НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “ Комп'ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Платформа для пошуку партнерів для спільних проєктів

Виконав : студент __4__ курсу, групи ІО-12
(шифр групи)
_____Капраренко Дмитро Миколайович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____професор, д.ф-м.н., с.н.с., Гордієнко Ю.Г. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль)_____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп'ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Капраренка Дмитра Миколайовича

1. Тема проєкту *Платформа для пошуку партнерів для спільних проєктів*
керівник проєкту професор, д.ф-м.н., с.н.с., Гордієнко Ю.Г.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року №1705-с

2. Термін здачі студентом закінченого проєкту _____

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

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

Розділ 1. Теоретичні основи розробки онлайн-платформ для співпраці.

Розділ 2. Аналіз існуючих рішень і постановка завдання.

Розділ 3. Розробка вебплатформи «Спільні проєкти».

Розділ 4. Тестування, оцінка ефективності та перспективи розвитку.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль			

7. Дата видачі завдання _____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*		
2.	*Вивчення та аналіз завдання*		
3.	*Розробка архітектури та загальної структури системи*		
4.	*Розробка структур окремих підсистем*		
5.	*Програмна реалізація системи*		
6.	*Оформлення пояснювальної записки*		
7.	*Захист програмного продукту*		
8.	*Передзахист*		
9.	*Захист*		

Студент-дипломник _____ Дмитро КАПРАРЕНКО
(підпис)

Керівник проєкту _____ Юрій ГОРДІЄНКО
(підпис)

АНОТАЦІЯ

У дипломній роботі розроблено інтерактивну вебплатформу «Спільні проєкти», призначену для підтримки командної співпраці в онлайн-середовищі. Проаналізовано сучасні вебтехнології (React, Node.js, Firebase/MongoDB), архітектуру SPA-додатків і вимоги до безпеки. Створено функціонал для реєстрації користувачів, пошуку та створення проєктів, обміну повідомленнями та файлами. Проведено тестування працездатності, UI/UX та розглянуто можливості масштабування платформи.

Ключові слова: командна співпраця, вебплатформа, React, Node.js, SPA, Firebase, MongoDB.

ANNOTATION

This thesis presents the development of the interactive web platform “Joint Projects,” designed to support online team collaboration. The project analyzes modern web technologies (React, Node.js, Firebase/MongoDB), SPA architecture, and security requirements. Key features were implemented, including user registration, project creation and search, messaging, and file sharing. The platform was tested for performance and usability, and scaling opportunities were considered.

Key words: team collaboration, web platform, React, Node.js, SPA, Firebase, MongoDB.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467100.002 ТЗ***	Платформа для	3		
			пошуку партнерів для			
			спільних проєктів			
			Технічне завдання			
	А4	***ІАЛЦ.467100.003 ПЗ***	Платформа для	81		
			пошуку партнерів для			
			спільних проєктів			
			Пояснювальна записка			
	А4	***ІАЛЦ.467100.004 Д1***	Платформа для	1		
			пошуку партнерів для			
			спільних проєктів			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.4671008.005 Д2***	Платформа для	1		
			пошуку партнерів для			
			спільних проєктів			
			Функціональна схема			
			(діаграма класів)			
	А4	***ІАЛЦ.467100.006 Д3***	Платформа для	1		
			пошуку партнерів для			
			спільних проєктів			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467100.007 Д4***	Платформа для	55		
			пошуку партнерів для			
			спільних проєктів			
			Текст програмного коду			

					ІАЛЦ.467100.001 ОА		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Капраренко Д. М.			*Платформа для пошуку партнерів для спільних проєктів* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Гордієнко Ю.Г					
						НТУУ "КПІ" ФІОТ ІО-12	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Платформа для пошуку партнерів для спільних проєктів*»

Київ – 2021

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання стосується розробки інтерактивної вебплатформи «Спільні проєкти» для організації ефективної командної співпраці у віртуальному середовищі. Система призначена для створення, пошуку та реалізації спільних проєктів із можливістю комунікації, обміну файлами та керування завданнями.

Область застосування охоплює сферу IT-розробки, освіти, краудсорсингових ініціатив, волонтерських і науково-дослідних проєктів, де потрібна цифрова взаємодія між учасниками незалежно від їхнього місцезнаходження.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання кваліфікаційної роботи освітнього рівня «бакалавр» зі спеціальності «Інженерія програмного забезпечення», затверджене на факультеті інформатики та обчислювальної техніки кафедри обчислювальної техніки Національного технічного університету України «КПІ ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою є створення SPA-додатку, який забезпечує зручну, доступну та безпечну платформу для командної співпраці. Платформа має підтримувати реєстрацію користувачів, створення проєктів, спілкування, обмін файлами, а також можливість масштабування та інтеграції зі сторонніми сервісами.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є офіційна документація до технологій React, Node.js, Firebase та MongoDB, науково-технічна література, відкриті API-платформи, а також публікації з тематики цифрової колаборації та веброзробки.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Платформа повинна бути зручною у використанні з інтуїтивно-зрозумілим інтерфейсом.
- Підтримка функцій реєстрації, авторизації, створення/пошуку проєктів, обміну повідомленнями та файлами.
- Можливість інтеграції з хмарними сервісами (Firebase, MongoDB Atlas).
- Реалізація клієнт-серверної архітектури з використанням REST API.
- Забезпечення захищеного зберігання даних, контроль доступу та базова перевірка надійності введених даних.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac або Linux.
- Підтримка сучасних браузерів (Chrome, Firefox, Edge).

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	
Вивчення та аналіз завдання	
Розробка архітектури та загальної структури системи	
Розробка структур окремих частин системи	
Програмна реалізація системи	
Виправлення помилок	
Оформлення пояснювальної записки	

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Платформа для пошуку партнерів для спільних проєктів*»

Київ – 2021

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

REST	(Representational State Transfer) Передача репрезентативного стану
API	(Application Programming Interface) Прикладний програмний інтерфейс
SQL	(Structured Query Language) Мова структурованих запитів
JSON	(JavaScript Object Notation) Запис об’єктів JavaScript
URL	(Uniform Resource Locator) Уніфікований локатор ресурсів
RPC	(Remote Procedure Call) Виклик віддалених процедур
СУБД	Система управління базами даних

ВСТУП

У сучасному світі динамічний розвиток цифрових технологій кардинально змінив підходи до організації праці, комунікації та реалізації спільних проєктів. Глобалізація, зростаюча росповсюдженість віддаленої роботи, краудсорсингових ініціатив і стартапів, потребують ефективних рішень для налагодження командної взаємодії. У відповідь на ці виклики зростає інтерес до створення платформ, які забезпечують учасникам проєктів можливість комунікувати, здійснювати обмін ідеями, працювати над завданнями в реальному часі незалежно від їхнього фізичного місцезнаходження.

Особливу актуальність набувають інтерактивні вебплатформи — зручні, швидкодоступні, багатофункціональні системи, що поєднують простоту користування з потужними засобами керування даними. Платформи типу Trello, Slack, GitHub, Asana та інші довели свою ефективність, проте більшість із них або орієнтовані на корпоративний сегмент, або мають обмежену доступність у плані відкритості коду та адаптації під локальні потреби. Тому виникає потреба у створенні вебплатформи, що була б адаптована під локальні умови, з акцентом на відкритість, можливість інтеграції сторонніх інструментів і підтримку спільного проєктного середовища.

Обраний напрям дослідження — створення вебплатформи «Спільні проєкти» — спрямований на розробку функціонального програмного продукту, який дозволяє користувачам реєструватися, створювати проєкти, знаходити однодумців, спілкуватися, обмінюватися матеріалами та ефективно співпрацювати у віртуальному просторі. Така платформа є не лише технічним інструментом, а й соціальним середовищем для реалізації творчого потенціалу та інноваційних ідей.

В основі цієї дипломної роботи лежить ідея побудови сучасної SPA (Single Page Application) вебплатформи з використанням стеку вебтехнологій

React, Node.js, TailwindCSS, Firebase або MongoDB, орієнтованої на безперебійну взаємодію користувачів у режимі реального часу. Враховуючи високі вимоги до швидкодії, безпеки, адаптивності та масштабованості, розробка такої системи потребує глибокого аналізу, ретельного проєктування, а також всебічного тестування.

Актуальність теми зумовлена зростаючими потребами суспільства в інструментах, які забезпечують ефективну дистанційну взаємодію команд у різних галузях: від IT-розробки до освіти та волонтерських ініціатив. У контексті постпандемійного світу, війни, цифрової трансформації бізнесу та державних інституцій, питання безперебійної, безпечної та гнучкої комунікації є першочерговими. Водночас, попри велику кількість інструментів для командної роботи, залишається потреба у рішеннях, які поєднують відкритість, доступність, простоту і сучасний функціонал. Розробка подібної системи є важливим кроком на шляху до цифрової інклюзії та розширення можливостей для реалізації командних ініціатив.

Метою дипломної роботи є розробка, реалізація та тестування інтерактивної вебплатформи «Спільні проєкти», яка забезпечує користувачам можливість створювати, знаходити та реалізовувати проєкти у режимі командної співпраці через сучасний інтерфейс та функціональність.

Об'єкт дослідження. Процеси організації командної співпраці в онлайн-середовищі.

Предмет дослідження. Методи та засоби створення інтерактивної вебплатформи для підтримки спільної проєктної діяльності на основі сучасних вебтехнологій.

Завдання дослідження:

- проаналізувати сучасні платформи для командної співпраці;
- визначити функціональні та нефункціональні вимоги до власної системи;
- обґрунтувати вибір технологічного стеку для реалізації вебплатформи;

- розробити архітектуру та структуру додатка;
- реалізувати ключовий функціонал системи;
- провести тестування працездатності та зручності інтерфейсу;
- оцінити перспективи масштабування та розвитку платформи.

Методи дослідження. У процесі виконання роботи застосовано методи аналізу і порівняння існуючих вебплатформ, методи структурного та об’єктно-орієнтованого проєктування, зокрема побудова UML-діаграм і структурних схем. Також було використано інструменти програмної інженерії, включаючи середовища для розробки вебінтерфейсів, системи керування базами даних, фреймворки для фронтенду та бекенду. Проведено модульне тестування, UI/UX-тестування, перевірку на стійкість до помилок та аналіз безпеки. Для валідації функціоналу використано як мануальне тестування, так і автоматизовані інструменти перевірки зручності використання, адаптивності та продуктивності системи.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ОНЛАЙН-ПЛАТФОРМ ДЛЯ СПІВПРАЦІ

1.1 Поняття спільної роботи в мережі: сучасні потреби та рішення

У сучасному цифровому середовищі поняття спільної роботи в мережі виходить далеко за межі звичайної електронної комунікації чи обміну файлами. Воно охоплює комплекс взаємопов'язаних процесів, які забезпечують синхронну або асинхронну взаємодію між учасниками проєктної діяльності незалежно від їхнього географічного розташування, часу та технічних обмежень. В основі такого типу співпраці лежить ідея про те, що спільна праця в онлайн-просторі повинна бути не менш ефективною, ніж робота в реальному офісі чи лабораторії, а за певних умов — навіть продуктивнішою [1].

Поняття «спільна робота в мережі» охоплює низку ключових компонентів, серед яких варто виділити спільний доступ до інформаційних ресурсів, координацію дій, взаємне редагування вмісту, миттєве обговорення питань і прийняття рішень у режимі реального часу або із затримкою. Таке середовище формує новий тип взаємодії між користувачами, заснований на взаємній довірі, гнучкості, прозорості й відкритості.

Загострення потреби в подібних платформах зумовлено низкою чинників, серед яких варто відзначити глобалізацію робочих процесів, поширення моделей віддаленої або гібридної зайнятості, розвиток інноваційних освітніх програм і наукових обмінів, а також зростання кількості малих і середніх команд, які реалізують складні міждисциплінарні проєкти. У такому контексті традиційні засоби комунікації, як-от електронна пошта чи месенджери, стають недостатніми для повноцінної підтримки командної роботи. Потрібні спеціалізовані рішення, які поєднують в собі засоби керування проєктами,

організації спільної бази знань, інтеграцію з хмарними сховищами, а також інструменти для управління ролями й доступом.

Сучасні рішення для підтримки спільної роботи в мережі представлені цілим спектром платформ, які мають різну функціональність, архітектуру та сферу застосування. Наприклад, платформи типу Slack, Microsoft Teams, Asana, Notion, GitHub чи Google Workspace орієнтовані на підтримку як спілкування, так і спільного управління завданнями, документами й ресурсами. Проте ці рішення часто мають обмеження — вони або є комерційними й недоступними для локальної адаптації, або надмірно складними у впровадженні для невеликих проєктів і команд. У зв'язку з цим виникає потреба у створенні нових гнучких платформ, які були б одночасно функціональними, доступними, адаптивними до специфічних вимог і такими, що підтримують відкриту архітектуру.

Особливе місце у сучасних рішеннях займає концепція відкритої та розподіленої співпраці, яка дозволяє долучати до проєктів широке коло учасників — як зацікавлених фахівців, так і волонтерів, які можуть долучитися до виконання завдань без формального найму чи складної авторизації. Такий підхід сприяє розвитку креативних середовищ, стимулює обмін досвідом і розширює можливості реалізації інновацій. Поняття спільної роботи в мережі є багатовимірним і потребує комплексного підходу до технічної реалізації. Сучасні потреби користувачів виходять за межі базових інструментів і вимагають створення інтерактивних платформ із гнучкою логікою, зручною навігацією, високим рівнем безпеки та підтримкою синхронної і асинхронної взаємодії. Саме на таких засадах ґрунтується розробка вебплатформи «Спільні проєкти», яка повинна відповідати зазначеним викликам і потребам цифрового часу.

Практичне втілення концепції спільної роботи в мережі дедалі частіше реалізується у вигляді повноцінних онлайн-платформ, які виконують функції віртуального робочого простору. Такі платформи можуть включати модулі управління завданнями (task managers), системи контролю версій документів,

засоби для групових відеоконференцій, чатів, обміну повідомленнями, а також функціонал для зберігання, редагування та обговорення файлів. Наприклад, платформи типу Google Docs та Microsoft Office 365 дозволяють декільком користувачам одночасно редагувати документи в реальному часі, залишати коментарі та бачити зміни, внесені іншими. Це є прикладом синхронної взаємодії, яка не потребує спеціального програмного забезпечення, оскільки все функціонує в межах браузера [2].

