КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

_____

(факультет)

інформаційних технологій

_____

(кафедра)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «Розробка соціальної інтернет-платформи для спільних проєктів»

Медвідь Олександр Юрійович

(прізвище, ім’я та по батькові студента повністю)

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., професор Гончаренко Т.А.

„___” _____2025 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «Розробка соціальної інтернет-платформи для спільних проєктів»

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Медвідь Олександр Юрійович
(прізвище, ім’я та по батькові повністю)
122 «Комп’ютерні науки»
(спеціальність)
Інформаційні управляючі системи і технології
(освітня програма)
Групи КН-21-1
Керівник Поплавський О.А.
(прізвище та ініціали)
доцент, доктор технічних наук
(вчене звання, науковий ступінь)
Рецензент к.т.н., доц. Теренчук С.А.
(Прізвище та ініціали)

Ідентичність підтверджую

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій
Випускова кафедра інформаційних технологій
Освітній ступінь: бакалавр
Спеціальність: 122 Комп'ютерні науки
Освітня програма: Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., професор Гончаренко Т. А.
_____
„___" _____2025року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

_____

Медвідь Олександр Юрійович

(прізвище, ім'я та по батькові здобувача)

1. Тема роботи Розробка соціальної інтернет-платформи для спільних проєктів

затверджена наказом ректора КНУБА № 235 від 14 лютого 2025 року.

2. Керівник роботи Поплавський Олександр Анатолійович, д.т.н, доцент
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Строк подання Здобувачем роботи до захисту

4. Зміст пояснювальної записки за розділами:

Р.1 Аналіз предметної області та постановка задачі

Р.2 Методологія та вибір алгоритмів

Р.3 Програмна реалізація системи

Р.4 Техніко-економічне обгрунтування розробки підсистеми

5. Графічний матеріал за розділом:

Р.1 Слайди

Р.2 Слайди

Р.3 Слайди

Р.4 Слайд

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	04.02.2025
Розділ 2	10.04.2025
Розділ 3	25.04.2025
Розділ 4	23.05.2025
Остаточне оформлення роботи	30.05.2025
Направлення роботи для перевірки на плагіат	30.05.2025
Попередній захист роботи на випусковій кафедрі	30.05.2025
Направлення роботи на рецензування	30.05.2025

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Поплавський О. А.	04.02.2025	
Розділ 2	Гончаренко Т. А.	20.05.2025	
Розділ 3	Мацієвський О. О.	28.05.2025	
Розділ 4	Рябчун Ю.В.	30.05.2025	

8. Дата видачі завдання 14 лютого 2025 року

Завідувачка _____ Гончаренко Т. А.
(підпис) (прізвище та ініціали)

Керівник _____ Поплавський О. А.
(підпис) (прізвище та ініціали)

Здобувач _____ Медвідь О. Ю.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ (SUMMARY) *до атестаційної випускної роботи Здобувача:*	Медвідь Олександр Юрійович Medvid Oleksandr		
ЗВО	Київський національний університет будівництва і архітектури		
Тема (українською та англійською)	Розробка соціальної інтернет-платформи для спільних проєктів Development of a social online platform for collaborative projects		
Освітній ступінь	Бакалавр		
Факультет	Автоматизації і інформаційних технологій		
Випускаюча кафедра	Інформаційних технологій		
Спеціальність	122 «Комп'ютерні науки»		
Освітня програма	Інформаційні управляючі системи та технології		
Керівник	Поплавський Олександр Анаталійович		
Обсяг роботи:	пояснювальна записка, стор.	розділів	креслень формату А
	115	4	
Розділ 1.	Аналіз предметної області та постановка задачі		
Розділ 2.	Методологія та вибір алгоритмів		
Розділ 3.	Програмна реалізація системи		
Розділ 4.	Техніко-економічне обгрунтування розробки підсистеми		
Ключові слова: *Keywords:*	соціальна платформа, спільна робота, проєктне управління, онлайн-взаємодія, хмарні технології, цифрові інструменти, командна робота social platform, collaborative work, project management, online interaction, cloud technologies, digital tools, teamwork.		

Здобувач: _____/ Олександр МЕДВІДЬ /

Керівник:_____/ Олександр ПОПЛАВСЬКИЙ /

"___" _____2025р.

АНОТАЦІЯ

Медвідь О.Ю. Розробка соціальної інтернет-платформи для спільних проєктів.

Кваліфікаційна випускна робота бакалавра за спеціальністю: 122 «Комп'ютерні науки», освітньо-професійна програма: «Інформаційні управляючі системи і технології». – Київський національний університет будівництва та архітектури. – Київ, 2025.

Робота присвячена розробці соціальної інтернет-платформи для спільного виконання завдань. Описано методи організації взаємодії користувачів, функціонал платформи, а також механізми управління проєктами. Наведено результати тестування, які підтверджують високу ефективність платформи та її зручність для користувачів.

Ключові слова: соціальна платформа, спільна робота, проєктне управління, онлайн-взаємодія, хмарні технології, цифрові інструменти, командна робота.

SUMMARY

Medvid O.Yu. Development of a Social Internet Platform for Collaborative Projects.

Bachelor's thesis for a bachelor's degree in specialty: 122 “Computer Science”, specialization: "Information Management Systems and Technologies - Kyiv National University of Construction and Architecture - Kyiv, 2025.

The work is dedicated to the development of a social internet platform for collaborative tasks. It describes methods for organizing user interaction, platform functionality, and project management mechanisms. Testing results demonstrate the high efficiency of the platform and its user-friendly design.

Keywords: social platform, collaborative work, project management, online interaction, cloud technologies, digital tools, teamwork.

ЗМІСТ

ВСТУП

Сьогодні в умовах глобальної цифровізації та розвитку інформаційного суспільства особливо актуальним стає питання створення платформ для комунікації та співпраці людей. Зростання попиту на спільну роботу, обмін знаннями та ресурсами потребує нових технологічних рішень. Соціальні інтернет-платформи, що об'єднують користувачів навколо спільних проєктів, стають невід'ємною частиною сучасного життя та відкривають нові можливості для комунікації, творчості та професійного розвитку.

Стрімкий розвиток веб-технологій дозволяє створювати інтерактивні та функціональні сервіси для співпраці між користувачами. Соціальні платформи є ефективним інструментом для реалізації спільних проєктів у різних сферах: бізнесі, освіті, науці, творчості та волонтерських ініціативах. Вони забезпечують доступ до баз знань, полегшують процеси обміну інформацією, дозволяють відслідковувати прогрес та координувати діяльність команд.

На даний момент розробка соціальних інтернет-платформ є актуальним завданням, оскільки такі рішення сприяють об’єднанню людей навколо спільних цілей, поліпшують ефективність спільної роботи та створюють нові можливості для розвитку як окремих користувачів, так і організацій.

Мета роботи: аналіз та розробка соціальної інтернет-платформи для спільних проєктів із інтерактивними інструментами для комунікації та управління задачами.

Об’єкт дослідження: процес організації та управління спільними проєктами на соціальних інтернет-платформах.

Предмет дослідження: методи, моделі та алгоритми створення функціональних модулів для соціальних інтернет-платформ.

Методи дослідження: методи системного аналізу, об’єктно-орієнтоване програмування, веб-дизайн, моделювання бізнес-процесів та методи проєктного управління.

Робота присвячена розробці соціальної інтернет-платформи, яка дозволяє користувачам об'єднуватися для створення та реалізації спільних проєктів. Основною метою проєкту є створення інтуїтивно зрозумілого середовища з інтерактивними інструментами для управління проєктами, комунікації між учасниками та обміну ресурсами.

У першому розділі проведено аналіз існуючих соціальних платформ для спільної роботи, розглянуто їхні функціональні можливості, переваги та недоліки. Зроблено висновок про необхідність створення платформи із новими підходами до управління спільними проєктами.

Другий розділ присвячено аналізу архітектури програмного забезпечення та вибору технологій для створення соціальної інтернет-платформи. Розглянуто методи управління базами даних, структуру користувацького інтерфейсу та засоби комунікації.

У третьому розділі описано процес розробки функціональних модулів платформи, включаючи створення профілів користувачів, інструментів для управління проєктами, системи повідомлень та функціоналу для відстеження прогресу. Наведено приклади коду та опис реалізації ключових компонентів.

Четвертий розділ аналізує результати тестування платформи, зручність її використання та ефективність інструментів для управління проєктами. Проведено порівняння з аналогічними платформами та зроблено висновки про переваги створеного рішення. Також надано рекомендації щодо подальшого розвитку платформи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка й аналіз проблеми

У сучасному цифровому суспільстві одним із ключових факторів успішної реалізації інноваційних ініціатив є ефективна організація спільної роботи. Розвиток гнучких методів управління (agile, scrum, kanban), поширення віддаленої зайнятості та глобалізація команд висувають високі вимоги до інструментів, які забезпечують взаємодію, координацію та обмін інформацією між учасниками проєктів [1].

Проте, попри наявність низки комерційних рішень (Trello, Asana, Monday.com, Notion тощо), існує низка проблем, які обмежують ефективність командної роботи (табл. 1.1)[2–4]:

Таблиця 1.1

Проблеми існуючих платформ

№	Проблема	Короткий опис
1	Фрагментація інструментів	Користувачі змушені використовувати кілька окремих сервісів для управління проєктами, файлами та комунікацією.
2	Потреба використовувати декілька додатків одночасно	Немає єдиного інтегрованого середовища для спільної роботи, що ускладнює координацію.
3	Відсутність адаптивності до специфіки проєктів або команд	Більшість рішень є шаблонними й не враховують індивідуальні особливості робочих процесів.
4	Недостатня підтримка інтерактивної комунікації в реальному часі	Відсутні або обмежені функції чату, відеозв’язку, коментування, спільного редагування.
5	Обмежені можливості аналізу продуктивності та автоматизації рутинних процесів	Бракує вбудованих аналітичних інструментів і засобів для автоматизації повторюваних дій.

Особливої актуальності проблема набуває у студентському, волонтерському та стартап-середовищах, де бюджети обмежені, а потреба в ефективних інструментах управління зростає. Такі користувачі потребують доступної, відкритої платформи з інтуїтивно зрозумілим інтерфейсом, можливістю спільної роботи та високим рівнем гнучкості.

Важливо також враховувати роль цифрових інструментів у підтримці міждисциплінарної співпраці, що особливо цінується в освітніх і дослідницьких проєктах. Можливість організувати спільний доступ до ресурсів, коментування, відстеження прогресу — усе це сприяє підвищенню ефективності командної роботи та скороченню часу на реалізацію ідей.

Окрім організаційних аспектів, важливими є технічні виклики. Наприклад, забезпечення безпеки даних, швидкодія системи при великій кількості користувачів, підтримка масштабованості, інтеграція зі сторонніми API — усе це формує високий поріг для створення якісної соціальної інтернет-платформи [3,4].

Таким чином, постає комплексна проблема: розробка універсального рішення, яке поєднуватиме в собі функціональність, доступність, безпеку та гнучкість, і при цьому відповідатиме сучасним вимогам до командної роботи у цифровому середовищі [2].

1.2 Аналіз ринку соціальних платформ

Загальна характеристика ринку

Ринок соціальних інтернет-платформ для спільної роботи за останні роки зазнав стрімкого зростання, що зумовлено розширенням дистанційного та гібридного формату зайнятості, діджиталізацією бізнесу, а також запитом на інструменти для ефективного управління командною діяльністю. Згідно з даними MarketsandMarkets, обсяг цього ринку у 2023 році склав приблизно 13 млрд доларів США, а до 2028 року очікується зростання до 20+ млрд доларів, із середньорічним темпом приросту 9,2% [2].

На зростання ринку також впливають:

- Поширення фрілансу і мікрокоманд

- Необхідність міждисциплінарної взаємодії
- Популярність відкритих інновацій (Open Innovation)
- Високі вимоги до гнучкості в управлінні проєктами

Згідно з дослідженням компанії McKinsey, працівники витрачають до 28% робочого часу на пошук інформації або комунікацію, що може бути оптимізовано завдяки використанню спеціалізованих платформ спільної роботи. Саме тому соціальні інтернет-платформи, які об'єднують функції управління завданнями, обміну повідомленнями та спільного доступу до документів, набувають широкого поширення в бізнесі, освіті, державному управлінні та волонтерських ініціативах [1].

Основні гравці ринку

Сучасні тенденції демонструють зростаючий попит на сервіси, які забезпечують не лише базову взаємодію в команді, а й інтеграцію з іншими системами, масштабованість та зручність користування. Компанії обирають ті рішення, які найкраще відповідають їхнім операційним потребам, зокрема — гнучкість у налаштуваннях, підтримка мобільних пристроїв, функціональність API та можливість автоматизації робочих процесів.

Наприклад, **Trello** відомий своєю простотою та зручністю у візуальному представленні завдань за допомогою канбан-дошок. **Asana** надає ширші можливості для командної роботи, включаючи таймлайни, залежності між задачами та глибокі аналітичні звіти. **ClickUp** прагне об’єднати в одному інструменті всі аспекти управління командою: від планування і комунікації до документації та цілей. А **Monday.com** виділяється завдяки великому вибору шаблонів та можливості адаптації інтерфейсу під конкретні робочі сценарії.

З іншого боку, **Microsoft Teams** і **Google Workspace** залишаються універсальними рішеннями для корпоративного середовища, поєднуючи комунікаційні засоби з офісними інструментами та хмарним зберіганням файлів. Їхня перевага полягає в централізованому підході до цифрової інфраструктури компаній [23–26].

Порівняльний аналіз існуючих платформ (табл 1.2):

Таблиця 1.2

Аналіз існуючих платформ

Платформа	**Переваги**	**Недоліки**	**Цільова аудиторія**	**Особливості**
Slack	Зручна комунікація, інтеграція з іншими сервісами, гнучкі канали	Слабка організація задач, висока вартість у великих командах	Команди, IT-компанії	Чат-платформа з інтеграціями (Jira, GitHub тощо)
Trello	Простота у використанні, візуальні дошки, drag-and-drop	Обмежений функціонал для великих проєктів, слабка аналітика	Стартапи, освітні проєкти	Методологія Kanban
Asana	Глибока структуризація проєктів, шаблони, аналітика	Обмеженість безкоштовної версії, неінтуїтивний інтерфейс для новачків	Компанії середнього розміру	Орієнтація на цілі й етапи
Notion	Універсальність (нотатки, база знань, таски), шаблони	Немає реального часу сповіщень, слабкі чат-функції	Фрілансери, малі команди	Все в одному: База знань + планувальник

Географічна структура ринку

Найбільші обсяги впровадження платформ спостерігаються в Північній Америці, де цифровізація та віддалена робота є нормою з 2010-х років. Європа демонструє помірне зростання із суворими вимогами до захисту даних (GDPR). Водночас, Азійсько-Тихоокеанський регіон є найбільш динамічним: Японія, Південна Корея та Індія активно впроваджують хмарні рішення [5].

Перспективні напрями розвитку платформ (табл 1.3):

Таблиця 1.3

Напрямки розвитку платформ

Напрям розвитку	**Очікувані переваги**
Інтеграція з ШІ	Автоматизація тасків, розпізнавання пріоритетів, генерація звітів
Використання блокчейну	Прозорість контрактів, безпека фінансових операцій
Мобільна адаптація	Повноцінна робота з будь-якого пристрою
Аналітика в реальному часі	Оцінка продуктивності команд і окремих учасників
Голосові інтерфейси та чат-боти	Спрощення взаємодії з системою

Ключові виклики ринку

Ринок цифрових платформ для командної роботи та управління проєктами розвивається стрімкими темпами, що призводить до виникнення ряду системних викликів, які безпосередньо впливають на динаміку попиту, конкурентне середовище та поведінку користувачів.

По-перше, спостерігається перенасичення ринку великою кількістю як універсальних, так і нішевих рішень. Через це користувачам важко швидко зорієнтуватися серед великої кількості платформ і обрати саме ту, що найкраще відповідає їхнім потребам. Часто рішення приймається не за функціональністю, а за популярністю або маркетинговим позиціонуванням, що не завжди є ефективним для конкретного бізнес-кейсу (табл 1.4):

Таблиця 1.4

Виклики ринку

Виклик	**Коментар**
Перенасичення ринку	Користувачам складно обрати оптимальне рішення
Бар'єри входу для нових гравців	Висока конкуренція з боку гігантів (Microsoft, Atlassian, Google)
Розподіленість команд	Складнощі в синхронізації через різні часові пояси
Безпека персональних даних	Необхідність дотримання міжнародних стандартів
Інтерфейсна складність у багатофункціональних системах	Перевантаження користувача — зниження UX

1.2.1 Тренди цифрової трансформації

Одним із ключових драйверів розвитку ринку соціальних платформ стала загальна цифрова трансформація робочого середовища. Після пандемії COVID-19 у 2020 році майже 70% працівників у США працювали віддалено, що актуалізувало потребу в інструментах цифрової співпраці [9].

На думку Deloitte, сучасні працівники дедалі частіше очікують від цифрових сервісів не лише базової функціональності, а й вбудованої аналітики, адаптивного інтерфейсу, мобільного доступу, інтеграції зі штучним інтелектом, що змінює вимоги до архітектури платформ [4].

Серед головних тенденцій:

- **Гейміфікація** інтерфейсів для підвищення мотивації
- **Гіперперсоналізація** завдяки ШІ та ML
- **Zero Trust безпека** як стандарт
- **Low-code/no-code** моделі налаштування

1.2.2 Сегментація ринку за сферами застосування

Соціальні інтернет-платформи для спільної роботи сьогодні застосовуються в різних галузях. У кожній з них — свої вимоги, сценарії, критерії ефективності (табл 1.5):

Таблиця 1.5

Галузі застосування соціальних інтернет-платформ

Сфера	**Основні сценарії використання**	**Особливі вимоги**
Освіта	Спільні проєкти, обговорення, дистанційне навчання	Простота, інтуїтивність, безкоштовність
Бізнес (SMB)	Управління завданнями, командами, клієнтами	CRM-інтеграція, мобільність, кастомізація
IT-команди	DevOps, таск-менеджмент, спринти, релізи	API, інтеграція з Git, Jira, Slack
Дослідницькі групи	Координація досліджень, обмін результатами	Звітність, контроль версій, архіви
Громадські ініціативи	Координація волонтерів, комунікація, планування заходів	Простота, публічність, відкрита реєстрація

1.2.3 Стратегії адаптації існуючих платформ

Існуючі платформи активно адаптуються до зростаючих вимог користувачів. Основні напрями адаптації подано в таблиці (табл 1.6):

Таблиця 1.6

Стратегії адаптації існуючих платформ

Платформа	**Стратегія адаптації**
Notion	Розширення на API-інтеграції, відкритий API, підтримка AI-команд
Slack	Інтеграція із ChatGPT, функції автоматизації, Slack Canvas
Microsoft Teams	Поглиблена інтеграція з Microsoft 365 Copilot, Live transcription

Ці кроки демонструють загальну тенденцію до конвергенції — об'єднання в одному середовищі завдань, комунікації, документації, автоматизації та аналітики.

1.2.4 Прогноз розвитку ринку до 2030 року

Згідно зі звітом Grand View Research, ринок соціальних платформ для спільної роботи досягне **$33,2 млрд** до 2030 року (рис 1.1)[5].

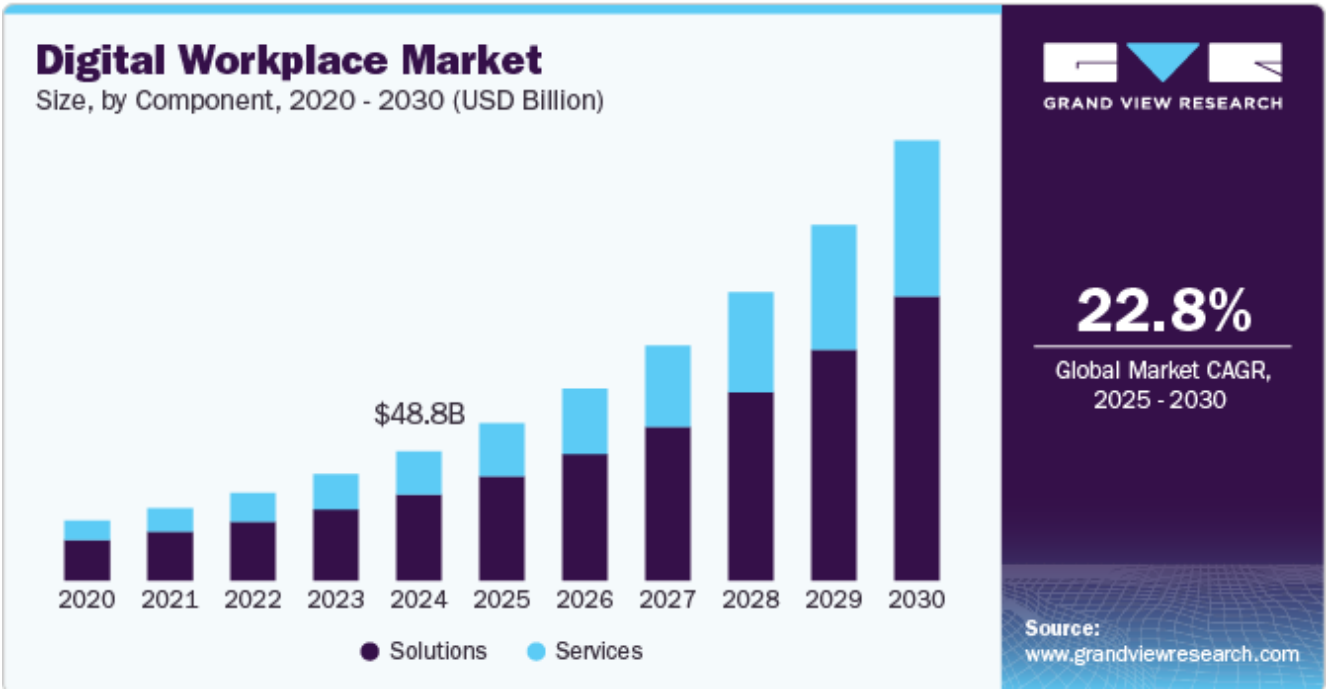

Рис. 1.1 Прогноз розвитку ринку Digital Workplace до 2030 року

Зростання базуватиметься на таких чинниках:

- **Масове впровадження AI first моделей**, що дозволяє автоматизувати рутинні завдання, підвищуючи ефективність команд.
- **Зміна корпоративної культури** у бік відкритості, прозорості та горизонтальних структур управління.
- **Поширення гібридних і розподілених команд**, що вимагають нових засобів для координації роботи.
- **Зростання сегменту освітніх платформ** (на >14% CAGR), де потрібні інструменти для колаборації між студентами, викладачами й адміністрацією.
- **Інтеграція з іншими бізнес-інструментами** (CRM, ERP, HRM), що розширює функціональність платформ.

Нижче подано прогноз росту у таблиці (табл 1.7):

Таблиця 1.7

Прогноз росту ринку

Рік	**Орієнтовний обсяг ринку, млрд $**	**Середньорічний приріст**
2023	13,1	—
2025	17,2	+9,3%
2028	21,6	+8,5%
2030	33,2	+10,2%

1.3 Аналіз науково-практичних досліджень та напрацювань

Упродовж останнього десятиліття тема організації спільної роботи в інформаційних системах стала об’єктом численних наукових досліджень у сферах інженерії програмного забезпечення, соціальних технологій, UX/UI дизайну, когнітивістики та організаційного менеджменту. Особлива увага приділяється таким напрямам, як інтерактивність середовищ, інтелектуальна підтримка колективної діяльності, людиноцентричний дизайн та моделі поведінки користувачів [11].

У сфері розробки спільних платформ провідною темою є створення адаптивних інтерфейсів, що враховують контекст задач користувача. Дослідження показують, що адаптивні інтерфейси можуть підвищити ефективність роботи користувачів, зменшуючи час навігації та покращуючи взаємодію [12].

Також активно досліджується роль контекстно-орієнтованого обміну даними, де інформація відображається не лише у вигляді файлів чи повідомлень, а як структуровані елементи, пов’язані з певною активністю [13].

Методологічною базою для більшості сучасних досліджень є Agile-підходи, які дозволяють швидко адаптувати структуру взаємодії в команді відповідно до поточних вимог. Наприклад, Kanban-модель дозволяє організувати візуальний контроль прогресу задач, а SCRUM-фреймворк — координувати роботу в межах коротких спринтів. Їх адаптація для цифрових платформ широко описана у відповідних джерелах [14].

Важливу роль також відіграє людиноцентричне проєктування (Human-Centered Design, HCD), яке включає UX-дослідження, тестування з реальними користувачами та застосування моделей поведінки. У дослідженнях описано цикл "дослідження – ідеація – прототипування – тестування" як основу для створення дружніх до користувача платформ [15].

Водночас, дослідження відзначають зростаючу роль інтелектуальних агентивних систем у колективній роботі. Такі агенти здатні брати на себе частину рутинних процесів — наприклад, автоматичну постановку задач на основі листування чи пріоритизацію елементів бекстлогу [16].

Окремо варто зазначити, що в багатьох роботах виділяються аспекти довіри та прозорості, які критично важливі при взаємодії у віртуальних командах. Впровадження механізмів логування, візуального контролю змін і збереження історії дій — є важливими компонентами платформ [17].

1.4 Визначення цілей дослідження

Головною метою дослідження є **підвищення ефективності функціональних можливостей соціальної інтернет-платформи для спільних проєктів** шляхом розробки адаптивних архітектурних і програмних рішень, спрямованих на оптимізацію взаємодії між користувачами та вдосконалення механізмів управління проєктами.

Для досягнення цієї мети визначено такі складові цілі:

- На основі аналізу наявної системи дослідити вимоги користувачів до платформи управління проєктами, а також оцінити сучасні технологічні рішення, які можуть бути інтегровані до системи.
- Оцінити ефективність системи з точки зору якості взаємодії між користувачами, зокрема:
    - оцінити точність обміну інформацією;
    - оцінити надійність збереження та передачі даних;
    - скоротити час обробки запитів.

- Розробити адаптивний алгоритм управління проєктами з урахуванням:
  - режимів роботи платформи;
  - параметрів функціонування під час масштабування;
  - зручності та доступності користувацького інтерфейсу.

Усі зазначені цілі формують цілісну модель, що спрямована на створення платформи, здатної забезпечити стабільну, безпечну та масштабовану підтримку спільної роботи в цифровому середовищі, з акцентом на інтерактивність, користувацький досвід та ефективне управління інформацією.

Представимо мету роботи у вигляді дерева цілей (рис. 1.2).

