формування відповідей у форматі JSON. Express привабливий саме своєю простотою – він не нав’язує жорсткої структури, що дозволяє побудувати компактний серверний застосунок без надлишкової складності. Для демонстраційного MVP це майже ідеальний вибір.

MongoDB – документо-орієнтована NoSQL-база даних, у якій інформація зберігається у вигляді JSON-подібних документів. Хоча у підрозділі 2.4 структуру даних було спроєктовано як умовно-реляційну (із таблицями та зв’язками через зовнішні ключі), MongoDB дозволяє реалізувати ті ж сутності у вигляді колекцій документів зі збереженням логіки зв’язків через посилання за ідентифікаторами. Для MVP MongoDB зручний тим, що не потребує складного попереднього налаштування схеми – структуру колекцій можна гнучко змінювати у процесі розробки. Безкоштовний хмарний сервіс MongoDB Atlas дозволяє розгорнути базу за кілька хвилин, що значно прискорює старт роботи.

Mongoose – це ODM-бібліотека (Object Data Modeling) для роботи з MongoDB у середовищі Node.js. Вона дозволяє описувати схеми документів,

додавати валідацію полів, керувати зв’язками між колекціями та виконувати запити у зручному об’єктному стилі. Без Mongoose довелося б писати більше «низькорівневого» коду для роботи з базою, тоді як ця бібліотека бере на себе значну частину рутини. Для проєкту, де є чітко окреслені сутності (користувач, проєкт, завдання, коментар, статус), використання Mongoose робить код помітно чистішим і безпечнішим.

JWT (JSON Web Token) використовується для реалізації механізму авторизації користувачів. Після успішного входу сервер генерує токен, що містить ідентифікатор користувача та підпис, який клієнт надсилає у заголовку кожного наступного запиту. Це дозволяє системі впізнавати користувача без потреби постійно перевіряти його пароль або зберігати сеанси на сервері. Для невеликого веб-застосунку JWT є простим і поширеним рішенням, яке добре документоване й має готові реалізації для Node.js.

bcrypt – бібліотека для безпечного хешування паролів. Пряме збереження паролів у базі даних у відкритому вигляді є грубим порушенням принципів безпеки, тому замість самого пароля у базі зберігається лише його хеш. bcrypt використовує алгоритм із «сіллю» та змінною складністю, що захищає паролі навіть у разі витоку бази. Для MVP це обов’язковий елемент: безпеку облікових записів не можна відкладати «на потім», навіть у демонстраційному продукті.

Figma – хмарний інструмент для проєктування інтерфейсів, у якому було підготовано усі макети, описані у попередньому підрозділі. Перевагою Figma є зручний браузерний інтерфейс, можливість працювати без встановлення громіздкого ПЗ та широкий вибір готових UI-китів. Для дипломної роботи Figma зручна ще й тим, що з неї легко експортувати макети як рисунки для додавання у пояснювальну записку.

Visual Studio Code – безкоштовний редактор коду від Microsoft, який фактично став стандартом для розробки на JavaScript та багатьох інших мовах. У проєкті він використовується як основне середовище написання коду. Завдяки

великій кількості розширень (для роботи з React, Node.js, MongoDB, Git тощо) VS Code дозволяє повністю покрити потреби розробника MVP без переходу між різними інструментами.

Git та GitHub застосовуються для контролю версій і зберігання коду. Git дає можливість фіксувати зміни у коді у вигляді послідовності комітів, повертатися до попередніх версій та працювати з різними гілками функціональності. GitHub, у свою чергу, виступає віддаленим репозиторієм, у якому код зберігається у хмарі – це гарантує, що результати роботи не будуть втрачені у разі проблем з локальним пристроєм, а також дозволяє продемонструвати історію розробки керівнику чи комісії під час захисту.

Postman – інструмент для тестування HTTP-запитів до API. Під час розробки серверної частини виникає потреба перевіряти, чи правильно працюють маршрути REST API ще до того, як буде написано клієнтський інтерфейс. Postman дозволяє надсилати GET, POST, PUT і DELETE запити, передавати у них дані та заголовки авторизації, а також бачити отримані відповіді. Це значно прискорює відлагодження backend-частини.

Узагальнений перелік обраних технологій з коротким описом їх призначення та обґрунтуванням вибору подано у таблиці 3.1.

Таблиця 3.1 – Обрані технології розробки інформаційної системи

Технологія	Призначення	Обґрунтування вибору
React	Розробка клієнтської частини: інтерфейс користувача, форми, відображення проєктів і задач	Компонентний підхід, велика спільнота, готові UI-елементи, висока поширеність у сучасній веб-розробці

Node.js	Серверне середовище виконання JavaScript для реалізації backend-логіки	Дозволяє використовувати одну мову на frontend і backend, достатня продуктивність для MVP, проста установка
Express.js	Веб-фреймворк для побудови REST API: визначення маршрутів і обробка HTTP-запитів	Мінімалістичний, простий у вивченні, не нав’язує жорсткої структури, добре підходить для невеликих застосунків
MongoDB	Зберігання даних користувачів, проєктів, задач, коментарів і статусів	Гнучка схема, простий старт через MongoDB Atlas, природно поєднується з Node.js та JavaScript
Mongoose	ODM-бібліотека для зручної роботи з MongoDB: схеми, валідація, запити	Спрощує опис сутностей, додає валідацію, зменшує обсяг шаблонного коду
JWT	Авторизація користувачів через токени, що передаються у заголовках HTTP	Простий і поширений стандарт, не потребує зберігання сеансів на сервері, добре підходить для REST API
bcrypt	Безпечне хешування паролів перед збереженням у базі	Алгоритм із сіллю та змінною складністю, що захищає паролі навіть у разі витоку даних
Figma	Проєктування макетів інтерфейсу майбутньої системи	Хмарний інструмент, безкоштовний для індивідуального використання, легкий експорт макетів для пояснювальної записки
Visual Studio Code	Середовище написання коду на JavaScript для frontend та backend	Безкоштовний, легкий, має багато розширень для React, Node.js, MongoDB та Git

Git / GitHub	Контроль версій вихідного коду та його зберігання у віддаленому репозиторії	Стандарт галузі, гарантує збереження результатів роботи, дозволяє вести історію розробки
Postman	Тестування REST API під час розробки серверної частини	Дозволяє надсилати запити різних типів, передавати токени авторизації та одразу бачити відповіді сервера

3.2 Реалізація серверної частини

Серверна частина є ключовим елементом розроблюваної інформаційної системи управління IT-проєктами малого бізнесу. Вона відповідає за прийом і обробку запитів від клієнтського застосунку, виконання бізнес-логіки, взаємодію з базою даних та формування відповідей у форматі JSON. Клієнтська частина, реалізована на основі React, взаємодіє із сервером виключно через HTTP-запити відповідно до архітектурного стилю REST, що забезпечує чіткий поділ між рівнем представлення та рівнем бізнес-логіки.

Серверна частина системи побудована на основі середовища виконання Node.js та веб-фреймворку Express.js. Node.js дозволяє виконувати JavaScript-код поза браузером – безпосередньо на сервері, забезпечуючи обробку HTTP-запитів, роботу з файловою системою та підключення до бази даних. Express.js спрощує організацію маршрутів і middleware, надаючи зручний інтерфейс для побудови REST API.

Точкою входу є файл index.js, у якому виконується ініціалізація застосунку: підключення необхідних залежностей, налаштування middleware, реєстрація маршрутів та встановлення з'єднання з базою даних. Структуру серверної частини подано нижче.

Директорія routes/ містить чотири файли маршрутів, згруповані за функціональним призначенням: authRoutes.js (авторизація), projectRoutes.js

(проєкти), taskRoutes.js (завдання) та commentRoutes.js (коментарі). Директорія models/ містить Mongoose-схеми, що описують структуру документів у базі даних. Конфігураційні параметри – рядок підключення до MongoDB та секретний ключ JWT – зберігаються у файлі .env, що запобігає їх потраплянню до системи контролю версій.

Для обробки тіл запитів у форматі JSON підключається middleware express.json(). Для забезпечення можливості міждоменних запитів з боку клієнтського застосунку використовується бібліотека cors. Після успішного підключення до MongoDB сервер починає прослуховувати вхідні HTTP-запити на порті 3001.