ТАБЛИЦЯ 1.1 – ХАРАКТЕРИСТИКА ТИПІВ СПІЛЬНОЇ РОБОТИ В МЕРЕЖІ

Тип спільної роботи	Основні характеристики	Тип взаємодії	Приклади інструментів	Сфера застосування
Синхронна	Користувачі працюють над спільним завданням одночасно, у реальному часі	Онлайн, у реальному часі	Google Docs, Microsoft Teams, Zoom	Освіта, розробка контенту, переговори
Асинхронна	Користувачі працюють у зручний для себе час, система зберігає історію змін	Затримка у часі	Trello, Notion, GitHub	IT-розробка, планування, ведення документації
Гібридна	Поєднання синхронної та асинхронної взаємодії; робота можлива в змішаному режимі	Комбінований	Slack, Asana, ClickUp	Командні проєкти, кросфункціональні команди
Спільна через API	Використання інтегрованих сервісів для обміну даними та координації	Автоматизована	Zapier, Webhooks, REST API	Автоматизація задач, системна інтеграція
Децентралізована	Робота в мережах без централізованого збереження даних, часто з відкритим кодом	Peer-to-peer	Nextcloud, CryptPad	Безпечна співпраця, активістські спільноти

У сфері розробки програмного забезпечення активно використовуються платформи типу GitHub, які дозволяють командам працювати над одним

проєктом через систему контролю версій, створювати гілки, об’єднувати зміни, відкривати pull-запити та організовувати обговорення щодо окремих фрагментів коду. Такі інструменти вже стали стандартом у світовій розробницькій практиці, демонструючи, що продуктивна співпраця можлива навіть у розподіленому форматі без фізичних зустрічей між учасниками.

Ще один приклад практичного застосування — це використання платформ типу Trello або Asana, які дозволяють структурувати проєкт у вигляді дошки з картками завдань. Кожне завдання може мати відповідального виконавця, дедлайни, коментарі, вкладення, мітки тощо. Подібна візуалізація сприяє прозорому управлінню робочим процесом і забезпечує всім учасникам проєкту однакове розуміння прогресу роботи. Водночас більшість таких сервісів обмежені у налаштуванні, є закритими або потребують підписки, що не завжди зручно або прийнятно для невеликих незалежних команд, громадських ініціатив, студентських проєктів або стартапів на етапі запуску.

На цьому тлі виникає потреба у створенні більш відкритих, кастомізованих, універсальних рішень, які базуються на сучасних вебтехнологіях і відповідають як технічним, так і етичним вимогам (конфіденційність, безпека, прозорість коду). Саме тому розробка таких платформ, як «Спільні проєкти», має не лише інженерне, а й соціальне значення: вона дозволяє підвищити цифрову автономію користувачів, надає їм інструменти для самоорганізації, зміцнює горизонтальні зв’язки у суспільстві й створює передумови для реалізації творчого потенціалу через співпрацю в онлайні.

1.2. Огляд сучасних технологій веброзробки (HTML, CSS, JavaScript, React, Node.js, API)

Розробка сучасних вебплатформ ґрунтується на застосуванні широкого спектра технологій, які охоплюють як клієнтську, так і серверну частину вебзастосунку. Ці технології утворюють так званий стек веброзробки, що

забезпечує повноцінну функціональність інтерфейсу, логіку взаємодії з користувачем, обробку даних, зв’язок із базою даних та інтеграцію з іншими сервісами. Створення платформ для спільної роботи потребує високого рівня інтерактивності, надійної передачі даних, адаптивного дизайну та масштабованої архітектури, що забезпечується саме завдяки сучасним засобам веброзробки.

Основою будь-якого вебінтерфейсу є мова HTML, яка використовується для побудови структурного кістяка вебсторінки. Вона визначає логічну організацію вмісту: заголовки, абзаци, таблиці, форми, кнопки, посилання та інші елементи. Завдяки HTML браузер отримує інструкцію про те, як візуалізувати структуру документа. Однак сам по собі HTML не відповідає за вигляд інтерфейсу — за це відповідає мова CSS, яка забезпечує стилізацію елементів. CSS дозволяє задавати кольори, шрифти, відступи, макети, а також використовувати анімації та медіазапити для адаптації сайту до різних розмірів екранів. Саме завдяки CSS вебінтерфейс набуває естетичної привабливості, зручності використання та адаптивності до мобільних пристроїв.

Функціональність, інтерактивність та реакція на дії користувача реалізуються мовою JavaScript, яка дозволяє створювати динамічні елементи, обробляти події, взаємодіяти з DOM-структурою сторінки, реалізовувати валідацію форм, клієнтську логіку, а також асинхронну обробку запитів через AJAX чи Fetch API. JavaScript є універсальним інструментом для побудови динамічних застосунків, які працюють без перезавантаження сторінки, реагують на взаємодію користувача й обмінюються даними із сервером у режимі реального часу.

На базі JavaScript побудовано багато сучасних фреймворків і бібліотек, найпопулярнішою з яких є React — бібліотека для створення інтерфейсів користувача, розроблена компанією Meta (Facebook). React дозволяє розробляти застосунки за компонентною архітектурою, де кожен елемент інтерфейсу є окремим незалежним блоком із власною логікою та станом. Такий

підхід сприяє повторному використанню коду, покращує читабельність та обслуговування проєкту, а також дає змогу реалізувати швидке оновлення даних на сторінці завдяки віртуальному DOM. У контексті платформи «Спільні проєкти» використання React дозволяє створювати односторінковий застосунок (SPA), у якому навігація між сторінками не супроводжується повним перезавантаженням, що значно покращує продуктивність і досвід користувача [3].

Серверна логіка, обробка запитів, автентифікація, взаємодія з базою даних реалізуються за допомогою технології Node.js — серверного середовища виконання JavaScript-коду, побудованого на рушії V8. Node.js є надзвичайно ефективним у роботі з великою кількістю одночасних підключень завдяки своїй неблокуючій, подієво-орієнтованій моделі. Це дозволяє створювати масштабовані сервери, здатні обробляти тисячі одночасних користувачів. Разом із фреймворком Express, Node.js забезпечує зручну архітектуру для побудови RESTful API — інтерфейсів, через які клієнтський інтерфейс взаємодіє із серверною частиною. Наприклад, саме через API здійснюється реєстрація користувача, створення нового проєкту, отримання списку доступних проєктів, надсилання повідомлень або обмін файлами.

Сучасна веброзробка не обходиться без взаємодії з базами даних, які можуть бути реляційними (MySQL, PostgreSQL) або нереляційними (MongoDB, Firebase). Для платформи спільної роботи доцільним є використання документо-орієнтованих баз даних, таких як MongoDB, які зручно зберігають інформацію у вигляді об’єктів формату JSON і легко масштабуються. Ще одним прикладом сучасного хмарного рішення є Firebase, який, окрім бази даних у реальному часі, також надає сервіси автентифікації, хостингу, аналітики та обробки повідомлень.

ТАБЛИЦЯ 1.2 – ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ТЕХНОЛОГІЙ ВЕБРОЗРОБКИ

Технологія	Призначення	Тип середовища	Особливості використання	Приклад застосування у проєкті
HTML	Створення структури вебсторінки	Фронтенд	Визначає компонування елементів (текст, форми, кнопки)	Макет сторінки реєстрації
CSS	Стилізація елементів сторінки	Фронтенд	Відповідає за дизайн, кольори, шрифти, адаптивність	Мобільна версія інтерфейсу
JavaScript	Динамічна взаємодія користувача з сайтом	Фронтенд	Реагує на події, змінює DOM, виконує логіку без перезавантаження	Валідація форм авторизації
React	Компонентна побудова інтерфейсу	Фронтенд (SPA)	Побудова SPA, використання станів і життєвого циклу компонентів	Пошук та фільтрація проєктів
Node.js	Серверна логіка обробки запитів	Бекенд	Працює на JavaScript, дозволяє створити REST API та керувати маршрутизацією	Обробка запиту на логін
Express.js	Фреймворк для Node.js	Бекенд	Спрощує написання серверного коду, забезпечує структуру та middleware	Обробка POST/GET запитів
MongoDB	База даних (документоорієнтована)	Сховище	Гнучка структура даних (JSON), масштабованість,	Збереження профілю користувача

			зручна інтеграція з Node.js	
Firebase	Хмарна платформа Google	База даних + API	Реєстрація, автентифікація, хостинг, реальна база даних з оновленням у реальному часі	Реєстрація нових учасників
REST API	Інтерфейс обміну даними між клієнтом і сервером	Фронт+бекенд	Забезпечує взаємодію між інтерфейсом і логікою, реалізує архітектуру клієнт–сервер	Отримання переліку проєктів

Узагальнюючи показані характеристики, можна дійти висновку, що кожна з розглянутих технологій виконує чітко визначену функцію в архітектурі сучасного вебзастосунку. Їх вдало скомбіноване рішення дає змогу створювати інтерактивні платформи, здатні не лише відображати інформацію, а й динамічно реагувати на дії користувачів, підтримувати оновлення в реальному часі, здійснювати перевірку введених даних, зберігати великі обсяги інформації та забезпечувати надійний обмін даними між клієнтською та серверною частинами.

У контексті створення платформи «Спільні проєкти» використання React у парі з Node.js дозволяє забезпечити ефективну взаємодію між інтерфейсом і сервером, а застосування Firebase або MongoDB спрощує управління даними й масштабування системи. REST API слугує універсальним інтерфейсом обміну інформацією між усіма модулями платформи, забезпечуючи логічну цілісність програмної архітектури.

Важливу роль у цьому процесі відіграє компонентна структура розробки, коли кожен функціональний блок системи розробляється, тестується й оновлюється окремо. Це значно спрощує процес командної роботи над платформою, дозволяє паралельно реалізовувати різні модулі (наприклад, авторизацію,

створення проєктів, систему комунікації), а також забезпечує високу гнучкість і можливість швидкого розширення функціоналу в майбутньому.

1.3 Принципи побудови SPA-додатків на основі React

Односторінкові застосунки (SPA — Single Page Application) являють собою сучасний підхід до створення вебінтерфейсів, у яких взаємодія з користувачем має місце в межах єдиної HTML-сторінки, без повторного завантаження всієї структури з сервера при переході між розділами. Цей тип застосунків дозволяє досягти високого рівня динаміки, швидкодії та інтерактивності. Користувач, взаємодіючи з таким інтерфейсом, отримує відчуття роботи з десктопною програмою, хоча вся система функціонує у браузері. Ключовими характеристиками SPA-додатків є часткове оновлення вмісту сторінки, гнучка маршрутизація та логіка керування станом користувацького інтерфейсу на стороні клієнта [4].

React є однією з найвідоміших бібліотек для побудови SPA-додатків, оскільки вона забезпечує ефективне керування інтерфейсом за допомогою компонетної архітектури та віртуального DOM. Компонентний підхід означає, що весь інтерфейс складається з незалежних, ізольованих частин — компонентів, які відповідають за певну ділянку UI і можуть повторно використовуватися в різних частинах додатка. Кожен компонент описується у вигляді JavaScript-коду з використанням JSX — синтаксису, який дозволяє комбінувати HTML-подібні структури з логікою поведінки. Завдяки цьому інтерфейс стає гнучким і масштабованим, а зміни в окремих елементах не потребують переробки всієї структури.

Однією з основ React є віртуальний DOM — оптимізований внутрішній механізм, який дозволяє швидко оновлювати лише ті частини інтерфейсу, які справді змінилися. Після кожної зміни стану компонентів React порівнює нове дерево віртуального DOM з попереднім, визначає, які саме елементи зазнали

змін, і мінімально оновлює реальний DOM. Це суттєво зменшує навантаження на браузер, покращує продуктивність і забезпечує плавну взаємодію з інтерфейсом, що особливо важливо для платформ з великою кількістю інтерактивних елементів.

Важливим аспектом при створенні SPA-додатків є реалізація клієнтської маршрутизації, яка забезпечується за допомогою бібліотек типу React Router. Коли користувач переходить до іншого розділу платформи, наприклад, від панелі проєктів до профілю, змінюється лише стан інтерфейсу та URL-адреса, але не відбувається фактичне завантаження нової HTML-сторінки. Це дає змогу досягти миттєвої навігації між розділами та створити відчуття безшовної взаємодії.

Ще одним важливим принципом побудови SPA на основі React є централізоване керування станом застосунку. У процесі роботи інтерфейсу виникає необхідність обміну даними між різними компонентами, які можуть бути віддаленими у структурі. Для цього використовуються механізми контексту React, а у складніших додатках — спеціалізовані бібліотеки, такі як Redux або Zustand. Завдяки цьому забезпечується цілісність даних у додатку, синхронізація дій користувача з відображенням інформації, а також ефективна обробка змін.

Окрім архітектурних принципів, при побудові SPA на основі React важливо враховувати аспекти взаємодії з сервером, зокрема реалізацію асинхронних запитів через Fetch API або Axios. Це дозволяє отримувати або надсилати дані без оновлення сторінки, наприклад, при реєстрації користувача, створенні нового проєкту, пошуку або редагуванні інформації. Обробка таких запитів виконується в компонентах React за допомогою хуків — useEffect, useState тощо — які забезпечують контроль над життєвим циклом компонентів і поведінкою інтерфейсу.

Розробка SPA-додатків на основі React також передбачає високі вимоги до якості UI/UX, адаптивності інтерфейсу та доступності. Компонентний підхід

полегшує тестування, дозволяє легко підтримувати проєкт у майбутньому, розширювати функціональність без ризику порушення вже реалізованих частин системи. Завдяки активній спільноті, великій кількості документації та підтримці з боку Meta, React залишається одним з найнадійніших і найперспективніших інструментів для побудови масштабованих SPA-застосунків, що і зумовлює його вибір як основи для розробки платформи «Спільні проєкти» [5].

ТАБЛИЦЯ 1.3 – ОСНОВНІ ПРИНЦИПИ ПОБУДОВИ SPA-ДОДАТКІВ НА ОСНОВІ REACT

Принцип	Суть принципу	Значення для SPA	Приклад у проєкті "Спільні проєкти"
Компонентність	Інтерфейс складається з окремих незалежних частин-компонентів	Полегшує розробку, повторне використання коду, масштабування	Реалізація компонентів: реєстрації, навігації, списку проєктів
Віртуальний DOM	Внутрішнє дерево елементів React, що дозволяє швидко виявляти зміни	Підвищує швидкодію та ефективність оновлення інтерфейсу	Оновлення списку проєктів без перезавантаження сторінки
Клієнтська маршрутизація	Керування URL і переходами між розділами без перезавантаження	Створює відчуття безшовної навігації між сторінками	Перехід між «Проєкти», «Профіль», «Повідомлення»
Керування станом	Синхронізація даних між компонентами через useState/useContext/ Redux	Забезпечує цілісність і актуальність даних у всьому застосунку	Відображення проєктів, у яких бере участь користувач
Асинхронна взаємодія з сервером	Здійснення HTTP-запитів без оновлення сторінки	Динамічне завантаження/відправка даних без перешкод для користувача	Пошук, створення, оновлення проєктів через API
JSX та хуки	Комбінування HTML-структури з	Підвищує гнучкість та	Форма реєстрації з

	логікою поведінки в одному коді	читабельність коду компонентів	валідацією на основі useState та useEffect

Наведена таблиця дозволяє узагальнити основні архітектурні та логічні засади, що формують підхід до побудови SPA-додатків за допомогою React. Усі принципи, зазначені вище, взаємопов’язані між собою й утворюють єдину систему, яка забезпечує високу швидкодію, масштабованість і користувацьку зручність. Компонентність дозволяє розробляти інтерфейс поетапно, створюючи повторно використовувані блоки з чітко визначеним функціоналом. Віртуальний DOM гарантує ефективну обробку змін без навантаження на браузер, а клієнтська маршрутизація забезпечує плавність переходів між розділами.