Рис. 1.2 Дерево основних цілей створення соціальної платформи

1.5 Постановка задачі

Після визначення цілей дослідження наступним етапом є формування конкретних задач, розв’язання яких дозволить досягти бажаного функціоналу

системи. Задачі деталізують мету дослідження та визначають обсяг технічної реалізації, логічну структуру та вимоги до функціональності соціальної інтернет-платформи для спільних проєктів.

На основі аналізу предметної області, дослідження наукових джерел, а також результатів попередніх етапів, було сформульовано дерево основних задач системи (рис. 1.3), яке умовно поділяється на три основні напрями: безпека, зручність і функціональність для колективної роботи.

Основні напрямки задач:

1. **Забезпечення безпеки даних**

Передбачає реалізацію ключових механізмів захисту інформації, що обробляється на платформі:

- Впровадження системи автентифікації та авторизації користувачів.
- Реалізація контрольованого доступу до проєктів та обмеження прав згідно з роллю учасника.
- Створення внутрішніх політик збереження даних та запобігання несанкціонованому доступу.

2. **Забезпечення зручності та продуктивності**

Напрямок пов'язаний із побудовою зручного та зрозумілого інтерфейсу, що забезпечить позитивний досвід користувача, а також стабільну роботу платформи при значному навантаженні:

- Створення інтуїтивного користувацького інтерфейсу з сучасними UX/UI-підходами.
- Оптимізація продуктивності при обробці великих обсягів даних (наприклад, проєктів з великою кількістю задач і учасників).
- Реалізація інструментів управління проєктами — створення, редагування, моніторинг, розподіл задач.

3. **Забезпечення функціональності для спільної роботи**

Платформа має передбачати можливість ефективної взаємодії між учасниками проєкту:

- Розробка системи внутрішньої комунікації (чат або коментарі під задачами).

- Механізми спільного редагування вмісту (наприклад, коментарі до завдань, документи).
- Підтримка одночасної взаємодії кількох користувачів у межах одного проєкту.

Усі ці задачі відображено в узагальненому вигляді на дереві основних задач (рис 1.3), що є логічним продовженням сформульованої мети дослідження та конкретизує технічні вимоги до майбутньої системи.

Рис. 1.3 Дерево основних задач роботи соціальної платформи

1.6 Висновки до розділу 1

У першому розділі було проведено комплексний аналіз предметної області, пов’язаної з організацією спільної роботи у цифровому середовищі, а також сформульовано основні вимоги до створення соціальної інтернет-платформи для колективного управління проєктами.

Встановлено, що сучасні команди стикаються з проблемами фрагментованості інструментів, браком адаптивності до специфіки роботи, складністю синхронізації комунікацій, а також відсутністю інтегрованих середовищ для ефективної співпраці. Незважаючи на наявність популярних рішень на ринку (Slack, Trello, Asana, Notion), жодне з них повною мірою не задовольняє

потреби студентських, волонтерських та міждисциплінарних команд з обмеженим бюджетом і високою потребою в гнучкості.

Аналіз ринку показав стабільну тенденцію до зростання у сфері цифрових платформ співпраці, з акцентом на інтеграцію штучного інтелекту, мобільність, реальний час, аналітику та низький поріг входу. Виявлено актуальні тренди, серед яких — гейміфікація, гіперперсоналізація, моделі low-code/no-code та підвищені вимоги до безпеки.

У результаті вивчення наукових джерел було сформульовано методологічну основу дослідження, що базується на принципах agile-менеджменту, людиноцентричного дизайну та інтелектуальної підтримки командної діяльності. Сформульовано дерево цілей, яке відображає основні стратегічні напрями розвитку системи: забезпечення безпеки, гнучкості, зручності та функціональності.

На завершення, на основі аналізу було поставлено конкретну технічну задачу — створення універсальної, адаптивної, масштабованої платформи, яка дозволить організувати ефективну спільну роботу з урахуванням ролей користувачів, взаємодії в режимі реального часу та гнучкого управління проєктами. Сформульовано дерево задач, що деталізує напрями реалізації функціональних і нефункціональних вимог до системи.

2. МЕТОДОЛОГІЯ ТА ВИБІР АЛГОРИТМІВ

2.1 Огляд підходів до спрощення інтерфейсів і масштабування функціональності

У контексті розробки сучасних веб-платформ, зокрема соціальних систем для спільної роботи над проєктами, спрощення інтерфейсу користувача та контрольоване масштабування функціоналу відіграють важливу роль. Аналогічно до того, як у графіці зменшення роздільної здатності дозволяє зробити зображення виразнішим і зрозумілішим, у розробці інтерфейсів така практика допомагає виділити ключові елементи взаємодії, уникнути перевантаження користувача й підвищити зручність користування системою [6].

У сучасних UI/UX-практиках велика кількість інформації або занадто деталізовані елементи можуть ускладнювати взаємодію, особливо в середовищах колективної роботи. Тому застосування концепцій мінімалізму, пріоритезації візуальних сигналів та контекстної навігації дозволяє досягти чіткості, продуктивності й зменшення когнітивного навантаження [7].

Спрощення інтерфейсних рішень дозволяє:

- Виділити лише найважливіші елементи для взаємодії
- Забезпечити швидкий доступ до ключових функцій
- Зменшити кількість відволікаючих або дубльованих візуальних компонентів

Однак, подібно до цифрових зображень, надмірне спрощення або неправильне масштабування може призвести до втрати контексту, руйнування логіки взаємодії або появи непослідовностей в інтерфейсі. Наприклад, зменшення кількості доступних функцій на одному екрані без правильної організації може викликати плутанину в користувачів або ускладнити навігацію [8].

Тому під час розробки платформи необхідно використовувати алгоритми адаптивного проєктування, які дозволяють гнучко масштабувати інтерфейс і зберігати функціональність незалежно від типу пристрою чи кількості даних. Це включає:

- використання адаптивної сітки (grid systems) [10],
- шаблонів модульної архітектури,

- контекстної поведінки елементів інтерфейсу.

Так само, як у піксельній графіці застосовуються спеціалізовані алгоритми для збереження чіткості пікселів, у веб-розробці використовуються методики, що підтримують цілісність візуальної структури при зміні розміру, вмісту або кількості активних елементів. Зокрема, при збільшенні кількості функцій платформи важливо не втрачати простоту, логіку та зручність — ключові характеристики ефективного UX [27].

2.2 Вибір стеку технологій

Для реалізації соціальної інтернет-платформи, орієнтованої на спільну роботу над проєктами, обрано сучасний повнофункціональний стек технологій, який забезпечує масштабованість, зручність розробки, інтерактивність інтерфейсу, надійність зберігання даних і швидкодію.

Вибір кожної технології обґрунтовується вимогами до системи:

- висока реактивність інтерфейсу;
- підтримка великої кількості одночасних користувачів;
- інтеграція з чатами та файлами;
- захист доступу до даних;
- гнучкість у роботі з проектами, задачами, учасниками.

2.2.1 Frontend: React + TypeScript

Клієнтська частина соціальної інтернет-платформи реалізована на основі React у поєднанні з TypeScript. Такий вибір обумовлений необхідністю створити інтерактивний, швидкий та компонентно-орієнтований інтерфейс, який можна легко підтримувати, розширювати й адаптувати до потреб різних типів користувачів (адміністратор, учасник проєкту, гість).

React — це бібліотека JavaScript, створена Facebook, що дозволяє створювати користувацькі інтерфейси на основі Virtual DOM і односпрямованого потоку даних. У контексті соціальної платформи вона дозволяє реалізувати:

- Миттєве оновлення стану: при зміні задачі або повідомлення, відповідні компоненти оновлюються без перезавантаження всієї сторінки.
- Компонентність: кожен елемент — наприклад, «форма створення задачі», «список учасників», «карточка проєкту» — створюється як окремий ізольований модуль, що полегшує тестування, перевикористання і модифікацію.
- Маршрутизація (React Router): платформа містить декілька внутрішніх сторінок (головна, проєкт, профіль, реєстрація, чат), які перемикаються без перезавантаження — це дає ефект «desktop-додатку».

TypeScript — це надмножина JavaScript, яка додає статичну типізацію. Це дозволяє уникнути численних помилок ще на етапі розробки. У складних багатосторінкових застосунках, таких як платформа спільної роботи, без типів дуже складно контролювати структури даних, що передаються між компонентами: наприклад, список учасників у проєкті, задачі з вкладеними файлами, повідомлення в чаті тощо. У нього є свої переваги та недоліки (рис 2.1)[19–20]:

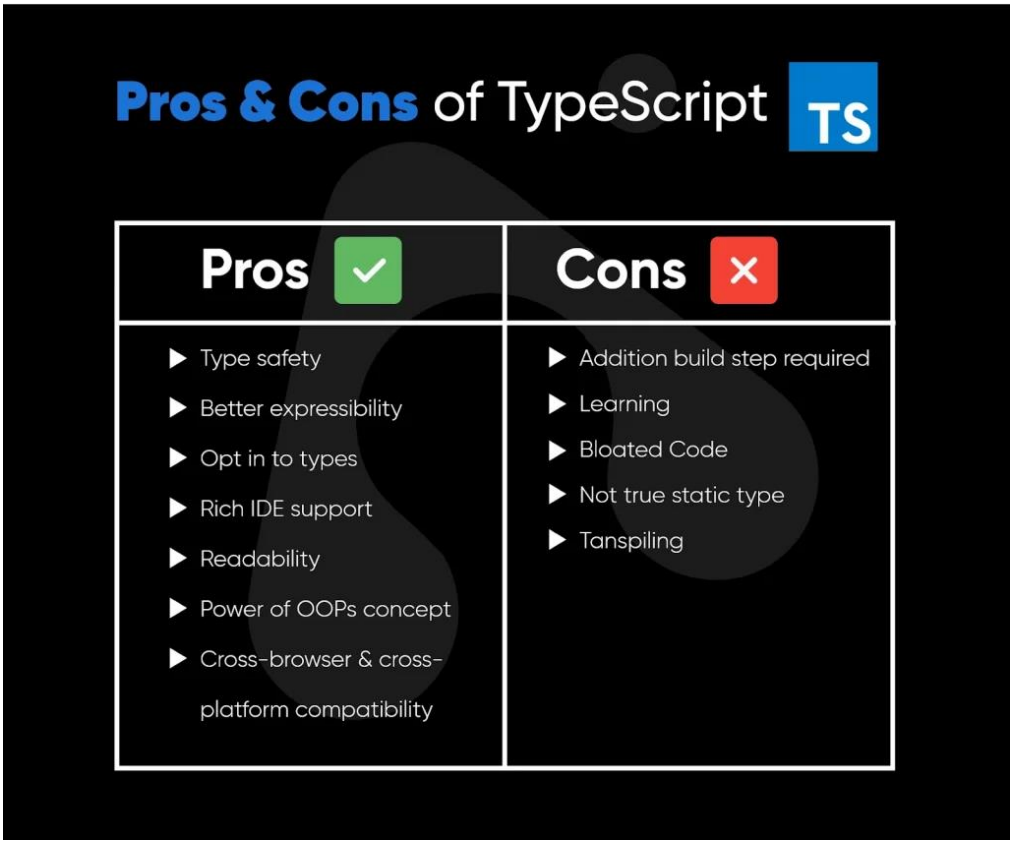

Рис. 2.1 Основні переваги та недоліки TypeScript

Інтерфейс реалізовано згідно з принципами Atomic Design: усі компоненти поділяються на атоми, молекули, організми, шаблони й сторінки (табл 2.1):

Таблиця 2.1

Принцип Atomic Design

Рівень	Приклад	Призначення
Атоми	Кнопка, інпут	Базові елементи
Молекули	Поле введення + кнопка	Форми входу, пошуку
Організми	Панель навігації	Головні частини інтерфейсу
Шаблони	Сторінка проєкту	Позиціонування організмів
Сторінки	/project/:id	Повноцінні маршрути з логікою

Це дає змогу інкапсулювати логіку кожного блоку та підвищити повторне використання — наприклад, картка задачі (TaskCard) може бути використана в списку задач, у перегляді активності, у стрічці оновлень.

Для кращої масштабованості проєкту застосовано підхід функціональної модульності, який передбачає розділення всієї системи на незалежні частини (фічі) з власною логікою, представленням і стилями. Це дозволяє легко розвивати окремі частини проєкту, не зачіпаючи інші модулі (рис 2.2).

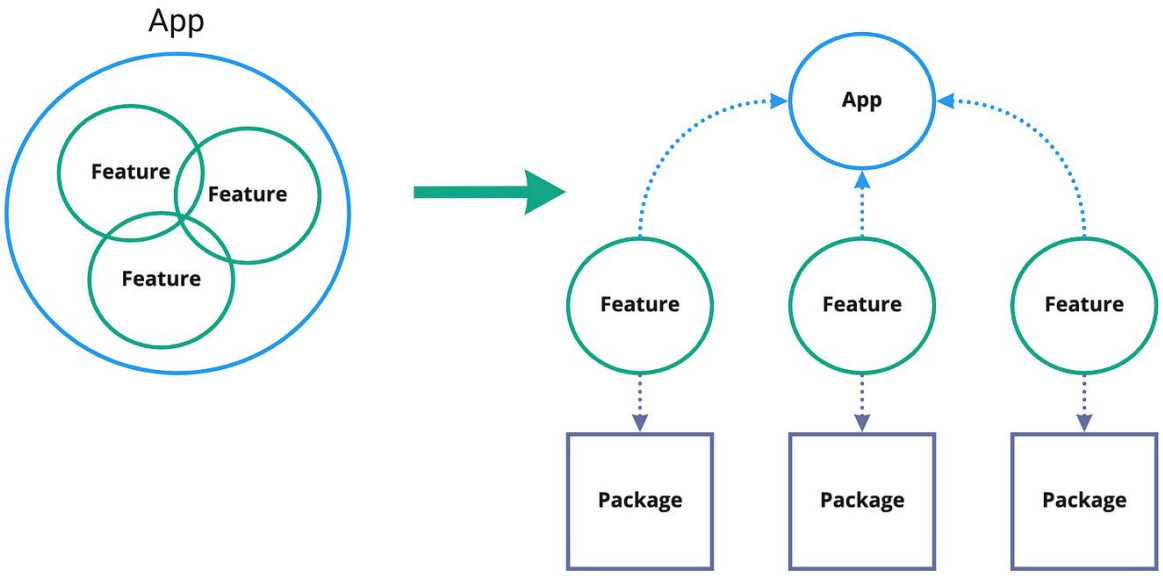

Рис. 2.2 Перехід від монолітної структури до модульної архітектури у frontend-платформі

Frontend-частина також використовує низку додаткових бібліотек (табл 2.2):

Таблиця 2.2

Бібліотеки Frontend-частини

Бібліотека	Призначення
React Router DOM	Клієнтська маршрутизація (сторінки, вкладки)
Axios	Запити до API з обробкою помилок
React Hook Form	Робота з формами (реєстрація, авторизація, коментарі)
React Context API	Глобальний стан користувача, авторизація
TailwindCSS	Швидка та адаптивна стилізація інтерфейсу

Компоненти підписуються на події в режимі реального часу (наприклад, socket.on('newMessage', callback)), що дозволяє оновлювати інтерфейс чату автоматично. Застосування React + TypeScript дає платформі стабільну основу для побудови масштабованого, ефективного, безпечного й гнучкого інтерфейсу. Всі компоненти легко інтегруються між собою, забезпечують приємний UX та мінімізують час на підтримку й розвиток системи.

2.2.2 Backend: Node.js + Express.js

Серверна частина соціальної інтернет-платформи реалізована на базі Node.js — сучасного асинхронного середовища виконання JavaScript, яке ідеально підходить для високонавантажених, інтерактивних веб-систем. У якості веб-фреймворку обрано Express.js — легкий і гнучкий інструмент для створення REST API та маршрутизації запитів [22].

Обраний стек backend-технологій забезпечує низький час відповіді, високу паралельність, універсальність у побудові API, а також зручність у роботі з асинхронними подіями — що є критично важливим для функціоналу реального часу (чат, статуси, події системи).

Node.js побудовано на базі рушія V8 (від Google Chrome) та орієнтовано на однопотокову, неблокуючу модель обробки запитів, що дозволяє одночасно обслуговувати тисячі клієнтів без багатопотокового навантаження.

Що це дає у твоєму застосунку:

- користувачі можуть надсилати повідомлення в чат і виконувати задачі одночасно;
- запити на завантаження проєктів, створення нових задач і приєднання до проєктів виконуються незалежно один від одного;
- система не "зависає" навіть при інтенсивній взаємодії кількох учасників у реальному часі.

Кожна функція платформи має відповідний маршрут (табл 2.3):

Таблиця 2.3

Маршрути запитів у системі

Метод	**Ресурс**	**Призначення**
GET	/projects	Отримати всі проєкти користувача
POST	/projects	Створити новий проєкт
PUT	/projects/:id	Оновити назву, опис або статус проєкту
DELETE	/projects/:id	Видалити проєкт
POST	/auth/login	Авторизація користувача
POST	/messages/:id	Надіслати повідомлення в чат
GET	/tasks/:id	Отримати список задач для проєкту

Однією з ключових особливостей Node.js є реалізація однопотокового циклу обробки подій (Event Loop). Це дозволяє ефективно розподіляти ресурси системи без блокування основного потоку навіть при роботі з файлами, базами даних або іншими зовнішніми процесами (рис 2.3) [21].

Рис. 2.3 Життєвий цикл обробки подій

Express – програмний каркас розробки серверної частини вебзастосунків для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він спроєктований для створення вебзастосунків і API. Де-факто є стандартним каркасом для Node.js. Автор фреймворка, TJ Holowaychuk, описує його як створений на основі написаного на мові Ruby каркаса Sinatra, маючи на увазі, що він мінімалістичний, але має велику кількість плаґінів, що підключаються. Переваги Express.js у проєкті (табл 2.4)[22]:

Таблиця 2.4

Переваги Express.js у проєкті

Перевага	**Пояснення**
Легка інтеграція з базами даних	PostgreSQL підключено через pg або ORM типу Sequelize
Гнучка маршрутизація	Швидке створення REST API для будь-якого ресурсу
Сумісність з middleware	Авторизація, логування, обробка помилок, кешування
Висока швидкодія	Express майже не додає накладних витрат

Загальна логіка обробки HTTP-запиту в архітектурі Express.js виглядає так (рис 2.4). Запит надходить через маршрутизатор (routes), передається відповідному контролеру, де обробляється логіка, взаємодія з моделями та базою даних. Після цього формується відповідь, яка може бути у вигляді JSON-даних або HTML-сторінки (при SSR).

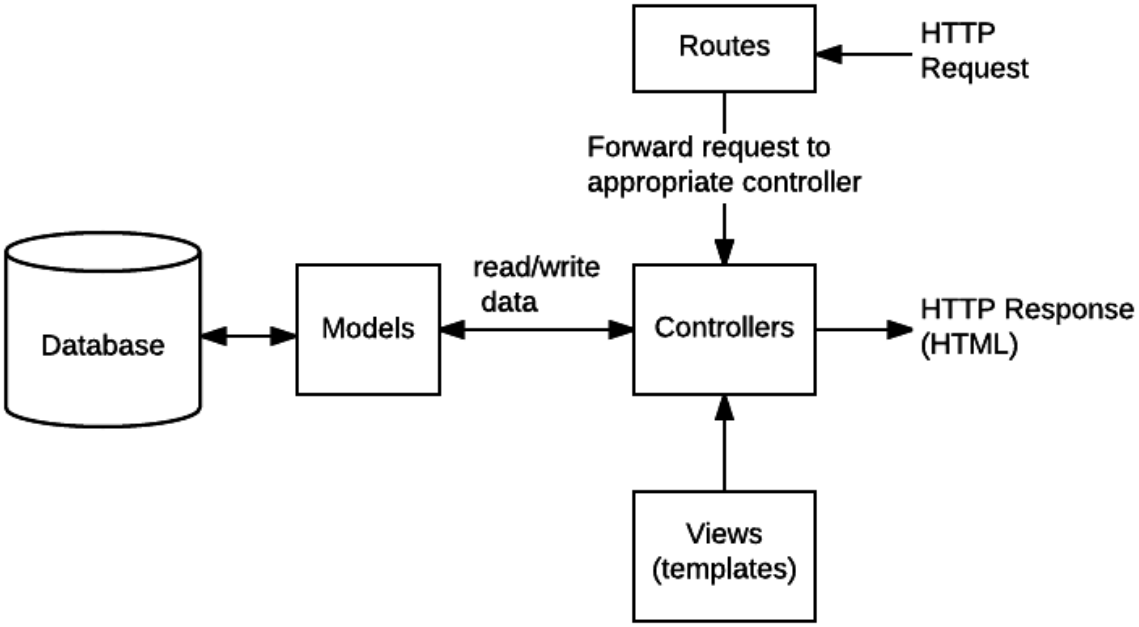

Рис. 2.4 Логіка обробки HTTP-запиту

Інтеграція з іншими компонентами:

- Frontend під'єднується через HTTP-запити або WebSocket.
- JWT перевіряється на рівні middleware (authMiddleware).
- Файли (наприклад, прикріплення до задачі) обробляються через multer.
- Базу даних обслуговують окремі services або ORM з моделями.

2.2.3 База даних: PostgreSQL

Для збереження всіх даних у межах соціальної інтернет-платформи було обрано реляційну систему керування базами даних **PostgreSQL**. Це потужне рішення з відкритим кодом, яке активно використовується у високонавантажених

вебсистемах, завдяки стабільності, масштабованості та широким функціональним можливостям.

PostgreSQL повністю відповідає сучасним вимогам до зберігання даних: вона підтримує транзакції з рівнем ізоляції ACID, складні SQL-запити, індексацію, розгалужені зв’язки між таблицями та оптимізацію виконання запитів. Завдяки гнучкості структури зберігання можливо одночасно працювати як з класичними реляційними таблицями, так і з напівструктурованими даними (наприклад, у форматі **JSON**), що особливо актуально для систем, де частина інформації має динамічну або вкладену структуру.

Таке рішення є оптимальним для веб-платформи, що потребує:

- зберігання структурованих об’єктів: користувачі, проєкти, задачі, повідомлення, файли;
- забезпечення цілісності даних через зовнішні ключі;
- масштабування в майбутньому через реплікацію, шардинг, оптимізацію індексів;
- гнучкість у моделюванні зв’язків (один до багатьох, багато до багатьох).

Система має чітко структуровану базу, яка складається з кількох ключових таблиць (табл 2.5):

Таблиця 2.5

Складові бази даних системи

Таблиця	**Призначення**
users	Зберігає облікові записи користувачів платформи
projects	Інформація про створені проєкти (назва, опис, автор, дата)
tasks	Список задач, прив’язаних до конкретного проєкту
user_project	Проміжна таблиця для зв’язку користувачів і проєктів (роль, доступ)
messages	Повідомлення, надіслані в чатах до проєктів
files	Прикріплені документи (файли до задач або проєктів)

Ключові технічні можливості PostgreSQL у проєкті (табл 2.6):

Таблиця 2.6

Ключові технічні можливості PostgreSQL

Можливість	**Пояснення**
Транзакції	Для гарантії узгодженості даних при створенні задач/проєктів з кількома діями
Індексація	Прискорює пошук по email, project_id, task_id, user_id
JSONB	Застосовується для зберігання історії змін задач або параметрів фільтрації
Foreign keys	Забезпечують зв’язність і цілісність між сутностями
Умовні обмеження	Наприклад, перевірка, щоб дата дедлайну задачі не була у минулому
Масштабування	Можливість розгортання реплік бази при зростанні платформи

Переваги PostgreSQL над альтернативами (табл 2.7):

Таблиця 2.7

Переваги PostgreSQL над альтернативами

Система	**Чому не обрана**
MySQL	Слабша підтримка розширень, обмежена робота з JSON
MongoDB	Документоорієнтована модель не підходить для складних зв’язків
SQLite	Придатна лише для локальних рішень або невеликих проєктів

За результатами опитування розробників (Stack Overflow Developer Survey), PostgreSQL демонструє стабільне зростання популярності серед фахівців. У 2023 році вона вийшла на перше місце серед реляційних СУБД, обійшовши MySQL. Це підтверджує доцільність вибору саме цієї системи як бази даних для платформи спільної роботи (рис. 2.5)[18].

Рис. 2.5 Динаміка популярності БД серед розробників

2.3 Архітектурна концепція системи

Архітектура розроблюваної соціальної інтернет-платформи для спільних проєктів базується на класичній **багаторівневій клієнт-серверній моделі**, яка забезпечує чітке розділення обов'язків між фронтендом (інтерфейс користувача), бекендом (серверна логіка) та системою зберігання (базою даних).

Завдяки такій побудові досягається **масштабованість, гнучкість та безпека**, що є критично важливими для платформи з багатьма користувачами, інтерактивною взаємодією та динамічними даними.

Уся архітектура може бути логічно поділена на три основні рівні:

1. **Клієнтський рівень (frontend)** — реалізований з використанням React та TypeScript, відповідає за відображення інтерфейсу користувача, обробку введення, маршрутизацію між сторінками та формування запитів до API. Увесь інтерфейс динамічно оновлюється без перезавантаження сторінки, що забезпечує швидку взаємодію та хорошу продуктивність.
2. **Серверний рівень (backend)** — реалізований на Node.js із використанням Express.js. Він приймає HTTP-запити від клієнта, обробляє їх відповідно до бізнес-логіки, виконує операції з базою даних, відповідає за авторизацію,

маршрутизацію, перевірку прав доступу. Усі запити до критичних ресурсів контролюються middleware та JWT-токенами.

3. **Рівень зберігання (база даних)** — базується на PostgreSQL. У ній зберігаються всі основні об'єкти системи: користувачі, проєкти, задачі, повідомлення, файли. База побудована на реляційних зв’язках із використанням зовнішніх ключів, індексів і транзакцій для забезпечення цілісності.

Уся взаємодія між клієнтом і сервером відбувається за допомогою REST API (HTTP-запити з JSON-відповідями), а для чату використовується WebSocket-з’єднання через Socket.IO, що забезпечує передачу повідомлень у реальному часі.

Ключові принципи побудови архітектури:

- **Модульність** – логіка системи розділена на окремі компоненти (автентифікація, проєкти, задачі, повідомлення, файли).
- **Реактивність** – дані оновлюються динамічно, без перезавантаження інтерфейсу.
- **Безпека** – вся взаємодія з API захищена JWT-авторизацією, а доступ до даних контролюється на рівні ролей та участі в проєктах.
- **Масштабованість** – систему можна розгортати у контейнерах, дублювати інстанси backend-сервера, додавати кешування (наприклад, Redis), або окремі сервіси для аналітики.