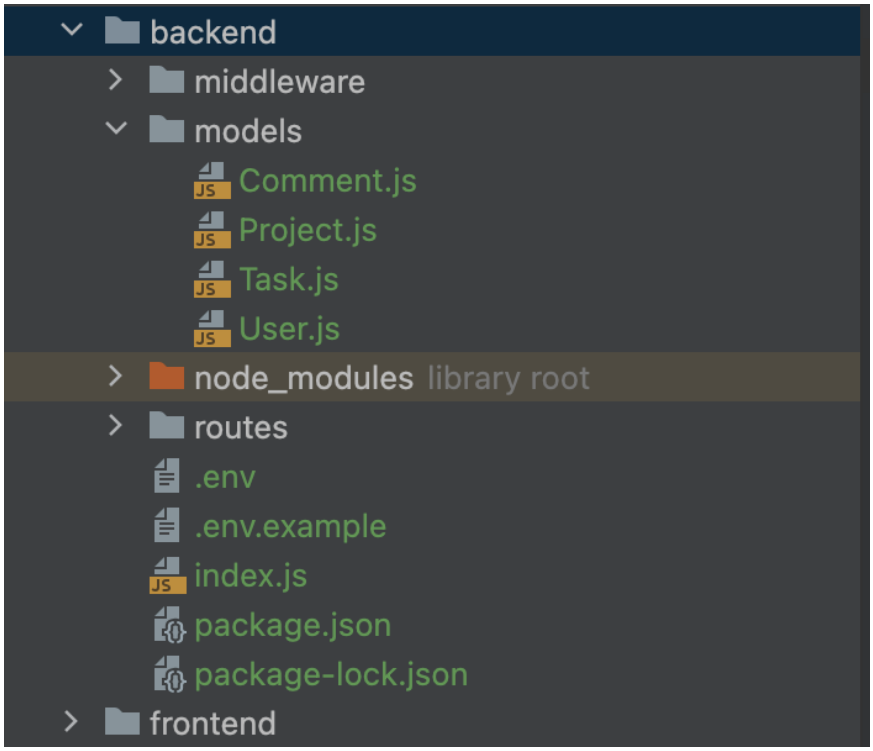
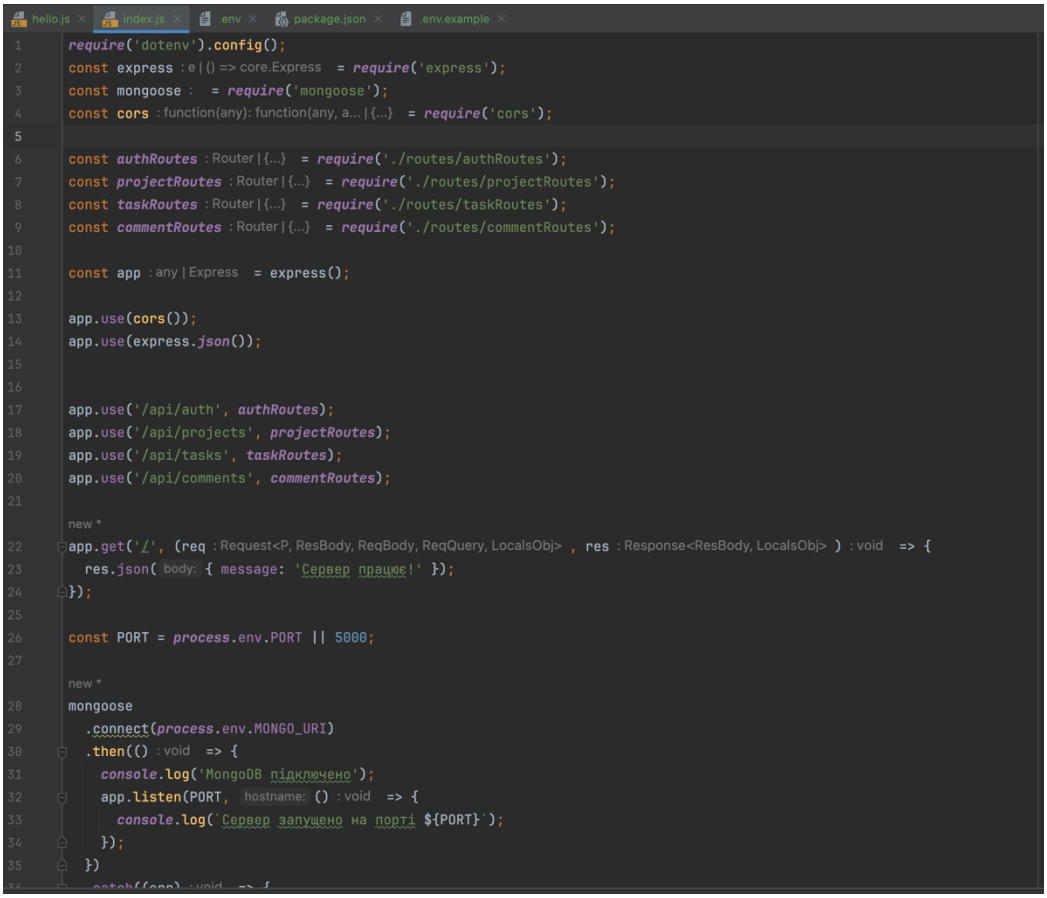


Рис. 3.1 – Файлова структура серверної частини проєкту

Авторизація користувачів реалізована на основі механізму JSON Web Token (JWT). При реєстрації введений користувачем пароль хешується за допомогою бібліотеки bcrypt перед збереженням у базі даних, що унеможливлює зберігання паролів у відкритому вигляді та підвищує захищеність системи.



```
hello.js | index.js | .env | package.json | .env.example

require('dotenv').config();
const express : e | () => core.Express  = require('express');
const mongoose :  = require('mongoose');
const cors : function(any): function(any, a... | {...}  = require('cors');

const authRoutes : Router | {...}  = require('./routes/authRoutes');
const projectRoutes : Router | {...}  = require('./routes/projectRoutes');
const taskRoutes : Router | {...}  = require('./routes/taskRoutes');
const commentRoutes : Router | {...}  = require('./routes/commentRoutes');

const app : any | Express  = express();

app.use(cors());
app.use(express.json());


app.use('/api/auth', authRoutes);
app.use('/api/projects', projectRoutes);
app.use('/api/tasks', taskRoutes);
app.use('/api/comments', commentRoutes);

new *
app.get('/', (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) : void  => {
  res.json( body: { message: 'Сервер працює!' });
});

const PORT = process.env.PORT || 5000;

new *
mongoose
  .connect(process.env.MONGO_URI)
  .then(() : void  => {
    console.log('MongoDB підключено');
    app.listen(PORT,  hostname: () : void  => {
      console.log(`Сервер запущено на порті ${PORT}`);
    });
  })
```

Рис. 3.2 – Фрагмент коду файлу index.js з налаштуванням Express-сервера та підключенням до MongoDB

При вході до системи сервер порівнює введений пароль з хешем, збереженим у базі даних, за допомогою функції bcrypt.compare(). У разі успішної перевірки генерується JWT-токен, підписаний секретним ключем. Токен містить ідентифікатор користувача та передається клієнту у відповіді. Клієнтський застосунок зберігає отриманий токен і надсилає його у заголовку Authorization при кожному зверненні до захищених маршрутів. На стороні сервера реалізовано middleware, яке перевіряє валідність токена: у разі відсутності або недійсності токена запит відхиляється із відповідним кодом помилки HTTP 401.