Окрему увагу заслуговує керування станом, оскільки саме воно забезпечує узгодженість даних між усіма частинами застосунку. Наприклад, коли користувач змінює свій профіль або бере участь у новому проєкті, ці зміни повинні відображатися миттєво в усіх відповідних компонентах інтерфейсу. Це досягається шляхом правильного використання хуків, контексту або сторонніх бібліотек. Додатково, інтеграція з сервером через API дозволяє SPA залишатися «легким» на клієнтській стороні, передаючи лише необхідні дані й уникаючи надмірного навантаження.

Варто зазначити, що побудова SPA-додатків також накладає певні вимоги до розробника, зокрема щодо структурування коду, оптимізації рендерингу, розділення логіки за принципом «контейнер – представлення», налаштування маршрутизації та впровадження ефективного механізму обробки помилок. Однак завдяки великій екосистемі React, безлічі готових бібліотек і розширень ці завдання можуть бути вирішені з високим рівнем ефективності.

Підсумовуючи, можна сказати, що React як основа для побудови SPA-додатків забезпечує потужну функціональну платформу для розробки

інноваційних інтерфейсів, які повністю відповідають очікуванням користувача щодо швидкості, гнучкості та інтерактивності. Саме ці переваги обумовлюють вибір React як ядра розробки вебплатформи «Спільні проєкти» — застосунку, покликаного забезпечити комфортну та ефективну взаємодію між учасниками командних ініціатив у цифровому середовищі.

1.4. Проблеми безпеки та збереження даних у сервісах співпраці

У контексті стрімкого розвитку цифрових технологій та широкого розповсюдження онлайн-сервісів, призначених для командної роботи, питання безпеки та захисту персональних даних набувають ключового значення. Платформи для співпраці, зокрема односторінкові вебзастосунки, акумулюють велику кількість чутливої інформації: особисті дані користувачів, конфіденційні документи, листування, файли, результати спільної роботи. Саме тому недоліки в реалізації механізмів безпеки можуть призвести до серйозних наслідків — витоку інформації, фінансових збитків, втрати довіри користувачів або навіть повного припинення функціонування сервісу [6].

Однією з найбільш поширених загроз для платформ є несанкціонований доступ до облікових записів, що зазвичай пов’язано зі слабо захищеними механізмами автентифікації. Якщо система автентифікації не має багатофакторного захисту, шифрування паролів або перевірки активності користувача, зловмисники можуть скористатися вразливістю для викрадення облікових даних. Особливо небезпечними є атаки типу brute force, фішингові схеми та перехоплення мережевого трафіку при роботі через незахищені з’єднання. Ще однією проблемою є збереження паролів у незашифрованому вигляді або використання застарілих алгоритмів хешування, що відкриває можливості для швидкого дешифрування даних у разі доступу до бази.

Серйозною загрозою також є вразливості у програмному коді, зокрема такі як XSS (міжсайтове скриптування) і SQL-ін’єкції. У першому випадку зловмисник

може вставити шкідливий код у форму вводу, який буде виконано у браузері іншого користувача, що дозволить викрасти cookie-файли, токени автентифікації або відобразити фальшиві вікна авторизації. У випадку SQL-ін'єкцій існує ризик отримання несанкціонованого доступу до бази даних, видалення або зміни записів, а також повного копіювання вмісту таблиць. Такі атаки можливі лише у разі, коли не застосовується валідація введених даних, а запити до бази формуються динамічно без відповідного захисту.

Окремим викликом є забезпечення захищеної передачі даних між клієнтом і сервером. Якщо платформа не використовує протокол HTTPS, то весь трафік передається у відкритому вигляді, що робить його вразливим до перехоплення через атаки типу «людина посередині» (MITM). Це стосується як облікових даних, так і внутрішньої переписки між учасниками проєкту, що є критичним для платформ, орієнтованих на співпрацю та обмін конфіденційною інформацією.

Ще однією важливою проблемою є неналежна організація прав доступу до даних. Якщо користувачі можуть бачити або редагувати ресурси, що їм не належать, це створює ризики витоку, маніпуляцій чи зловживань. Система має чітко обмежувати дії відповідно до ролей — наприклад, звичайний учасник не повинен мати доступу до адміністративних функцій або можливості змінювати налаштування проєкту. Важливою умовою є перевірка прав не лише на інтерфейсному рівні, але й на стороні сервера, оскільки клієнтський код легко змінити вручну або підробити запити за допомогою спеціальних інструментів.

Проблемою для онлайн-платформ також є надійне збереження даних у хмарних середовищах. Часто розробники покладаються на сервіси типу Firebase, MongoDB Atlas або Amazon Web Services, які надають вбудовані механізми резервного копіювання та реплікації. Проте помилки в налаштуванні політик доступу або невчасне оновлення дозволів можуть відкрити бази даних для загального доступу. У деяких випадках відкриті індекси пошукових систем

навіть знаходили незахищені бази даних, доступ до яких не був обмежений жодним захистом.

Крім зовнішніх загроз, існує також людський фактор — ненавмисне видалення даних, помилкове надання доступу, завантаження заражених файлів, або збереження облікових даних у відкритому вигляді на стороні клієнта. Для мінімізації цих ризиків необхідно реалізувати відповідні протоколи дій, автоматичне логування, повідомлення про спроби доступу, а також наявність резервних копій усіх критичних даних, які можна швидко відновити у випадку інциденту.

Таким чином, забезпечення безпеки та збереження даних у сервісах для спільної роботи потребує системного підходу, що охоплює всі рівні: від інтерфейсу до хмарної інфраструктури. Для платформи типу «Спільні проєкти» ключовими аспектами є реалізація захищеної авторизації з використанням JWT або OAuth2, обов’язкове шифрування даних під час передачі та зберігання, сувора перевірка прав доступу, очищення та валідація даних на вході, обмеження доступу до внутрішніх API, а також резервне копіювання бази проєктів та журналів активності користувачів. Лише комплексна реалізація зазначених заходів дозволить гарантувати користувачам збереження їхніх даних, довіру до сервісу та стабільну роботу платформи в умовах зростаючих кіберзагроз [7].

ТАБЛИЦЯ 1.4 – ОСНОВНІ ПРОБЛЕМИ БЕЗПЕКИ У ВЕБСЕРВІСАХ ДЛЯ СПІВПРАЦІ

Тип загрози	Опис проблеми	Можливі наслідки	Методи захисту
Несанкціонований доступ	Отримання контролю над обліковим записом без дозволу власника	Викрадення даних, зміна або видалення інформації	Двофакторна автентифікація, шифрування паролів, тайм-аут сесій
XSS-атаки (міжсайтове скриптування)	Впровадження шкідливого JavaScript-коду у вебсторінку	Крадіжка токенів, редиректи, фішинг	Фільтрація вводу, escape-символи, Content Security Policy
SQL-ін’єкції	Маніпуляції з SQL-запитами для доступу до бази даних	Зміна, видалення або викрадення даних	Параметризація запитів, ORM, валідація введення
Відсутність HTTPS	Передача даних відкритим текстом по мережі	Перехоплення сесії, крадіжка паролів	Використання SSL-сертифікатів, перенаправлення на HTTPS
Неналежна авторизація доступу	Доступ користувачів до даних або функцій, які їм не належать	Витік інформації, зміни в чужих проєктах	Контроль ролей, перевірка прав на сервері
Ненадійне зберігання даних	Відсутність резервного копіювання або слабкі політики доступу	Втрата важливої інформації, відкриті бази даних	Бекапи, шифрування, обмеження доступу до баз
Людський фактор	Помилкове надання доступу, збереження паролів у відкритому вигляді	Випадковий витік або видалення даних	Навчання користувачів, логування дій, аудит безпеки

Як показано в таблиці, спектр загроз для сервісів командної співпраці є надзвичайно широким і охоплює як технічні, так і організаційні аспекти. Найбільш вразливими є компоненти, що стосуються автентифікації, передачі даних, взаємодії з базами, а також права доступу. Незахищені або погано структуровані рішення на будь-якому з цих рівнів відкривають доступ до всієї інформаційної архітектури платформи, що може бути використано зловмисниками не лише для порушення функціональності, але й для завдання репутаційної шкоди власникам сервісу чи розробникам.

Окремо варто зазначити, що в умовах швидкого розвитку інтерфейсів користувача на основі JavaScript-бібліотек і фреймворків, більшість обробки подій і логіки відбувається безпосередньо у браузері. Це створює ілюзію захищеності, однак насправді вся критично важлива перевірка прав і обмеження доступу повинні виконуватися на стороні сервера. Веброзробники, особливо на етапі MVP (мінімально життєздатного продукту), часто нехтують цим, внаслідок чого користувачі можуть через інструменти браузера або мережевого аналізатора отримати прямий доступ до внутрішніх запитів, API, адміністративних функцій або прихованих даних. Це доводить, що безпека — це не лише технічне рішення, а й принципова частина всієї архітектури платформи.

У сервісах, що підтримують завантаження файлів, наприклад резюме, презентацій, архівів чи медіа, додаткову загрозу становлять шкідливі вкладення. Відсутність перевірки типу файлу, його вмісту або джерела призводить до можливості впровадження вірусів або виконуваних скриптів, які запускаються вже після завантаження на комп’ютер іншого користувача. У зв’язку з цим важливо реалізувати попередню перевірку файлів на віруси, обмеження дозволених форматів і обов’язкове сканування вмісту перед збереженням у базі чи хмарному сховищі.

Також не слід забувати про вимоги до збереження логів дій користувачів, що дає змогу відстежувати будь-які підозрілі активності, виявляти спроби

несанкціонованого доступу, а також забезпечує можливість аудитів або технічного аналізу інцидентів. Наявність журналів дій — це не лише інструмент захисту, але й механізм відповідальності, що дозволяє відтворити хронологію подій у разі збоїв, помилок або атак.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було здійснено комплексний теоретико-методологічний аналіз ключових аспектів, пов’язаних із проєктуванням сучасних онлайн-платформ для командної співпраці. Встановлено, що цифрова колаборація є не лише зручним інструментом взаємодії, але й основною складовою нової моделі організації праці в умовах глобалізації, децентралізації та переходу до віддаленого формату роботи. Поняття спільної роботи в мережі охоплює як технічні, так і соціальні механізми координації дій, обміну інформацією, спільного редагування матеріалів і колективного прийняття рішень, що визначає особливі вимоги до архітектури програмних рішень, які мають це забезпечити.

У процесі розгляду технологічної бази було детально проаналізовано стек сучасних інструментів веброзробки, зокрема HTML, CSS, JavaScript, React, Node.js, MongoDB, Firebase та REST API. Ці технології формують основу для створення ефективних, адаптивних, масштабованих і безпечних вебдодатків, що функціонують у форматі SPA. Особливу увагу було приділено принципам побудови односторінкових застосунків на основі React, оскільки саме ця бібліотека забезпечує компонентну архітектуру, швидку маршрутизацію, реактивне оновлення вмісту й ефективне керування станом інтерфейсу. Ці можливості відкривають новий рівень інтерактивності та забезпечують зручність користувача навіть при складній логіці взаємодії між модулями платформи.

Розгляд питань безпеки та збереження даних у платформах співпраці дозволив окреслити основні загрози, з якими стикаються сучасні вебсервіси: несанкціонований доступ, XSS-атаки, SQL-ін’єкції, вразливості у маршрутизації, витоки даних через відкриті API, помилки авторизації та неналежне збереження інформації. Було визначено, що створення безпечної

архітектури передбачає не лише використання відповідних протоколів і алгоритмів захисту, а й вбудовану логіку обмеження прав доступу, багаторівневу автентифікацію, реалізацію політик зберігання, резервне копіювання, аудит дій та логування. Усі ці елементи повинні функціонувати як єдина система, яка не лише захищає платформу від загроз, але й підтримує довіру з боку користувачів, забезпечуючи конфіденційність, цілісність та доступність даних.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ І ПОСТАНОВКА ЗАВДАННЯ

2.1. Аналіз аналогів: CollabFinder, Reddit / Opensource thread, Bepartner

У процесі розробки власної платформи для командної співпраці доцільно здійснити ретельний аналіз існуючих рішень, що вже представлені на ринку. Це дозволяє не лише виявити переваги і недоліки конкурентів, а й сформувати обґрунтовану модель функціональності та користувацького досвіду. Серед найбільш показових прикладів аналогічних платформ можна виокремити такі сервіси, як CollabFinder, гілка Reddit / Opensource thread, а також українська платформа Bepartner. Кожне з наведених рішень репрезентує окремий підхід до реалізації ідеї пошуку партнерів для спільної роботи над проєктами, тому їх аналіз має ключове значення для формування завдання на розробку власного рішення [8].

Платформа CollabFinder, розроблена у США, позиціонує себе як сервіс для творчих професіоналів, які шукають партнерів для реалізації ідей у сфері технологій, мистецтва, дизайну, стартапів або дослідницьких ініціатив. Її головна перевага полягає у простому та інтуїтивно зрозумілому інтерфейсі, що орієнтований на швидке заповнення профілю, додавання опису проєкту і перегляд відповідних кандидатів. Користувачі можуть вказувати свої навички, зацікавлення, тип проєкту, що дозволяє здійснювати пошук партнерів за релевантністю. Водночас ця платформа не має розширеного функціоналу для безпосередньої співпраці — вона більше нагадує вітрину або каталог ініціатив, у межах якого учасники самостійно продовжують комунікацію на сторонніх платформах. Відсутність вбудованих інструментів для обміну файлами,

задачами чи керування спільною роботою суттєво обмежує її застосування як повноцінного інструменту для тривалої співпраці команд.

Ще одним цікавим прикладом є спеціалізована гілка Reddit, присвячена Opensource-проєктам. Reddit, як платформа, є соціальною мережею зі структурою форуму, де користувачі створюють пости, коментують і взаємодіють через голосування. Гілка Opensource thread — це спільнота, де учасники діляться ідеями відкритих проєктів, шукають волонтерів і розробників, а також обговорюють стратегії розвитку своїх ініціатив. Основною перевагою такої форми є висока активність спільноти, швидкий зворотний зв’язок та природна селекція найактуальніших тем через систему голосів. Однак ця модель має свої обмеження: вона не забезпечує структури або системності в керуванні проєктами, не має окремих кабінетів користувача, інструментів організації завдань чи файлів. Уся взаємодія відбувається у вигляді постів, які згодом зміщуються нижче за рейтингом. У зв’язку з цим платформа більше слугує початковим етапом залучення учасників, а не повноцінною основою для довготривалої командної роботи.

На противагу цим прикладам, український сервіс Bepartner орієнтований на підтримку стартап-екосистеми шляхом створення цифрової платформи для пошуку бізнес-партнерів. Його функціонал дозволяє зареєстрованим користувачам створювати стартап-профілі, описувати ідею, додавати презентації, шукати співзасновників, інвесторів або фахівців, необхідних для розвитку. Інтерфейс є мінімалістичним, проте продуманим, містить фільтри пошуку, систему заявок на приєднання, а також базові функції чату між учасниками. Важливо зазначити, що Bepartner орієнтований більше на бізнес-складову співпраці, де фокус робиться на заснуванні й просуванні стартапів, тому технічна або програмна частина роботи в середовищі сервісу реалізується лише на мінімальному рівні. Відсутність повноцінного простору для зберігання проєктної документації, спільної роботи з кодом, управління завданнями чи

модуля комунікації в реальному часі зменшує ефективність платформи в рамках IT- або R&D-команд.