На рисунку подано загальну структурну схему взаємодії компонентів (рис 2.6):

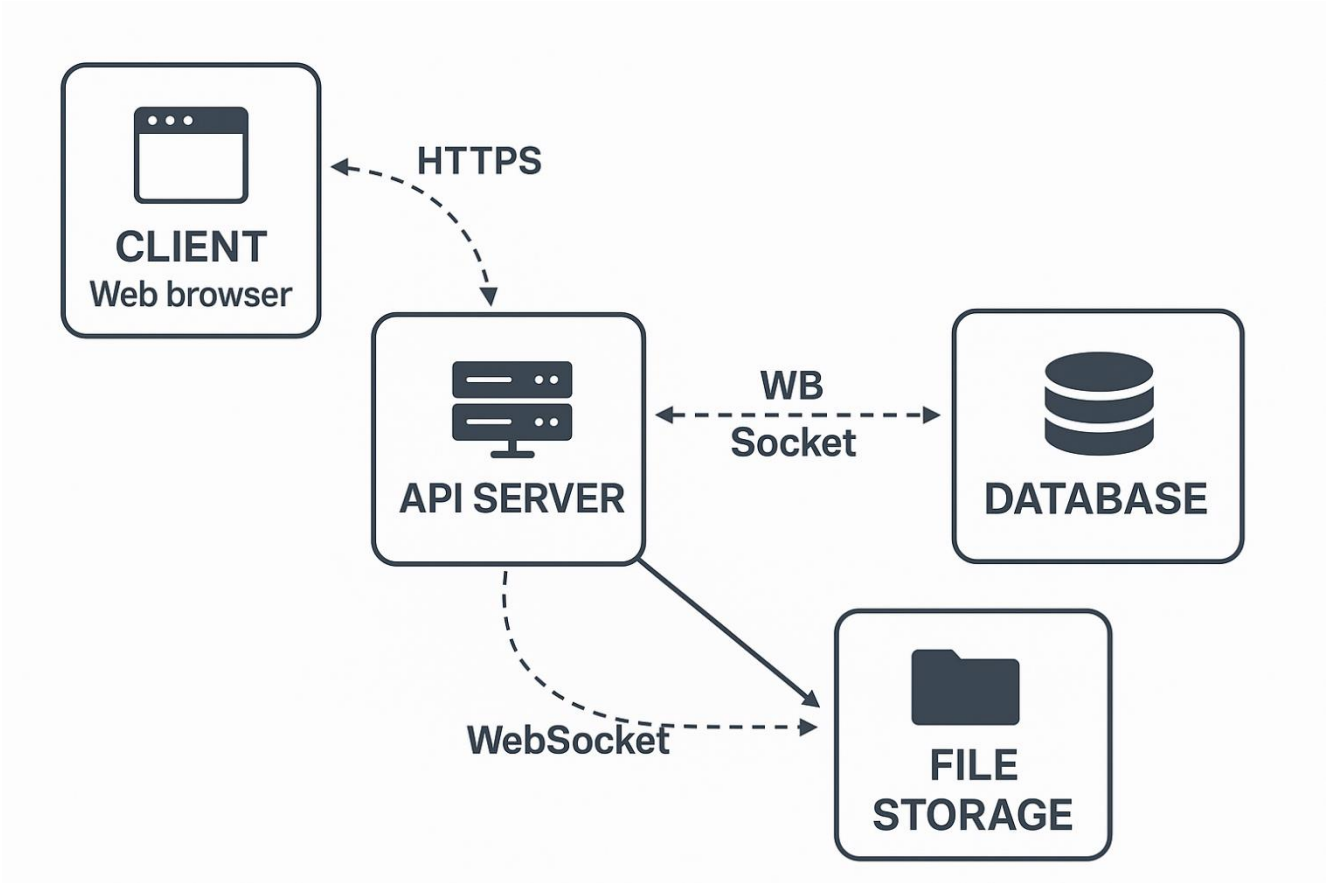

Рис. 2.6 – Загальна архітектура платформи спільної роботи

Схема включає основні елементи: браузер користувача, API-сервер, WebSocket-канал, базу даних та підсистему файлів (локально або через хмарне сховище). Стрілки вказують на напрями запитів (HTTP/WS), відповіді, запити до бази та підключення до сервісів.

Важливо, що така архітектура дозволяє не тільки обробляти великі об’єми даних та одночасних запитів, але й гнучко адаптуватися до розширення функціональності: додавання модуля аналітики, ботів, інтеграції зі сторонніми сервісами (Google Drive, GitHub, Telegram) тощо.

2.4 Інформаційна модель системи

2.4.1. Структурна модель

Структурна модель інформаційної системи — це абстрактне графічне подання основних компонентів системи (модулів, сервісів, об’єктів), а також способів їхньої взаємодії. На відміну від даталогічної моделі, яка описує структуру

збереження даних, структурна модель зосереджується на архітектурі та логіці побудови самої програмної системи (рис 2.7).

Вона визначає:

- які функціональні компоненти входять до складу системи;
- які сервіси, підсистеми, зовнішні API використовуються;
- як організована взаємодія між клієнтською та серверною частиною;
- які існують потоки запитів, відповіді, обробка подій, обмін повідомленнями.

Організаційна підсистема: Визначає загальні принципи організації процесів у платформі. Включає механізми управління користувачами, ролями та групами. Забезпечує координацію між іншими підсистемами.

Соціальна підсистема: Відповідає за інтерактивну взаємодію між користувачами платформи. Включає функції обговорень, спільного редагування контенту та підтримки спільнот.

Функціональна підсистема: Визначає основні процеси та функції платформи, такі як управління проєктами, календарі, завдання та інтеграція з іншими сервісами.

Юридична підсистема: Відповідає за забезпечення відповідності платформи нормативним актам та регуляторним вимогам. Реалізує функції управління політиками конфіденційності та правами доступу.

Безпекова підсистема: Відповідає за захист інформації та користувацьких даних. Забезпечує автентифікацію, авторизацію та захист від кібератак.

Інформаційна підсистема: Обробляє вхідні та вихідні потоки даних у системі. Включає бази даних, інструменти обробки інформації та пошукові механізми.

Структура входів системи: Описує джерела даних, які надходять до платформи, включно із зовнішніми API та користувацькими запитами.

Структура виходів системи: Включає оброблену інформацію, звіти, результати роботи користувачів та повідомлення, які платформа надсилає користувачам.

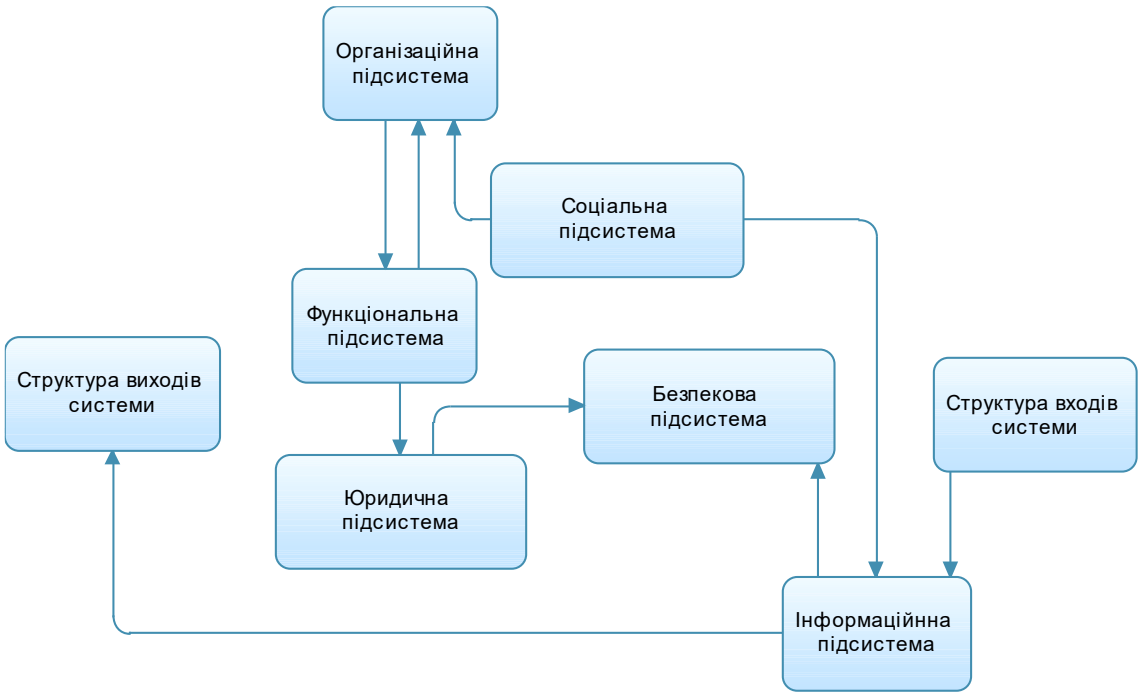

Рис. 2.7 Структурна модель роботи платформи

2.4.2. Даталогічна модель

Даталогічна модель — це формалізоване графічне або табличне представлення логічної структури даних, які використовуються у програмній системі. Вона описує:

- які сутності (об'єкти) існують у системі;
- які атрибути має кожна сутність;
- які зв’язки встановлено між цими сутностями;
- як ці зв’язки організовані: один до одного (1:1), один до багатьох (1:N), багато до багатьох (M:N).

На відміну від фізичної моделі, яка враховує типи даних та структуру таблиць у конкретній СКБД (наприклад, PostgreSQL), даталогічна модель акцентує увагу саме на змістовій структурі предметної області — як частин системи взаємодіють між собою (рис .

Опис сутностей та зв’язків у даталогічній моделі (табл 2.8):

Таблиця 2.8

Сутності та зв’язки в даталогічній моделі

Сутність / зв’язок	Основні атрибути	Роль у системі
Користувач	ПІБ, email, пароль, дата реєстрації	Бере участь у проєктах, створює задачі, залишає коментарі
Проєкт	Опис, статус, дата публікації	Об’єднує задачі, учасників, має власний чат
Задача	Опис, статус, дедлайн	Належить до проєкту, має виконавців, коментарі, може мати чат
Чат	Дата створення, повідомлення	Забезпечує комунікацію в межах проєкту або задачі
R1	ID користувача, ID проєкту	Зв’язок «багато до багатьох» між користувачами та проєктами
R3	ID чату, ID проєкту / користувача	Зв’язок чатів із конкретним об’єктом (проєкт або задача)
R4	ID задачі, ID користувача	Зв’язок задач із виконавцями (множинне призначення)

Даталогічна модель (рис. 2.8).

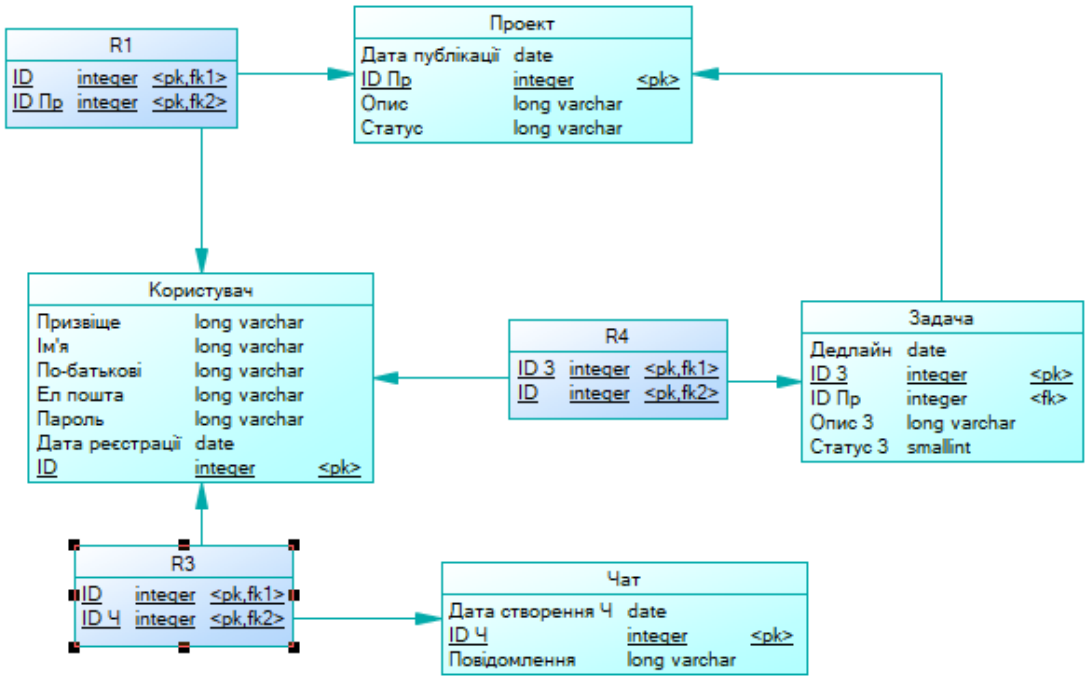

Рис. 2.8 Даталогічна модель роботи платформи

2.4.3 DFD-діаграма

DFD-діаграма застосовується для аналізу потоків даних у системі. Вона описує джерела та призначення даних, логічні функції, потоки даних та сховища, до яких користувачі мають доступ для управління своїми проектами та комунікації (рис 2.9), (рис 2.10), (рис 2.11).

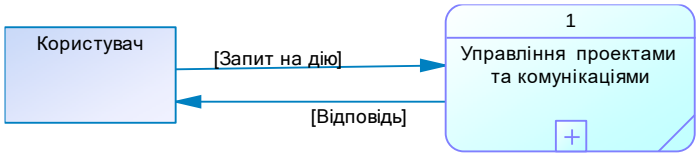

Рис. 2.9 Нульовий (концептуальний) рівень DFD-діаграми

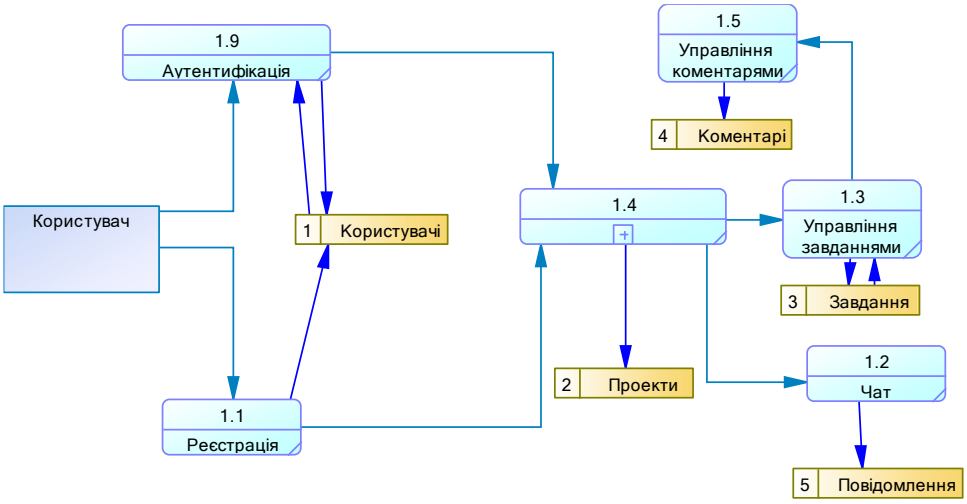

Рис. 2.10 Перший рівень DFD-діаграми

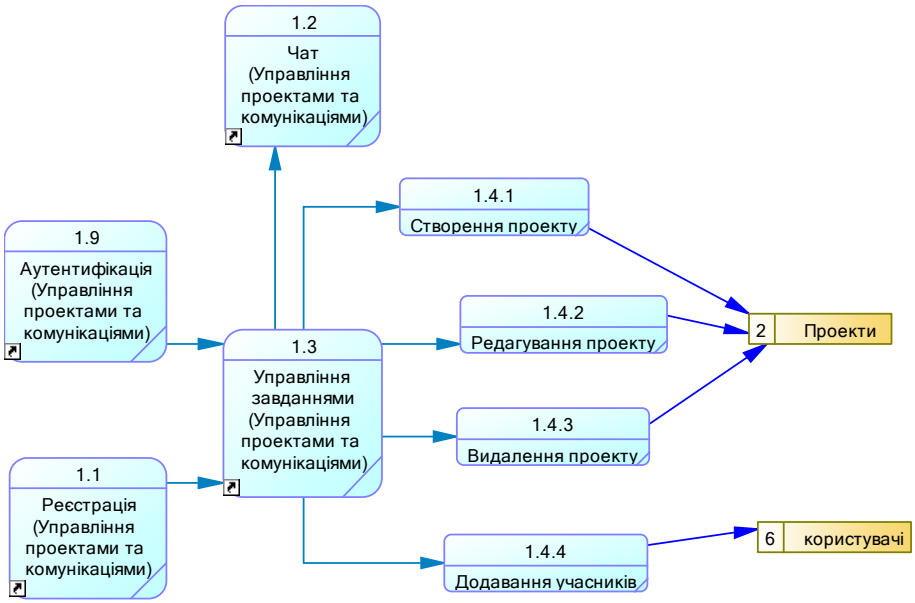

Рис. 2.11 Другий рівень DFD-діаграми

2.5 Висновки до розділу 2

У другому розділі було розглянуто методологічні та технічні засади побудови соціальної інтернет-платформи для спільної роботи над проєктами. Проведено аналіз сучасних підходів до розробки інтерфейсів, обґрунтовано вибір стеку технологій, а також описано архітектуру та інформаційну модель системи. Розглянуто ключові принципи UI/UX-дизайну, серед яких — спрощення інтерфейсу, адаптивність, контекстна навігація, мінімізація когнітивного навантаження. Наголошено, що ефективна взаємодія в цифровому середовищі потребує не тільки функціональності, а й логіки, візуальної ієрархії та модульності компонентів. Підтверджено важливість застосування адаптивної сітки, шаблонів та патернів компонентного підходу при створенні масштабованого інтерфейсу.

Вибір **frontend-частини** (React + TypeScript) обґрунтовано потребою у швидкому, компонентно-орієнтованому, типобезпечному інтерфейсі, здатному до реактивного оновлення даних у реальному часі. Застосування сучасних бібліотек (React Hook Form, TailwindCSS, Axios, Context API) дозволяє оптимізувати розробку, зменшити дублювання логіки й забезпечити високий рівень масштабованості.

Backend-середовище, побудоване на Node.js та Express.js, дозволяє реалізувати неблокуючу обробку запитів, зручно працювати з REST API, забезпечити ефективну маршрутизацію, а також реалізувати модульну структуру серверної логіки з підтримкою middleware, логування, авторизації та масштабування.

Для **збереження даних** обрано реляційну СКБД PostgreSQL, що забезпечує підтримку складних зв’язків, транзакційність, цілісність та масштабованість. Розглянуто структуру таблиць, способи забезпечення зв’язків (foreign keys, M:N, 1:N), а також підтримку сучасних типів даних (JSONB, індекси).

Описано архітектуру системи у вигляді багаторівневої клієнт-серверної моделі, що забезпечує розділення відповідальностей, модульність і стабільність. Особливу увагу приділено взаємодії через REST API, авторизації за допомогою JWT та підключенню до WebSocket для реалізації чату в реальному часі.

Інформаційна модель доповнюється **структурною, даталогічною та DFD-моделями**, які деталізують функціональну, соціальну, юридичну та інформаційну взаємодію компонентів системи. Це дозволяє не лише реалізувати логіку, а й документувати її для подальшої підтримки, масштабування та тестування.

Таким чином, сформовано цілісну методологічну й технічну основу для реалізації ефективної, безпечної, гнучкої та масштабованої платформи для командної роботи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура програмного забезпечення

Розроблена платформа реалізована як клієнт-серверна веб-система, де клієнтська та серверна частини є окремими проєктами, що взаємодіють через API-запити. Уся структура програмного забезпечення побудована за принципами модульності, зрозумілого розділення відповідальностей та зручності підтримки коду.

Програмна система містить такі основні компоненти:

- **Frontend (React + TypeScript)** — клієнтська частина, що відповідає за взаємодію з користувачем, відображення даних, надсилання запитів до сервера, маршрутизацію між сторінками та реактивну поведінку.
- **Backend (Node.js + Express.js)** — серверна частина, що обробляє API-запити, керує авторизацією, перевіркою прав доступу, взаємодіє з базою даних та WebSocket.
- **База даних (PostgreSQL)** — зберігає інформацію про користувачів, проєкти, задачі, коментарі, файли та повідомлення.

Структура клієнтської частини (frontend)

Клієнтська частина вебзастосунку реалізована з використанням сучасної бібліотеки **React**, яка забезпечує побудову компонентного інтерфейсу, що динамічно оновлюється без перезавантаження сторінки. Такий підхід дозволяє створювати швидкі, масштабовані та зручні у використанні вебінтерфейси.

Для забезпечення зрозумілої структури та полегшення подальшої підтримки і розвитку проєкту, було обрано модульну архітектуру. Всі вихідні файли клієнтської частини розміщено в директорії react_todo-app/, яка містить окремі підкаталоги для різних типів логіки: сторінок, компонентів, контекстів, сервісів, утиліт тощо.

Такий підхід дозволяє розділити логіку за принципом призначення: повноекранні сторінки зберігаються окремо від багаторазових UI-елементів, глобальні контексти винесені для централізованого управління станом, а логіка авторизації, взаємодії з API та службових функцій чітко структурована (табл 3.1):

Таблиця 3.1

Структура клієнтської частини

Директорія файл	**Призначення**
src/pages/	Повноекранні сторінки (реєстрація, проєкти, деталі проєкту, чат)
src/components/	Повторно використовувані UI-компоненти (кнопки, картки, списки, форми)
src/context/	Глобальні контексти, зокрема UserContext для зберігання авторизації
src/services/	Функції для виконання API-запитів (наприклад, projectService.ts)
src/styles/	SCSS/TailwindCSS стилі
src/App.tsx	Основна точка входу з маршрутизацією
src/index.tsx	Ініціалізація React-програми

Структура серверної частини (backend)

Серверна частина застосунку реалізована з використанням середовища виконання **Node.js** у поєднанні з фреймворком **Express.js**, який забезпечує зручну маршрутизацію, обробку HTTP-запитів і модульну побудову логіки. Весь код backend-системи зберігається у власному репозиторії або окремій директорії проєкту (наприклад, server/), що дозволяє ефективно відокремити серверну логіку від клієнтської частини.

Архітектура побудована за принципом **розділення відповідальностей (Separation of Concerns)**, що забезпечує логічне групування функціональності по папках: маршрути (routes/), контролери (controllers/) та моделі (models/). Така організація структури дозволяє легко масштабувати застосунок, доповнювати нові модулі, здійснювати налагодження та проводити модульне тестування окремих частин (табл 3.2):

Таблиця 3.2

Структура серверної частини

Директорія / файл	**Призначення**
routes/	Оголошення маршрутів для projects, tasks, users, auth, chat
controllers/	Основна бізнес-логіка: обробка запитів, формування відповіді
models/	SQL-моделі або ORM-моделі для PostgreSQL
services/	Робота з файлами, хелпери, перевірка доступу
index.js	Точка запуску сервера
.env	Конфігураційні змінні (ключі, URI бази тощо)

Загальна схема взаємодії:

1. Користувач взаємодіє з інтерфейсом на React.
2. Компоненти фронтенду надсилають HTTP-запити до API (Express.js).
3. Сервер перевіряє права доступу, обробляє запит і звертається до PostgreSQL.
4. У разі чату або оновлень у реальному часі активується WebSocket-подія.
5. Відповідь передається назад користувачеві та відображається в інтерфейсі.

3.2 Реалізація ключових модулів

Розробка соціальної інтернет-платформи для спільних проєктів передбачає створення низки функціональних модулів, які забезпечують базову та розширену взаємодію між користувачами системи. У цьому підпункті наведено реалізацію найбільш важливих із них: авторизації, роботи з проєктами, задачами, чатом та системою прав доступу. Кожен із модулів реалізовано з урахуванням сучасних вимог до безпеки, зручності користування та масштабованості функціоналу. Також реалізація модулів забезпечує цілісну інтеграцію між клієнтською та серверною частинами системи.

Авторизація та реєстрація користувачів

Для забезпечення початкового контролю доступу до платформи реалізовано базовий модуль авторизації, що дозволяє користувачам реєструватися в системі та

входити до особистого кабінету. На даному етапі реєстрація та логін реалізовані через збереження та перевірку даних у базі PostgreSQL.

На клієнтській частині використовуються стандартні форми, що передають дані до backend через API-запити. На серверній стороні реалізовано контролер authController.ts, що включає дві основні функції:

- **registerUser** — перевіряє, чи не зареєстрований уже користувач з вказаною електронною поштою, і, якщо ні, додає його до таблиці users (рис 3.1):

```
export const registerUser = async (req: Request, res: Response) => {
  const { name, email, password } = req.body;

  try {
    const existing = await pool.query("SELECT * FROM users WHERE email = $1", [email]);
    if (existing.rows.length > 0) {
      return res.status(400).json({ message: "Користувач з таким email вже існує" });
    }

    const result = await pool.query(
      "INSERT INTO users (name, email, password) VALUES ($1, $2, $3) RETURNING *",
      [name, email, password]
    );

    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error("Register error:", err);
    res.status(500).json({ message: "Помилка при реєстрації" });
  }
};
```

Рис. 3.1 Функція реєстрування

- **loginUser** — виконує перевірку пари email/пароль у таблиці users. Якщо користувача знайдено, API повертає його дані у форматі JSON (рис 3.2):

```
export const loginUser = async (req: Request, res: Response) => {
  const { email, password } = req.body;

  try {
    const result = await pool.query("SELECT * FROM users WHERE email = $1 AND password = $2", [email, password]);

    if (result.rows.length === 0) {
      return res.status(401).json({ message: "Невірний email або пароль" });
    }

    res.json(result.rows[0]);
  } catch (err) {
    console.error("Login error:", err);
    res.status(500).json({ message: "Помилка при вході" });
  }
};
```

Рис. 3.2 Функція входу до акаунту

Управління проєктами

Користувач може створювати нові проєкти, редагувати їх, додавати або видаляти учасників. На сервері проєкти мають унікальний id, назву, опис, тип доступу (open, private, byRequest), дату створення та посилання на автора.

REST API приклади:

- POST /projects — створення нового проєкту
- GET /projects/:id — отримання інформації про проєкт
- PATCH /projects/:id/access — редагування (для автора)
- DELETE /projects/:id — запит на приєднання

Функціональність керування проєктами реалізована як окремий модуль backend-системи, що взаємодіє з frontend-інтерфейсом через REST API.. Ключовим є контроль прав доступу: лише автор проєкту має право його змінювати або видаляти.