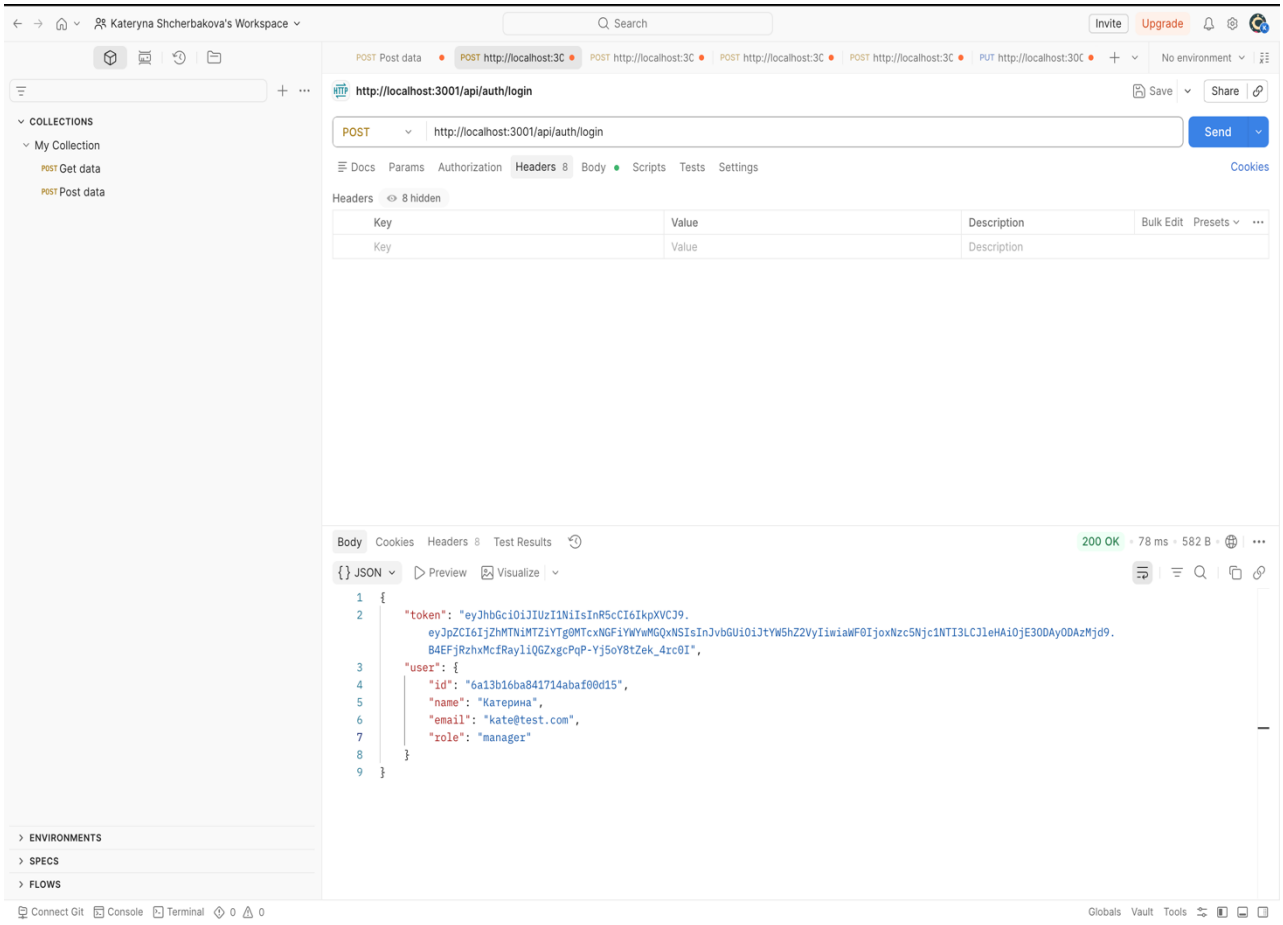


Рис. 3.3 – Результат тестування ендпоінту авторизації у Postman

API серверної частини побудовано відповідно до принципів архітектурного стилю REST. Кожна функціональна сутність системи – проєкти, завдання, коментарі та користувачі – має виділений набір маршрутів. Маршрути реалізують стандартні HTTP-методи: GET – для отримання даних, POST – для створення нових записів, PATCH – для часткового оновлення існуючих об'єктів, DELETE – для їх видалення.

У файлі index.js маршрути підключено за такими префіксами: /api/auth – операції авторизації та реєстрації, /api/projects – управління проєктами, /api/tasks – управління завданнями, /api/comments – додавання та отримання коментарів. Така організація коду забезпечує його структурованість і зручність розширення у майбутньому.

Для роботи з MongoDB використовується бібліотека Mongoose – ODM (Object Data Modeling) для Node.js. Mongoose дозволяє описувати структуру документів у вигляді схем з визначенням типів, обов'язковості та валідації полів.

На основі схем формуються моделі, через які виконується взаємодія з відповідними колекціями бази даних.

У директорії models/ описано чотири основні схеми: User (ім'я, email, хешований пароль), Project (назва, опис, статус, автор), Task (назва, опис, статус, пріоритет, виконавець, дедлайн, прив'язка до проєкту) та Comment (текст, автор, прив'язка до завдання). Сервер виконує повний набір CRUD-операцій: створення документів через save() або create(), отримання даних через find() та findById(), оновлення через findByIdAndUpdate(), а при необхідності – видалення через findByIdAndDelete(). Mongoose автоматично перетворює результати запитів у JavaScript-об'єкти, що спрощує їх подальшу обробку та серіалізацію у JSON.

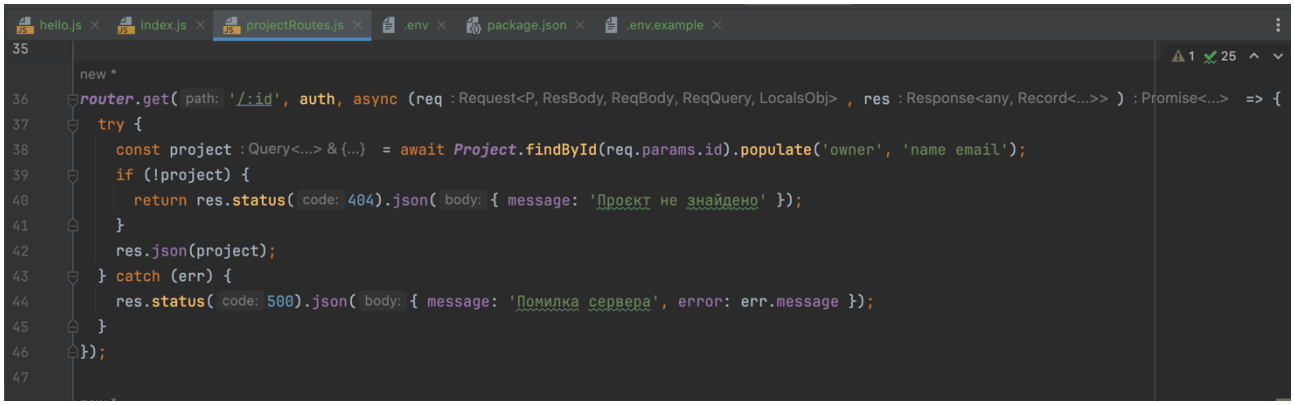


Рис. 3.4 – Фрагмент коду маршруту отримання проєкту за ідентифікатором

Таблиця 3.2 – Основні API-маршрути серверної частини

HTTP метод	Маршрут	Призначення
POST	/api/auth/register	Реєстрація нового користувача
POST	/api/auth/login	Вхід до системи та отримання JWT-токена
GET	/api/projects	Отримання списку проєктів поточного користувача
POST	/api/projects	Створення нового проєкту
GET	/api/tasks	Отримання списку завдань

POST	/api/tasks	Створення нового завдання
PATCH	/api/tasks/:id/status	Оновлення статусу завдання
POST	/api/comments	Додавання коментаря до завдання

3.3 Реалізація клієнтської частини

Клієнтська частина є рівнем представлення інформаційної системи та забезпечує безпосередню взаємодію користувача із системою. Вона відображає дані, отримані від серверної частини, та надає зручний інтерфейс для виконання основних операцій: авторизації, управління проєктами й завданнями, перегляду стану робіт та додавання коментарів. Якість інтерфейсу безпосередньо впливає на зручність використання системи, тому під час розробки приділялась увага простоті, логічності навігації та візуальній узгодженості елементів.

Клієнтська частина реалізована на основі бібліотеки React, яка дозволяє будувати інтерфейс у вигляді незалежних компонентів. Кожен компонент відповідає за окремий елемент або сторінку системи та може повторно використовуватись у різних частинах застосунку.

Структура проєкту організована за функціональним призначенням. Директорія pages/ містить п'ять основних сторінок: Login, Dashboard, Projects, Tasks та TaskDetails. Директорія components/ включає допоміжні компоненти, що використовуються на кількох сторінках: Sidebar (бічна навігаційна панель), ProjectCard (картка проєкту) та TaskCard (картка завдання). У директорії services/ розміщено файл api.js, який відповідає за взаємодію з серверним API.

Навігація між сторінками реалізована за допомогою бібліотеки React Router. У файлі App.js визначено маршрути для кожної сторінки: /login, / (dashboard), /projects, /tasks та /tasks/:id. Для захищених маршрутів реалізовано компонент PrivateRoute, який перевіряє наявність токена авторизації у сховищі браузера: якщо токен відсутній – користувача автоматично перенаправляють на сторінку входу.

Взаємодія з серверною частиною здійснюється через HTTP-запити за допомогою бібліотеки Axios. У файлі api.js налаштовано єдиний екземпляр Axios з

базовою адресою серверного API. Завдяки механізму перехоплювачів (interceptors) JWT-токен автоматично додається до заголовка кожного запиту, що звільняє від необхідності вручну передавати токен у кожному виклику. Сервер повертає дані у форматі JSON, які клієнтська частина отримує та відображає користувачу.

Інтерфейс системи складається з п'яти основних сторінок, кожна з яких реалізує окремий функціональний блок.

Сторінка входу (рисунок 3.5) є першою сторінкою, яку бачить користувач при зверненні до системи. Вона містить форму з полями для введення електронної пошти та пароля, а також кнопку для входу. Передбачено можливість перемикання між режимами входу та реєстрації – у режимі реєстрації додатково з'являється поле для введення імені. При виникненні помилки (наприклад, невірний пароль або вже зайнятий email) користувач отримує відповідне повідомлення безпосередньо на сторінці. Дизайн сторінки виконано у мінімалістичному стилі: картка форми розміщена по центру сторінки на світлому фоні з декоративними круговими елементами.