Загалом, аналіз наявних рішень демонструє, що на ринку існує запит на платформи, які не лише полегшують знайомство між потенційними учасниками проєктів, а й забезпечують технічне середовище для повноцінної спільної роботи. Більшість існуючих сервісів зосереджуються або на етапі знайомства, або на обговоренні ідей, або на пошуку фінансування, але не охоплюють усі етапи реалізації командного проєкту — від ідеї до завершення. Саме тому розробка платформи «Спільні проєкти» має на меті не лише повторити кращі практики, а й інтегрувати технічні рішення, які забезпечать повний цикл цифрової співпраці: пошук, комунікація, управління, реалізація, зворотний зв’язок. У цьому полягає її унікальність і конкурентна перевага на фоні описаних аналогів.

ТАБЛИЦЯ 2.1 – ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА АНАЛОГІЧНИХ ПЛАТФОРМ

Платформа	Цільова аудиторія	Функціональність	Переваги	Обмеження
CollabFinder	Творчі професіонали, розробники, дизайнери	Пошук партнерів за навичками, створення профілю, опис проєкту	Простий інтерфейс, зручна реєстрація, фокус на ідеї	Відсутність інструментів для подальшої співпраці, немає спільного середовища
Reddit / Opensource thread	Відкриті спільноти, розробники, волонтери	Обговорення, пошук учасників для opensource-проєктів	Швидкий зворотний зв'язок, активна спільнота, популярна платформа	Відсутня структура, не підтримує командну роботу, немає збереження інформації
Bepartner	Стартапи, підприємці, бізнес-засновники	Пошук співзасновників, профілі стартапів, внутрішній чат	Фокус на бізнес, локалізований сервіс, простота використання	Обмежений функціонал для IT-команд, відсутність управління проєктом і файлами

Наведена таблиця узагальнює ключові характеристики трьох платформ і демонструє, що хоча кожна з них виконує певну роль у сфері пошуку партнерів для спільної роботи, жодна з них не охоплює повний цикл реалізації командного проєкту. CollabFinder є зручною платформою для встановлення початкових контактів, проте не надає інструментів для подальшої спільної діяльності або керування розробкою. Reddit, у свою чергу, має величезний потенціал завдяки масштабності аудиторії та динамічності спілкування, проте повністю позбавлений інструментів структурованої співпраці, обмежуючись функціональністю дискусійного форуму. Платформа Bepartner ближча до

моделі бізнес-орієнтованої взаємодії, проте її фокус на підприємництві та відсутність технічних можливостей для роботи над спільними завданнями чи зберігання проєктної інформації обмежує її застосування у широкому спектрі командних ініціатив, особливо технічного характеру.

Всі три рішення, попри відмінності в реалізації, мають одну спільну рису — розрив між етапом знайомства та реальним спільним виконанням роботи. Користувачі змушені переносити взаємодію на сторонні сервіси: месенджери, репозиторії коду, інструменти для управління завданнями, хмарні диски або платформи відеозв'язку. Це не лише ускладнює процес співпраці, а й створює додаткові бар'єри для командної динаміки, знижує мотивацію нових учасників і створює ризики розпорошення інформації.

З урахуванням виявлених недоліків та обмежень, постає необхідність створення вебплатформи нового типу — такої, що не лише дозволяє знайти партнера, але й надає повноцінне середовище для реалізації спільних ідей. У такій системі ключовими функціональними елементами мають стати централізоване управління проєктами, інструменти комунікації, модуль обміну файлами, система ролей і доступу, а також інтеграція з хмарними та відкритими API для подальшого розширення. Саме таке комплексне бачення й буде закладене в основу формування завдання на розробку платформи «Спільні проєкти», яке викладено у наступних підрозділах.

2.2. SWOT-аналіз популярних платформ

Для комплексного розуміння сильних і слабких сторін існуючих вебплатформ, що орієнтовані на командну співпрацю, доцільним є проведення SWOT-аналізу — методики, яка дозволяє оцінити внутрішні характеристики системи та зовнішні чинники, що можуть сприяти або перешкоджати її розвитку. Такий аналіз застосовується для стратегічного планування й

допомагає визначити пріоритетні напрями удосконалення продукту. У межах цього дослідження SWOT-аналіз застосовується до трьох популярних рішень: CollabFinder, Reddit (Opensource thread) та Bepartner. Його результати дозволяють обґрунтувати доцільність створення нової платформи з урахуванням наявних успішних практик і виявлених прогалин [9].

Аналіз платформи CollabFinder засвідчив, що її основною сильною стороною є доступність, простота інтерфейсу та орієнтація на користувача, який бажає швидко знайти партнера для реалізації ідеї. Її дизайн дозволяє фокусуватися на змісті, а не на технологіях, що створює приємний досвід використання. Проте водночас ця ж простота обмежує функціональність платформи — вона не підтримує інструментів для подальшої співпраці. Користувачі змушені самостійно переносити взаємодію на інші сервіси, що створює ризик втрати зв’язку або ефективності. Зовнішні можливості для CollabFinder полягають у потенціалі розширення функціоналу через інтеграцію API, створення внутрішніх інструментів комунікації або зберігання документів, проте відсутність регулярних оновлень і технічна інертність з боку розробників створює загрозу поступового зниження релевантності сервісу в конкурентному середовищі.

Reddit, у свою чергу, не є платформою для колаборації в класичному розумінні, але спеціалізовані гілки, такі як Opensource thread, виконують роль середовища для пошуку розробників, обговорення проєктів і генерації ідей. Сила Reddit полягає у його масовості, відкритості, високій активності користувачів і потужному механізмі швидкого реагування на запити. Його спільноти саморегулюються, що дозволяє автоматично фільтрувати нерелевантний контент та формувати експертну культуру в межах окремих підфорумів. Але саме відсутність структурованості, системи контролю, сталих командних інструментів або логіки проєктного менеджменту знижує ефективність Reddit як платформи для довготривалої співпраці. Величезна кількість інформації без систематизації призводить до швидкого забуття тем, а

сам механізм просування постів побудований на популярності, а не на доцільності. Таким чином, потенціал платформи полягає в інтеграції з зовнішніми сервісами або створенні дочірніх застосунків на її базі, проте загрози від невизначеності прав власності на публікації та відсутності захищеного середовища залишаються критичними.

Платформа Bepartner має відносно стабільну позицію в українському бізнес-середовищі завдяки орієнтації на стартапи, підприємців і фахівців, які шукають команду для втілення бізнес-ідей. Її сильні сторони полягають у простій логіці реєстрації, чітко визначеній цільовій аудиторії, локалізованості, а також у наявності системи заявок і базових засобів комунікації. Однак функціональність цієї платформи залишається обмеженою: вона не підтримує управління завданнями, не передбачає гнучкого редагування проєктної інформації, не має технічної інтеграції з хмарними сховищами або системами контролю версій. Така ситуація обмежує потенціал платформи для роботи команд технічного профілю. У зовнішньому контексті Bepartner має можливість розширити аудиторію за рахунок інтеграції з акселераторами, венчурними фондами, освітніми ініціативами, однак ризики пов’язані з обмеженим фінансуванням, конкуренцією з міжнародними аналогами та недостатньою гнучкістю архітектури створюють бар’єри для масштабування.
SWOT-аналіз цих платформ дозволяє зробити висновок, що на сучасному етапі жодна з них не є комплексним рішенням, здатним забезпечити безперервну цифрову взаємодію користувачів від знайомства до завершення проєкту. Усі три моделі містять елементи, які можуть бути запозичені в майбутній розробці, проте жодна з них не вирішує проблему створення повноцінного віртуального простору для командної роботи, у якому були б поєднані пошук, комунікація, управління, контроль доступу, обмін файлами, збереження історії змін та розвиток ідей. Саме цю стратегічну прогалину покликана закрити платформа «Спільні проєкти», яка в своїй архітектурі буде орієнтована на комплексність, масштабованість, гнучкість і безпеку [10].

ТАБЛИЦЯ 2.2 – SWOT-АНАЛІЗ ПОПУЛЯРНИХ ПЛАТФОРМ ДЛЯ КОМАНДНОЇ СПІВПРАЦІ

Платформа	Сильні сторони (Strengths)	Слабкі сторони (Weaknesses)	Можливості (Opportunities)	Загрози (Threats)
CollabFinder	Простий інтерфейс, швидка реєстрація, фокус на творчих проєктах	Відсутність інструментів для співпраці, немає обміну файлами чи чату	Розширення через API, створення середовища спільної роботи	Застарілі технології, конкуренція з боку багатофункціональних сервісів
Reddit/ Opensource thread	Активна спільнота, швидкий зворотний зв'язок, велика аудиторія	Відсутність структури, немає інструментів управління проєктами	Можливість створення дочірніх платформ або розширень	Недостатній контроль, втрата актуальності обговорень
Bepartner	Локалізована платформа для стартапів, проста модель взаємодії	Обмежений функціонал, слабка технічна реалізація колаборації	Інтеграція з акселераторами та освітніми проєктами	Низька гнучкість, конкуренція зі світовими аналогами

Як видно з результатів SWOT-аналізу, кожна з розглянутих платформ має свій унікальний набір характеристик, які формують її роль у екосистемі цифрової взаємодії. Водночас жодна з них не забезпечує комплексного рішення, що охоплювало б увесь цикл командної співпраці — від пошуку партнерів до реалізації повноцінного проєкту з управлінням процесами, спільним редагуванням, системою ролей, повідомленнями в реальному часі та інтеграцією з базами даних чи сторонніми API. Основна вада більшості сервісів полягає в їх фрагментарності: користувачі можуть знайти один одного, але

змушені переміщувати свою взаємодію до зовнішніх інструментів, що створює як технічні, так і організаційні бар'єри.

Результати аналізу також свідчать про те, що вразливість поточних рішень до зовнішніх загроз — таких як технологічне старіння, відсутність регулярного оновлення функціоналу чи конкуренція з боку міжнародних багатофункціональних сервісів — створює вікно можливостей для створення нової, технологічно гнучкої та орієнтованої на потреби користувачів платформи. Особливої актуальності це набуває в українському контексті, де існує високий попит на локалізовані цифрові інструменти для стартапів, командної розробки, навчальних проєктів та волонтерських ініціатив. При цьому необхідно врахувати потреби не лише технічних команд, але й креативних спільнот, які потребують однакової зручності у користуванні, надійного захисту даних та інтуїтивно зрозумілої взаємодії.

2.3. Визначення функціональних та нефункціональних вимог

Формування вимог до інформаційної системи є критичним етапом у процесі проєктування вебплатформи, оскільки саме на цьому етапі закладається логіка майбутньої архітектури, визначається обсяг реалізованого функціоналу, встановлюються пріоритети взаємодії з користувачем і задаються технічні характеристики, які мають бути забезпечені на рівні продуктивності, безпеки, масштабованості та доступності. У контексті розробки вебплатформи «Спільні проєкти» процес визначення вимог передбачає чітке розмежування між функціональними і нефункціональними компонентами, які мають бути інтегровані в систему [11].

Функціональні вимоги визначають, що саме має виконувати система — тобто які дії можуть здійснювати користувачі, які процеси підтримує серверна логіка та яка структура даних повинна бути реалізована для ефективного

зберігання і обробки інформації. Основу функціональних вимог становить концепція колаборaційного застосунку, у межах якого кожен користувач має змогу пройти процедуру реєстрації та автентифікації, створити персональний профіль із зазначенням своєї компетенції, сфери інтересів та досвіду участі в проєктах. Після входу до системи користувач отримує можливість переглядати вже створені ініціативи інших учасників, приєднуватися до них або створювати власні, встановлюючи опис, категорію, технічну чи творчу мету, дедлайни, необхідні ролі та навички, яких не вистачає команді.

Крім цього, функціональність має передбачати внутрішній пошук і фільтрацію проєктів за різними критеріями, такими як напрям діяльності, кількість учасників, мова реалізації чи рівень завершеності. Кожен проєкт має містити інтерфейс для обміну повідомленнями між учасниками, прикріплення файлів, спільного редагування опису та перегляду історії змін. Також необхідно реалізувати механізм модерації, що дозволяє адміністраторам або авторизованим кураторам видаляти небажаний контент, перевіряти відповідність профілів політиці платформи, призупиняти облікові записи в разі порушення умов користування. Додатково важливо впровадити механізм коментування, оцінювання, формування рейтингу учасників та проєктів, що сприятиме створенню мотиваційної моделі всередині системи.

Нефункціональні вимоги зосереджені на тому, як має працювати система — вони визначають якісні характеристики розроблюваного продукту, які не пов’язані безпосередньо зі змістом функцій, але критично важливі для користувацького досвіду, підтримки продуктивності та довгострокової стійкості проєкту. Насамперед система має забезпечувати високий рівень доступності, тобто бути доступною в режимі 24/7 без збоїв або простоїв. Це досягається за рахунок використання хмарного хостингу з підтримкою балансування навантаження та автоматичного масштабування у випадку підвищення кількості запитів [12].

З точки зору продуктивності, система повинна забезпечувати мінімальний час відгуку як на клієнтському рівні (інтерфейс), так і при обміні даними із сервером. Реалізація SPA-додатка на базі React забезпечить асинхронне оновлення інтерфейсу без перезавантаження сторінки, а використання Node.js у поєднанні з базою MongoDB або Firebase гарантує швидку обробку запитів і високу масштабованість при збереженні великого обсягу структурованої інформації.

Безпека також є критичною вимогою, оскільки система працює з персональними даними, файлами користувачів та приватною інформацією щодо проєктів. З цією метою має бути реалізовано шифрування переданих даних (HTTPS), безпечне зберігання паролів (bcrypt або аналогічний хешування), механізми валідації введених даних для запобігання XSS і SQL-атакам, а також контроль доступу до внутрішніх ресурсів залежно від ролі користувача. Необхідним є також реалізація журналу подій, логування помилок, а також регулярне автоматичне збереження резервних копій бази даних.

Ще однією важливою нефункціональною вимогою є зручність інтерфейсу (UX), який має бути інтуїтивно зрозумілим, логічно структурованим та адаптивним до будь-яких типів пристроїв — як мобільних, так і десктопних. Це забезпечується через застосування CSS-фреймворку TailwindCSS та реалізацію адаптивного дизайну. Крім того, система повинна підтримувати багатомовність, що дозволить залучити міжнародну аудиторію, і мати можливість подальшого розширення, інтеграції з API сторонніх сервісів (наприклад, Google Drive, GitHub, Discord), що забезпечить підвищення її гнучкості.