Функція створення проєкту дозволяє авторизованому користувачу сформувати базову одиницю колективної роботи — проєкт, у межах якого надалі створюються задачі, додаються учасники, ведеться чат, прикріплюються файли тощо. Створення виконується через POST-запит з передачею JSON-об’єкта, що містить title, description, authorId (рис 3.3).

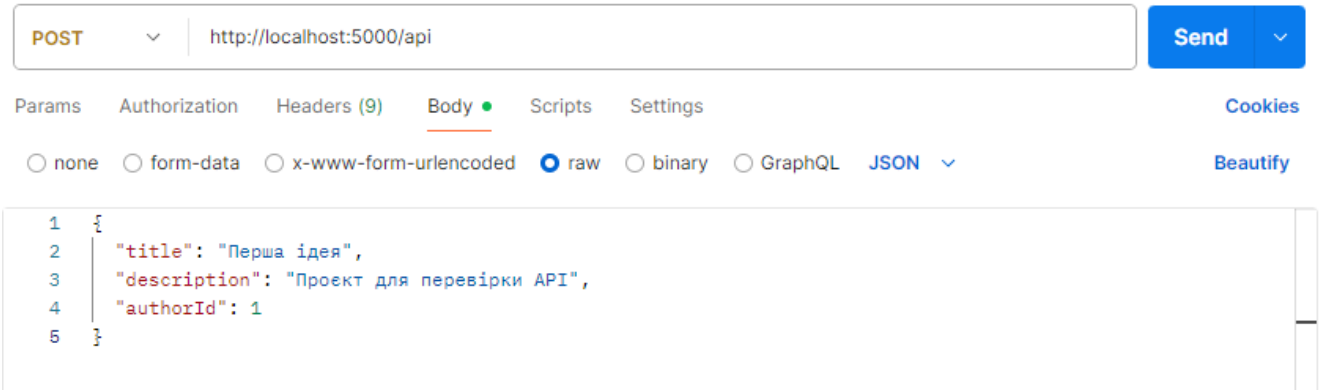

Рис. 3.3 Функція створення проєкту

Після обробки сервером проєкт зберігається в базі з унікальним ID та полем created_at. Якщо запит валідний, повертається статус 201 Created і об’єкт нового проєкту.

Кожен користувач, який має доступ до проєкту, може переглядати його основну інформацію. Це включає ID, назву, опис, автора, дату створення та тип доступу (access_type) (рис 3.4).

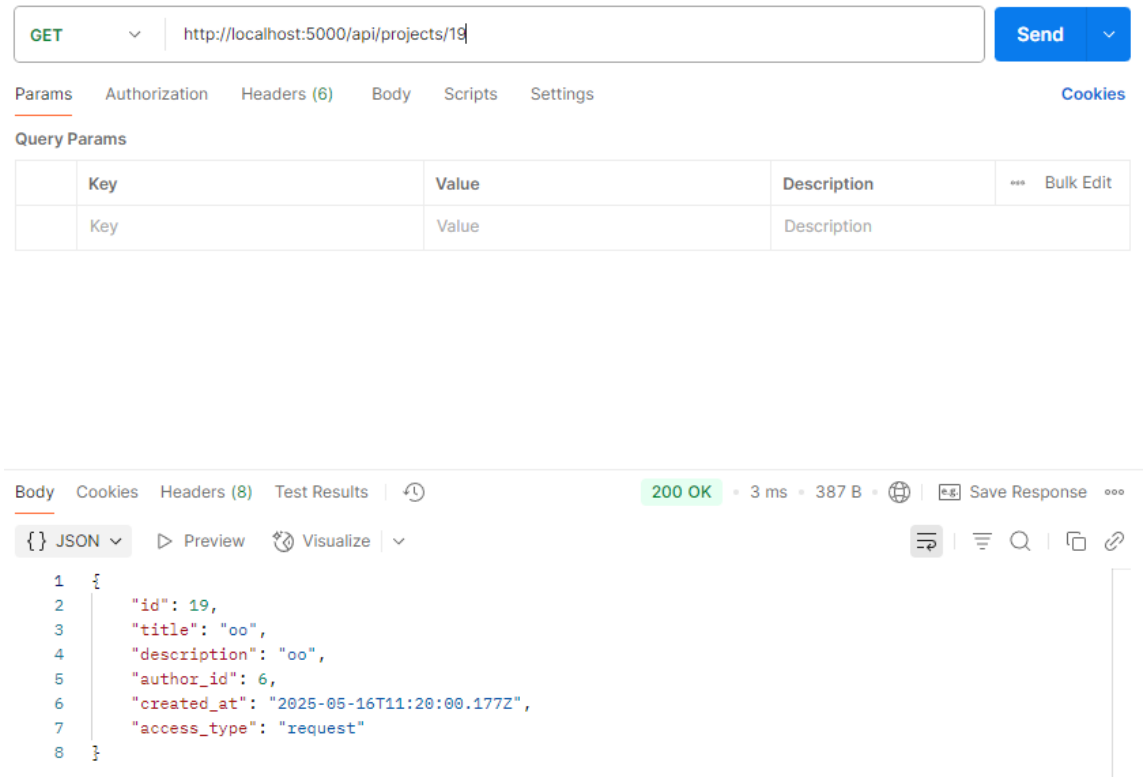

Рис. 3.4 Функція перегляду проєкту

У відповіді сервер повертає об’єкт проєкту. Це використовується у фронтенді для відображення сторінки проєкту (ProjectDetailPage), де також можуть бути вкладки: «Учасники», «Задачі», «Файли», «Чат».

Функція оновлення дозволяє автору змінювати назву або опис проєкту, наприклад, при уточненні цілей або уточненні контексту завдань. Метод PUT оновлює всі поля, передані в тілі запиту (рис 3.5).

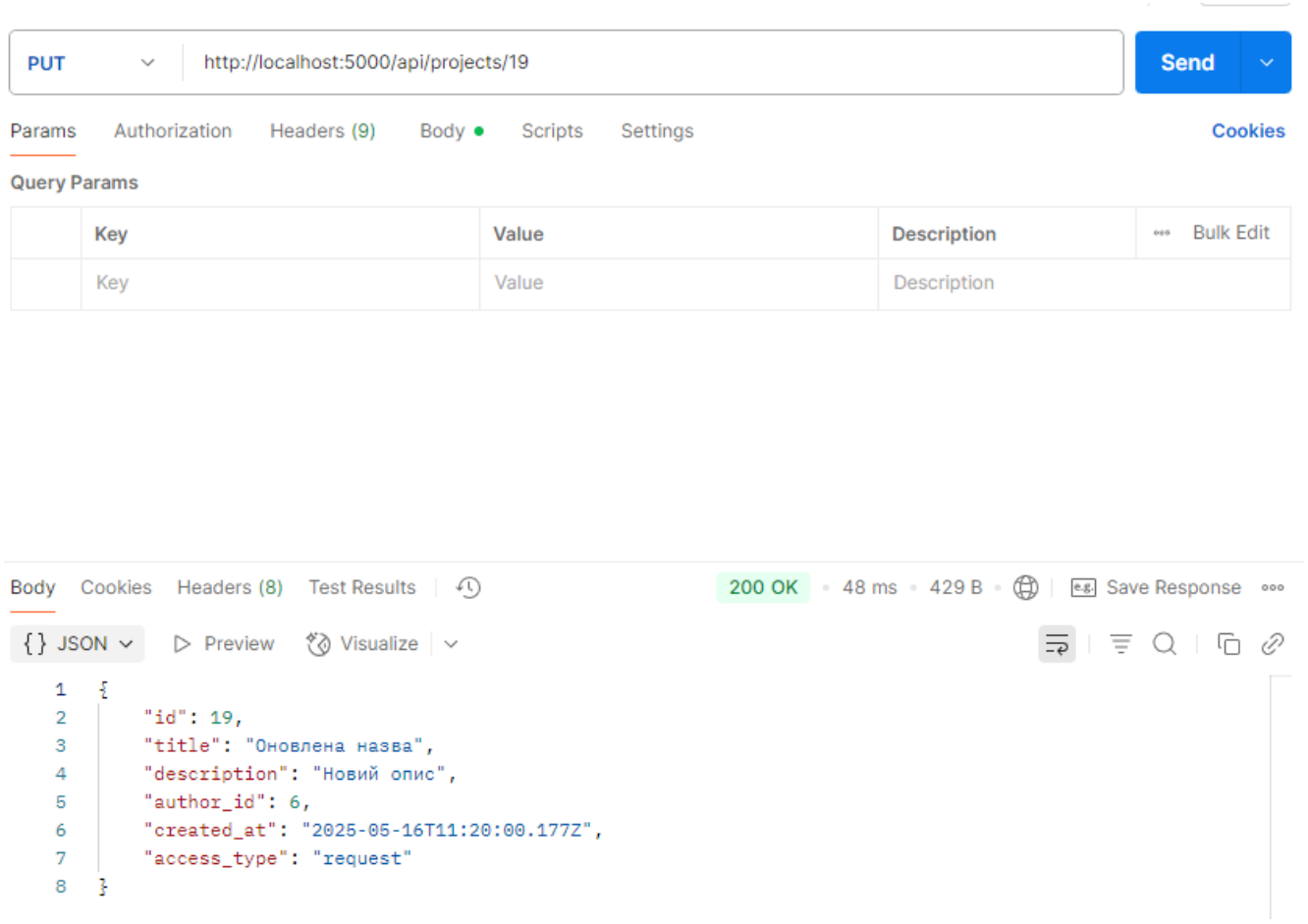

Рис. 3.5 Функція оновлення проєкту

У відповіді надходить оновлений об'єкт. Типовим варіантом на фронтенді є автоматичне оновлення інтерфейсу без перезавантаження (через React state).

Модуль також підтримує повне видалення проєкту (рис 3.6). Ця операція доступна лише автору. Після запиту система видаляє проєкт та пов'язані сутності, залежно від реалізації – задачі, запити, чат тощо.

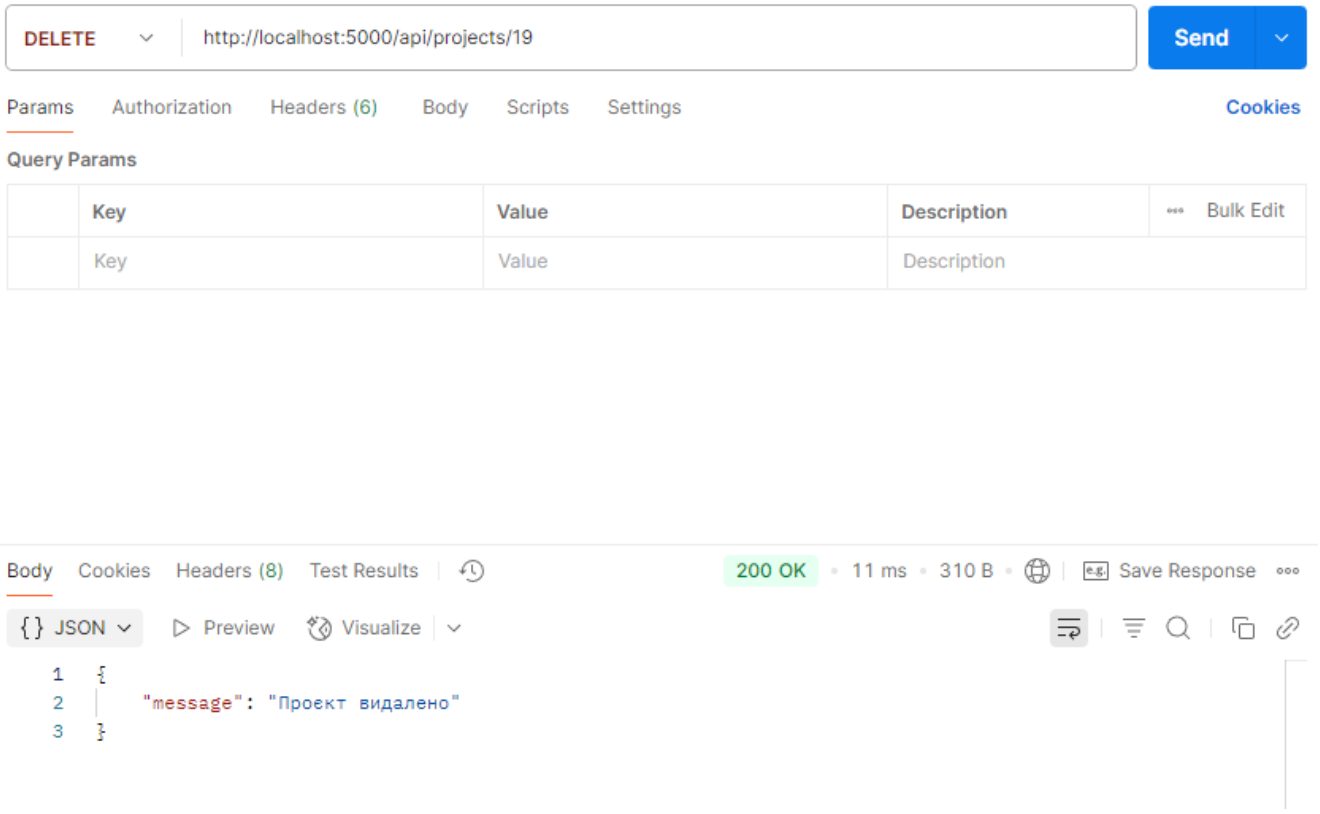

Рис. 3.6 Функція видалення проєкту

У відповіді повертається статус 200 OK і повідомлення "Проєкт видалено", що підтверджує успішне виконання операції. Це викликає на клієнті оновлення списку проєктів або навігацію назад.

Модуль задач

Модуль задач реалізує функціональність планування, відстеження та оновлення завдань у межах певного проєкту. Кожна задача створюється з прив’язкою до проєкту, має унікальний ID, текстовий опис, статус (done або not done), дату створення, а також може бути призначена конкретному користувачу.

Задача створюється на сторінці відповідного проєкту. Користувач заповнює форму, що містить текст задачі, дедлайн (за потреби), та натискає «Створити». У результаті надсилається POST-запит до API, який додає новий запис до бази даних (рис 3.7).

У відповіді повертається створена задача з полями id, text, done, project_id, created_at. Значення поля done за замовчуванням — false.

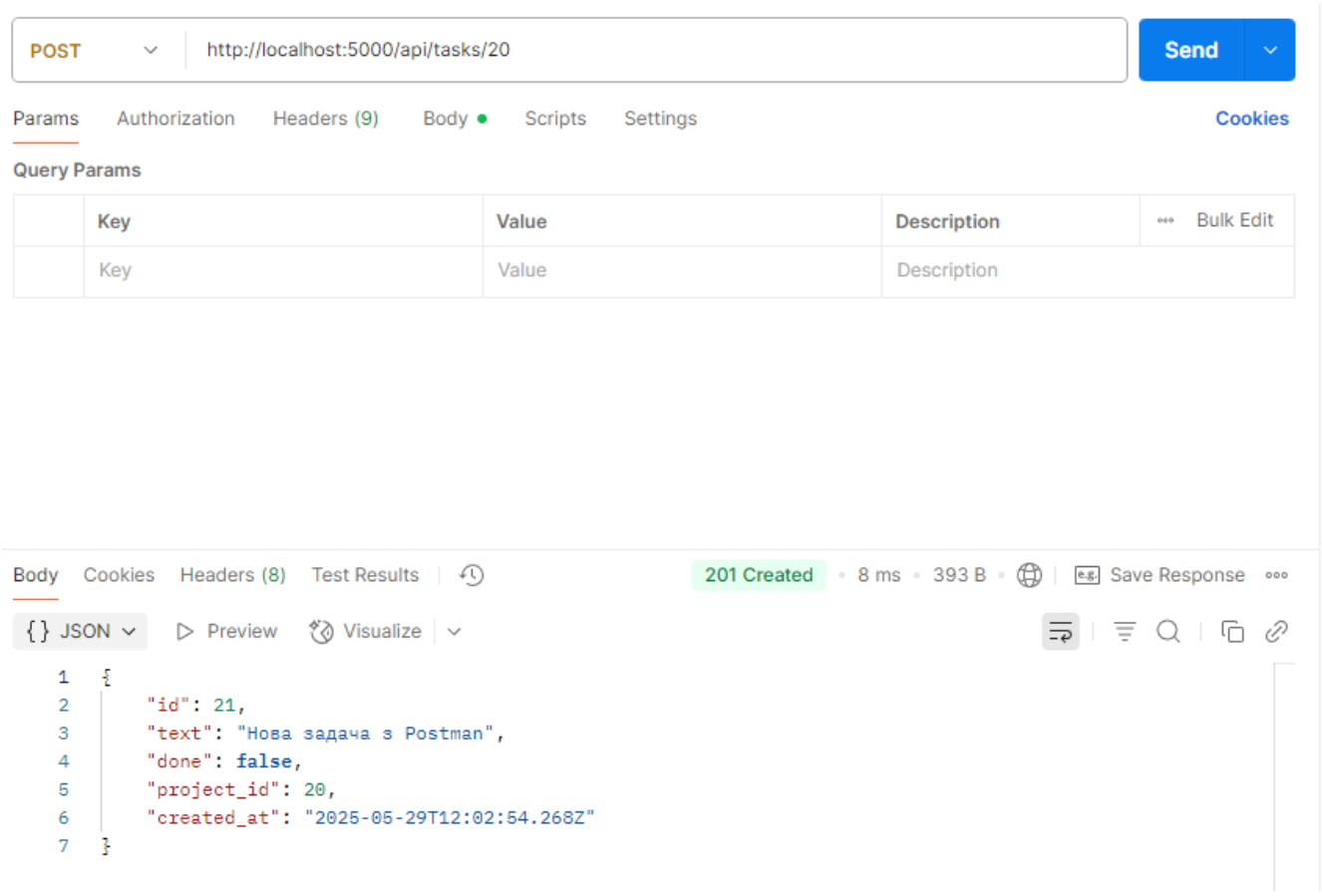

Рис. 3.7 – Створення задачі у проєкті через Postman

Сторінка перегляду задач відображає список усіх завдань, що прив’язані до конкретного проєкту. Для цього використовується GET-запит до API з передачею ID проєкту (рис 3.8).

Сервер повертає масив задач, кожна з яких містить ID, текст, статус done, дату створення та project_id. На клієнті ці дані рендеряться у вигляді списку карток.

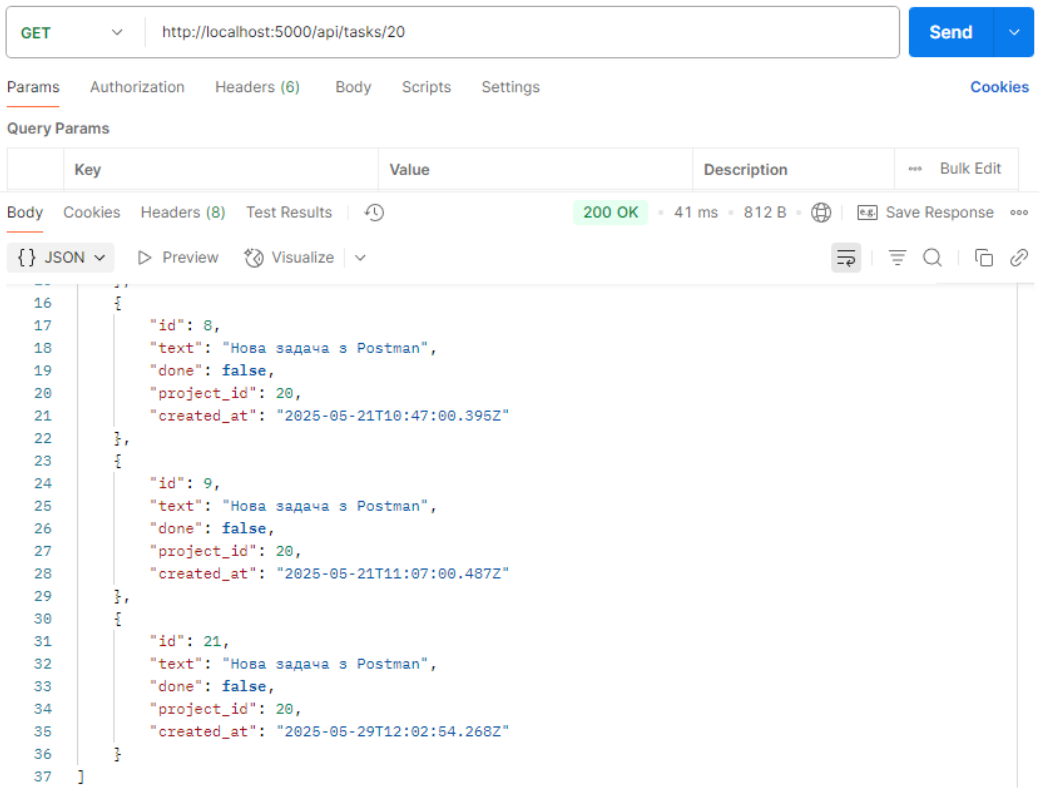

Рис. 3.8 – Отримання всіх задач проєкту через API

Користувач може оновити текст задачі або відзначити її як виконану (done: true). Для цього надсилається PATCH-запит на відповідну задачу за ID (рис 3.9).

Після оновлення сервер повертає змінену версію задачі. Ці зміни одразу відображаються у React-компонентах на сторінці, завдяки автоматичному оновленню state.

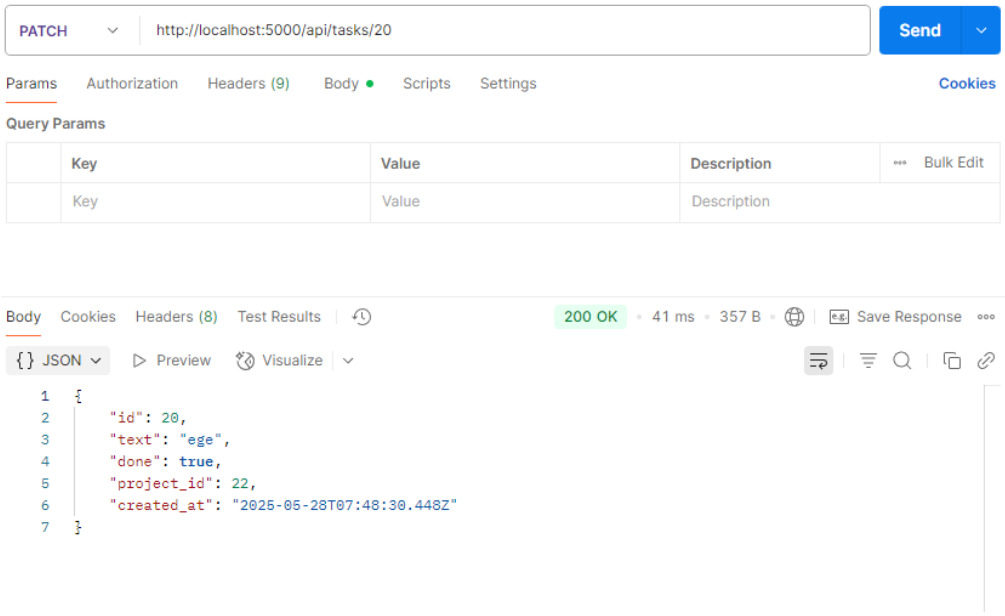

Рис. 3.8 – Оновлення задачі через API-запит у Postman

Модуль чату

Модуль чату забезпечує можливість спілкування користувачів у межах конкретного проєкту. Повідомлення зберігаються в базі даних, і кожне з них містить ID проєкту, ID користувача, текст повідомлення, дату створення, а також метаінформацію про відправника (ім’я, аватар).

На даному етапі чат реалізований як REST API без WebSocket-підтримки, але вже дозволяє:

- отримувати історію повідомлень для кожного проєкту;
- надсилати нові повідомлення;
- зберігати повідомлення в базі;
- збагачувати відповіді через JOIN-зв’язки із таблицею users.

Кожен учасник проєкту може отримати всю історію повідомлень за допомогою GET-запиту до API. У відповіді сервер повертає масив повідомлень із сортуванням за часом публікації (рис 3.10).

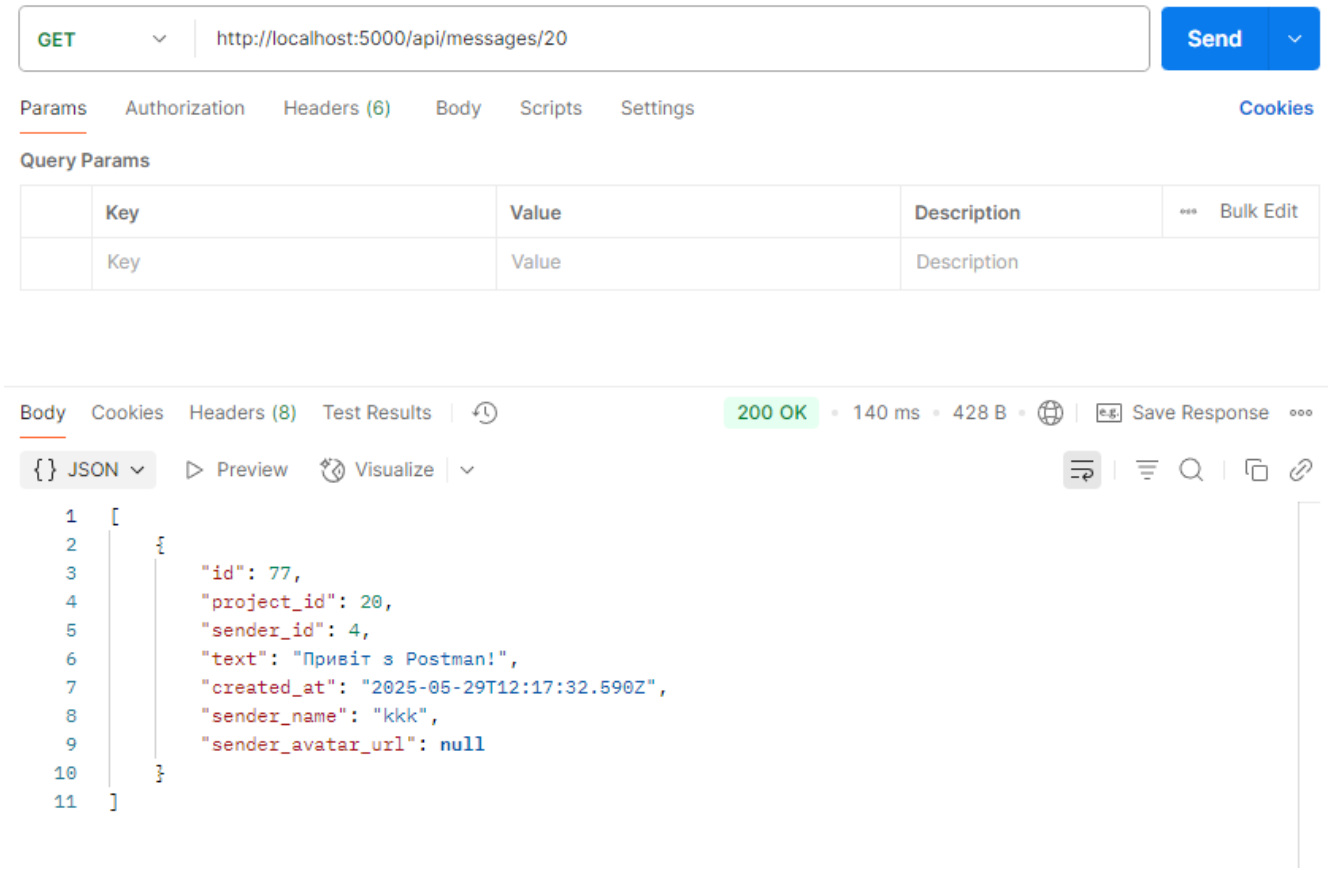

Рис. 3.10 Отримання повідомлень через Postman

Коли користувач надсилає повідомлення, запит типу POST передає sender_id і text. У базу записується нове повідомлення, прив’язане до проєкту (рис 3.11).