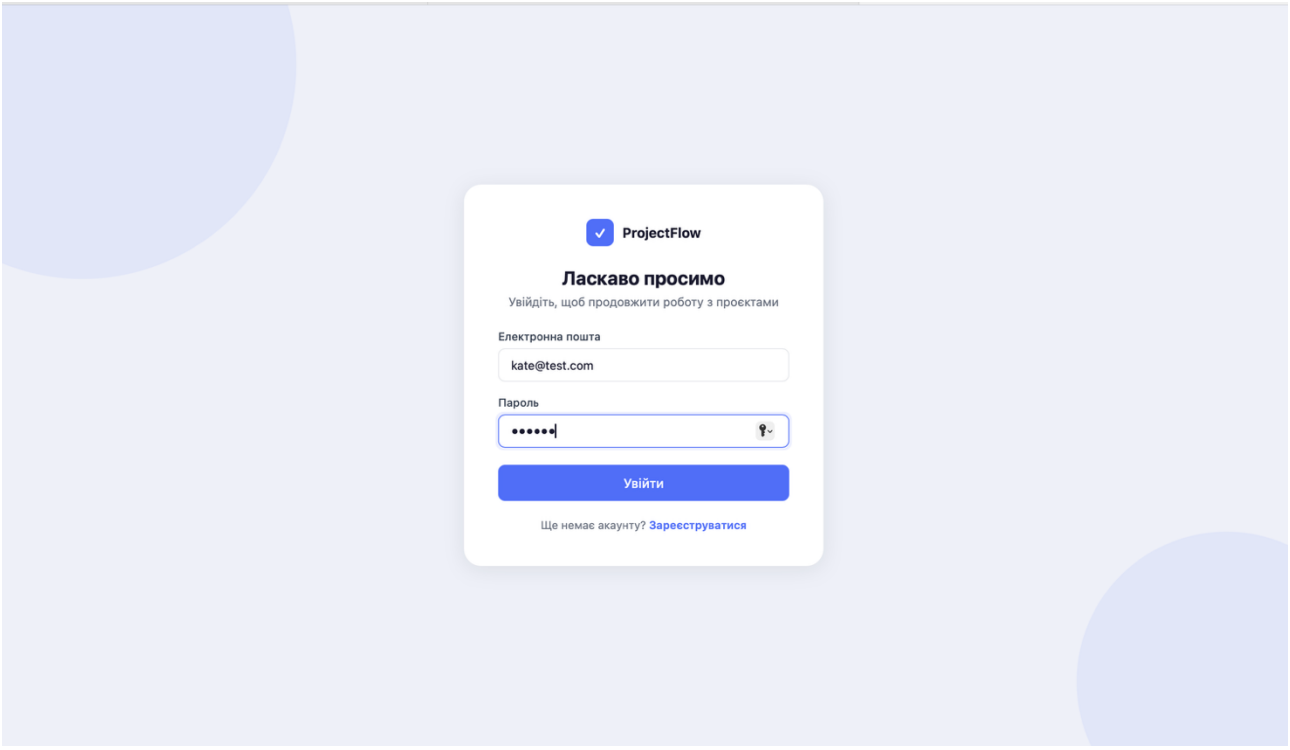


Рис. 3.5 – Реалізація сторінки входу в систему

Головна сторінка – Dashboard (рисунок 3.6) – відображається одразу після успішного входу. Вона містить блок статистики у вигляді чотирьох карток: кількість проєктів, активні завдання, завершені завдання та прострочені завдання. Дані для карток завантажуються при відкритті сторінки через звернення до серверного API і відображаються в реальному часі. Нижче розташовано два блоки: список останніх проєктів зі статусами та перелік найближчих дедлайнів із сортуванням за датою. Кожен елемент є інтерактивним і при натисканні переходить на відповідну сторінку або завдання.

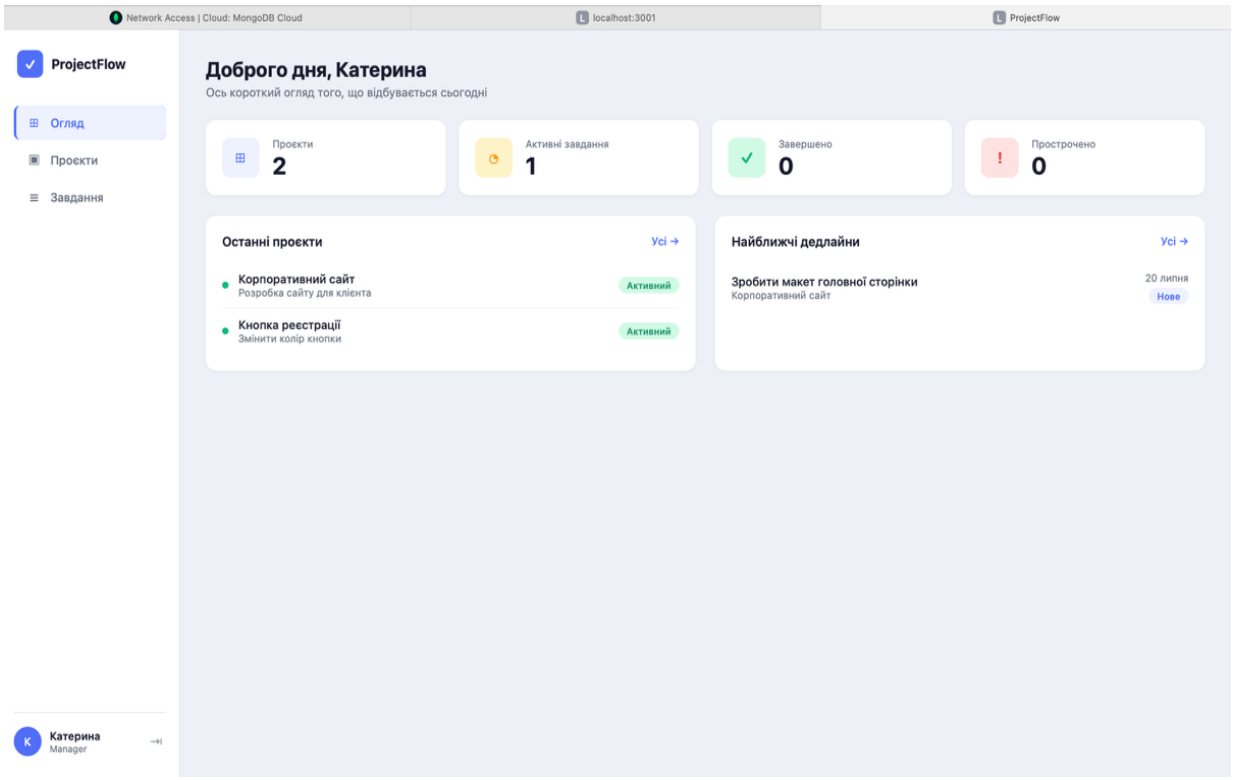


Рис. 3.6 – Реалізація головної сторінки системи

Сторінка проєктів (рисунок 3.7) відображає повний список проєктів у вигляді сітки карток. Кожна картка містить назву проєкту, короткий опис та поточний статус. Реалізовано фільтрацію проєктів за статусом: «Усі», «Активні», «На паузі» та «Завершені». Для створення нового проєкту передбачено кнопку «Створити проєкт», після натискання якої відкривається модальне вікно з формою, що містить

поля для назви, опису, дат початку й завершення та статусу. Після збереження список проєктів оновлюється автоматично.

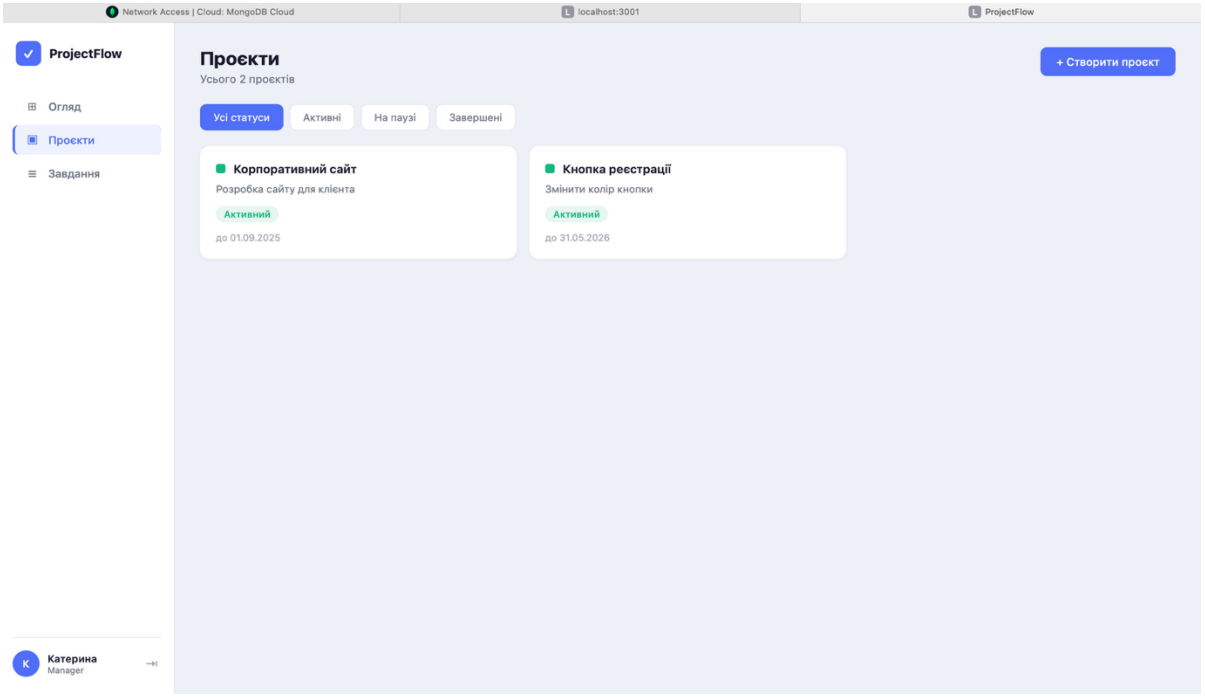


Рис. 3.7 – Реалізація сторінки проєктів

Сторінка завдань (рисунок 3.8) відображає всі завдання, згруповані за статусами: «Нові», «У роботі», «На перевірці» та «Завершено». Підтримується фільтрація завдань за конкретним проєктом через параметр у URL, що використовується при переході зі сторінки проєктів. Для створення нового завдання передбачено модальну форму з полями для назви, опису, вибору проєкту, пріоритету, статусу та дедлайну.