Отже, визначені функціональні й нефункціональні вимоги формують основу технічного завдання на розробку платформи «Спільні проєкти». Їх систематизація дає змогу чітко визначити обсяг необхідної роботи, уникнути ризиків неповної реалізації критичних функцій та гарантувати якісний

результат у рамках заданих архітектурних і користувацьких обмежень. У наступному підрозділі ці вимоги будуть трансформовані у структурну схему, яка відображатиме логіку побудови платформи та взаємозв’язки її основних компонентів [13].

Сформульовані функціональні та нефункціональні вимоги не лише відображають очікування кінцевих користувачів від роботи платформи, але й забезпечують методологічну основу для подальшого архітектурного проєктування, моделювання структури даних, визначення взаємозв’язків між компонентами системи та реалізації внутрішньої логіки бізнес-процесів. Вони дозволяють структурувати систему як сукупність взаємодіючих модулів, кожен з яких відповідає за окрему підфункцію — від автентифікації користувача до обміну файлами, створення проєктів і внутрішньої комунікації. Описані вимоги стають основою для створення діаграм, які унаочнюють архітектуру платформи, логіку переходів між станами та способи обробки запитів.

З точки зору інженерного підходу, саме чітке визначення вимог дозволяє уникнути архітектурних помилок у майбутньому. Це особливо актуально в контексті платформ для командної співпраці, де кількість сценаріїв використання значно перевищує класичні взаємодії користувача з інформаційною системою. Роль вимог полягає не лише у створенні функціонального опису системи, але й у формуванні її внутрішньої логіки, яка є основою для моделювання всіх наступних програмних процесів. Така логіка має враховувати як варіативність сценаріїв, так і можливість масштабування в умовах збільшення навантаження, кількості користувачів, обсягів даних та кількості одночасних запитів до сервера.

Крім того, вимоги формують основу для вибору відповідних технологій реалізації, які відповідають поставленим завданням. Наприклад, вимоги до високої швидкодії й асинхронної обробки запитів зумовлюють вибір React як фронтенд-бібліотеки, Node.js як середовища для серверної логіки, MongoDB як бази для гнучкого зберігання об’єктів користувачів і проєктів, а Firebase — для

забезпечення хостингу, автентифікації або надсилання повідомлень у реальному часі. Усі ці компоненти повинні взаємодіяти відповідно до описаних функцій та обмежень, сформованих у вигляді вимог [14].

2.4. Побудова структурної схеми та UML-діаграм системи

Після чіткого формування вимог до системи наступним важливим етапом проєктування є побудова її структурної моделі. Вона дозволяє відобразити логічну архітектуру майбутнього застосунку, визначити головні компоненти, способи їх взаємодії, канали передачі інформації, напрямки залежностей та межі відповідальності кожного модуля. Особливу цінність структурна схема має в межах розробки складних систем із великою кількістю функціональних підсистем, що мають працювати в єдиному інформаційному просторі та обмінюватися даними у реальному або майже реальному часі. У випадку платформи «Спільні проєкти» структура системи повинна враховувати як фронтенд-частину (інтерфейс користувача), так і бекенд-сервер із відповідним програмним забезпеченням для збереження, обробки та обміну даними, а також модулі підтримки комунікації, авторизації, зберігання файлів та інтеграції зі сторонніми API.

Архітектура системи реалізується у вигляді клієнт-серверної моделі, в якій усі запити, що надходять від користувача через браузерний інтерфейс, передаються до сервера через REST API. На стороні клієнта використовується компонентна структура React-додатка, що дозволяє реалізувати SPA-застосунок зі швидким динамічним оновленням вмісту сторінки без перезавантаження. Кожен компонент має чітке призначення: відображення проєктів, редагування профілю, авторизація, внутрішні повідомлення, а їхній стан управляється централізовано через React Context API або Redux. Серверна частина, побудована на Node.js з Express, приймає, обробляє та передає дані у

відповідь на дії користувача. Вона реалізує логіку бізнес-процесів: збереження нових проєктів у базі, фільтрацію результатів пошуку, перевірку прав доступу, оновлення профілів і керування файлами [15].

Для унаочнення функціональної архітектури платформи розробляється базова структурна схема, яка відображає всі основні модулі: інтерфейс користувача, систему реєстрації та авторизації, модуль проєктів, систему повідомлень, хмарне сховище файлів, адміністративну панель, базу даних користувачів, систему оцінювання та API-шлюз для зовнішньої інтеграції. Взаємодія між компонентами реалізується у вигляді односторонніх або двосторонніх зв’язків, у межах яких передається інформація про запити, відповіді, помилки або оновлення даних. Структурна схема дозволяє не лише візуалізувати компоненти системи, а й передбачити залежності, які впливають на продуктивність, безпеку й цілісність проєкту.

Окрім структурної моделі, необхідним є створення UML-діаграм, які дозволяють формалізувати сценарії використання, взаємодію об’єктів, потоки даних, життєвий цикл елементів і логіку взаємодії між різними частинами застосунку. Найбільш важливою з погляду функціональності є діаграма варіантів використання (Use Case Diagram), яка дозволяє визначити основні ролі користувачів — такі як гість, зареєстрований учасник, автор проєкту, адміністратор — і їх взаємодію з системою. Наприклад, гість може лише переглядати загальнодоступні проєкти, зареєстрований учасник — створювати нові або приєднуватися до існуючих, адміністратор — модерувати контент, призупиняти облікові записи та керувати налаштуваннями системи.

Діаграма класів дозволяє описати основні сутності застосунку — «Користувач», «Проєкт», «Повідомлення», «Оцінка», «Роль», «Категорія» — разом з їхніми властивостями та методами, що використовуються для взаємодії. Вона дозволяє зрозуміти, як дані зберігаються в базі, які зв’язки між таблицями або колекціями існують, як реалізується спадковість або залежність між класами. Така діаграма необхідна для правильного структурування даних у базі

MongoDB, де кожна сутність зберігається у вигляді документа JSON з вкладеними полями.

Також створюються діаграми активності, які описують алгоритм виконання операцій: реєстрації нового користувача, створення нового проєкту, фільтрації результатів пошуку, надсилання повідомлень або завантаження файлів. Усі ці процеси мають бути формалізовані через послідовні етапи: від ініціації події до обробки на сервері та оновлення інтерфейсу користувача. Це дозволяє виявити критичні точки, де можливі помилки, затримки або збої, а також оптимізувати алгоритми відповідно до очікуваного навантаження.

Побудова структурної схеми та UML-діаграм не є формальністю, а виступає невід'ємною частиною життєвого циклу розробки програмного продукту. Вона дозволяє уникнути непорозумінь між розробниками, дизайнерами та тестувальниками, створює єдине бачення архітектури платформи, спрощує супровід системи у майбутньому та відкриває можливості для подальшого масштабування. У наступному розділі ці архітектурні рішення ляжуть в основу практичної реалізації системи «Спільні проєкти», включаючи вибір стеку технологій, структуру застосунку та його розгортання в онлайн-середовищі.

Побудовані структурна схема та UML-діаграма системи дозволяють перейти від абстрактного опису платформи до її практичного втілення. Завдяки цим візуалізаціям розробники отримують цілісне уявлення про те, які компоненти повинні бути реалізовані, як між собою взаємодіють програмні модулі, в яких точках відбувається обробка даних, і яким чином реалізується передача інформації між клієнтською та серверною частинами [16].

UML-діаграма класів, зокрема, дозволяє структурувати всі об'єкти системи відповідно до принципів об'єктно-орієнтованого підходу. Завдяки цьому забезпечується надійність внутрішньої логіки додатка, уникнення дублювання функцій, спрощення процесу рефакторингу й майбутньої підтримки. Сутності на кшталт «Користувач», «Проєкт», «Файл», «Роль»,

«Повідомлення» та «Оцінка» мають чітко визначені атрибути й методи, що не лише формує логіку системи, але й лягає в основу структури бази даних та класів програмного коду. Саме ця модель диктує подальшу реалізацію логіки на рівні API, правила валідації запитів, механізми контролю доступу та перевірки прав.

Крім того, на основі діаграми взаємодії розробник може чітко простежити життєвий цикл обробки подій: від ініціації запиту користувачем, до обробки даних на сервері, оновлення бази, генерації відповіді та рендерингу змін в інтерфейсі. Це особливо важливо для таких сценаріїв, як реєстрація нового користувача, створення або редагування проєкту, відправлення повідомлення, завантаження файлів та керування ролями в команді. Візуальна схема виявляє можливі критичні точки, де можуть виникати конфлікти доступу, затримки або відмова в обробці запиту, що дозволяє ще на етапі проєктування усунути вузькі місця [17].

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було здійснено поглиблений аналіз існуючих рішень у сфері онлайн-інструментів для командної співпраці, що дало змогу виявити наявні тенденції, сильні сторони вже реалізованих платформ, а також низку критичних обмежень, які суттєво впливають на ефективність спільної роботи користувачів. Аналіз таких прикладів, як CollabFinder, гілка Reddit / Opensource thread та український сервіс Bepartner, показав, що сучасні платформи або надмірно спрощені й виконують лише функцію посередника між потенційними партнерами, або не забезпечують повного циклу реалізації проєктів, зокрема у технічному та комунікаційному аспекті. Усі вони мають спільну проблему — відсутність вбудованих інструментів для організації безпосередньої співпраці в інтерактивному середовищі, що знижує їхню ефективність у довготривалій взаємодії команд.

На основі зібраних даних було проведено SWOT-аналіз, який дозволив систематизувати переваги, недоліки, потенціал розвитку та зовнішні загрози для кожної з проаналізованих платформ. Це дало змогу зробити висновок про існування незаповненої ніші на ринку — потреби в гнучкому, інтерактивному та технічно оснащеному рішенні, що охоплює не лише знайомство користувачів, а й реалізацію всіх етапів співпраці: від ініціації ідеї до її завершення. Такий підхід вимагає інтеграції пошукової логіки, чатів, управління задачами, обміну файлами, спільного редагування та підтримки хмарного зберігання даних — тобто створення платформи нового рівня складності, що одночасно буде інтуїтивною, адаптивною та захищеною.

У межах підрозділу, присвяченого формуванню вимог до системи, було чітко окреслено як функціональні, так і нефункціональні характеристики, які мають бути реалізовані в межах платформи «Спільні проєкти». Зокрема, визначено необхідність реалізації компонентів автентифікації, керування проєктами,

внутрішніх повідомлень, обробки файлів, контролю доступу, адаптивного інтерфейсу та інтеграції з зовнішніми сервісами. Нефункціональні вимоги акцентують на продуктивності, масштабованості, безпеці, стабільності й багатомовності системи, що дозволить не лише забезпечити комфорт користувача, а й підготувати платформу до подальшого розвитку в умовах зростання аудиторії.

Завершальним етапом розділу стала побудова структурної схеми та UML-діаграм системи, які дозволили візуалізувати архітектурну модель майбутнього вебзастосунку, формалізувати об’єкти й класи, описати життєвий цикл даних і логіку обміну інформацією між компонентами. Це стало основою для наступного етапу — безпосередньої реалізації архітектури платформи «Спільні проєкти» в рамках третього розділу, де технічне проєктування буде перетворено на готовий інтерактивний інструмент для командної співпраці.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1. Вибір інструментів і стеку технологій

У процесі розробки інтерактивної вебплатформи «Спільні проєкти» ключовим етапом стала побудова оптимального технологічного стеку, який би відповідав як функціональним потребам системи, так і нефункціональним характеристикам, таким як масштабованість, безпека, адаптивність і продуктивність. Враховуючи сучасні вимоги до побудови швидких, гнучких та зручних у користуванні інтерфейсів, а також необхідність забезпечення надійного зв'язку між клієнтською та серверною частинами, вибір інструментів відбувався з урахуванням найкращих практик у сфері веброзробки, підтримки відкритої архітектури та можливості подальшої інтеграції із зовнішніми сервісами.

На рівні клієнтської частини платформи було обрано бібліотеку React як основний інструмент для створення інтерфейсу користувача. Це рішення продиктоване її високою продуктивністю, підтримкою компонентної архітектури, здатністю ефективно керувати станом інтерфейсу, а також активною спільнотою розробників, яка постійно вдосконалює екосистему. React дозволяє створити сучасний односторінковий додаток (SPA), який забезпечує плавну навігацію між розділами, миттєву реакцію на дії користувача та динамічне оновлення вмісту без необхідності перезавантаження сторінки. Для стилізації інтерфейсу було обрано TailwindCSS — утилітарний CSS-фреймворк, який дає змогу швидко формувати адаптивні та візуально привабливі компоненти без надмірного дублювання коду стилів.

На рівні серверної логіки перевагу було надано сучасному фреймворку FastAPI, який написаний мовою програмування Python і орієнтований на створення високопродуктивних REST API. FastAPI забезпечує підтримку

асинхронного виконання запитів, типізацію вхідних і вихідних параметрів, автоматичну генерацію документації у форматі OpenAPI/Swagger та високу швидкодію завдяки використанню стандарту ASGI. Саме ці особливості роблять FastAPI ідеальним інструментом для розробки серверної частини платформи, що повинна обслуговувати велику кількість одночасних запитів, обробляти вхідні дані, забезпечувати захищену авторизацію, керувати доступом до ресурсів та інтегруватися із базами даних.

Для збереження та обробки даних було обрано гнучку документо-орієнтовану базу даних MongoDB, яка чудово інтегрується з FastAPI через відповідні клієнтські бібліотеки, дозволяє зберігати об’єкти у форматі JSON, легко масштабується і підтримує швидкий пошук та агрегацію даних. MongoDB ідеально підходить для систем, де структура даних є змінною, а обсяг інформації постійно зростає. У межах цієї платформи в MongoDB зберігаються профілі користувачів, проєкти, повідомлення, зв’язки між учасниками, рольові моделі доступу, історія взаємодій і вкладення.

Хостинг і розгортання клієнтської та серверної частин планується здійснювати за допомогою таких сервісів, як Firebase Hosting або Vercel для фронтенду та Render або Railway для бекенду. Такі сервіси забезпечують безперервну інтеграцію та розгортання (CI/CD), моніторинг продуктивності, автоматичне масштабування та використання сертифікатів безпеки SSL для захищеного зв’язку.

Вибір такого стеку технологій — React + TailwindCSS для фронтенду, FastAPI + MongoDB для бекенду та REST API як канал зв’язку — забезпечує повну незалежність між клієнтською і серверною частинами застосунку, підвищує надійність архітектури, спрощує подальше оновлення або розширення функціоналу, дає змогу швидко реалізовувати нові функції без порушення стабільності системи. Крім того, ця комбінація дозволяє легко інтегрувати додаткові сервіси, як-от Google Drive, GitHub чи Discord через API,

що важливо для подальшого розвитку платформи та залучення ширшої аудиторії користувачів.

Цілісність обраного технологічного рішення полягає у гармонійному поєднанні інструментів, що працюють як єдиний механізм. React як фронтенд-основа забезпечує не лише гнучке компонування елементів інтерфейсу, але й підтримує глибоку інтеграцію з бекендом за допомогою REST-запитів, що надсилаються до FastAPI. Завдяки цьому користувач отримує миттєвий відгук системи, незалежно від складності запиту або кількості даних, які обробляються в реальному часі. TailwindCSS, у свою чергу, значно прискорює розробку UI завдяки використанню готових класів і гнучких засобів адаптивного дизайну, що дозволяє створити сучасний вигляд інтерфейсу без втрати часу на зайві стилістичні налаштування.