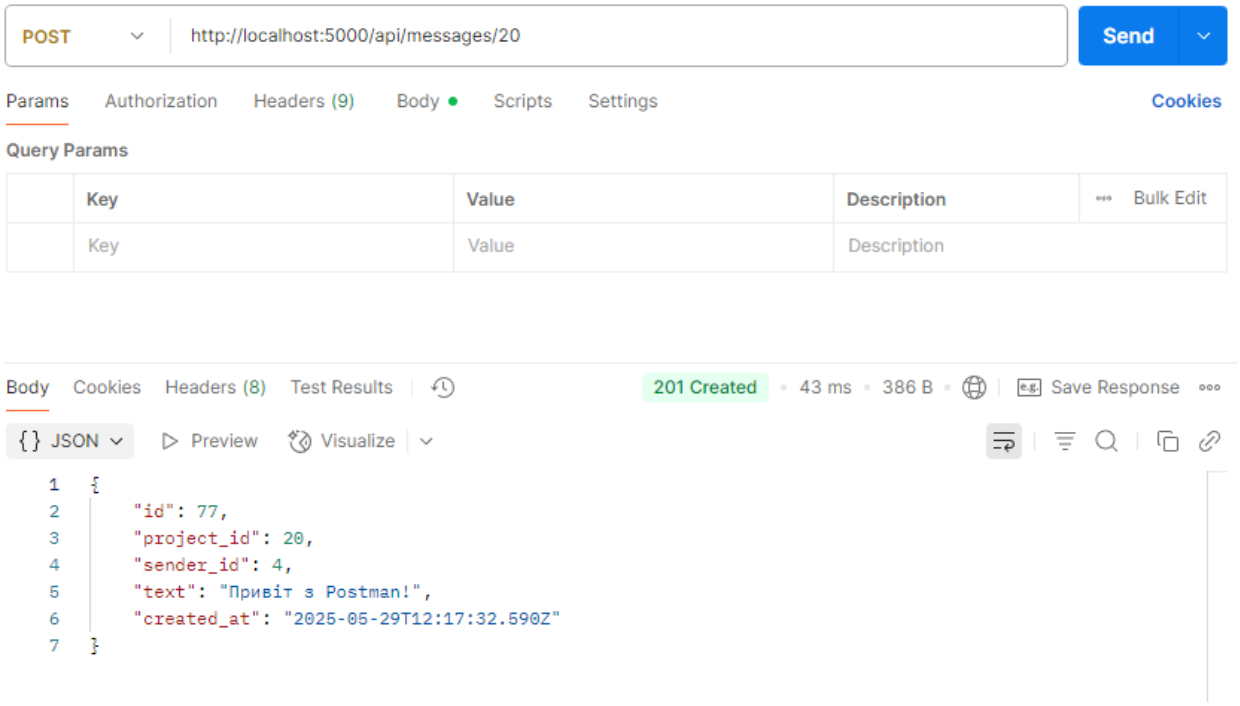

Рис. 3.11 – Надсилання повідомлення через Postman

Всі маршрути винесені в окремий модуль messageRoutes.ts (рис 3.12).

```
import express from "express";
import * as controller from "../controllers/messageController";

const router = express.Router();

router.get("/:projectId", controller.getMessages);
router.post("/:projectId", controller.sendMessage);

export default router;
```

Рис. 3.12 – Маршрути модуля чату

Таким чином, реалізований REST-модуль чату дозволяє вести базову текстову комунікацію між учасниками проєкту. У майбутньому він може бути доповнений підтримкою WebSocket для передачі повідомлень у режимі реального часу, а також функціями видалення/редагування повідомлень та прикріплення файлів.

Керування доступом до проєкту

Для гнучкого регулювання рівнів доступу до проєкту реалізовано функцію зміни типу доступу. Кожен проєкт у базі має поле access_type, яке може набувати одного з таких значень:

- "open" – будь-хто може приєднатися;
- "request" – приєднання потребує підтвердження автора;
- "closed" – доступ лише за прямим запрошенням.

Змінювати тип доступу може лише автор проєкту, через PUT-запит до API.

Контролер updateAccessType перевіряє коректність переданого значення (access_type) та викликає відповідну функцію моделі. Цей механізм дозволяє ефективно контролювати склад учасників та рівень відкритості кожного окремого проєкту. Якщо значення некоректне або проєкт не знайдено, API повертає повідомлення про помилку (рис 3.13).

```
export const updateAccessType = async (req: Request, res: Response) => {
  const { id } = req.params;
  const { access_type } = req.body;

  if (!["open", "closed", "request"].includes(access_type)) {
    return res.status(400).json({ message: "Неприпустиме значення доступу" });
  }

  try {
    const updated = await projectModel.updateAccessType(Number(id), access_type);
    if (!updated) {
      return res.status(404).json({ message: "Проєкт не знайдено" });
    }
    res.json(updated);
  } catch (err) {
    console.error("Update access type error:", err);
    res.status(500).json({ message: "Помилка при оновленні доступу" });
  }
};
```

Рис. 3.13 – Контролер оновлення типу доступу

Оновлення доступу (модель)

У моделі відбувається запит до бази PostgreSQL, який оновлює поле access_type в таблиці projects (рис 3.14).

```
export const updateAccessType = async (id: number, access_type: string) => {
  const res = await pool.query(
    "UPDATE projects SET access_type = $1 WHERE id = $2 RETURNING *",
    [access_type, id]
  );
  return res.rows[0];
};
```

Рис. 3.14 – SQL-запит у моделі updateAccessType

Оновлений маршрут PUT /api/projects/:id/access підключений до контролера та дозволяє змінити тип доступу для конкретного проєкту (рис 3.15).

```
1   import { Router } from "express";
2   import * as projectController from "../controllers/projectController";
3
4   const router = Router();
5
6   router.get("/", projectController.getProjects);
7   router.post("/", projectController.create);
8   router.put("/:id", projectController.update);
9   router.delete("/:id", projectController.remove);
10  router.get("/:id", projectController.getOne);
11  router.put("/:id/access", projectController.updateAccessType);
12  export default router;
```

Рис. 3.15 – Маршрут зміни доступу до проєкту у projectRoutes.ts

Запити на приєднання до проєкту

У проєктах із доступом типу "request" реалізована система керування запитами на приєднання. Користувач, який хоче приєднатися до такого проєкту, надсилає запит, який повинен підтвердити автор. Ця логіка реалізована через окрему таблицю project_join_requests, контролери, моделі та маршрути.

Користувач надсилає запит через POST-метод, передаючи user_id, project_id, а також додаткове повідомлення до автора проєкту. Якщо в базі вже є запит у статусі pending, створення буде відхилено (рис 3.16).

```
export const createJoinRequest = async (req: Request, res: Response) => {
  const { user_id, project_id, message } = req.body;

  try {
    const existing = await model.getExistingPendingRequest(user_id, project_id);
    if (existing) {
      return res.status(400).json({ message: "Ви вже подали запит" });
    }

    const request = await model.createJoinRequest(user_id, project_id, message);
    res.status(201).json(request);
  } catch (err) {
    console.error("createJoinRequest error:", err);
    res.status(500).json({ message: "Не вдалося створити запит" });
  }
};
```

Рис. 3.16 – Контролер createJoinRequest

Автор проєкту може переглянути всі вхідні запити до своїх проєктів через GET /author/:authorId. У відповіді повертається масив запитів з даними користувача та назвою проєкту (рис 3.17).

```
export const getRequestsForAuthor = async (req: Request, res: Response) => {
  const { authorId } = req.params;
  try {
    const requests = await model.getJoinRequestsByAuthor(Number(authorId));
    res.json(requests);
  } catch (err) {
    console.error("getRequestsForAuthor error:", err);
    res.status(500).json({ message: "Помилка при отриманні запитів" });
  }
};
```

Рис. 3.17 – SQL-запит getJoinRequestsByAuthor

Автор може підтвердити або відхилити запит через PUT /:id, передавши новий статус (approved або rejected). У випадку підтвердження, користувача автоматично додають до зв’язку user_projects (рис 3.18).

```
export const updateRequestStatus = async (req: Request, res: Response) => {
  const { id } = req.params;
  const { status } = req.body;

  if (!["approved", "rejected"].includes(status)) {
    return res.status(400).json({ message: "Недопустимий статус" });
  }

  try {
    const updated = await model.updateJoinRequestStatus(Number(id), status);

    // якщо підтверджено — додати до user_projects
    if (updated && status === "approved") {
      await userProjectModel.joinProject(updated.user_id, updated.project_id);
    }

    res.json(updated);
  } catch (err) {
    console.error("updateRequestStatus error:", err);
    res.status(500).json({ message: "Помилка при оновленні статусу" });
  }
};
```

Рис. 3.18 – Контролер updateRequestStatus

Користувач може переглянути список своїх запитів через GET /user/:userId. Це дозволяє бачити, які запити ще не розглянуто, а які відхилено або підтверджено (рис 3.19).

```
export const getRequestsByUser = async (req: Request, res: Response) => {
  const { userId } = req.params;
  try {
    const result = await pool.query(
      `SELECT * FROM project_join_requests
       WHERE user_id = $1
       ORDER BY created_at DESC`,
      [userId]
    );
    res.json(result.rows);
  } catch (err) {
    console.error("getRequestsByUser error:", err);
    res.status(500).json({ message: "Помилка при отриманні запитів користувача" });
  }
};
```

Рис. 3.19 – Контролер getRequestsByUser

Переваги реалізації:

- Запобігання дублю запитів.
- Автоматичне додавання учасника після підтвердження.
- Зручне сортування за автором або користувачем.
- Розширюваність — можливість додати причину відхилення, сповіщення тощо.

Вся логіка запитів на приєднання до проєктів реалізована у вигляді окремої групи маршрутів у backend-системі. Вона дозволяє створювати, переглядати та змінювати статуси запитів як зі сторони користувача, так і зі сторони автора проєкту.

Створення запиту на приєднання. Користувач надсилає POST-запит із параметрами: user_id, project_id, message. Система перевіряє, чи не було вже надіслано запит у статусі pending (рис 3.20).

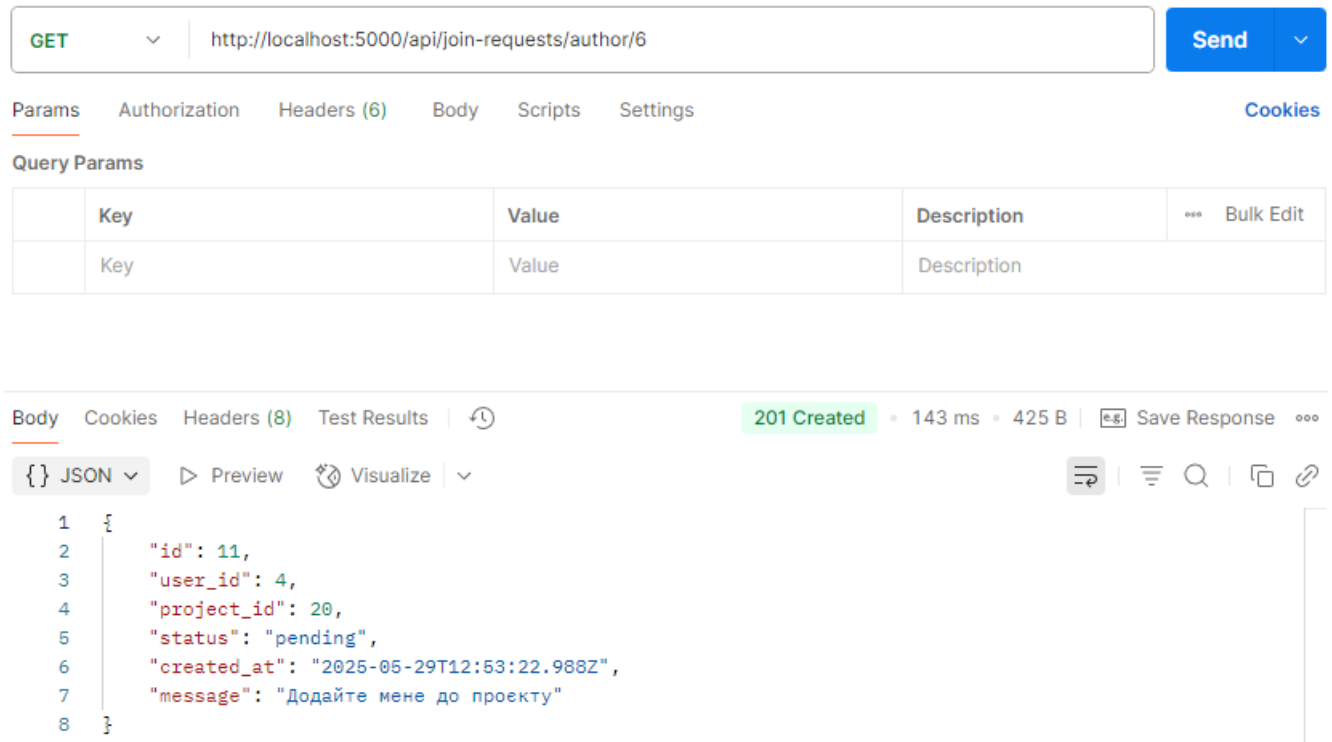

Рис. 3.20 – Створення запиту на приєднання (POST /api/join-requests)

201 Created – повертає створений запит із ID, статусом, датою. Автор проєкту може побачити всі запити до своїх проєктів через GET-запит із передачею свого authorId (рис 3.21).

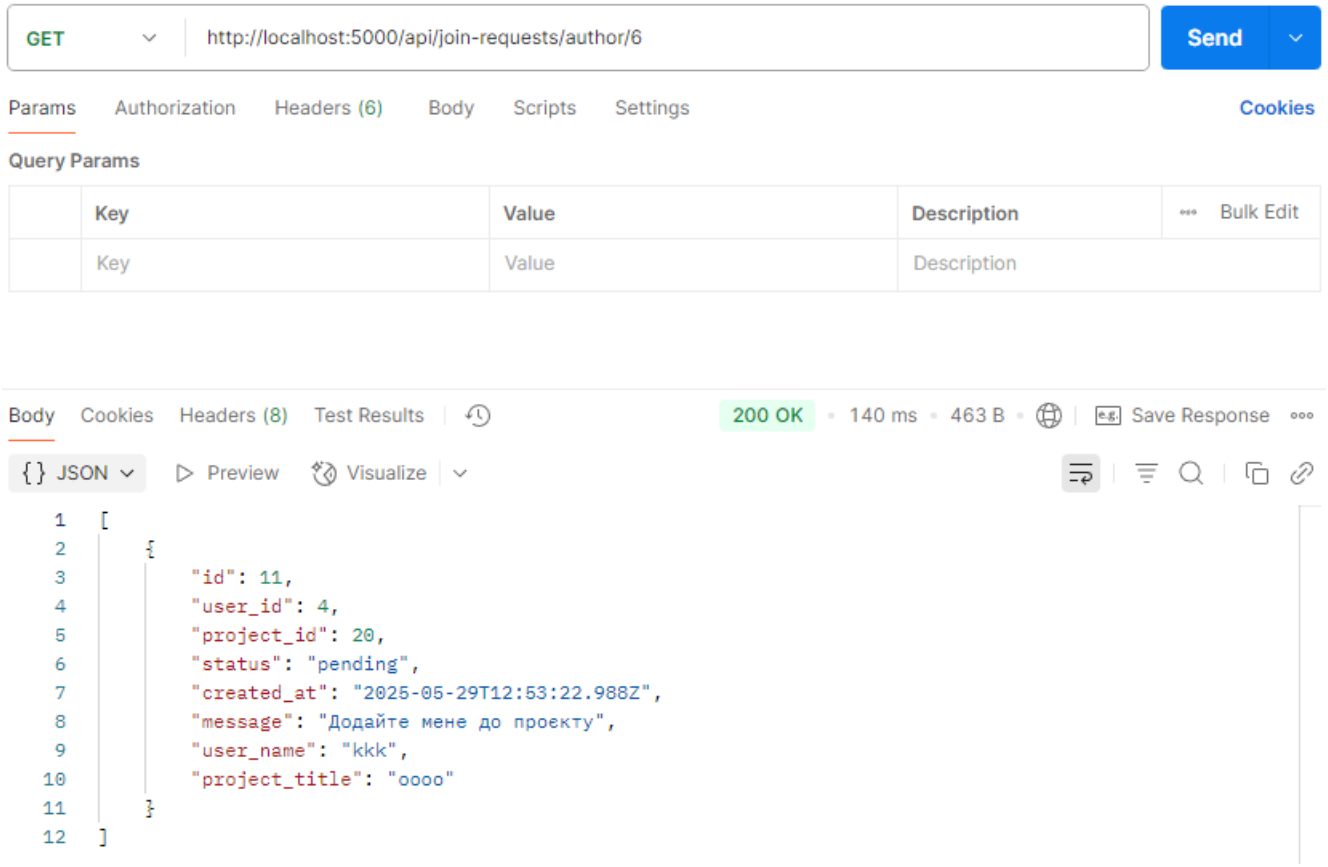

Рис. 3.21 – Отримання запитів для автора (GET /api/join-requests/author/6)

У відповіді: ім’я користувача, назва проєкту, статус, дата та повідомлення. Автор має можливість підтвердити ("approved") або відхилити ("rejected") запит. У разі підтвердження користувача додають у проєкт автоматично (рис 3.22).

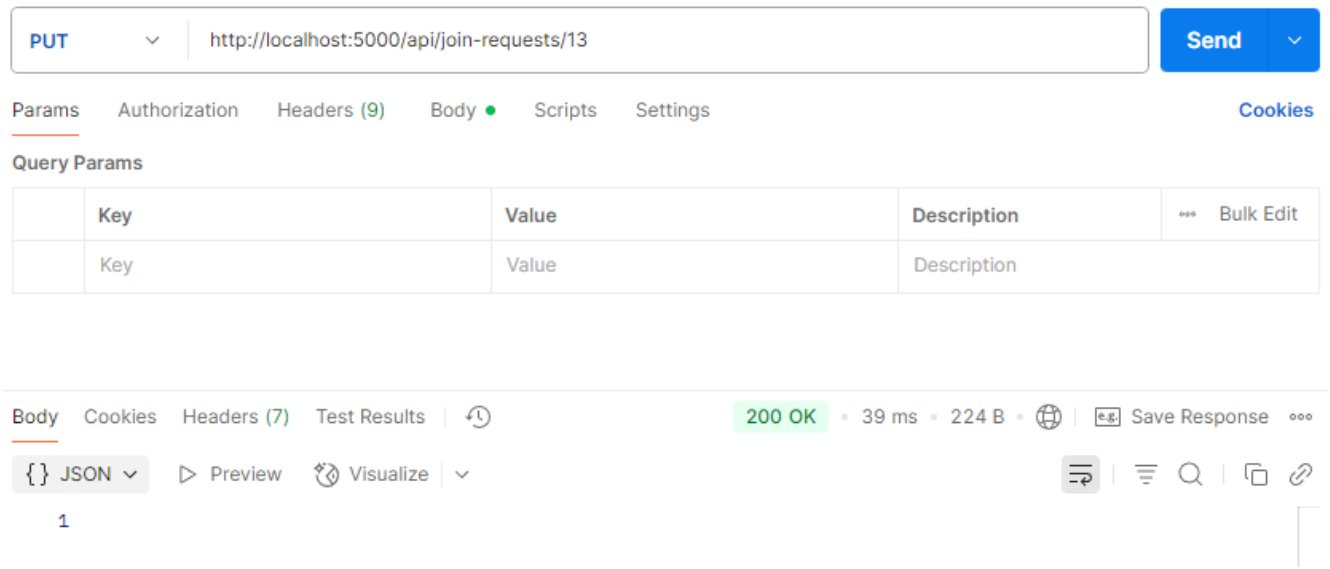

Рис. 3.22 – Оновлення статусу запиту (PUT /api/join-requests/:id)

200 OK – повертає оновлений запит. Кожен користувач може переглядати всі свої запити, включаючи статус і час подачі, що дозволяє стежити за перебігом розгляду (рис 3.23).

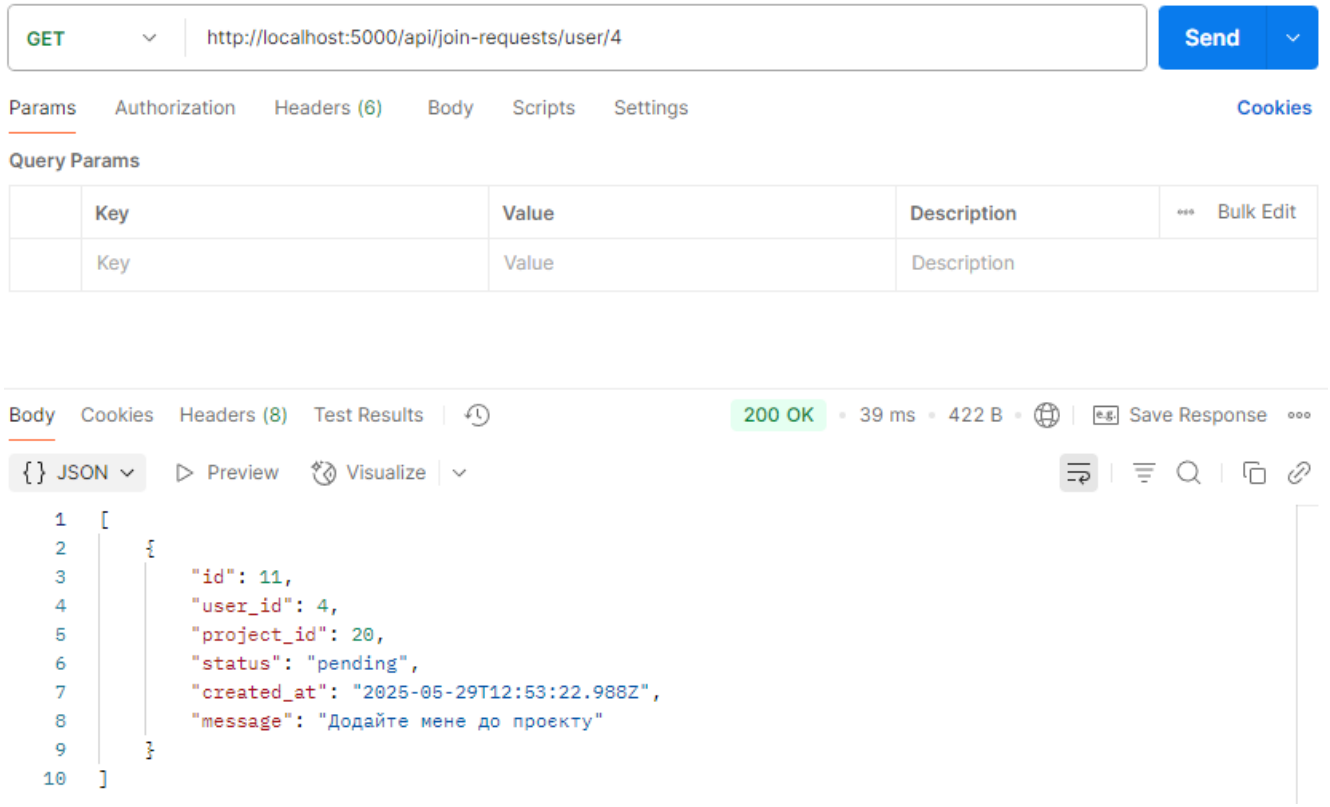

Рис. 3.23 – Отримання власних запитів (GET /api/join-requests/user/4)

Відповідь — масив об’єктів із status, message, created_at. Таким чином, маршрути API для запитів підтримують:

- Контроль дій із обох сторін (користувач/автор);
- Неповторюваність запитів (перевірка на pending);
- Автоматичне додавання учасника при підтвердженні;
- Прозоре логування всіх запитів для обох сторін.

3.3 Інтерфейс користувача

Інтерфейс користувача платформи розроблений з урахуванням принципів простоти, інтуїтивності та адаптивності. Він реалізований на основі React з використанням TailwindCSS, що дозволяє створити легкий, гнучкий і сучасний дизайн. Усі ключові дії користувача — авторизація, керування проєктами, робота з задачами та спілкування — зведені до чітких візуальних форм, з акцентом на зручність.

Навігаційна панель (Navbar). Верхнє меню є постійно доступним для навігації між основними сторінками:

- Головна, Проєкти, Створити, Запити, Профіль (рис 3.24).
- Праворуч – ім’я користувача, аватар та кнопка «Вийти» (рис 3.25).

Це дозволяє швидко переміщуватись між модулями системи незалежно від того, на якій сторінці перебуває користувач.

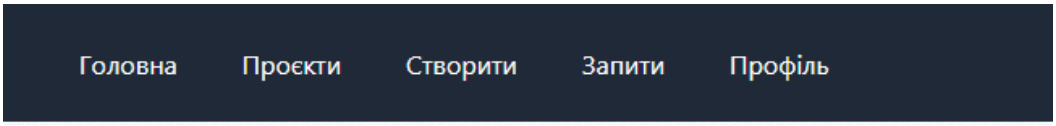

Рис. 3.24 Ліва сторона навігації

Рис. 3.25 Права сторна навігації

Авторизація користувача. Проста форма входу містить поля:

- Email

- Пароль

Синя кнопка "Увійти" викликає запит до API та зберігає токен. Стилістично форма вписується в загальний дизайн: біла картка, центрована, з великим заголовком (рис 3.26).