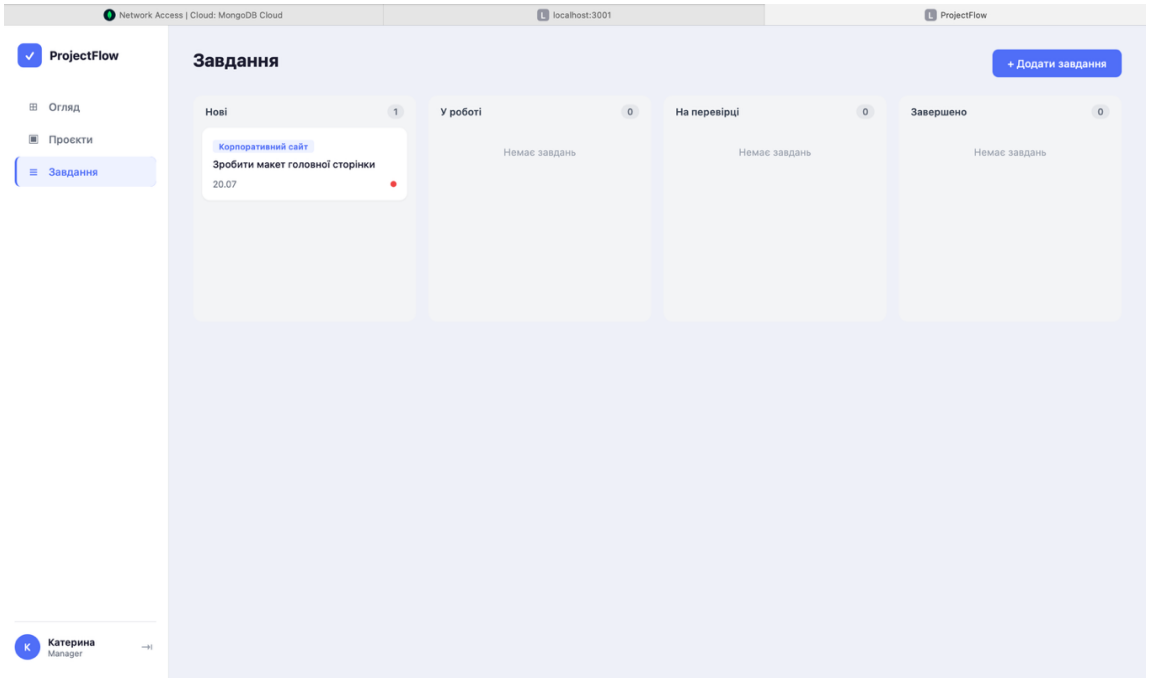


Рис. 3.8 – Реалізація сторінки завдань

Детальна сторінка завдання (рисунок 3.9) відкривається при натисканні на окреме завдання. Вона містить повну інформацію про задачу: назву, опис, пріоритет, дедлайн та поточний статус. Для зміни статусу реалізовано кнопки-перемикачі, натискання на які надсилає PATCH-запит до серверного API та оновлює стан задачі без перезавантаження сторінки. Окремий блок відведено для коментарів: відображається перелік наявних коментарів із іменами авторів та датами, а також форма для додавання нового коментаря.

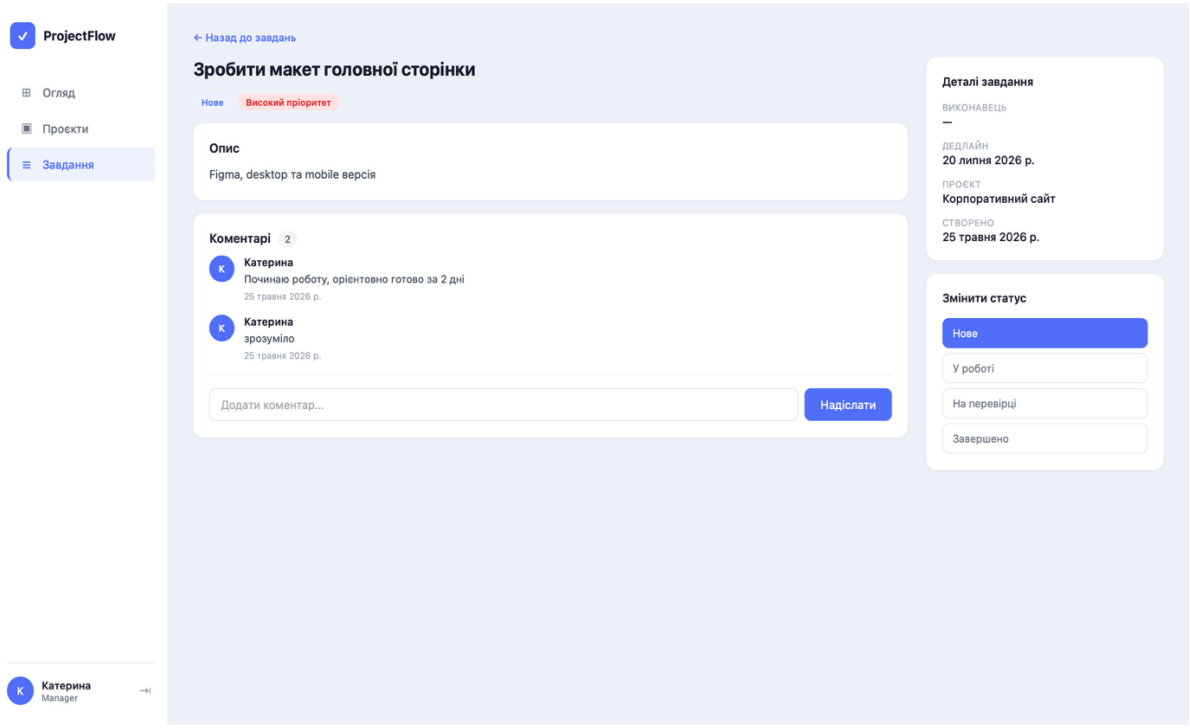


Рис. 3.9 – Реалізація детальної сторінки завдань

Вся взаємодія клієнтської частини з серверним API реалізована через бібліотеку Axios. У файлі services/api.js створено єдиний налаштований екземпляр Axios із базовою адресою сервера. Перед відправленням кожного запиту interceptor автоматично зчитує JWT-токен зі сховища браузера (localStorage) та додає його до заголовка Authorization у форматі Bearer <token>. Завдяки цьому не потрібно вручну передавати токен при кожному зверненні до захищених маршрутів.

Після успішної авторизації сервер повертає токен та дані користувача, які зберігаються у localStorage. При виході з системи або закінченні терміну дії токена дані очищаються, а користувач перенаправляється на сторінку входу. Запити до API виконуються при завантаженні кожної сторінки за допомогою хука useEffect, що забезпечує актуальність відображуваних даних при кожному відкритті сторінки.

Візуальне оформлення системи виконано у сучасному мінімалістичному стилі з використанням біло-синьої кольорової гами. Світлий фон забезпечує читабельність і не перевантажує увагу користувача, тоді як синій колір застосовується для акцентування активних елементів, кнопок та індикаторів

навігації. Всі сторінки мають єдину структуру: фіксована бічна панель (Sidebar) зліва та основна зона контенту праворуч, що формує звичний для веб-застосунків управлінського типу layout.

Компоненти карток, кнопок, форм та модальних вікон оформлені єдино на всіх сторінках, що забезпечує візуальну узгодженість інтерфейсу. Модальні вікна для створення проєктів і завдань відкриваються поверх поточного контенту, не порушуючи навігаційного контексту. Реалізовано адаптивне розташування елементів, що дозволяє комфортно працювати із системою на екранах різної ширини. Загальний дизайн орієнтований на щоденне використання: мінімальна кількість дій для виконання типових операцій та зрозуміла структура навігації між розділами.