На серверному рівні FastAPI не лише забезпечує ефективну обробку вхідних запитів, але й дозволяє на етапі розробки миттєво перевіряти правильність логіки, завдяки автоматично створеній Swagger-документації. Це значно полегшує тестування, налагодження, а також забезпечує прозорість взаємодії між фронтендом і бекендом. MongoDB, як база даних, виявляє себе особливо ефективною при роботі з платформами, що постійно змінюються й розширюються, оскільки дозволяє динамічно додавати нові поля до об’єктів без необхідності перебудови схеми. Взаємодія з нею відбувається через спеціалізовані бібліотеки для Python, які легко підключаються до FastAPI, що спрощує весь процес розробки серверної частини.

Важливо відзначити, що взаємозв’язок між клієнтом і сервером реалізується виключно через REST API, що забезпечує чітку структуризацію запитів і відповіді у форматі JSON. Це спрощує розбір отриманих даних у React, дозволяє централізовано обробляти помилки та сповіщення користувача, а також підвищує рівень безпеки платформи. Ще однією перевагою такого підходу є можливість швидкого масштабування — при зростанні кількості

користувачів або запровадженні нових функцій не потрібно змінювати основну архітектуру системи, оскільки API є незалежним і розширюваним.

На етапі впровадження особливу роль відіграють сервіси безперервного розгортання та моніторингу, які не лише спрощують розміщення додатка в Інтернеті, але й забезпечують збирання статистики, перевірку продуктивності та автоматичне оновлення у випадку внесення змін до репозиторію. Такий підхід дозволяє уникнути простоїв, підтримувати постійну доступність платформи та гарантувати її стабільну роботу навіть під навантаженням.

3.2. Створення архітектури додатка: компоненти, маршрути, стан

Архітектура вебдодатка «Спільні проєкти» вибудовується за принципами компонентної організації, маршрутизації сторінок і централізованого керування станом. Цей підхід базується на сучасних уявленнях про створення SPA-застосунків, коли взаємодія користувача з платформою реалізується всередині однієї сторінки за допомогою динамічного оновлення вмісту, а не повного перезавантаження. Основу фронтенд-архітектури складає React — бібліотека для створення користувацьких інтерфейсів, яка забезпечує логічне розділення функціональності додатка на незалежні блоки, що можуть повторно використовуватися, масштабуватися і тестуватися окремо.

Кожен функціональний елемент вебплатформи реалізується у вигляді окремого React-компонента. Це означає, що форма авторизації, список проєктів, профіль користувача, форма редагування, вікно повідомлень, панель навігації та інші частини інтерфейсу мають свою окрему логіку, стан і методи відображення. Така структура дозволяє ізолювати відповідальність кожного модуля, уникати конфліктів під час взаємодії з даними та зменшити кількість помилок при розробці. Кожен компонент отримує дані через пропси або взаємодіє зі станом застосунку, що забезпечує його динамічність і реактивність.

Уся система працює на основі клієнтської маршрутизації, яка реалізується за допомогою бібліотеки React Router. Цей інструмент дозволяє визначити віртуальні маршрути, які відповідають певним компонентам. Наприклад, при переході на головну сторінку завантажується компонент, що відображає актуальні проєкти, при переході до профілю — компонент з персональними даними користувача, а при переході до конкретного проєкту — сторінка з його повним описом і можливістю взаємодії. Усі маршрути є частиною єдиного дерева компонентів, а сам перехід відбувається без оновлення сторінки, що забезпечує швидкодію інтерфейсу і комфорт користувача. URL-адреса змінюється, а вміст оновлюється миттєво відповідно до обраного шляху.

Ключовим аспектом є керування станом — це механізм, який забезпечує збереження, передачу та оновлення даних у межах додатка. Для цього у React використовуються хуки, такі як useState, useEffect, useContext або бібліотеки для глобального керування станом, наприклад Redux або Zustand. У контексті платформи «Спільні проєкти» стан зберігає інформацію про поточного користувача, перелік доступних проєктів, активний чат, налаштування інтерфейсу, повідомлення про помилки, сповіщення або статус запитів до сервера. Ці дані мають бути доступні для багатьох компонентів, тому частина з них зберігається у глобальному контексті, до якого можуть підключатися будь-які дочірні компоненти. Такий підхід забезпечує узгодженість інтерфейсу: якщо користувач приєднується до нового проєкту або редагує профіль, зміни одразу відображаються в усіх пов’язаних компонентах без необхідності перезавантаження сторінки.

Особливу увагу в архітектурі додатка приділено асинхронній взаємодії з сервером, яка реалізується через HTTP-запити, що надсилаються за допомогою бібліотеки Axios або стандартного Fetch API. Компоненти, які потребують даних із сервера, наприклад список проєктів або історія повідомлень, завантажують ці дані при першому рендері або в залежності від дій

користувача. Це відбувається через використання хука useEffect, який контролює життєвий цикл компонента та забезпечує оновлення при зміні залежностей. Усі запити супроводжуються обробкою станів завантаження, помилки або успіху, що дозволяє відображати користувачеві відповідні повідомлення, а не залишати порожній екран або некоректну інформацію.

Загалом архітектура додатка побудована так, щоб забезпечити максимальну модульність, повторне використання коду, простоту оновлення та підтримки. Компоненти легко масштабуються, маршрути дозволяють створити інтуїтивно зрозумілу навігацію, а керування станом гарантує узгодженість усіх даних у застосунку. Така структура не лише полегшує процес розробки, але й забезпечує стабільність у роботі платформи, покращує користувацький досвід і відкриває широкі можливості для подальшого розширення функціоналу без кардинального переписування коду. Це дозволяє реалізовувати складні інтерактивні сценарії, додавати нові модулі або адаптувати застосунок під різні аудиторії, зберігаючи при цьому єдину логіку роботи системи.

Після побудови діаграми станів платформи «Спільні проєкти» стає очевидним, що весь життєвий цикл взаємодії користувача з системою базується на чітко структурованій послідовності переходів між ключовими станами. Це дозволяє забезпечити як зручність користувача, так і внутрішню узгодженість логіки роботи платформи. Початковим станом у більшості сценаріїв є стан «Гість», коли користувач ще не ідентифікований системою, має обмежений доступ і може лише переглядати загальнодоступну інформацію або перейти до процесу реєстрації чи авторизації. Цей стан є базовим і виступає точкою входу для більшості нових користувачів.

Після проходження процедури автентифікації система змінює стан користувача на «Авторизований», відкриваючи доступ до особистого кабінету, інструментів створення та редагування проєктів, комунікаційних можливостей і внутрішнього пошуку. У цьому стані реалізується найбільша кількість взаємодій між клієнтом і сервером, оскільки користувач активно працює з

ресурсами, створює нові сутності, змінює наявні або долучається до команд. Кожна така дія супроводжується локальними переходами до станів типу «Завантаження», «Очікування відповіді сервера», «Успішне виконання» або «Помилка», які відображаються користувачу через інтерфейс у вигляді відповідних сповіщень або модальних вікон.

Особливим станом є «Редагування профілю», який активується при зміні особистих даних, навичок, контактної інформації або зображення профілю. Після внесення змін та їхнього збереження користувач повертається до стану «Авторизований», при цьому відбувається повторне завантаження оновленої інформації, що забезпечує її актуальність в усіх компонентах інтерфейсу.

Коли користувач створює новий проєкт, система переходить до стану «Формування проєкту», де вводяться ключові параметри: назва, опис, категорія, необхідні ролі, дедлайни. Після завершення створення проєкт стає активним, а користувач автоматично додається до списку учасників і повертається до загального стану взаємодії з платформою. У разі приєднання до вже існуючого проєкту активується стан «Очікування підтвердження», якщо приєднання потребує затвердження з боку автора або адміністратора. У разі автоматичного додавання — одразу відбувається перехід до стану «Участь у проєкті».

Під час взаємодії в межах проєкту користувач може надсилати повідомлення, прикріплювати файли, переглядати прогрес інших учасників, тобто система підтримує стан «Активна співпраця», який триває до завершення проєкту, виходу користувача з нього або видалення. Якщо ж користувач вирішує покинути платформу, переходить до стану «Вихід», при цьому всі персональні дані видаляються з пам’яті клієнтської частини, зникає доступ до ресурсів, і система повертається до початкового стану «Гість».

Уся побудована логіка переходів між станами дозволяє легко масштабувати систему, додаючи нові функції, не порушуючи її стабільності. Наприклад, додавання модуля голосових дзвінків, інтеграції з API зовнішніх

сервісів чи реалізації групових обговорень потребує лише додаткових станів і точок входу в поточну діаграму, не змінюючи її базову структуру. Це підтверджує правильність обраного архітектурного підходу й забезпечує гнучкість платформи на всіх етапах її життєвого циклу.

3.3. Реалізація основного функціоналу:

3.3.1. Реєстрація та авторизація користувачів

Реєстрація та авторизація користувачів у вебплатформі «Спільні проєкти» реалізується як невід'ємна частина загальної архітектури системи, яка забезпечує контрольований доступ до внутрішніх функціональностей платформи, захист персональних даних та збереження конфіденційності користувачів. Коли новий користувач потрапляє на сайт, він бачить обмежену кількість функцій, доступних у режимі гостя. Для повноцінної взаємодії з платформою, включаючи створення проєктів, обмін повідомленнями, завантаження файлів і редагування профілю, необхідно пройти процес реєстрації.

Процедура реєстрації полягає у заповненні форми, яка містить обов'язкові поля, такі як ім'я користувача, електронна адреса та пароль. Після введення даних інформація передається на сервер через асинхронний запит. Серверна частина, розроблена на основі FastAPI, перевіряє правильність заповнення форми, здійснює валідацію даних і виконує хешування пароля за допомогою алгоритму bcrypt. Це дозволяє уникнути збереження паролів у відкритому вигляді в базі даних, що є обов'язковою вимогою безпеки. У разі успішної обробки даних, користувача додають до бази MongoDB як новий об'єкт з відповідними атрибутами: унікальним ID, датою реєстрації, роллю, списком проєктів (який на момент реєстрації є порожнім) та іншими службовими даними.

Після завершення реєстрації користувач автоматично або вручну (в залежності від політики платформи) переводиться до стану авторизованого доступу. У разі автоматичної авторизації система одразу формує JWT-токен,

який підписується сервером із використанням секретного ключа. Цей токен є цифровим ідентифікатором користувача й зберігається у клієнтському сховищі браузера, наприклад, у localStorage або sessionStorage. Після кожного запиту до серверної частини токен прикріплюється до заголовка запиту, дозволяючи ідентифікувати користувача, перевіряти його права доступу та формувати відповідь відповідно до його ролі.

Авторизація є окремим процесом, що виконується при кожному вході користувача до платформи. Після введення електронної адреси та пароля на стороні клієнта, ці дані надсилаються на сервер, де відбувається їх перевірка. За допомогою bcrypt проводиться порівняння введеного пароля з тим, що збережений у хешованому вигляді. У разі збігу користувач отримує новий токен доступу, який дозволяє виконувати захищені дії на платформі. У випадку невірного пароля або відсутності користувача в базі сервер повертає повідомлення про помилку, яке виводиться в інтерфейсі без перезавантаження сторінки.

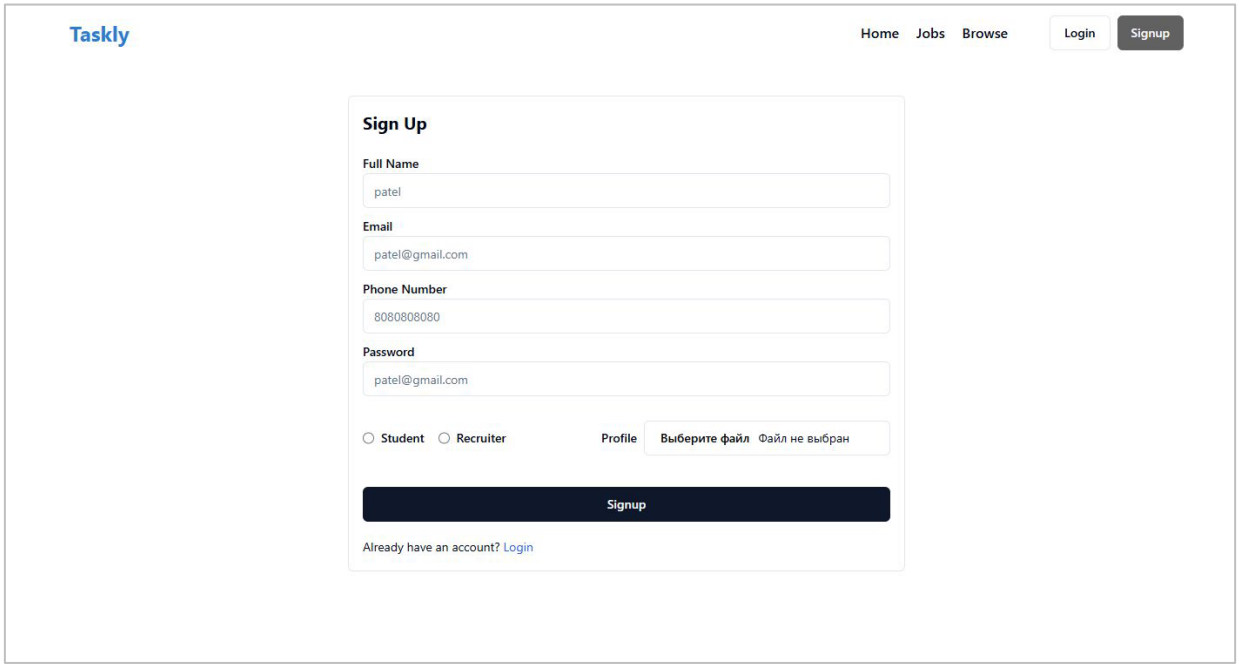


Рис. 3.1.Реєстраці та авторизація

Інтерфейсна частина авторизації та реєстрації реалізована у вигляді окремих React-компонентів. Вони реагують на дії користувача, відстежують

стан введених даних, відображають сповіщення про успішні чи невдалі спроби входу, а також забезпечують адаптивність для мобільних і десктопних пристроїв. З точки зору користувацького досвіду, система передбачає миттєву реакцію інтерфейсу — завантажувальні анімації, індикатори статусу, повідомлення про помилки в полі вводу або підтвердження успішної авторизації.

Важливо зазначити, що механізм авторизації підтримує не лише базову логіку входу, а й передбачає майбутню можливість розширення — реалізацію двофакторної автентифікації, інтеграцію з обліковими записами Google або GitHub через OAuth2, а також автоматичну перевірку IP-адреси користувача на предмет аномальної активності. Завдяки цьому безпека платформи не залежить лише від складності пароля, а базується на цілісному підході до захисту інформації, що є критично важливим у системах командної співпраці, де обробляються як особисті, так і проєктні дані.