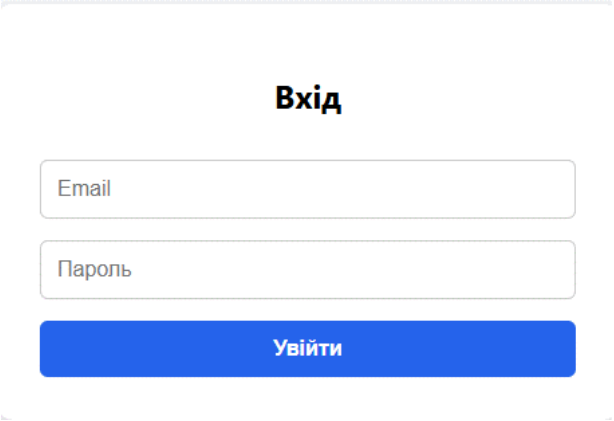

Рис. 3.26 Авторизація користувача

Реєстрація користувача.

Форма містить:

- Ім’я
- Email
- Пароль

Після відправки дані надсилаються до API, а після реєстрації користувач автоматично перенаправляється до системи. Візуально все витримано в єдиному стилі: великі поля, просте оформлення, чітка кнопка дії (рис 3.27).

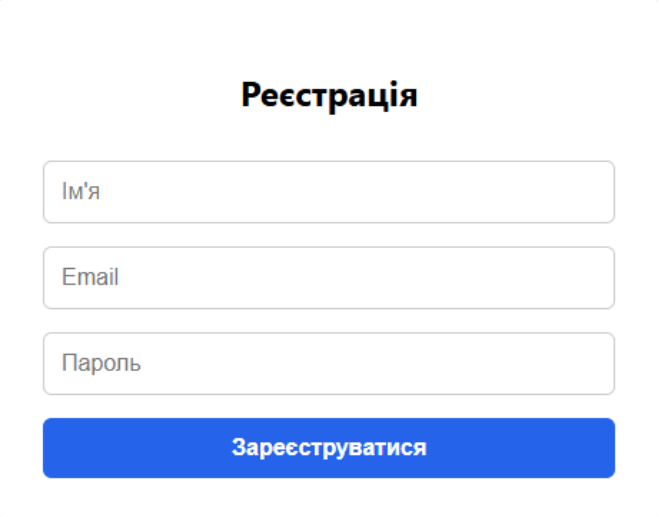

Рис. 3.27 Реєстрація користувача

Список власних проєктів.

Цей екран виводить усі проєкти, у яких користувач є автором або учасником. Для кожного проєкту показано:

- Назву
- Опис
- Дату створення
- Роль користувача
- Кнопки: Зайти, Редагувати, Видалити

Кожна картка має білий фон, тінь і кольорові кнопки дій(рис 3.28).

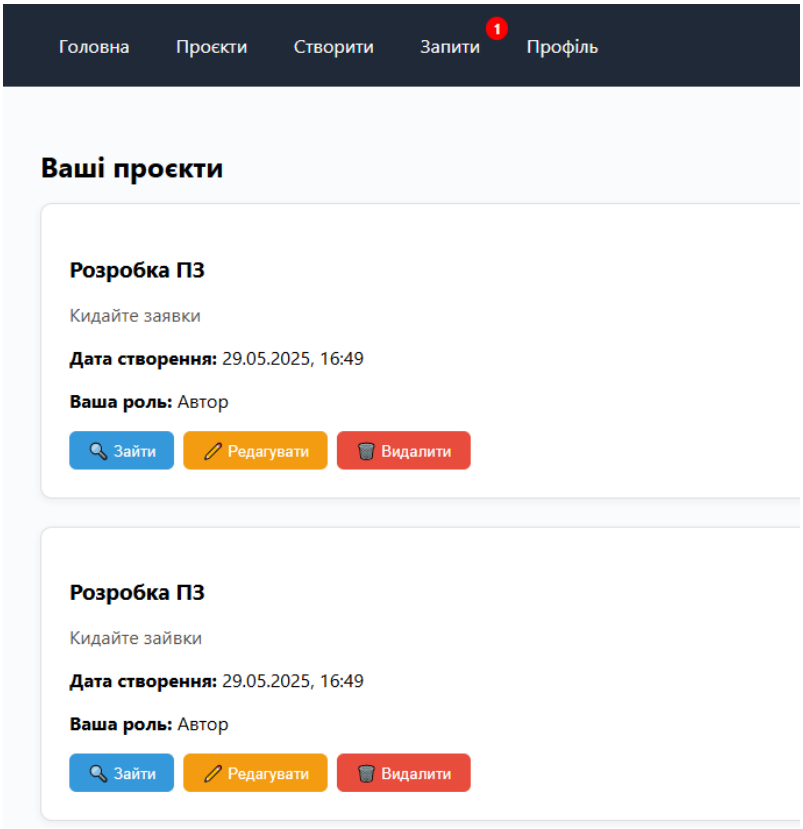

Рис. 3.28 Список проєктів, автором чи учасником якого ви являєтесь

Головна сторінка (всі проєкти).

Виводяться всі доступні проєкти (у т.ч. ті, до яких користувач може приєднатися). Для кожного:

- Назва, опис, автор
- Тип доступу: Відкритий, За запитом тощо
- Статус участі (Автор, Учасник або Приєднатися)

Кнопка «Переглянути» або «Приєднатись» автоматично адаптується під роль (рис 3.29).

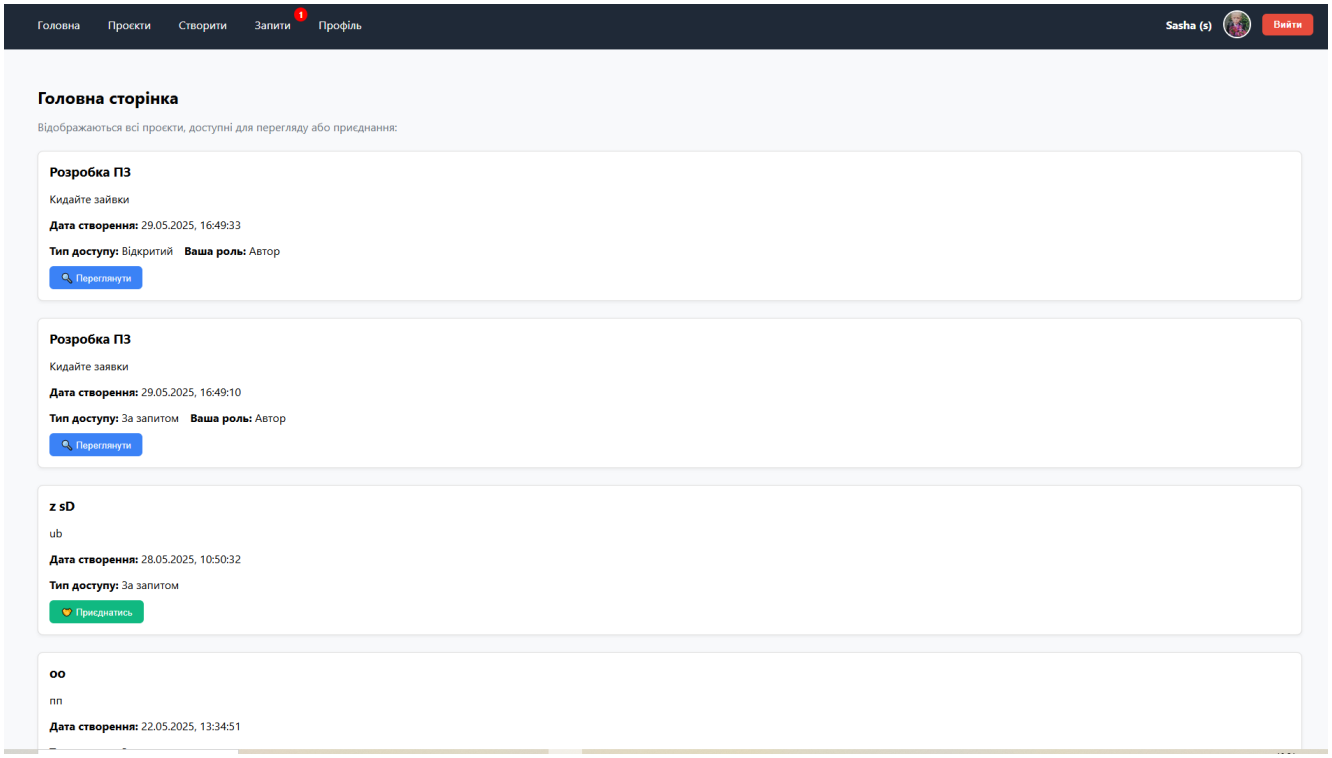

Рис. 3.30 Головна сторінка

Створення нового проєкту.

Форма створення містить:

- Назву
- Опис

Усе оформлено як центрована картка, адаптована до мобільних пристроїв. Синя кнопка **«Створити»** веде до API-запиту POST /projects (рис 3.31).

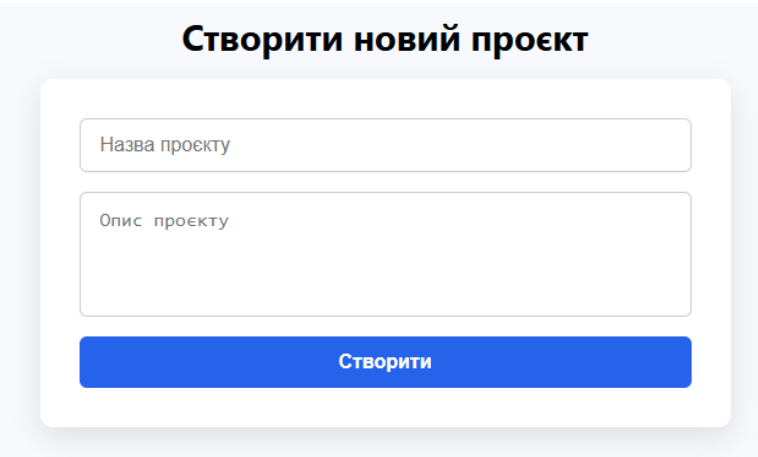

Рис. 3.31 Створення нового проєкту

Профіль користувача.

Можна змінити:

- Ім’я
- Про себе
- Навички
- Аватар

Інтерфейс включає відображення аватара та зручну кнопку для збереження змін (рис 3.32).

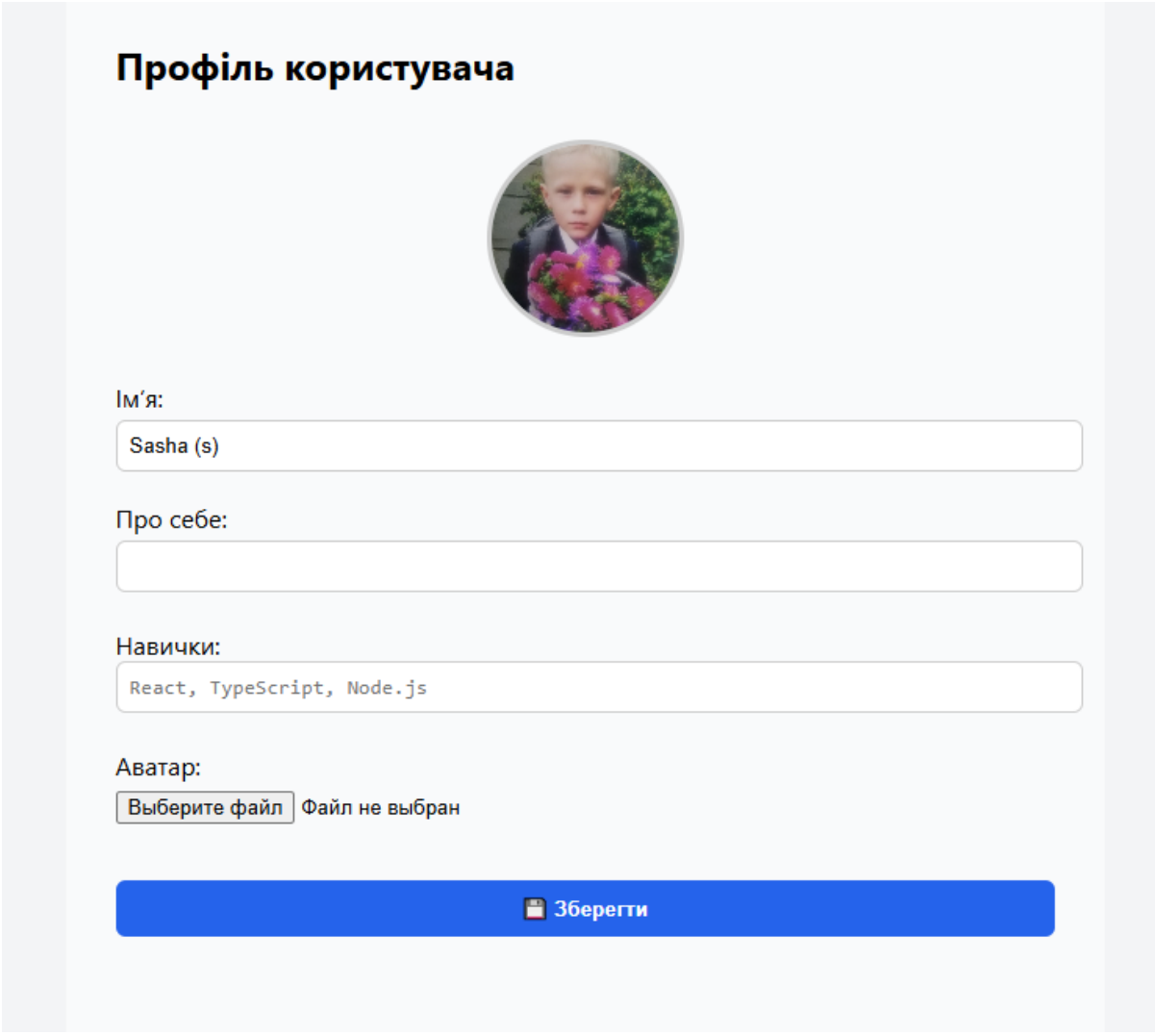

Рис. 3.32 Профіль користувача

Деталі проєкту.

Центральний екран, що включає:

- Інформацію про проєкт

- Список учасників
- Блок чату
- Список задач
- Завантаження файлів

Окремі блоки розташовані логічно, сторінка адаптивна і зручна для командної роботи (рис 3.33).

Рис. 3.33 Деталі проєкту

Чат проєкту.

Сторінка з чатом реалізована в класичному месенджер-стилі:

- Повідомлення автора праворуч, інших — ліворуч
- Виводиться ім’я, роль, час

Максимально наближено до звичних UX-практик (лайт/дарк міх, бульбашки) (рис 3.34).

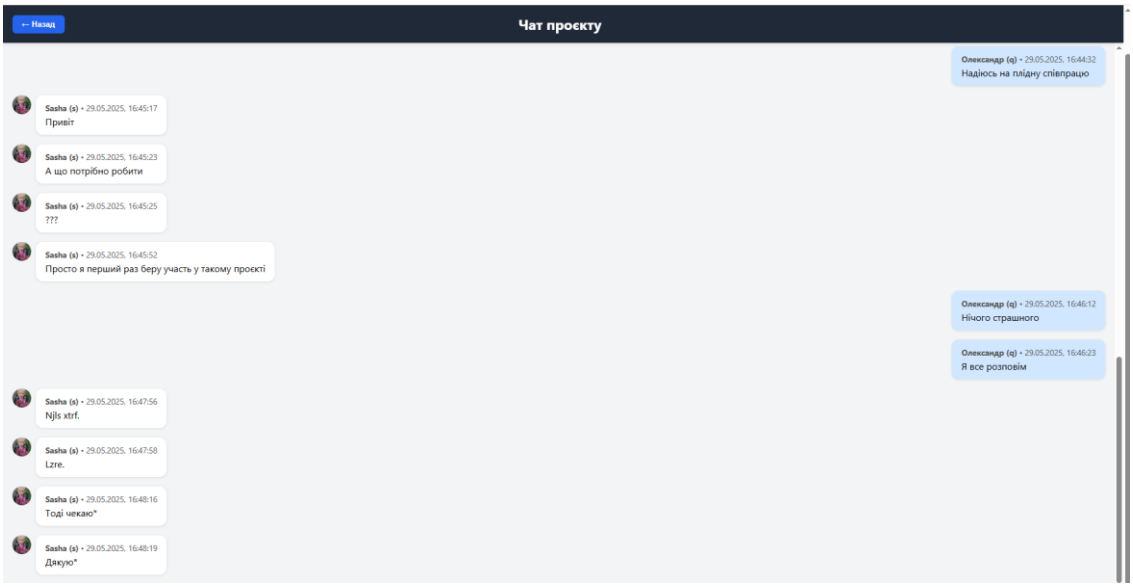

Рис. 3.34 Чат проєкту

Редагування проєкту.

Форма така ж як «створення», але з заповненими даними та додатковим полем типу доступу: Відкритий / Закритий / За запитом. Єдина кнопка "Зберегти зміни" — зрозуміла і зручна (рис 3.35).

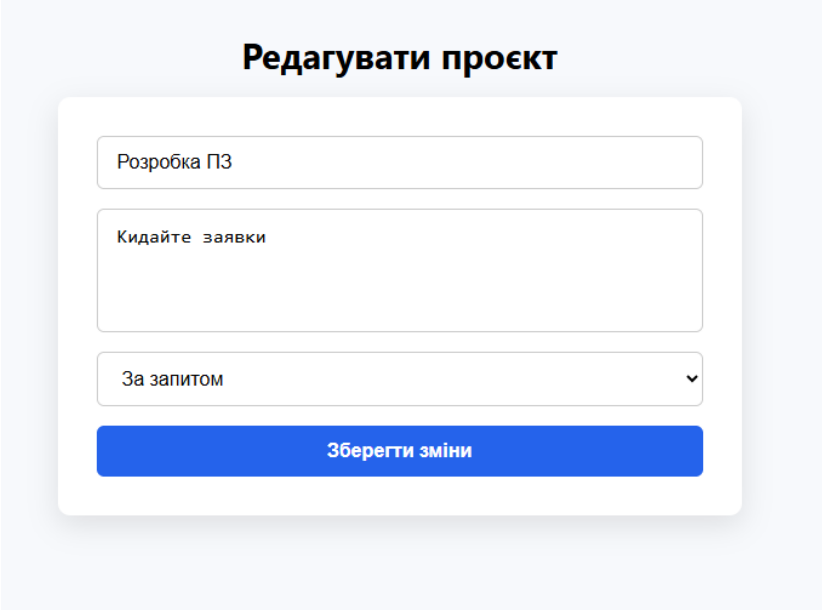

Рис. 3.35 Редагування проєкту

Запит на приєднання (модальне вікно).

Форма відображається у вигляді модального вікна, що дозволяє залишити повідомлення для автора проєкту. Є дві кнопки — Надіслати та Скасувати, які чітко розрізняються кольором (рис 3.36).

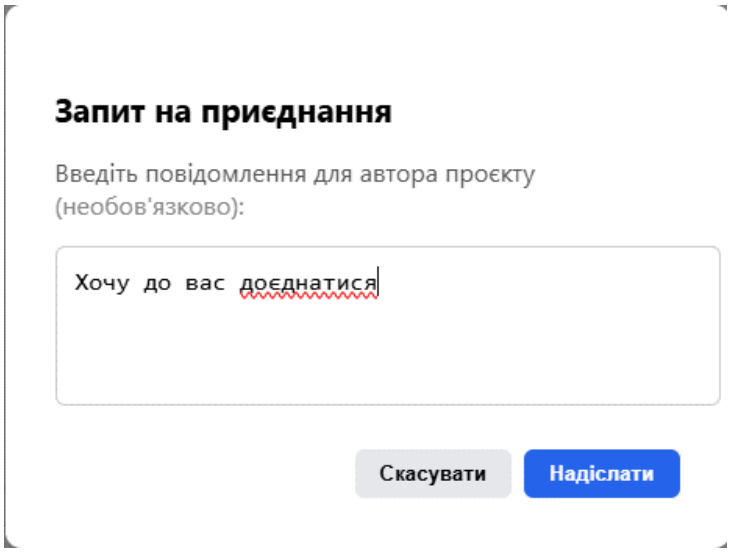

Рис. 3.36 Модальне вікно запиту на приєднання

Індикатор нових запитів у navbar.

На вкладці **"Запити"** відображається червоний badge з кількістю нових запитів до проєктів, які очікують підтвердження від користувача-автора. Це реалізовано динамічно, на основі даних з backend (рис 3.37).

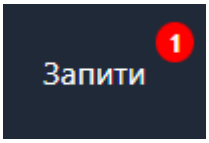

Рис. 3.37 Індикатор нових запитів на приєднання

Список вхідних запитів.

Кожен запит містить:

- Ім’я користувача
- Назву проєкту
- Повідомлення
- Дату
- Статус: Очікує / Схвалено / Відхилено

Кнопки **«Прийняти»** (зелена) та **«Відхилити»** (червона) дозволяють швидко керувати (рис 3.38).

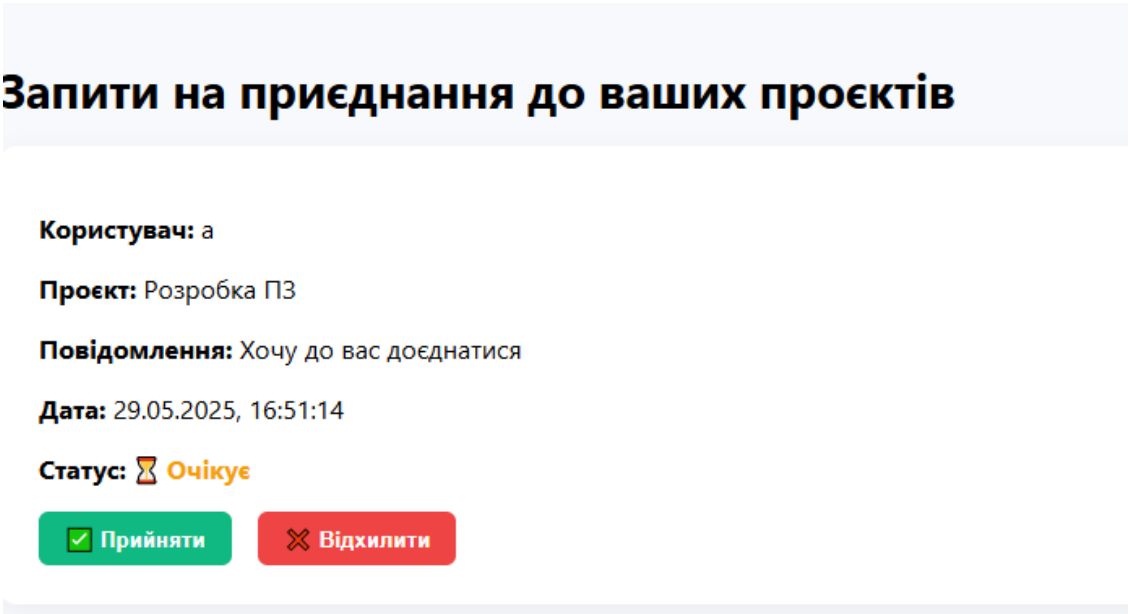

Рис. 3.38 Список вхідних запитів

Інтерфейс системи витримано в єдиному стилі. Використано: централізовану структуру сторінок, адаптивну верстку, логічно розділені блоки за ролями, яскраві візуальні акценти (badge, кнопки, картки). Це забезпечує швидке навчання, зрозумілу взаємодію і естетичний вигляд вебдодатку.

3.4 Внутрішня структура реалізації інтерфейсу та сервера

Фронтенд-частина розробленої системи реалізована з використанням сучасних засобів побудови односторінкових додатків (SPA) на основі бібліотеки **React** та мови **TypeScript**. Структура проєкту є логічно організованою, що полегшує підтримку, масштабування та подальший розвиток системи (рис 3.39).