3.4 Реалізація основних модулів системи

Модуль користувачів реалізує реєстрацію та авторизацію. При реєстрації користувач вказує ім'я, електронну пошту та пароль; пароль хешується бібліотекою bcrypt перед збереженням у базі даних. У схемі User передбачено три ролі: admin, manager та member – за замовчуванням нові користувачі отримують роль member. При вході сервер перевіряє пароль і у разі успіху повертає JWT-токен. Токен зберігається у localStorage та автоматично додається до заголовка кожного запиту до захищених маршрутів. Якщо токен відсутній або недійсний, сервер повертає код 401 і доступ до системи блокується (див. рисунок 3.5).

Модуль управління проєктами підтримує операції створення, перегляду, редагування та видалення. Схема Project містить поля: назва, опис, дати початку й завершення, статус (Active, Paused, Completed) та посилання на користувача-власника. На сторінці проєктів реалізовано фільтрацію за статусом і відображення у вигляді карток. Новий проєкт створюється через модальну форму – після збереження запис одразу з'являється у списку без перезавантаження сторінки (див. рисунок 3.7).

Модуль управління завданнями є основним функціональним блоком

системи. Схема Task включає: назву, опис, статус, пріоритет, дедлайн, посилання на проєкт та виконавця. Підтримується чотири статуси: New – нове завдання, In Progress – виконується, Review – очікує перевірки, Done – завершено. Пріоритет може бути Low, Medium або High. На сторінці завдання згруповані за статусами. Зміна статусу виконується на детальній сторінці кнопками-перемикачами: надсилається PATCH-запит і стан оновлюється без перезавантаження (див. рисунок 3.8).

Модуль коментарів дозволяє учасникам залишати повідомлення в межах конкретного завдання. Схема Comment зберігає текст коментаря, посилання на завдання, посилання на автора та дату створення. На детальній сторінці завдання коментарі виводяться у хронологічному порядку із зазначенням імені автора та дати. Новий коментар додається через форму внизу блоку і одразу з'являється у списку після надсилання POST-запиту до API (див. рисунок 3.9).

Dashboard відображає зведену інформацію про стан системи. У верхній частині сторінки розміщено чотири картки: кількість проєктів, кількість активних завдань (New та In Progress), кількість завершених завдань (Done) та кількість прострочених – тих, де дедлайн минув, а статус залишається незавершеним. Нижче виводяться два блоки: список останніх п'яти проєктів зі статусами та перелік найближчих дедлайнів – до п'яти завдань, відсортованих за датою дедлайну. Усі дані завантажуються при відкритті сторінки через запити до API проєктів і завдань (див. рисунок 3.6).

3.5 Тестування програмного продукту

Тестування є обов'язковим етапом розробки будь-якого програмного продукту. Його головна мета полягає у виявленні невідповідностей між фактичною поведінкою системи та очікуваними результатами ще до введення продукту в експлуатацію. Відсутність перевірки може призвести до збоїв у роботі, некоректного збереження даних або порушення логіки взаємодії між компонентами системи.

Для веб-орієнтованих інформаційних систем тестування є особливо важливим, оскільки такі системи складаються з декількох рівнів: клієнтської частини, серверного API та бази даних. Коректна робота системи в цілому залежить від узгодженої взаємодії всіх трьох рівнів – збій на будь-якому з них може порушити виконання сценаріїв використання. Тому перевірка повного ланцюжка від дій користувача в інтерфейсі до збереження даних у MongoDB є ключовим кроком перед завершенням розробки.

Для перевірки розробленої системи обрано функціональне тестування – перевірку поведінки системи на відповідність визначеним функціональним вимогам. Такий підхід передбачає послідовне відтворення реальних сценаріїв роботи користувача: реєстрацію, авторизацію, створення та редагування проєктів і завдань, зміну статусів, додавання коментарів і перегляд статистики на dashboard. Автоматизоване тестування у межах цієї роботи не застосовувалося – функціональна складність системи та обмежений обсяг дозволили повноцінно перевірити усі сценарії вручну.

Тестування серверної частини виконувалося за допомогою інструменту Postman. Для кожного маршруту API надсилався відповідний HTTP-запит із необхідними параметрами та тілом запиту. Перевірялися: код стану відповіді (200, 201, 401, 404), структура та вміст JSON-відповіді, поведінка захищених маршрутів за наявності та відсутності JWT-токена, а також коректність збереження даних у базі після кожної операції. Тестування клієнтської частини проводилося безпосередньо у браузері: перевірялося відображення даних в інтерфейсі, реакція на введення некоректних значень у форми, а також оновлення стану сторінки після виконання операцій без перезавантаження.

Тестування модуля авторизації охопило кілька сценаріїв. При реєстрації перевірялося, що пароль зберігається у базі у хешованому вигляді, а не у відкритому тексті. При вході до системи з коректними даними сервер повертав JWT-токен, який зберігався у localStorage та використовувався у подальших запитах. При спробі звернутися до захищених маршрутів без токена або з недійсним токеном сервер коректно повертав код 401.

Для модуля проєктів перевірялися всі операції: створення нового проєкту через модальну форму, отримання списку проєктів з підстановкою даних власника через populate, редагування полів та оновлення запису в базі, видалення проєкту та його зникнення зі списку. Фільтрація за статусом (Active, Paused, Completed) також перевірялася в інтерфейсі – результати відповідали очікуваним. Аналогічно тестувався модуль завдань: створення задачі з прив'язкою до конкретного проєкту та виконавця, відображення задач у згрупованому за статусами вигляді, зміна статусу через кнопки-перемикачі на детальній сторінці та оновлення запису в MongoDB без перезавантаження сторінки.

Модуль коментарів перевірявся на детальній сторінці завдання: після відправки форми коментар одразу з'являвся у списку із коректно підставленим іменем автора та поточною датою. Під час тестування dashboard перевірялася точність підрахунку кожного з чотирьох показників – кількість проєктів, активні, завершені та прострочені завдання – шляхом порівняння значень на сторінці з фактичним вмістом бази даних. Список найближчих дедлайнів також відповідав очікуваному сортуванню за датою.

Таблиця 3.3 – Результати функціонального тестування системи

№	Функція	Очікуваний результат	Результат тестування
1	Реєстрація користувача	Обліковий запис створено, пароль збережено у хешованому вигляді	Функція працює коректно
2	Авторизація користувача	Сервер повертає JWT-токен, користувач отримує доступ до системи	Функція працює коректно
3	Створення проєкту	Проєкт збережено у MongoDB, відображається у списку	Функція працює коректно

4	Редагування проєкту	Оновлені дані збережено, зміни відображаються в інтерфейсі	Функція працює коректно
5	Видалення проєкту	Запис видалено з бази даних, зникає зі списку проєктів	Функція працює коректно
6	Створення завдання	Завдання збережено із прив'язкою до проєкту та виконавця	Функція працює коректно
7	Зміна статусу завдання	Статус оновлено через PATCH-запит без перезавантаження сторінки	Функція працює коректно
8	Додавання коментаря	Коментар збережено, відображається з іменем автора та датою	Функція працює коректно
9	Відображення dashboard	Картки статистики відповідають фактичним даним у базі	Функція працює коректно
10	Отримання даних із MongoDB	API повертає коректні JSON-дані, frontend відображає їх без помилок	Функція працює коректно

За підсумками проведеного функціонального тестування усі десять перевірених сценаріїв виконуються коректно. Серверна частина стабільно опрацьовує вхідні запити, повертає відповіді з очікуваними кодами стану та забезпечує надійний захист маршрутів через перевірку JWT-токена. Клієнтська частина правильно відображає отримані від API дані та реагує на дії користувача без помилок у консолі браузера. Операції запису, читання, оновлення та видалення виконуються коректно на рівні MongoDB – зміни, внесені через інтерфейс, одразу відображаються в базі даних. Помилок у роботі основних сценаріїв під час тестування не виявлено. Отримані результати підтверджують, що розроблена система відповідає визначеним функціональним вимогам і готова до практичного застосування.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі розглянуто теоретичні засади управління ІТ-проєктами у сфері малого бізнесу, проведено аналіз наявних програмних рішень, сформульовано вимоги до інформаційної системи та здійснено її повну практичну реалізацію.

У першому розділі досліджено специфіку діяльності малих ІТ-підприємств, визначено характерні особливості управління ІТ-проєктами та проведено порівняльний аналіз сучасних систем управління – Trello, Jira, Asana, Monday.com і ClickUp. Встановлено, що більшість наявних рішень орієнтована на середній і великий бізнес та має суттєві обмеження для невеликих команд: висока вартість підписки, надмірна складність інтерфейсу, недостатня локалізація та залежність від хмарних сервісів. Це підтвердило доцільність розроблення окремої системи, адаптованої до потреб малого ІТ-підприємства.

У другому розділі визначено функціональні та нефункціональні вимоги до системи, побудовано діаграму прецедентів, обґрунтовано вибір трирівневої клієнт-серверної архітектури та спроєктовано базу даних у складі чотирьох основних сутностей: користувач, проєкт, завдання і коментар. Розроблено макети всіх сторінок інтерфейсу користувача в мінімалістичному стилі з біло-синьою кольоровою гамою, що відповідає вимогам простоти та зручності використання.