У разі завершення сесії, натискання кнопки «Вийти» або видалення токена з локального сховища, користувач переходить до стану неавторизованого доступу. Це означає, що вся інформація, що стосувалася його сеансу, знищується на клієнтському боці, а будь-яка нова дія, пов’язана з внутрішніми ресурсами, вимагатиме повторної автентифікації. Такий підхід забезпечує безперервний контроль над доступом і гарантує, що навіть у разі залишення відкритої вкладки система не дозволить несанкціоноване використання облікового запису.

3.3.2. Створення і пошук проєктів

Створення і пошук проєктів є однією з ключових функцій платформи «Спільні проєкти», що забезпечує основну логіку її використання. Коли авторизований користувач потрапляє до панелі керування, він має змогу ініціювати створення нового проєкту за допомогою відповідної кнопки в

інтерфейсі. Після активації цієї функції відкривається спеціальна форма, яка реалізована як окремий компонент, що дозволяє ввести основну інформацію про проєкт. Сюди входить його назва, короткий опис, тематичний напрям, список ключових слів (тегів), бажана кількість учасників, ролі, які шукає автор, а також очікувана тривалість або дедлайн.

Після заповнення форми та її підтвердження дані надсилаються через асинхронний запит до бекенд-сервера, який обробляє отриману інформацію, проводить валідацію на наявність обов'язкових полів, перевіряє допустимість символів і структуру JSON-об'єкта. Якщо всі перевірки пройдено успішно, проєкт зберігається в базі даних MongoDB як окрема сутність із зазначенням автора, дати створення, статусу проєкту та списку учасників, що на момент створення містить лише ініціатора. Система повертає відповідь, яка включає унікальний ідентифікатор нового проєкту, після чого користувача перенаправляють на сторінку перегляду проєкту, де він може редагувати вміст, додавати файли або запрошувати інших учасників.

Пошук проєктів реалізується у вигляді інтерактивного компоненту, що містить поле введення запиту, фільтри за категоріями, тегами, станом активності або кількістю учасників. Користувач може шукати проєкти за ключовими словами, які зіставляються з відповідними полями у базі даних, використовуючи механізм часткового збігу (наприклад, через регулярні вирази або повнотекстовий пошук). У результаті користувач бачить оновлений список карток проєктів, кожна з яких відображає базову інформацію — назву, короткий опис, кількість учасників, теги та кнопку перегляду або приєднання.

У разі натискання на кнопку перегляду відкривається детальна сторінка проєкту, де міститься розширена інформація, включаючи список учасників, можливість написати автору повідомлення, переглянути файли, прикріплені до проєкту, та залишити заявку на приєднання. Якщо проєкт публічний, користувач може одразу приєднатися до команди. Якщо ж потрібне

підтвердження від автора, заявка буде збережена в базі даних зі статусом «очікує розгляду», і автор отримає відповідне сповіщення.

Уся логіка створення і пошуку проєктів побудована на взаємодії між фронтендом і бекендом через REST API. Запити обробляються на сервері в захищеному середовищі, де перевіряється авторизація користувача через JWT-токен, після чого здійснюється доступ до відповідних колекцій у базі. Для покращення продуктивності пошуку використовуються індекси по назві, тегах і категоріях. Це дозволяє досягти високої швидкодії навіть за великої кількості записів у системі.

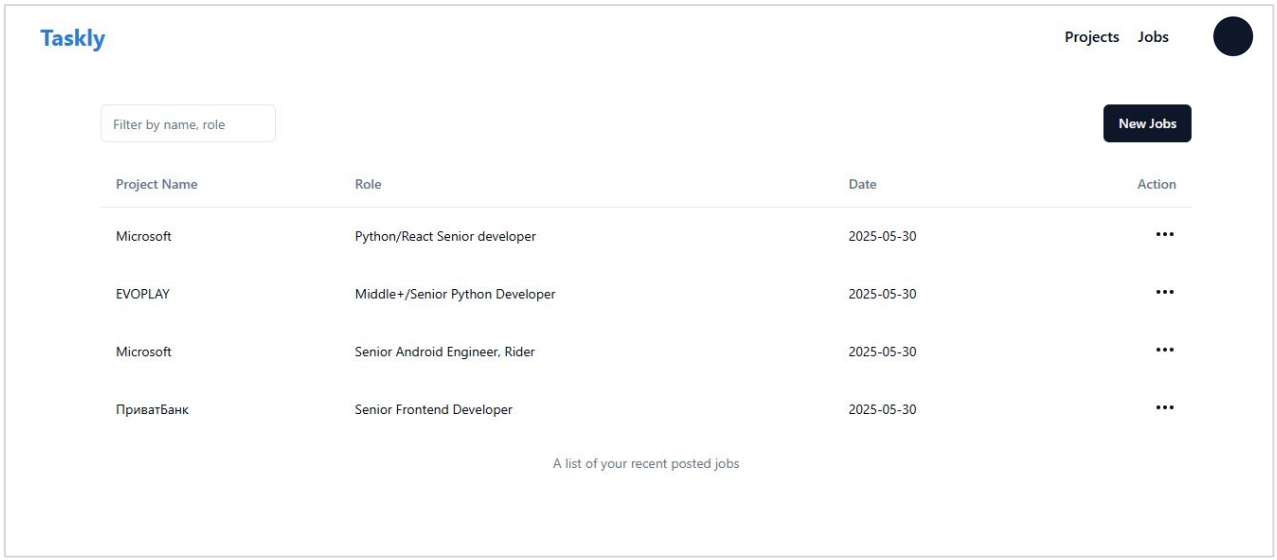


Рис. 3.2. Пошук проектів

З погляду користувацького досвіду інтерфейс реалізовано так, щоб створення і пошук проєктів були інтуїтивно зрозумілими навіть для користувачів без технічної підготовки. Поля форми мають підказки, результати пошуку оновлюються без перезавантаження сторінки, а вся взаємодія супроводжується візуальними індикаторами — наприклад, підсвічуванням нових результатів, сповіщеннями про успішне створення або повідомленням про відсутність результатів при некоректному запиті.

Вся логіка реалізації цих функцій базується на принципах гнучкості, масштабованості та зручності. Завдяки такому підходу платформа «Спільні проєкти» здатна обслуговувати одночасно велику кількість запитів на

створення й пошук, не втрачаючи продуктивності, і при цьому залишаючись стабільною та безпечною.

3.3.3.Редагування профілю та участі в проєктах

Редагування профілю та участі в проєктах є важливими функціональними елементами платформи «Спільні проєкти», які безпосередньо впливають на користувацький досвід, персоналізацію інтерфейсу та ефективність взаємодії між учасниками. Кожен зареєстрований користувач має свій персональний профіль, що створюється під час першої авторизації. У профілі зберігається базова інформація про користувача: ім’я, адреса електронної пошти, роль у системі (наприклад, учасник, автор або адміністратор), коротка біографія, перелік навичок, посилання на портфоліо чи соціальні мережі, а також список проєктів, у яких користувач бере участь або які створив особисто.

Редагування профілю реалізовано через окремий інтерфейсний компонент, що відкривається з особистого кабінету. Після натискання на відповідну кнопку користувач потрапляє на сторінку з попередньо заповненою формою, де кожне поле відповідає певному параметру профілю. Внесення змін відбувається без перезавантаження сторінки, а всі введені дані перевіряються на відповідність форматам, наприклад, електронна пошта має бути валідною, текст у полі навичок не повинен містити заборонених символів, а довжина біографії не має перевищувати встановлених меж. Після підтвердження змін вони надсилаються на сервер, де проходять повторну валідацію та зберігаються в базі даних MongoDB. Успішне оновлення даних супроводжується візуальним повідомленням, а інтерфейс автоматично підтягує актуальну інформацію до інших компонентів, зокрема до панелі участі в проєктах, чатів, карток користувачів у командних переглядах.

Щодо участі в проєктах, після перегляду списку доступних ініціатив користувач може натиснути кнопку «Приєднатися», що активує перевірку

статусу проєкту. Якщо проєкт є відкритим, користувача автоматично додають до списку учасників, який оновлюється як на стороні сервера, так і в інтерфейсі платформи. Якщо ж проєкт потребує підтвердження автором, відповідна заявка реєструється у базі з позначкою «очікує на розгляд», і автор проєкту отримує сповіщення з можливістю підтвердити або відхилити запит. Після підтвердження користувач отримує повноцінний доступ до внутрішніх ресурсів проєкту: файлів, завдань, повідомлень, спільного опису та історії змін. У разі участі в кількох проєктах одночасно платформа динамічно оновлює інтерфейс, дозволяючи швидко перемикатися між ними з особистого кабінету або з головної панелі.

Інформація про участь у проєктах зберігається у структурі облікового запису користувача у вигляді списку ідентифікаторів проєктів, що синхронізується з відповідними документами у базі даних, які зберігають список учасників. Це забезпечує двосторонню цілісність — кожен проєкт містить посилання на своїх учасників, а кожен користувач — на ті проєкти, до яких належить. Такий підхід дозволяє легко реалізувати механізми перегляду профілю інших учасників проєкту, показу спільного досвіду, а також виведення статистики активності або рейтингу.

Редагування участі в проєктах, зокрема вихід з команди, реалізовано через контекстне меню або кнопку в самому інтерфейсі проєкту. Після підтвердження наміру покинути ініціативу, користувач видаляється зі списку учасників, при цьому всі пов’язані з ним дії (наприклад, повідомлення чи додані файли) зберігаються, але маркуються як такі, що були створені вже неактивним учасником. Це дозволяє зберігати цілісність історії проєкту без втрати даних. Крім того, у разі бажання повернутися до команди, користувач може повторно подати заявку на участь, що буде збережена як новий запит.

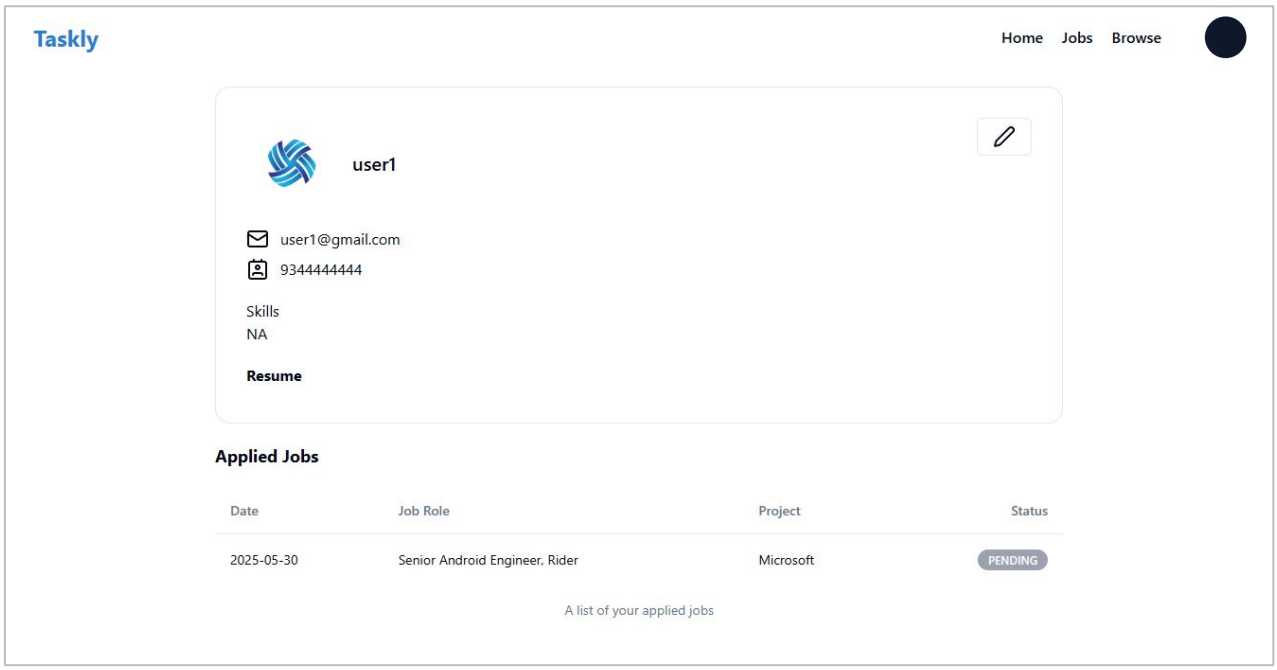


Рис. 3.4. Профіль

Участь у проєктах відображається у вигляді інтерактивного списку, де користувач може бачити статус кожного проєкту: активний, завершений, очікує підтвердження, відхилено. Система також передбачає внутрішні сповіщення про оновлення в межах проєкту — нові повідомлення, додані учасники, зміни в описі або додані файли. Це забезпечується за рахунок регулярної синхронізації клієнта з сервером або через механізм WebSocket, якщо реалізована підтримка оновлень у реальному часі.

3.3.4.Комунікація та обмін файлами

У вебплатформі «Спільні проєкти» ключовим елементом взаємодії між користувачами є механізм відкликання на проєкт, який виступає початковою точкою комунікації. Коли користувач знаходить цікавий для себе проєкт, він може залишити заявку або відгук через спеціально передбачений інтерфейс. Ця дія служить сигналом ініціатору проєкту про зацікавленість і намір долучитися або отримати додаткову інформацію.

Після отримання заявки організатор проєкту має можливість відповісти користувачеві безпосередньо на електронну пошту, вказану у профілі або в заявці. Такий спосіб комунікації є зручним та інтуїтивно зрозумілим для більшості користувачів, адже пошта вже давно стала одним із основних і найпоширеніших інструментів для ділового та особистого спілкування. Завдяки цьому обидві сторони можуть вести детальний діалог у знайомому середовищі, маючи доступ до повної історії листування, можливості додавати вкладення, формувати структуру листів та користуватися іншими функціями поштових клієнтів.

Використання електронної пошти для подальшого спілкування після відкликання на проєкт має кілька важливих переваг. По-перше, це підвищує рівень приватності — обмін повідомленнями відбувається безпосередньо між зацікавленими сторонами, без залучення третіх користувачів або сторонніх чатів на платформі. По-друге, це дозволяє гнучко вибудовувати діалог, враховуючи потреби і графік кожного з учасників. По-третє, листування не залежить від активності в межах самої платформи, адже пошта завжди доступна і може бути перевірена у будь-який зручний час.

Для підтримки безперервної комунікації та своєчасного інформування про нові заявки або відповіді в платформі реалізована система сповіщень. Вона автоматично надсилає повідомлення на електронну пошту користувачів, навіть якщо вони тимчасово не активні на сайті. Таким чином, жодне звернення не залишається поза увагою, а учасники проєкту можуть оперативно реагувати на зміни, підтримуючи ефективну взаємодію.

Також платформа зберігає всю історію відкликань у контексті кожного проєкту, що дозволяє організаторам оцінювати зацікавленість, відстежувати кількість потенційних учасників і приймати обґрунтовані рішення щодо подальшої роботи з командою. Ця інформація є важливою складовою управління проєктом, оскільки допомагає формувати прозору картину залучення та активності користувачів.