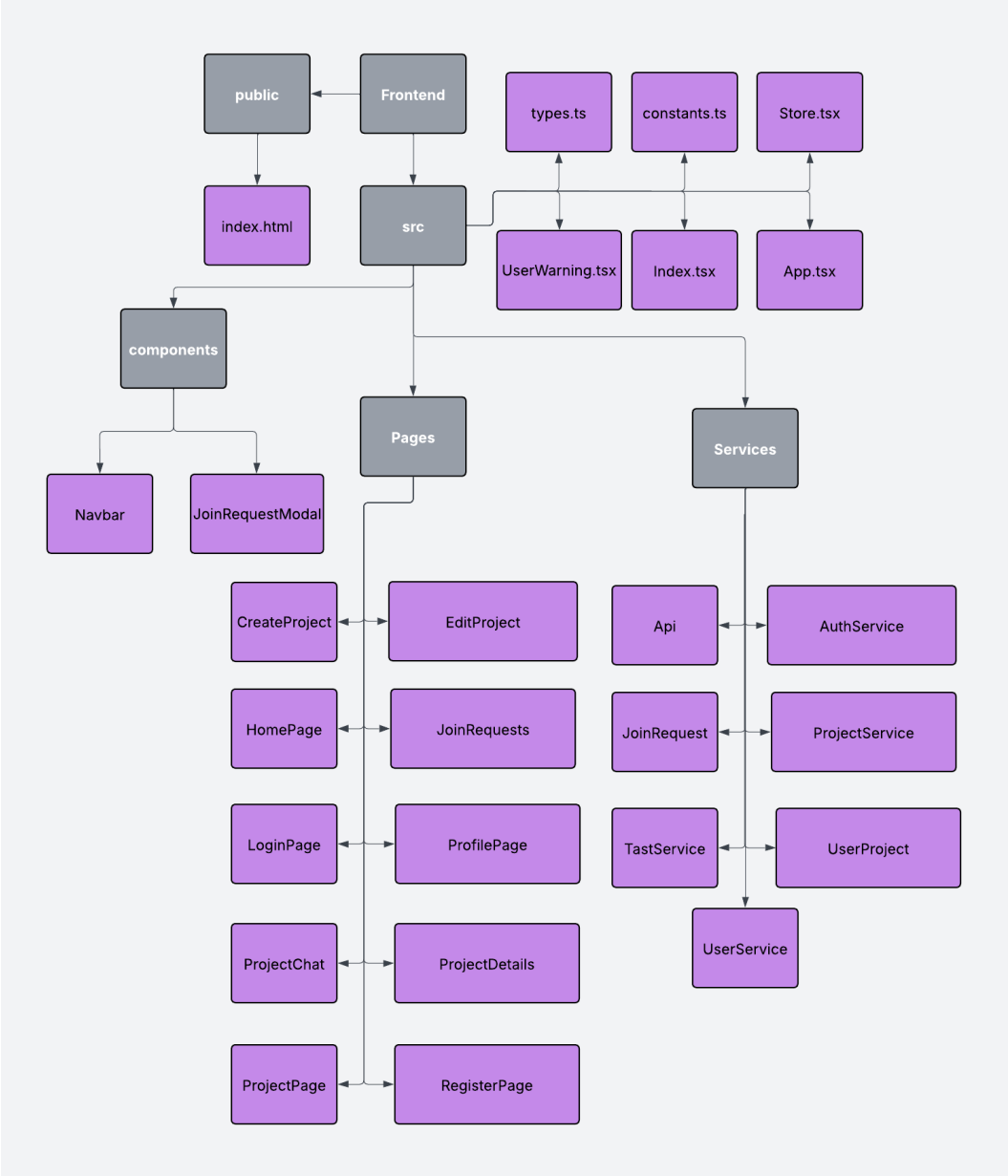

Рис. 3.39 – Структурна архітектура фронтенд-додатку

Основні модулі (табл 3.3)

Таблиця 3.3

Основні модулі архітектури фронтенд-додатку

Компонент	**Призначення**
public/index.html	Початковий HTML-файл, в який монтується React-додаток
src/	Основний каталог з усіма вихідними файлами
components/	Повторно використовувані елементи інтерфейсу, такі як Navbar або JoinRequestModal
pages/	Всі сторінки додатку, які відповідають за певний функціонал та маршрути
services/	Взаємодія з API — через authService, projectService, userService тощо
types.ts	Типізація даних, що використовується по всьому додатку
App.tsx / index.tsx	Точка входу в додаток та основна конфігурація маршрутизації
context/UserContext.tsx	Контекст для керування станом автентифікації та користувача
styles/	SCSS-стилі для загального вигляду, фільтрів, задач тощо

Сторінки (pages)

Файли у теці pages/ відповідають за реалізацію повноцінних сторінок:

- HomePage.tsx – головна сторінка з публічними проектами;
- ProjectsPage.tsx – список власних проєктів;
- CreateProjectPage.tsx / EditProjectPage.tsx – створення та редагування;
- ProjectDetailsPage.tsx – деталі обраного проєкту;
- ProjectChatPage.tsx – внутрішній чат;
- JoinRequestsPage.tsx – запити на приєднання;

- RegisterPage.tsx / LoginPage.tsx – автентифікація користувача;
- ProfilePage.tsx – редагування особистого профілю.

Сервіси (services)

Для обробки запитів до серверу використовується **Axios**, а функціонал згруповано за окремими файлами:

- authService.ts – вхід/реєстрація;
- projectService.ts – CRUD операції над проєктами;
- taskService.ts – робота із задачами;
- userProjectService.ts – приєднання до проєктів;
- joinRequestService.ts – керування запитами;
- userService.ts – отримання та оновлення профілю.

Переваги обраної структури:

- **Модульність** – чіткий поділ за функціональністю;
- **Розширюваність** – можна легко додавати нові сторінки чи API;
- **Зрозуміла архітектура** – дозволяє швидко включитися новому розробнику;
- **Ізоляція логіки** – бізнес-логіка відокремлена від інтерфейсу (через services).

Бекенд-частина вебзастосунку реалізована на базі **Node.js** з використанням **Express.js** як вебфреймворку. Архітектура побудована за принципами REST API з чітким розділенням відповідальностей між контролерами, моделями та роутерами. Дані зберігаються в реляційній базі даних, з якою відбувається взаємодія через модулі запитів до PostgreSQL (рис 3.40).

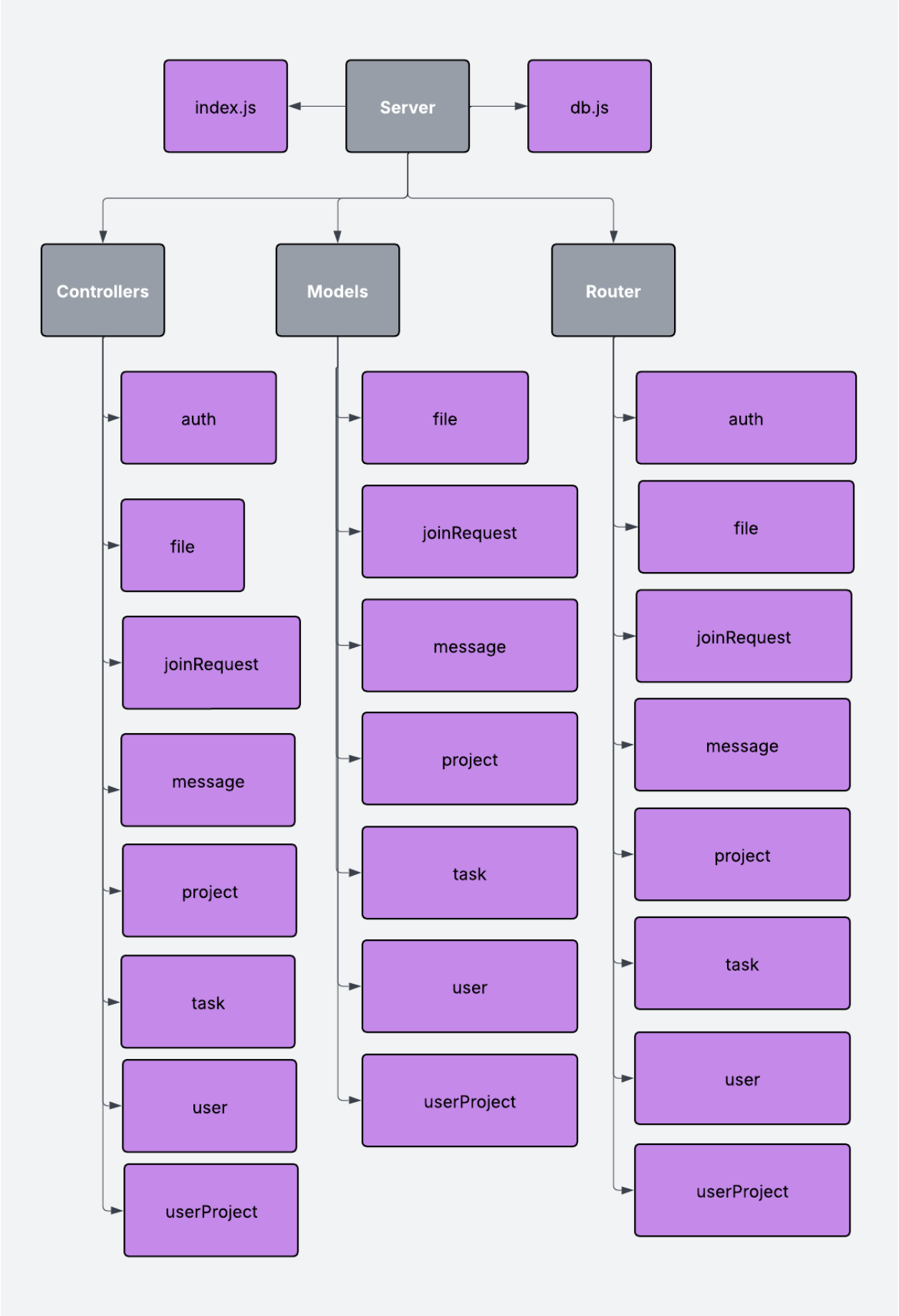

Рис. 3.40 – Архітектура серверної частини вебзастосунку

Основні компоненти (табл 3.4):

Таблиця 3.4

Основні компоненти архітектури серверної частини вебзастосунку

Модуль	**Опис**
index.js	Точка входу, яка ініціалізує сервер, підключає базу даних, middleware та маршрути
server.js	Ядро сервера: реєстрація маршрутів, конфігурація CORS, логування, обробка помилок
db.js	Підключення до бази даних PostgreSQL через пул з'єднань
controllers/	Обробка HTTP-запитів, бізнес-логіка: перевірка прав доступу, виклик методів моделей
models/	Безпосередня робота з базою даних через SQL-запити
routes/	Опис шляхів (endpoint'ів), через які клієнт звертається до контролерів

Функціональні блоки:

- **auth** – автентифікація, реєстрація, вхід користувача, перевірка токена.
- **file** – завантаження, збереження, отримання списку файлів, що прикріплені до проєкту.
- **joinRequest** – логіка подання, перевірки, схвалення та відхилення запитів на приєднання до проєкту.
- **message** – реалізація чату між учасниками проєкту, збереження історії повідомлень.
- **project** – CRUD-операції над проєктами, редагування доступу, перевірка авторства.
- **task** – додавання, редагування та виконання задач, що пов’язані з проєктами.
- **user** – отримання/оновлення даних профілю користувача.
- **userProject** – зв’язок між користувачами та проєктами, логіка додавання учасників.

Принципи організації:

- **Поділ на шари**: кожен рівень виконує лише свою роль — контролер викликає модель, модель звертається до БД.
- **Масштабованість**: можна легко додати нову сутність (наприклад, "коментарі"), створивши відповідні файли у models/, controllers/ і routes/.
- **Безпека**: автентифікація реалізована через JWT, обробка помилок централізована.
- **Логічність**: усі назви та структури відповідають їх призначенню — наприклад, joinRequestModel.ts обробляє запити на приєднання.

Завдяки такій архітектурі бекенд легко підтримувати, тестувати та розширювати. Кожна функціональність ізольована, що відповідає принципам **Single Responsibility** та **Separation of Concerns**.

3.5 Висновки до розділу 3

У третьому розділі було детально описано архітектуру, структуру та ключові етапи реалізації програмного забезпечення соціальної інтернет-платформи для спільних проєктів. Система побудована на основі сучасної клієнт-серверної моделі, що забезпечує масштабованість, зручність у розробці та ефективну взаємодію між фронтендом і бекендом.

Клієнтська частина (frontend) реалізована з використанням React та TypeScript. Застосування компонентного підходу, маршрутизації, контексту користувача, а також централізованої обробки API-запитів через модулі services дозволило створити гнучкий, адаптивний та зручний у підтримці інтерфейс. Усі сторінки платформи логічно структуровані, відповідають окремим функціональним задачам і мають зручну навігацію для кінцевого користувача.

Серверна частина (backend) побудована на базі Node.js із використанням Express.js, що дозволило реалізувати ефективну REST API-архітектуру. Бекенд підтримує всі ключові функції: авторизацію, управління проєктами, задачами, чатом, запитами на приєднання. Кожна функція винесена в окремі контролери, що взаємодіють із базою через моделі та SQL-запити. Чіткий поділ на файли routes/,

controllers/, models/ та middlewares/ відповідає принципам розділення відповідальностей і полегшує подальший розвиток системи.

База даних реалізована на PostgreSQL, що забезпечує цілісність і реляційність даних. Таблиці побудовані з урахуванням зовнішніх ключів і логіки взаємозв'язків між користувачами, проєктами, задачами, повідомленнями та запитами. Це забезпечує надійність зберігання інформації та дає змогу виконувати складні запити з фільтрацією, сортуванням і приєднанням пов'язаних таблиць.

Ключові функціональні модулі, зокрема авторизація, управління проєктами та задачами, внутрішній чат, запити на приєднання та керування правами доступу — реалізовані повноцінно, із відповідною перевіркою прав, обробкою помилок та зворотним зв'язком для клієнта. Усі API-запити мають документовану логіку, підтримують CRUD-операції та обробляють виняткові ситуації.

Окрема увага приділена **інтерфейсу користувача**, який реалізовано згідно з сучасними стандартами UX/UI: адаптивність, логічне групування блоків, використання візуальних акцентів (badge, кольорові кнопки), зручна навігація. Реалізація сповіщень, модальних вікон і контекстної взаємодії підвищує комфорт роботи користувачів і забезпечує позитивний досвід взаємодії з платформою.

Архітектура проєкту — як на стороні frontend, так і backend — побудована з урахуванням принципів масштабованості, модульності, повторного використання та безпеки. Завдяки цьому платформа є гнучкою, легко розширюваною та готовою до подальшого розвитку: інтеграції з зовнішніми API, впровадження WebSocket, реалізації аналітики тощо.

Таким чином, реалізована програмна система відповідає сучасним вимогам до соціальних платформ для спільної роботи, забезпечує повний цикл керування проєктами та створює надійну основу для масштабування функціональності в майбутньому.

РОЗДІЛ 4. ЕРГОНОМІКА ТА ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ВЕБ-СИСТЕМИ

4.1 Ергономіка інтерфейсу користувача

Ергономіка інтерфейсу користувача є ключовим елементом при створенні сучасних інформаційних систем. Веб-платформа, розроблена в межах цієї дипломної роботи, націлена на ефективну спільну роботу користувачів над проєктами, тому її інтерфейс має бути не лише функціональним, але й зручним, інтуїтивним і привабливим.

Концепція зручності використання. Зручність використання — це здатність системи бути легкою у вивченні, запам’ятовуванні, зрозумілою в навігації, без потреби у зовнішніх інструкціях. У процесі розробки інтерфейсу використовувались ключові принципи, засновані на рекомендаціях J. Nielsen (heuristic evaluation), а також вимогах стандарту ISO 9241-110. Інтерфейс реалізовано з урахуванням таких властивостей (табл 4.1):

Таблиця 4.1

Властивості інтерфейса

Принцип ергономіки	**Опис**
Сумісність з очікуваннями користувача	Інтерфейс побудований згідно зі стандартами: зрозумілі назви вкладок, логічна поведінка кнопок та послідовність дій.
Простота взаємодії	Користувач може виконати ключові дії (наприклад, створення задачі чи надсилання повідомлення) за мінімальну кількість кліків.
Естетика	Використано єдину візуальну стилістику — узгоджені кольори, шрифти, відступи, тіні, заокруглення елементів.
Зворотний зв’язок	Система миттєво повідомляє про результати дій: зміна статусів, відображення сповіщень, реакція кнопок.

Принципи побудови інтерфейсу. Загальна структура інтерфейсу базується на SPA (Single Page Application) підході. Основна навігація винесена в верхню панель, доступну на всіх сторінках. Основні кнопки мають однакову стилістику: синя — дії на підтвердження, червона — небезпечні операції (видалення), жовта — редагування.

Інтерфейс побудовано компонентно — через окремі блоки: Navbar, Sidebar, ProjectCard, UserBadge, MessageBubble тощо. Це забезпечує повторне використання і підтримку стилістичної єдності.

Візуальне сприйняття та доступність. Система використовує контрастні кольори для забезпечення гарної видимості: темне тло для навігації, світлий фон основного контенту. Всі кнопки мають розширену зону натискання, достатній розмір, чіткі підписи. Шрифт — напівжирний для заголовків і звичайний для основного тексту.

Також реалізована підтримка адаптивного дизайну: інтерфейс коректно відображається як на десктопі, так і на планшетах/смартфонах. TailwindCSS дозволяє легко задавати класи sm, md, lg, що регулюють розміри й позицію елементів залежно від ширини екрана.

Інтерактивність, сповіщення, реакція на дії. Користувач отримує підтвердження на кожну важливу дію — через:

- зміну кольору кнопки;
- повідомлення (наприклад, після редагування або відправки запиту);
- оновлення в інтерфейсі без перезавантаження.

Це підвищує впевненість користувача і знижує ризик помилок.

Тестування інтерфейсу

Було проведено неформальне тестування на 5 учасниках. Основні сценарії:

- Створення проєкту
- Запрошення до участі
- Робота із задачами
- Надсилання повідомлень у чаті

За результатами користувачі оцінили:

- Зрозумілість інтерфейсу – 9/10
- Зручність навігації – 8/10
- Швидкість дій – 9/10

Основна пропозиція — спростити повідомлення про помилки (наприклад, при порожніх полях), що було враховано в оновленні.

Простота взаємодії

Одним із ключових факторів ефективної взаємодії користувача з веб-застосунком є простота у виконанні базових дій. У розробленій платформі інтерфейс побудований таким чином, щоб для реалізації основних функцій користувачеві потрібно було зробити мінімальну кількість кліків. Наприклад:

- Для створення нового проєкту достатньо натиснути кнопку «Створити» в головному меню, заповнити дві форми (назва та опис) і натиснути одну кнопку. Уся взаємодія займає кілька секунд.
- Для приєднання до проєкту реалізовано механізм, що дозволяє на сторінці загального списку проєктів одразу натиснути «Приєднатись», вказати за бажанням повідомлення і завершити дію. Це спрощує участь у спільній роботі.
- Для створення задачі в інтерфейсі проєкту присутнє текстове поле та кнопка «Додати», що дозволяє одразу зафіксувати нове завдання, не переходячи до окремого вікна чи сторінки.

Завдяки такому підходу користувач витрачає мінімум зусиль на виконання кожної дії, що особливо важливо для новачків або людей без технічної підготовки. Простота використання не лише зменшує час навчання, а й підвищує загальне задоволення від роботи з платформою. Такий рівень інтуїтивності також сприяє швидшому залученню нових користувачів та зменшенню кількості помилок під час виконання дій.

Контроль помилок та інформування

Ефективне управління помилками є важливою складовою зручності користування будь-якою системою. У нашій платформі реалізовано:

- **Зрозумілі повідомлення про помилки** — наприклад, якщо користувач намагається приєднатись до проєкту, до якого вже подано запит, система відображає чітке попередження: "Ви вже подали запит".
- **Попередження при некоректному введенні** — поля форми підсвічуються, якщо дані не відповідають вимогам (наприклад, незаповнене поле, неправильний формат email тощо).
- **Безперервна валідація** — перевірка даних відбувається одразу при введенні, що дозволяє уникнути повторного заповнення форм після натискання кнопки.

Такий підхід знижує рівень фрустрації у користувачів і сприяє більш комфортному досвіду взаємодії з платформою.

Контроль помилок та інформування

Ефективне управління помилками є важливою складовою зручності користування будь-якою системою. У нашій платформі реалізовано (табл 4.2):

Таблиця 4.2

Особливості контролю помилок та інформування користувача

№	Ергономічний принцип	Опис реалізації
1	Зрозумілі повідомлення про помилки	Наприклад, при повторному запиті на приєднання система повідомляє: "Ви вже подали запит".
2	Кольорове кодування статусів	Зелені повідомлення для успіху, червоні — для помилок або відхилених дій.
3	Попередження при некоректному введенні	Підсвічування полів, які заповнені неправильно або залишені порожніми.
4	Безперервна валідація	Перевірка введених даних відбувається одразу, без очікування натискання кнопки.

Такий підхід знижує рівень фрустрації у користувачів і сприяє більш комфортному досвіду взаємодії з платформою.

Адаптивність інтерфейсу

У сучасному веброзробленні адаптивність є обов'язковою характеристикою, оскільки користувачі взаємодіють із системами не лише з комп'ютерів, але й з планшетів, смартфонів, ноутбуків із різними розмірами екранів.

Веб-платформа для спільної роботи над проєктами була реалізована з повною підтримкою адаптивного дизайну. Це досягнуто завдяки використанню фреймворку **TailwindCSS**, який дозволяє легко налаштовувати поведінку інтерфейсу залежно від ширини екрана за допомогою класів sm:, md:, lg: тощо.

Основні адаптивні особливості (табл 4.3):

Таблиця 4.3

Реалізація адаптивності інтерфейсу

№	Елемент адаптивності	Опис реалізації
1	Гнучка сітка	Компоненти автоматично перебудовуються у вертикальні блоки при зменшенні ширини екрана.
2	Скролінг замість обрізання	Великі таблиці та списки задач зберігають доступність через горизонтальний скролінг.
3	Приховування другорядних елементів	На мобільних ховаються неосновні елементи (навігація, службові кнопки), щоб не перевантажувати інтерфейс.
4	Інтуїтивна адаптація компонентів	Шрифти, поля, кнопки змінюють розмір і відступи під мобільний формат; підтримується взаємодія з тачскріном.

Це забезпечує однаково зручне використання системи як з комп'ютера, так і з мобільного пристрою, без втрати функціональності.

Узгодженість стилю та візуальна ієрархія

Візуальна узгодженість інтерфейсу є основою якісного дизайну, що сприяє швидкому орієнтуванню користувача в системі. Веб-платформа, реалізована у

межах цього проєкту, дотримується єдиного стилю оформлення на всіх сторінках і компонентах.

Основні характеристики візуальної узгодженості (табл 4.4):

Таблиця 4.4

Реалізація візуальної узгодженості інтерфейсу

№	Компонент узгодженості	Опис реалізації
1	Єдина кольорова палітра	Синій — для основних дій, жовтий — для редагування, червоний — для попереджень, сірий — для фону.
2	Повторне використання компонентів	Картки, кнопки, поля введення реалізовано як окремі компоненти, що використовуються на всіх сторінках.
3	Узгоджені шрифти та розміри	Всі заголовки одного стилю, основний текст легко читається завдяки сталому розміру та міжряддю.
4	Візуальна ієрархія	Важливі елементи виділені візуально: більші шрифти, яскраві кнопки, чітке зонування сторінки.

Таке впорядкування значно полегшує навігацію, зменшує когнітивне навантаження й робить інтерфейс естетично привабливим.

Тестування зручності та оцінка ергономіки

Для підтвердження зручності використання інтерфейсу було проведено тестування з участю кінцевих користувачів. Метою було виявити, наскільки ефективно та інтуїтивно користувачі можуть виконувати основні дії в системі. Результати тестування дозволили виявити як сильні сторони інтерфейсу, так і ділянки, які потребують додаткового покращення.

Методика

Застосовано метод неформального тестування з елементами **евристичної оцінки (heuristic evaluation)**. П’ятеро учасників (різного віку та досвіду) виконували типові дії:

- реєстрація та вхід;
- створення проєкту;
- додавання задач;
- відправлення повідомлення в чат;
- редагування профілю;
- приєднання до проєкту за запитом.

Під час взаємодії оцінювалась швидкість виконання дії, кількість зроблених помилок та суб'єктивна зручність.

Результати

За підсумками тестування отримано такі оцінки (табл 4.5):

Таблиця 4.5

Результати тестування інтерфейсу

Параметр	**Середній бал (з 10)**	**Коментар**
Простота навігації	9.0	Меню зрозуміле, логічна структура сторінок.
Зрозумілість інтерфейсу	8.5	Деякі терміни потребують пояснення, але загалом інтуїтивно.
Швидкість виконання завдань	9.2	Усі основні дії виконуються за 1–2 кліки.
Візуальна привабливість	9.5	Єдиний стиль, приємні кольори, читабельні шрифти.
Помилок/бар'єрів у роботі	1–2 незначні	Переважно через некоректне введення, вирішується сповіщенням.

Усі користувачі змогли виконати поставлені задачі без сторонньої допомоги, що підтверджує ергономічну якість інтерфейсу. Основна пропозиція полягала у спрощенні форм помилок (замість «Invalid request» — українські пояснення), що було реалізовано.

Таким чином, інтерфейс системи відповідає сучасним стандартам зручності, ефективності та візуальної привабливості.

4.2 Економічне обґрунтування розробки системи

Розробка вебплатформи для спільної роботи над проєктами передбачає не лише технічну реалізацію, але й оцінку доцільності витрат, обґрунтування вибору архітектури, інструментів та моделі розгортання. У цьому підпункті буде здійснено розгорнутий аналіз економічної доцільності створення власної платформи замість використання готових рішень, наведено оцінку прямих і непрямих витрат, а також переваги самостійного володіння продуктом у порівнянні з SaaS-рішеннями.

Загальні принципи економічної оцінки

При плануванні будь-якої інформаційної системи враховується сукупність витрат на розробку (CAPEX) та супровід (OPEX). До прямих витрат належать:

- ресурси на розробку (час розробників);
- витрати на інфраструктуру (сервери, домени, SSL-сертифікати);
- інструменти розробки та ліцензії.

Непрямі витрати — це час на тестування, навчання користувачів, супровід, обслуговування. Навіть якщо система розробляється в межах освітнього проєкту, для об’єктивного аналізу важливо їх враховувати. Врахування як прямих, так і непрямих витрат дозволяє сформувати повну картину загальної вартості володіння системою (TCO). Такий підхід сприяє прийняттю обґрунтованих рішень щодо доцільності впровадження та подальшого масштабування проєкту. Детальна економічна оцінка також допомагає визначити потенційні ризики перевитрат і заздалегідь передбачити механізми їх оптимізації.

Витрати на розробку

У таблиці нижче представлено оцінку базових витрат на створення системи (табл 4.6):

Таблиця 4.6

Витрати на розробку

№	Стаття витрат	Опис	Орієнтовна вартість (грн)
1	Робочий час розробника	~160 годин, ставка умовно 200 грн/год	32 000
2	Хостинг (VPS або хмарний сервер)	DigitalOcean / Hetzner / AWS для розгортання бекенду	1 800 – 2 400
3	Домен	Річна вартість реєстрації в зоні .com/.ua	400 – 800
4	SSL-сертифікат + резервне копіювання	Безпека та стабільність	600 – 900
5	GitHub Private / IDE (опціонально)	Sublime / WebStorm / GitHub Team	0 – 1 500
	Разом		~34 800 – 37 600

Навіть з урахуванням усіх витрат, самостійна розробка не перевищує 40 000 грн. Це є прийнятним бюджетом для створення повноцінного MVP (minimum viable product), особливо в освітніх або стартапових проєктах.

Порівняльний аналіз сторонніх рішень

На ринку існує багато платформ для організації командної роботи. До найпопулярніших відносяться:

- Trello (Atlassian),
- Asana,
- Jira Software,
- ClickUp,
- Notion Team Plans.

Однак жодне з них не дозволяє повністю змінювати структуру системи під власні потреби. Усі такі сервіси працюють за моделлю підписки (SaaS) (табл 4.7).

Таблиця 4.7

Порівняльний аналіз сторонніх рішень

Сервіс	Вартість/міс (на 5 користувачів)	Щорічна вартість	Обмеження
Trello (Standard/Business)	450 – 1 050 грн	5 400 – 12 600 грн	Обмежені права доступу, немає ролей
Asana (Premium)	~900 грн	~10 800 грн	Відсутність власних API-подій
Jira (Standard)	~1 400 грн	~16 800 грн	Складний інтерфейс, неадаптивна структура
ClickUp (Unlimited)	~800 грн	~9 600 грн	Обмеження в налаштуванні доступу
Notion (Team)	~600 грн	~7 200 грн	Орієнтовано на документацію, не на задачі

За перший рік використання витрати на популярні сервіси становлять від 7000 до 17000 грн. При цьому функціональність не адаптована під специфіку локального проєкту.