У третьому розділі реалізовано повнофункціональний веб-застосунок на основі технологічного стеку JavaScript: React на клієнтській стороні та Node.js з Express на серверній. Для зберігання даних використано MongoDB у зв'язці з бібліотекою Mongoose. Авторизацію реалізовано на основі JWT із хешуванням паролів засобами bcrypt. Серверна частина надає REST API з маршрутами для управління проєктами, завданнями, коментарями та обліковими записами. Клієнтська частина складається з п'яти сторінок: входу до системи, головної інформаційної панелі, сторінок проєктів, завдань та детальної сторінки завдання. За результатами функціонального тестування всі десять перевірених сценаріїв відпрацьовують коректно, помилок не виявлено.

Практична цінність роботи полягає в тому, що розроблена система вирішує конкретну прикладну задачу – організацію проєктної роботи невеликої ІТ-команди без необхідності платити за дорогі корпоративні сервіси. Система може бути розгорнута локально або на хмарному сервері і за потреби розширена додатковим функціоналом: ролевим розмежуванням прав, сповіщеннями, звітністю або інтеграцією зі сторонніми сервісами.

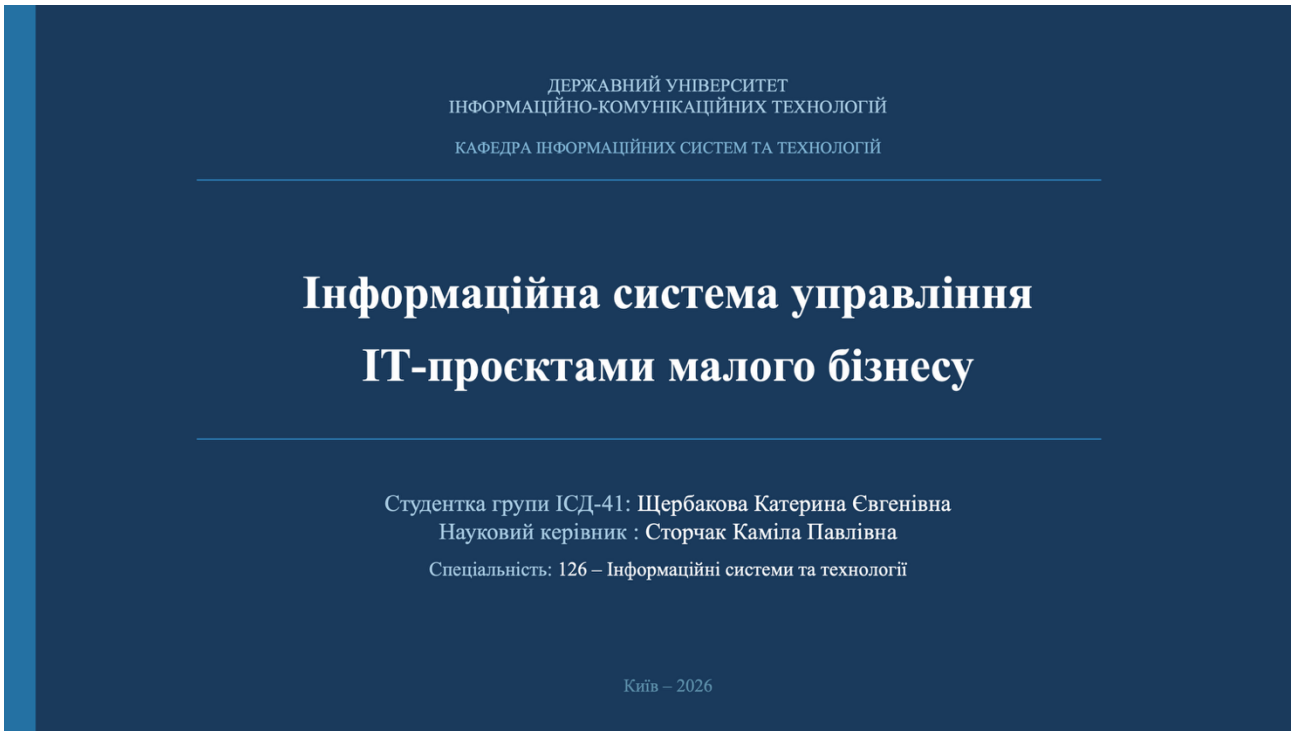
Мету бакалаврської роботи досягнуто: створено інформаційну систему управління ІТ-проєктами малого бізнесу, що забезпечує планування робіт, контроль виконання завдань і координацію діяльності команди. Усі завдання, визначені у вступі, виконано в повному обсязі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Asana. Official website [Електронний ресурс]. URL: https://asana.com (дата звернення: 20.05.2026).
2. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston : Addison-Wesley, 2005. 496 p.
3. ClickUp. Official website [Електронний ресурс]. URL: https://clickup.com (дата звернення: 20.05.2026).
4. Codd E. F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM. 1970. Vol. 13, No. 6. P. 377–387.
5. Date C. J. An Introduction to Database Systems. 8th ed. Boston : Pearson, 2003. 1024 p.
6. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Hoboken : Pearson, 2016. 1272 p.
7. European Commission. SME definition [Електронний ресурс]. URL: https://single-market-economy.ec.europa.eu/smes/sme-fundamentals/sme-definition_en (дата звернення: 20.05.2026).
8. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD dissertation. Irvine : University of California, 2000. 162 p.
9. Jira. Official website [Електронний ресурс]. URL: https://www.atlassian.com/software/jira (дата звернення: 20.05.2026).
10. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 22.05.2026).
11. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Upper Saddle River : Prentice Hall, 2004. 736 p.

12. Monday.com. Official website [Електронний ресурс]. URL: https://monday.com (дата звернення: 22.05.2026).
13. MongoDB Manual [Електронний ресурс] / MongoDB Inc. URL: https://www.mongodb.com/docs/manual (дата звернення: 07.05.2026).
14. Node.js Documentation [Електронний ресурс] / OpenJS Foundation. URL: https://nodejs.org/en/docs (дата звернення: 22.05.2026).
15. OECD. Digitalisation of SMEs [Електронний ресурс]. URL: https://www.oecd.org/en/topics/digitalisation-of-smes.html (дата звернення: 22.05.2026).
16. OECD. The Digital Transformation of SMEs. Paris : OECD Publishing, 2021. 280 p.
17. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). 7th ed. Newtown Square : PMI, 2021. 250 p.
18. React Documentation [Електронний ресурс] / Meta Platforms, Inc. URL: https://react.dev (дата звернення: 22.05.2026).
19. Schwaber K., Sutherland J. The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game [Електронний ресурс]. URL: https://scrumguides.org/scrum-guide.html (дата звернення: 22.05.2026).
20. Sommerville I. Software Engineering. 10th ed. Harlow : Pearson Education, 2016. 816 p.
21. Tanenbaum A. S., Van Steen M. Distributed Systems: Principles and Paradigms. 3rd ed. Hoboken : Pearson, 2017. 596 p.
22. Trello. Official website [Електронний ресурс]. URL: https://trello.com (дата звернення: 22.05.2026).
23. Міністерство цифрової трансформації України. Дія.Бізнес: цифрові рішення для підприємців [Електронний ресурс]. URL: https://business.diia.gov.ua (дата звернення: 22.05.2026).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)



Аналіз існуючих систем управління проєктами

Критерій	Trello	Jira	Asana	Monday	ClickUp
Безкоштовний план	✓ (обмеж.)	✓ (обмеж.)	✓ (обмеж.)	✗	✓
Проста адаптація	✓	✗	✓	✓	✗
Управління ролями	✗	✓	✓	✓	✓
Звітність	✗	✓	частково	частково	✓
Для малих команд	✓	✗	✓	✗	✗

Висновок: наявні рішення або надто складні й дорогі, або функціонально обмежені для потреб малих IT-команд → доцільна розробка власної ІС

3

Вимоги до інформаційної системи

ФУНКЦІОНАЛЬНІ

- Реєстрація та авторизація з ролями (керівник / виконавець)
- Створення та управління проєктами
- Управління завданнями (додавання, редагування, видалення)
- Призначення виконавців та відстеження статусів
- Можливість коментування завдань
- Перегляд проєктів та задач за роллю

НЕФУНКЦІОНАЛЬНІ

- Простий інтерфейс без потреби у навчанні
- Швидкодія при навантаженні до 50 користувачів
- Захист даних: хешування паролів (bcrypt), JWT-токени
- Реалізація системи як веб-застосунку, що працює у сучасних браузерах без потреби встановлення додаткового програмного забезпечення
- Надійність: стабільна робота з мінімальним відсотком збоїв та резервне копіювання даних