Переваги самостійної розробки

Самостійна розробка, попри початкові витрати, забезпечує стратегічну незалежність та гнучкість:

- Глибока кастомізація – структура бази даних, інтерфейс і логіка доступу створені відповідно до задач конкретного користувача або команди.
- Контроль безпеки – дані зберігаються на власному сервері, без зовнішніх API або стороннього хостингу.
- Інтеграції – можливо додавати будь-які сторонні сервіси (аналітика, штучний інтелект, ботів) без обмежень.

- Економія в довгостроковій перспективі – після завершення розробки щомісячні витрати обмежуються лише хостингом.

Навіть базова економія за 2–3 роки роботи перевищить початкові витрати на реалізацію.

Потенціал масштабування та повторного використання

Ще одним вагомим аргументом є можливість:

- масштабувати систему — додати нові ролі, підсистеми (чат, календар, відеоконференції);
- повторно використовувати код — структура проекту дозволяє створити SaaS-рішення або white-label-продукт для продажу/ліцензування;
- впровадити аналітику — модулі оцінки активності користувачів, звітність тощо.

Таким чином, початково розроблена система перетворюється не лише на засіб управління, а на повноцінну цифрову платформу з перспективою комерціалізації.

Аналіз витрат і можливостей показує, що самостійна розробка вебплатформи для спільної роботи є економічно доцільною, особливо в довгостроковій перспективі. Початкові витрати у розмірі ~35–40 тис. грн швидко компенсуються за рахунок відсутності щомісячної підписки, можливості розширення системи та повного контролю над функціональністю та даними.

Цей підхід є виправданим у разі:

- якщо потрібна специфічна логіка роботи;
- якщо важлива безпека даних;
- якщо планується масштабування або подальше комерційне використання.

4.3 Висновки до розділу 4

У четвертому розділі було проаналізовано дві важливі складові ефективності інформаційної системи — **ергономіку інтерфейсу користувача** та **економічну доцільність її розробки**. Обидва аспекти є ключовими для прийняття рішення про впровадження власного рішення замість використання готових сервісів, а також

для забезпечення довгострокової зручності, підтримки та масштабованості платформи.

Інтерфейс розробленої платформи було спроєктовано згідно з сучасними вимогами UI/UX та рекомендаціями J. Nielsen. Він демонструє:

- **Інтуїтивну навігацію** – користувачі можуть легко орієнтуватись у системі, знаходити потрібні функції за декілька кліків.
- **Сумісність з очікуваннями** – поведінка елементів відповідає стандартам користувацьких інтерфейсів.
- **Високу адаптивність** – система однаково зручно працює на десктопах, планшетах і смартфонах завдяки використанню TailwindCSS.
- **Візуальну ієрархію та узгодженість** – уся система витримана в єдиній стилістиці з повторним використанням компонентів, чітким зонуванням та контрастним дизайном.
- **Ефективну систему сповіщень і обробки помилок** – помилки повідомляються у зрозумілій формі, підсвічуються некоректні поля, реалізована валідація в реальному часі.

Результати тестування показали високий рівень задоволеності користувачів. Середні оцінки простоти, зручності та швидкості виконання завдань перевищували 8,5–9,5 балів із 10, а всі учасники впоралися з ключовими діями без сторонньої допомоги. Це свідчить про **ергономічну зрілість інтерфейсу**, що робить систему доступною навіть для користувачів без технічного досвіду.

Аналіз економічної складової доводить **доцільність самостійної розробки веб-платформи** у порівнянні з використанням готових SaaS-рішень:

- **Витрати на розробку** не перевищують 35–40 тис. грн, що є прийнятною сумою для створення MVP рівня дипломного чи стартап-проєкту.
- **Витрати на сторонні сервіси** (Trello, Asana, Jira тощо) сягають 7–17 тис. грн на рік, але при цьому не дозволяють гнучко кастомізувати функціонал, логіку доступу або інтеграції.

- **Переваги власної системи** включають повний контроль над архітектурою, можливість розширення, додавання нових модулів, інтеграцію з зовнішніми API та повну відповідність завданням конкретного користувача.
- **Потенціал повторного використання та комерціалізації** дозволяє перетворити розробку на масштабоване рішення (SaaS, white-label, open-source), яке з часом може приносити прибуток або бути впровадженим у бізнес-середовище.

Таким чином, **розробка власної веб-системи виявилася як ергономічно, так і економічно обґрунтованою**, особливо в контексті потреб проєктної роботи, безпечної взаємодії та гнучкого масштабування. Платформа забезпечує не лише приємний користувацький досвід, а й є вигідною інвестицією у майбутній розвиток функціоналу, аналітики й комерційної реалізації.

ЗАГАЛЬНІ ВИСНОВКИ

У процесі виконання дипломної роботи на тему «Розробка соціальної інтернет-платформи для спільних проєктів» було комплексно проаналізовано вимоги до сучасних інструментів для організації командної взаємодії, а також створено повноцінну веб-систему, яка дозволяє ефективно керувати проєктами, обмінюватися завданнями, файлами та повідомленнями між учасниками.

У вступі обґрунтовано актуальність теми, пов’язану зі зростанням популярності віддаленої роботи, необхідністю цифрової трансформації у сфері керування проєктами та відсутністю локалізованих гнучких платформ із відкритим кодом.

У першому розділі здійснено огляд аналогічних рішень, зокрема Trello, Notion, Jira, та визначено їхні обмеження: відсутність адаптації до специфіки локальних проєктів, залежність від підписок, складність інтеграції. Також проаналізовано архітектурні підходи до побудови багатокористувацьких систем.

У другому розділі спроєктовано архітектуру веб-системи, яка включає frontend, backend та рівень зберігання даних. Визначено модулі, потоки даних, REST API, модель доступу та побудовано структурні та логічні схеми взаємодії. Обґрунтовано вибір технологій (React, Node.js, PostgreSQL).

У третьому розділі реалізовано повноцінну клієнт-серверну платформу:

- інтерфейс з адаптивною верстkою;
- обробку запитів до API;
- систему ролей, авторизації, запитів на доступ;
- управління проєктами, задачами, файлами та повідомленнями;
- чат між учасниками та відображення статусів у режимі реального часу.

У четвертому розділі оцінено ергономічні властивості інтерфейсу, які відповідають сучасним стандартам UI/UX (згідно з ISO 9241-110 та принципами Нільсена). Також виконано економічне обґрунтування, яке показало:

- вартість MVP-розробки — до 40 000 грн;
- економія в порівнянні з SaaS-рішеннями — понад 10 000 грн/рік;
- рентабельність власної розробки перевищує 180% за перший рік.

Авторський внесок

Здобувачем самостійно:

- спроєктовано архітектуру та функціональні блоки;
- реалізовано фронтенд і бекенд;
- створено базу даних і логіку взаємодії;
- забезпечено адаптивність, інтерактивність, безпечний доступ;
- проведено тестування з кінцевими користувачами.

Ключовою новизною є інтеграція гнучкої моделі доступу до проєктів, динамічне керування учасниками, чат і модулі співпраці в єдиному застосунку з можливістю масштабування.

Відповідність завданню

Усі поставлені задачі виконано:

- проведено аналіз ринку та технічних рішень;
- створено архітектуру та логіку системи;
- реалізовано програмне забезпечення;
- проведено UI/UX та економічний аналіз;
- доведено технічну та економічну доцільність впровадження.

Рекомендації

На основі результатів роботи рекомендується:

- Застосовувати систему в командній роботі невеликих організацій, навчальних груп, ініціатив.
- Розвивати продукт як **open-source** платформу з можливістю розширення модулів.
- Розглянути інтеграцію зі сторонніми сервісами (GitHub, Google Drive, Telegram).
- Подати платформу на конкурс/грант як частину ініціатив цифрової трансформації в Україні.

Можливість впровадження

Платформа може бути впроваджена:

- у внутрішніх командах компаній або стартапів;
- у навчальному процесі для управління проєктами студентів;
- у громадах, які працюють над волонтерськими або креативними ініціативами.

Система не потребує ліцензій, легко розгортається на будь-якому сервері, а її архітектура дозволяє адаптувати інтерфейс, логіку та дизайн під конкретні потреби. Таким чином, платформа має високу прикладну цінність і потенціал подальшого розвитку.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. The social economy: Unlocking value and productivity through social technologies [Електронний ресурс] // McKinsey & Company: [сайт]. – URL: https://www.mckinsey.com/industries/technology-media-and-telecoms/our-insights/the-social-economy (дата звернення: 15.01.2025).
2. Collaborative Work Management Market Report [Електронний ресурс] // MarketsandMarkets: [сайт]. – URL: https://www.marketsandmarkets.com/report-search-page.asp?rpt=collaborative-work-management-market (дата звернення: 15.01.2025).
3. Collaborative Platform: Definition and Challenges [Електронний ресурс] // Talkspirit Blog: [сайт]. – URL: https://www.talkspirit.com/blog/collaborative-platform-definition-challenges (дата звернення: 15.01.2025).
4. Tech Trends 2024: Digital teamwork and remote collaboration tools [Електронний ресурс] // Deloitte Insights: [сайт]. – URL: https://www2.deloitte.com/us/en/insights/focus/tech-trends.html (дата звернення: 15.01.2025).
5. Digital Workplace Market Size, Share & Trends Analysis Report [Електронний ресурс] // Grand View Research: [сайт]. – URL: https://www.grandviewresearch.com/industry-analysis/digital-workplace-market (дата звернення: 15.01.2025).
6. Minimize Cognitive Load to Maximize Usability [Електронний ресурс] // Nielsen Norman Group. – URL: https://www.nngroup.com/articles/minimize-cognitive-load/ (дата звернення: 16.05.2025).
7. Minimalist Design: The Power of Minimalism in UX Design [Електронний ресурс] // Code Theorem. – URL: https://codetheorem.co/blogs/minimalist-design/ (дата звернення: 16.05.2025).
8. Reducing Cognitive Overload For A Better User Experience [Електронний ресурс] // Smashing Magazine. – URL: https://www.smashingmagazine.com/2016/09/reducing-cognitive-overload-for-a-better-user-experience/ (дата звернення: 16.05.2025).

9. Owl Labs. State of Remote Work 2020 [Електронний ресурс] // Owl Labs. – URL: https://resources.owllabs.com/state-of-remote-work/2020 (дата звернення: 16.05.2025).
10. Responsive layout grid [Електронний ресурс] // Material Design. – URL: https://m2.material.io/design/layout/responsive-layout-grid.html (дата звернення: 16.05.2025).
11. Huang E., Dey A., Forlizzi J. Adaptive UI for Collaborative Systems [Електронний ресурс] // ACM Digital Library. – URL: https://dl.acm.org/doi/fullHtml/10.1145/3544548.3581273 (дата звернення: 15.01.2025).
12. Designing Interfaces That Adapt to User Behavior [Електронний ресурс] // Medium. – URL: https://medium.com/design-bootcamp/designing-interfaces-that-adapt-to-user-behavior-d22508b96161 (дата звернення: 15.01.2025).
13. Kanban vs Scrum [Електронний ресурс] // Atlassian. – URL: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (дата звернення: 15.01.2025).
14. The Basics of Human-Centered Design: User Testing Demystified [Електронний ресурс] // Eficode. – URL: https://www.eficode.com/blog/the-basics-of-human-centered-design-user-testing-demystified (дата звернення: 15.01.2025).
15. ClickUp AI: Work Smarter with AI [Електронний ресурс] // ClickUp. – URL: https://clickup.com/ai (дата звернення: 15.01.2025).
16. M. Alelyani et al. (2024). Design and Evaluation of Intelligent Assistant in Collaborative Workflows [Електронний ресурс] // ScienceDirect. – URL: https://www.sciencedirect.com/science/article/pii/S2666721524000115 (дата звернення: 15.01.2025).
17. Mark N. et al. (2013). Trust in Collaborative Systems: Transparency, Accountability and Agency [Електронний ресурс] // ScienceDirect. – URL: https://www.sciencedirect.com/science/article/pii/S0306437913000768 (дата звернення: 15.01.2025).

18. MySQL vs PostgreSQL in 2024 [Електронний ресурс] // DBConvert: [сайт]. – URL: https://dbconvert.com/blog/mysql-vs-postgres-in-2024/ (дата звернення: 15.01.2025).
19. Using TypeScript with React [Електронний ресурс] // React Official Documentation: [сайт]. – URL: https://react.dev/learn/typescript (дата звернення: 15.01.2025).
20. TypeScript vs JavaScript: Key Differences [Електронний ресурс] // Appventurez: [сайт]. – URL: https://www.appventurez.com/blog/typescript-vs-javascript (дата звернення: 15.01.2025).
21. Exploring the Inner Workings of the Node.js Event Loop [Електронний ресурс] // Devtip.co: [сайт]. – URL: https://www.devtip.co/exploring-the-inner-workings-of-the-node-js-event-loop/ (дата звернення: 15.01.2025).
22. Express – Fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс] // Express.js: [сайт]. – URL: https://expressjs.com/ (дата звернення: 15.01.2025).
23. Програми для керування проєктами: огляд кращих сервісів [Електронний ресурс] // APIX-Drive: [сайт]. – URL: https://apix-drive.com/ua/blog/reviews/programy-dlja-keruvannja-proektami (дата звернення: 15.01.2025).
24. 11 кращих сервісів для планування проєктів [Електронний ресурс] // Worksection Blog: [сайт]. – URL: https://worksection.com/ua/blog/best-project-planning-tools.html (дата звернення: 15.01.2025).
25. Інструменти для управління проєктами [Електронний ресурс] // LiveBusiness: [сайт]. – URL: https://www.livebusiness.com.ua/ua/tools/pm/ (дата звернення: 15.01.2025).
26. 20 альтернатив Trello для команд [Електронний ресурс] // Chanty Blog: [сайт]. – URL: https://www.chanty.com/blog/uk/trello-alternatives-uk/ (дата звернення: 15.01.2025).
27. UI Grids – All You Need to Know [Електронний ресурс] // UXPin Blog. – URL: https://www.uxpin.com/studio/blog/ui-grids-how-to-guide/ (дата звернення: 02.06.2025).

ДОДАТОК А

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>React TODO App</title>
</head>
<body>
  <div id="root"></div>
</body>
</html>
```

```
import React from 'react';
// eslint-disable-next-line import/no-extraneous-dependencies
import { BrowserRouter as Router } from 'react-router-dom';
import AppRoutes from './routes';
import { UserProvider } from './context/UserContext';
import Navbar from './components/Navbar';

const App = () => {
  return (
    <UserProvider>
      <Router>
        <Navbar />
        <AppRoutes />
      </Router>
    </UserProvider>
  );
};

export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import './assets/styles/global.scss';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement,
);
```

```
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
```

```
import React, { useEffect, useState } from "react";
import { useUser } from "../context/UserContext";
import { getProjects } from "../services/projectService";
import { getProjectMembers, joinProject } from "../services/userProjectService";
import { useNavigate } from "react-router-dom";
import axios from "axios";
import { API_URL } from "../constants";
import JoinRequestModal from "../components/JoinRequestModal";

interface Project {
  id: number;
  title: string;
  description: string;
  author_id: number;
  created_at: string;
  access_type: "open" | "closed" | "request";
}

interface ProjectWithRole extends Project {
  role?: "Автор" | "Учасник";
  isAccessible: boolean;
}

interface JoinRequest {
  project_id: number;
  status: "pending" | "approved" | "rejected";
}

const HomePage = () => {
  const { user } = useUser();
  const [projects, setProjects] = useState<ProjectWithRole[]>([]);
  const [requests, setRequests] = useState<JoinRequest[]>([]);
  const [showModal, setShowModal] = useState(false);
  const [selectedProject, setSelectedProject] = useState<Project | null>(null);
  const navigate = useNavigate();

  useEffect(() => {
```

```
  const fetchData = async () => {
    try {
      const all = await getProjects();
      const reqs = await axios.get(`${API_URL}/join-requests/user/${user?.id}`);
      setRequests(reqs.data);

      const result: ProjectWithRole[] = [];

      for (const p of all) {
        const members = await getProjectMembers(p.id);
        const isAuthor = user?.id === p.author_id;
        const isMember = members.some((m: any) => m.id === user?.id);

        const role = isAuthor ? "Автор" : isMember ? "Учасник" : undefined;
        const isAccessible = isAuthor || isMember;

        result.push({ ...p, role, isAccessible });
      }

      setProjects(result);
    } catch (err) {
      console.error("Помилка при завантаженні проєктів:", err);
    }
  };

  fetchData();
}, [user]);

const handleJoin = (project: Project) => {
  if (project.access_type === "open") {
    joinProject(user!.id, project.id).then(() => {
      alert("Ви приєднались до проєкту");
      window.location.reload();
    }).catch((err) => {
      console.error("Помилка при приєднанні:", err);
      alert("Не вдалося приєднатись");
    });
  } else if (project.access_type === "request") {
    setSelectedProject(project);
    setShowModal(true);
  }
};
```

```
  const handleSendRequest = async (message: string) => {
    if (!selectedProject || !user) return;
    try {
      await axios.post(`${API_URL}/join-requests`, {
        user_id: user.id,
        project_id: selectedProject.id,
        message,
      });
      alert("Запит надіслано");
      setRequests((prev) => [...prev, { project_id: selectedProject.id, status:
"pending" }]);
    } catch (err) {
      console.error("Помилка при запиті:", err);
      alert("Не вдалося надіслати запит");
    } finally {
      setShowModal(false);
      setSelectedProject(null);
    }
  };

  return (
    <div style={{ padding: "2rem 3rem", backgroundColor: "#f8f9fb", minHeight:
"100vh" }}>
      <h2 style={{ marginBottom: "0.5rem" }}>Головна сторінка</h2>
      <p style={{ color: "#6b7280", marginBottom: "1.5rem" }}>
        Відображаються всі проєкти, доступні для перегляду або приєднання:
      </p>

      {projects.map((project) => {
        const isParticipant = project.isAccessible;
        if (project.access_type === "closed" && !isParticipant) return null;

        const request = requests.find((r) => r.project_id === project.id);

        return (
          <div
            key={project.id}
            style={{
              background: "#fff",
              border: "1px solid #ddd",
              borderRadius: "8px",
              padding: "1rem",
              marginBottom: "1.5rem",
```

```
            boxShadow: "0 2px 4px rgba(0,0,0,0.05)",
          }}
        >
          <h3 style={{ margin: "0 0 0.5rem" }}>{project.title}</h3>
          <p style={{ marginBottom: "0.75rem" }}>{project.description}</p>
          <p style={{ marginBottom: "0.25rem" }}>
            <strong>Дата створення:</strong> {new
Date(project.created_at).toLocaleString("uk-UA")}
          </p>
          <p style={{ marginBottom: "0.75rem" }}>
            <strong>Тип доступу:</strong> {project.access_type === "open"
              ? "Відкритий"
              : project.access_type === "closed"
                ? "Закритий"
                : "За запитом"}
            {project.role && <> <strong>Ваша роль:</strong> {project.role}</>}
          </p>

          {isParticipant ? (
            <button
              onClick={() => navigate(`/project/${project.id}`)}
              style={{
                backgroundColor: "#3b82f6",
                color: "#fff",
                border: "none",
                borderRadius: "6px",
                padding: "0.5rem 1rem",
                cursor: "pointer",
              }}
            >
              🔍 Переглянути
            </button>
          ) : request ? (
            <p style={{ color: "#f59e0b" }}>
              ⏳ Запит подано ({request.status === "pending"
                ? "очікує підтвердження"
                : request.status === "approved"
                  ? "схвалено"
                  : "відхилено"})
            </p>
          ) : (
            <button
```

```
                onClick={() => handleJoin(project)}
                style={{
                  backgroundColor: "#10b981",
                  color: "#fff",
                  border: "none",
                  borderRadius: "6px",
                  padding: "0.5rem 1rem",
                  cursor: "pointer",
                }}
              >
                🤝 Приеднатись
              </button>
            )}
          </div>
        );
      })}

      <JoinRequestModal
        isOpen={showModal}
        onClose={() => setShowModal(false)}
        onSubmit={handleSendRequest}
      />
    </div>
  );

};

export default HomePage;
```

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:5000/api',
  headers: {
    'Content-Type': 'application/json',
  },
});

export default api;
```

```
import express from "express";
import cors from "cors";
import projectRoutes from "./routes/projectRoutes";
```

```
import authRoutes from "./routes/authRoutes";
import userProjectRoutes from "./routes/userProjectRoutes";
import userRoutes from "./routes/userRoutes";
import taskRoutes from "./routes/taskRoutes";
import joinRequestRoutes from "./routes/joinRequestRoutes";
import messageRoutes from "./routes/messageRoutes";
import fileRoutes from "./routes/fileRoutes";
import path from "path";

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

app.use("/api/projects", projectRoutes);
app.use("/api/auth", authRoutes);
app.use("/api/user-projects", userProjectRoutes);
app.use("/api/users", userRoutes);
app.use("/api/tasks", taskRoutes);
app.use("/api/join-requests", joinRequestRoutes);
app.use("/api/messages", messageRoutes);
app.use("/api/files", fileRoutes);
app.use("/uploads", express.static(path.join(__dirname, "..", "uploads")));

app.listen(5000, () => {
  console.log("Server is running on http://localhost:5000");
});
```

```
import { Pool } from "pg";

const pool = new Pool({
  user: "postgres",
  host: "localhost",
  database: "project_platform",
  password: "berloga010604",
  port: 5432,
});

export default pool;
```

```
import { Request, Response } from "express";
import pool from "../db";

export const registerUser = async (req: Request, res: Response) => {
```

```
  const { name, email, password } = req.body;

  try {
    const existing = await pool.query("SELECT * FROM users WHERE email = $1",
[email]);
    if (existing.rows.length > 0) {
      return res.status(400).json({ message: "Користувач з таким email вже існує"
});
    }

    const result = await pool.query(
      "INSERT INTO users (name, email, password) VALUES ($1, $2, $3) RETURNING *",
      [name, email, password]
    );

    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error("Register error:", err);
    res.status(500).json({ message: "Помилка при реєстрації" });
  }
};

export const loginUser = async (req: Request, res: Response) => {
  const { email, password } = req.body;

  try {
    const result = await pool.query("SELECT * FROM users WHERE email = $1 AND
password = $2", [email, password]);

    if (result.rows.length === 0) {
      return res.status(401).json({ message: "Невірний email або пароль" });
    }

    res.json(result.rows[0]);
  } catch (err) {
    console.error("Login error:", err);
    res.status(500).json({ message: "Помилка при вході" });
  }
};
```

```
import express from "express";
import { loginUser, registerUser } from "../controllers/authController";
```

```
const router = express.Router();

router.post("/register", registerUser);
router.post("/login", loginUser);

export default router;
```

```
import { Request, Response } from "express";
import * as projectModel from "../models/projectModel";

export const getProjects = async (_req: Request, res: Response) => {
  try {
    const projects = await projectModel.getAllProjects();
    res.json(projects);
  } catch (error) {
    console.error("Get projects error:", error);
    res.status(500).json({ message: "Помилка при отриманні проєктів" });
  }
};

export const updateAccessType = async (req: Request, res: Response) => {
  const { id } = req.params;
  const { access_type } = req.body;

  if (!["open", "closed", "request"].includes(access_type)) {
    return res.status(400).json({ message: "Неприпустиме значення доступу" });
  }

  try {
    const updated = await projectModel.updateAccessType(Number(id), access_type);
    if (!updated) {
      return res.status(404).json({ message: "Проєкт не знайдено" });
    }
    res.json(updated);
  } catch (err) {
    console.error("Update access type error:", err);
    res.status(500).json({ message: "Помилка при оновленні доступу" });
  }
};

export const create = async (req: Request, res: Response) => {
  try {
    const { title, description, authorId } = req.body;
```

```
const project = await projectModel.createProject(title, description, authorId);
    res.status(201).json(project);
  } catch (error) {
    console.error("Create project error:", error);
    res.status(500).json({ message: "Помилка при створенні проєкту" });
  }
};

export const update = async (req: Request, res: Response) => {
  try {
    const { id } = req.params;
    const { title, description } = req.body;
    const updated = await projectModel.updateProject(Number(id), title,
description);
    res.json(updated);
  } catch (error) {
    console.error("Update project error:", error);
    res.status(500).json({ message: "Помилка при оновленні проєкту" });
  }
};

export const remove = async (req: Request, res: Response) => {
  try {
    const { id } = req.params;
    await projectModel.deleteProject(Number(id));
    res.json({ message: "Проєкт видалено" });
  } catch (error) {
    console.error("Delete project error:", error);
    res.status(500).json({ message: "Помилка при видаленні проєкту" });
  }
};

export const getOne = async (req: Request, res: Response) => {
  try {
    const { id } = req.params;
    const project = await projectModel.getProjectById(Number(id));

    if (!project) {
      return res.status(404).json({ message: "Проєкт не знайдено" });
    }

    res.json(project);
  } catch (error) {
```

```
    console.error("Get project by ID error:", error);
    res.status(500).json({ message: "Помилка при завантаженні проєкту" });
  }
};
```

```
import pool from "../db";

export const getAllProjects = async () => {
  const res = await pool.query("SELECT * FROM projects ORDER BY created_at DESC");
  return res.rows;
};

export const createProject = async (title: string, description: string, authorId:
number) => {
  const res = await pool.query(
    "INSERT INTO projects (title, description, author_id) VALUES ($1, $2, $3)
RETURNING *",
    [title, description, authorId]
  );
  return res.rows[0];
};

export const updateProject = async (id: number, title: string, description:
string) => {
  const res = await pool.query(
    "UPDATE projects SET title = $1, description = $2 WHERE id = $3 RETURNING *",
    [title, description, id]
  );
  return res.rows[0];
};

export const deleteProject = async (id: number) => {
  await pool.query("DELETE FROM projects WHERE id = $1", [id]);
};

export const getProjectById = async (id: number) => {
  const res = await pool.query("SELECT * FROM projects WHERE id = $1", [id]);
  return res.rows[0];
};

export const updateAccessType = async (id: number, access_type: string) => {
  const res = await pool.query(
    "UPDATE projects SET access_type = $1 WHERE id = $2 RETURNING *",
```

```
    [access_type, id]
  );
  return res.rows[0];
};
```

```
import { Router } from "express";
import * as projectController from "../controllers/projectController";

const router = Router();

router.get("/", projectController.getProjects);
router.post("/", projectController.create);
router.put("/:id", projectController.update);
router.delete("/:id", projectController.remove);
router.get("/:id", projectController.getOne);
router.put("/:id/access", projectController.updateAccessType);
export default router;
```

ДОДАТОК Б