4

Архітектура системи та стек технологій

КЛІЄНТСЬКА ЧАСТИНА — **React**

↓

СЕРВЕРНА ЧАСТИНА — **Node.js + Express.js + JWT**

↓

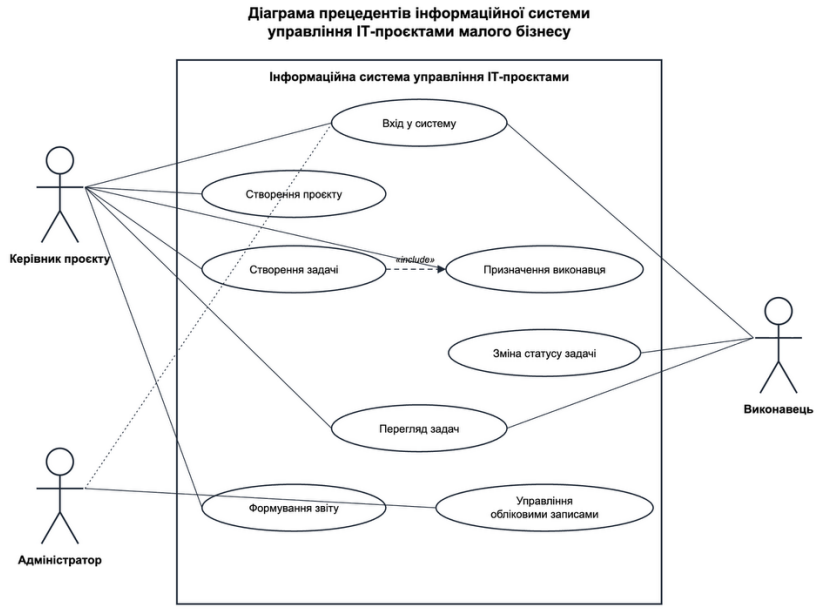
БАЗА ДАНИХ — **MongoDB + Mongoose**

ТЕХНОЛОГІЧНИЙ СТЕК

React UI-бібліотека (Meta)	**Node.js** Серверне середовище JS
Express.js REST API фреймворк	**MongoDB** NoSQL база даних
Mongoose ODM для MongoDB	**JWT** Авторизація / токени
bcrypt Хешування паролів	**Postman** Тестування API

5

Діаграма прецедентів (Use Case Diagram)



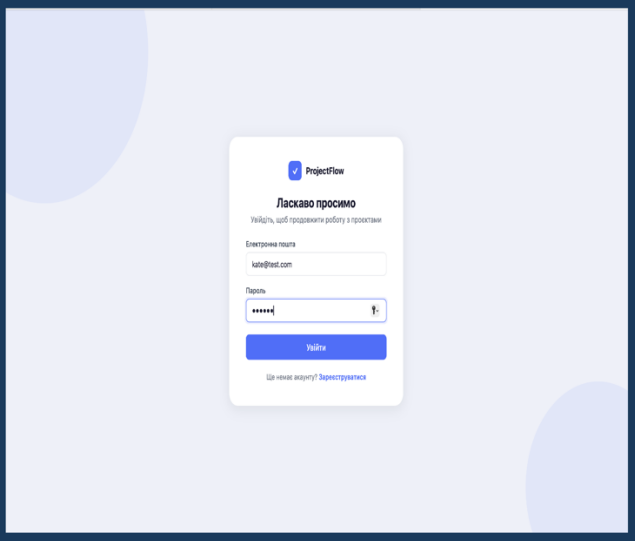
6

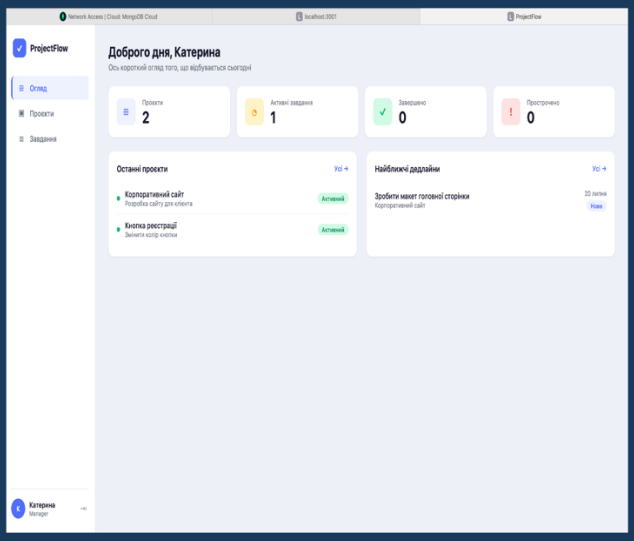
Проєктування бази даних (ER-діаграма)

7

Інтерфейс застосунку

Авторизація та головна сторінка

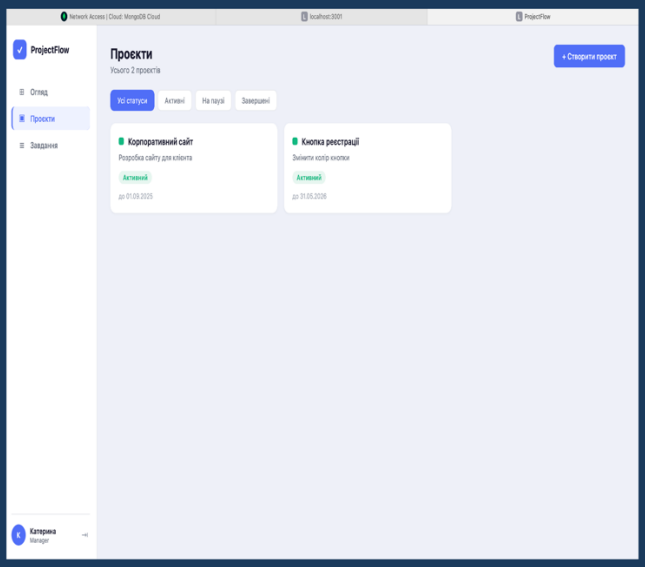


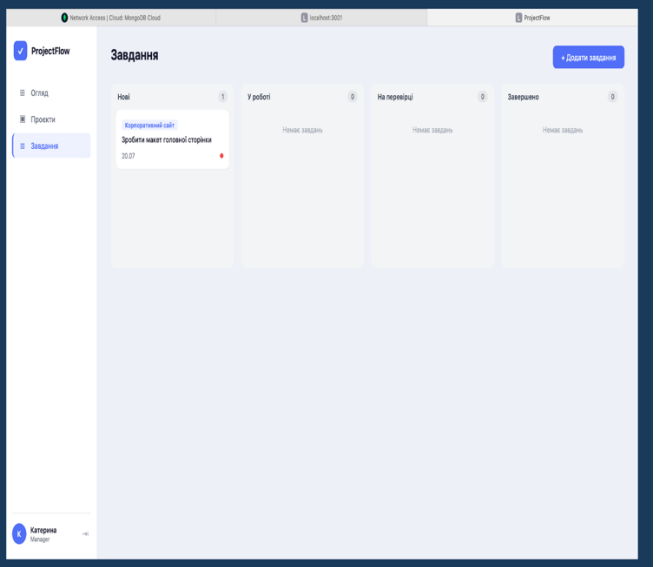


8

Інтерфейс застосунку

Управління проєктами та завданнями





9

Результати функціонального тестування

№	Тест-кейс	Очікуваний результат	Статус
1	Реєстрація нового користувача	Обліковий запис створено, JWT-токен видано	✓ OK
2	Авторизація з валідними даними	Перехід на захищену сторінку	✓ OK
3	Створення нового проєкту	Проєкт збережено, відображається у списку	✓ OK
4	Додавання завдання до проєкту	Завдання з'явилось у kanban-дошці	✓ OK
5	Зміна статусу завдання	Статус оновлено в базі та на UI	✓ OK
6	Додавання коментаря до завдання	Коментар збережено та відображається	✓ OK
7	Доступ без авторизації (захист маршрутів)	Перенаправлення на сторінку входу	✓ OK

Результат: 7/7 тестів пройдено успішно. Усі ключові функції системи підтверджено.

10

Висновки

01 Досліджено специфіку малих IT-підприємств та виявлено ключові обмеження існуючих систем управління проєктами для даного сегменту

02 Визначено функціональні та нефункціональні вимоги, побудовано UML-діаграму прецедентів для трьох ролей користувачів

03 Спроєктовано трирівневу клієнт-серверну архітектуру та базу даних із 6 сутностей на основі MongoDB

04 Реалізовано повнофункціональний веб-застосунок на MERN-стеку з JWT-авторизацією та розмежуванням ролей

05 Проведено функціональне тестування: 7/7 тест-кейсів пройдено успішно, система готова до практичного використання

06 Мету досягнуто: створено ІС управління IT-проєктами, що забезпечує планування, контроль та координацію роботи команди

11

Апробація результатів

Щербакова К.Є.

«Системний аналіз та порівняльна оцінка ефективності Agile та гібридних методологій управління IT-проєктами» // III Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу». — Київ : ДУІКТ, 2025.

Щербакова К.Є.

«Аналітичні інструменти для підвищення ефективності малого бізнесу» // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». — Київ : ДУІКТ, 2026.

Дякую за увагу!

Щербакова К. Є. | ДУІКТ, 2026