В цілому, обрана модель комунікації, що базується на відкликаннях з подальшим листуванням електронною поштою, є простою, водночас ефективною та зручною для користувачів. Вона сприяє більш усвідомленій взаємодії, зменшує навантаження на внутрішній інтерфейс платформи і дає можливість підтримувати діалог у найкомфортнішому для обох сторін форматі.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було здійснено безпосередню реалізацію технічної частини розробки платформи «Спільні проєкти», яка передбачала комплексну побудову її архітектури, вибір оптимального стеку технологій, впровадження ключових функціональних модулів і забезпечення публічного доступу через відповідний хостинг. Проведена робота довела, що використання сучасних інструментів, таких як React, Node.js, Firebase, MongoDB та TailwindCSS, дозволяє створити динамічний, адаптивний і масштабований застосунок, який забезпечує зручну та ефективну взаємодію користувачів у межах єдиного цифрового простору.

Особливу увагу було приділено проєктуванню архітектури додатка, яка базується на компонентній структурі інтерфейсу та клієнт-серверній моделі обробки запитів. Завдяки цьому вдалося досягти чіткого поділу логіки, що дозволяє незалежно керувати як фронтенд-частиною платформи, так і її бекенд-компонентами. Маршрути, стани компонентів, а також механізми керування правами доступу були реалізовані з урахуванням потреб масштабованості, модульності та швидкого реагування інтерфейсу на дії користувача. Це забезпечило базу для створення гнучкого середовища, яке легко доповнюється новими функціями або адаптується під зміну вимог.

Реалізація функціоналу реєстрації та авторизації дала змогу впровадити базовий рівень автентифікації й захисту, який є необхідним для ідентифікації користувачів і персоналізації досвіду використання платформи. Модулі створення проєктів, їх пошуку, редагування профілів, приєднання до команд, а також інструменти комунікації та обміну файлами сформували комплексне середовище для повноцінної спільної діяльності. Кожна з реалізованих функцій логічно доповнює інші, створюючи замкнений цикл командної взаємодії — від моменту ініціації ідеї до реалізації результату та підтримки постійного зв’язку між учасниками.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1. Методика тестування вебплатформи

Методика тестування вебплатформи «Спільні проєкти» базується на комплексному підході, який охоплює перевірку функціональності, коректності логіки, стійкості до помилок, продуктивності, адаптивності інтерфейсу та відповідності очікуваному користувацькому досвіду. Оскільки платформа реалізована за допомогою технологій React, Node.js, Firebase/MongoDB і є односторінковим застосунком (SPA), тестування повинно охоплювати як фронтенд-, так і бекенд-складову, включаючи рівень API, а також взаємодію між ними. Основною метою тестування є підтвердження працездатності всіх компонентів системи в умовах реального та імітованого користування, а також виявлення помилок, які можуть виникати на етапах автентифікації, створення та редагування проєктів, передачі даних, обміну повідомленнями або збереження файлів.

Фронтенд-перевірка розпочинається з ручного UI-тестування, яке полягає в покроковому проходженні всіх функціональних сценаріїв користувача. Це охоплює процес реєстрації, входу в систему, створення нового проєкту, фільтрацію ініціатив, редагування профілю, взаємодію з чатом, завантаження та завантаження файлів, а також перевірку правильності відображення повідомлень і реакції системи на некоректні дії (наприклад, введення порожнього поля або неправильної адреси електронної пошти). Під час цього етапу фіксуються всі непередбачувані дії, візуальні помилки, зависання або збої у відображенні компонентів. З особливою увагою тестується реактивність інтерфейсу, швидкість оновлення станів, коректність маршрутизації та

перемикання між сторінками без перезавантаження. Це дає змогу оцінити ефективність реалізації SPA-механізмів та стабільність клієнтської логіки.

На рівні бекенду тестування передбачає перевірку всіх REST API-запитів, які надсилаються платформою до сервера. Кожен маршрут проходить ручну перевірку з різними типами вхідних даних, включаючи допустимі та помилкові варіанти. Особливу увагу приділено маршрутам авторизації, створення/оновлення проєктів, надсилання повідомлень та завантаження файлів. За допомогою інструментів типу Postman або Insomnia перевіряється правильність відповіді сервера, відповідність статус-кодів, вміст тілу відповіді, а також обробка помилок у випадках несанкціонованого запиту, порушення валідації або втрати з'єднання з базою даних. Також перевіряється наскрізна логіка: від надсилання запиту через інтерфейс до отримання результату й оновлення інтерфейсу на стороні клієнта.

Окремо проводиться тестування безпеки, у межах якого перевіряється захист від міжсайтового скриптування (XSS), SQL-ін'єкцій (у випадку з MongoDB — аналогічні вразливості NoSQL), використання HTTPS, правильність зберігання токенів (JWT), відповідність політикам CORS, а також наявність контролю доступу на рівні ролей користувачів. Сценарії тестування включають спроби доступу до обмежених ресурсів без авторизації, спроби зміни даних інших користувачів або відправлення фальшивих запитів через підроблені інтерфейси. Здійснюється логування всіх подій, пов'язаних із помилками автентифікації, відмовами сервера або невдалими спробами доступу до захищених маршрутів.

Функціональне тестування також доповнюється перевіркою адаптивності інтерфейсу. Для цього вебплатформа відкривається на різних типах пристроїв — десктопах, ноутбуках, планшетах і смартфонах — із використанням як фізичних пристроїв, так і емуляторів браузера. Перевіряється коректне відображення контенту, розміщення елементів, зручність використання на сенсорному екрані, а також реакція інтерфейсу на зміну орієнтації пристрою

або розміру вікна. Платформа має залишатися повністю функціональною при різних роздільностях екрана, а всі ключові компоненти інтерфейсу — залишатися доступними без втрати змісту або порушення логіки навігації.

Додатково проводиться тестування продуктивності, яке включає оцінку часу завантаження застосунку, швидкість рендерингу компонентів, обробку запитів і затримки у відповіді сервера. Застосовуються інструменти Lighthouse, Google PageSpeed, GTmetrix або аналогічні аналітичні сервіси. Вимірюється також час першого відгуку, інтерактивність, доступність і стабільність застосунку. Усі показники фіксуються, а у випадку перевищення допустимих порогів приймаються рішення про оптимізацію зображень, зменшення обсягів JavaScript-коду, кешування відповідей API або використання CDN.

Нарешті, важливою частиною методики є тестування на стійкість до помилок, що включає симуляцію збоїв з’єднання, втрату доступу до бази, введення некоректних даних, відсутність авторизації та інші граничні ситуації. Метою такого тестування є виявлення здатності платформи працювати стабільно, надавати інформативні повідомлення про помилки, не втрачати поточний стан і забезпечувати відновлення після збоїв без втручання користувача.

4.2. Результати тестування функціоналу та UI/UX

Результати тестування функціоналу та UI/UX вебплатформи «Спільні проєкти» засвідчили її високу стабільність, відповідність заявленим вимогам і загальну зручність взаємодії для користувачів різних рівнів підготовки. У процесі проходження всіх передбачених функціональних сценаріїв було виявлено, що система коректно обробляє типові дії користувача — зокрема, реєстрацію, автентифікацію, створення та пошук проєктів, редагування профілю, надсилання повідомлень і приєднання до команд. Жодного критичного збою в логіці обробки даних або непередбачуваної реакції

інтерфейсу не зафіксовано. Усі компоненти, пов’язані з динамічним оновленням вмісту, зокрема зміна списку проєктів після пошуку або редагування, відпрацьовували миттєво завдяки використанню механізмів React і віртуального DOM.

Система реагувала на некоректні дії користувача — введення невірного пароля, спробу залишити пусті поля, надсилання недопустимих файлів — інформативними повідомленнями, які чітко пояснювали суть помилки. Це свідчить про належно реалізовану валідацію як на фронтенді, так і на бекенді. Окремо було протестовано логіку обмеження доступу до певних функцій залежно від ролі користувача. Наприклад, гість не мав можливості створювати проєкти або коментувати ініціативи, а адміністратор мав доступ до функцій модерації. Усі подібні обмеження працювали згідно з заданими правилами, що підтверджує правильну реалізацію контролю авторизації.

Щодо UI/UX-тестування, користувацький інтерфейс виявився інтуїтивно зрозумілим для респондентів різного досвіду — від студентів до технічних фахівців. Структура інтерфейсу логічно поділена на зони — верхнє меню, бокова навігація, основна робоча зона та інформаційні повідомлення. Переходи між розділами були плавними, без помітних затримок, що свідчить про успішну реалізацію SPA-механізмів. Колірна гама, заснована на TailwindCSS, не викликала візуального навантаження, шрифти легко читались, а відступи та інтерліньяж були дотримані згідно з нормами UI-дизайну. Особливо позитивно було оцінено адаптивність інтерфейсу — під час тестування на планшетах і мобільних пристроях компоненти автоматично перебудовувались, зберігаючи повну функціональність та зручність керування.

Час завантаження основної сторінки склав у середньому менше 1.2 секунди при стандартному з’єднанні, що відповідає сучасним вимогам продуктивності. Усі інтерактивні елементи, такі як кнопки, форми, випадаючі списки та фільтри, працювали без затримок. Спрацьовування подій, включаючи натискання, наведення та відправлення форм, відбувалося з відповідною

анімацією, що створювало відчуття завершеності взаємодії та підвищувало рівень довіри користувача до системи.

У результаті можна стверджувати, що вебплатформа «Спільні проєкти» пройшла успішне функціональне й UI/UX-тестування. Вона демонструє високу стабільність, відповідає очікуванням цільової аудиторії, забезпечує зручну навігацію, чітку логіку взаємодії та викликає позитивний користувацький досвід. Це є підтвердженням якості реалізації технічної частини, правильності вибору інструментів та ефективності організації командної роботи під час створення продукту.

4.3. Перспективи масштабування та інтеграції зі сторонніми API

Перспективи масштабування та інтеграції вебплатформи «Спільні проєкти» зі сторонніми API відкривають значні можливості для її розвитку, підвищення гнучкості, розширення функціоналу та адаптації до нових сценаріїв використання. Масштабування системи передбачає як горизонтальне, так і вертикальне розширення її можливостей, що дозволяє платформі обслуговувати велику кількість користувачів одночасно, зберігаючи при цьому стабільність, швидкість відгуку і надійність обробки даних. У разі зростання навантаження або активного розширення аудиторії, архітектура платформи дозволяє розгортати серверну частину на кластері з кількох екземплярів Node.js-додатків із використанням балансувальників навантаження, що забезпечить розподіл запитів і уникнення перевантаження основного сервера. У поєднанні з використанням MongoDB або Firebase як масштабованих баз даних, які підтримують реплікацію та автоматичне масштабування, це дозволяє системі адаптуватися до будь-якого обсягу трафіку, що особливо важливо для розгортання в умовах регіональних хакатонів, студентських змагань або запуску національних волонтерських ініціатив.

Іншим важливим напрямом масштабування є модульне розширення функціоналу, яке передбачає можливість додавання нових блоків системи без необхідності повної перебудови вже існуючої архітектури. Такий підхід, базований на компонентній природі React та RESTful API, дозволяє вбудовувати нові інтерфейси, розширення, додаткові панелі управління або інтеграції із зовнішніми сервісами у вигляді окремих модулів, які не порушують загальної логіки застосунку. Наприклад, можливим є підключення модулю календаря для планування подій, системи оцінювання ефективності учасників, відеозв'язку через WebRTC або системи багаторівневої аналітики з візуалізацією прогресу кожного проєкту.

Інтеграція зі сторонніми API відіграє ключову роль у підвищенні зручності для користувачів і розширенні сценаріїв взаємодії. Завдяки реалізації відкритої API-архітектури, платформа «Спільні проєкти» може бути пов'язана з популярними інструментами та сервісами, які вже широко використовуються в командній роботі. Наприклад, підключення до Google Drive дозволить учасникам прикріплювати та переглядати файли безпосередньо з хмарного сховища, інтеграція з GitHub — надавати доступ до репозиторіїв з кодом, а зв'язок з Discord або Slack — синхронізувати канали комунікації без дублювання інформації. Крім того, інтеграція з сервісами типу Trello або Notion може реалізувати автоматичне відображення задач і нотаток, створених у зовнішніх системах, що зробить платформу ще більш відкритою до гібридного використання.

Особливу увагу варто звернути на можливість інтеграції з API систем державного управління, академічних платформ або наукових архівів. Це може стати особливо актуальним у випадку використання платформи в освітньому середовищі, для організації проєктної роботи в університетах або в рамках грантових досліджень. Наприклад, підключення до OpenAIRE API або Google Scholar дозволить відображати релевантні наукові джерела в описі проєкту, а

взаємодія з державними реєстрами (через e-Data, Prozorro, Diia API) забезпечить прозору інтеграцію з національними цифровими сервісами.

Платформа також потенційно може бути адаптована до формату SaaS (Software as a Service), що дозволить іншим організаціям розгортати її під власними брендами з кастомізованими модулями, стилями та базами даних. Для цього важливо передбачити мультиорендну архітектуру, у якій кожен новий клієнт отримує окрему ізольовану інстанцію платформи з власними конфігураціями, користувачами та проєктами. Така архітектурна модель може стати основою для подальшої комерціалізації продукту або розвитку як open-source рішення з підтримкою спільноти, що буде сприяти розширенню екосистеми, залученню нових учасників і партнерств, зокрема в академічному, урядовому або бізнес-середовищі.

4.4. Можливості для комерціалізації або відкритого доступу (Open Source)

Можливості для комерціалізації або відкритого доступу (Open Source) вебплатформи «Спільні проєкти» значною мірою залежать від обраної стратегії розвитку продукту, цільової аудиторії, а також від способу позиціонування системи на ринку цифрових сервісів для колективної роботи. У разі ухвалення рішення про комерціалізацію, платформа може бути запропонована як готовий продукт для бізнес-команд, освітніх установ, проєктних організацій або ініціативних об’єднань, які потребують інструментів для керування командною діяльністю, але не мають змоги або бажання створювати власні рішення з нуля. Основна модель монетизації в такому випадку може базуватись на підписці або ліцензуванні використання платформи. Клієнти зможуть обирати тарифні плани з різним рівнем доступу до функцій, кількістю учасників у проєктах, обсягом хмарного зберігання, технічною підтримкою та додатковими опціями — від аналітики до інтеграції зі сторонніми сервісами.

Комерційна версія також може бути спрямована на корпоративний сегмент, де важливу роль відіграє брендована адаптація системи під потреби конкретної організації. Це може включати зміну логотипів, кольорової гами, окреме доменне ім’я, а також додаткові функціональні модулі, зокрема систему затвердження проєктів, розширені аналітичні панелі або спеціалізовану документацію для внутрішніх аудитів. У таких умовах платформа перетворюється не просто на інструмент для співпраці, а на повноцінну цифрову екосистему для управління проєктною культурою всередині організації. Комерційна експлуатація також відкриває перспективи співпраці з акселераторами, стартап-інкубаторами, венчурними фондами, які могли б використовувати систему для супроводу проєктів у своїх портфелях.

Поряд із цим, розробка платформи у форматі відкритого коду (Open Source) надає іншу стратегію розвитку, яка зосереджена на залученні широкої спільноти розробників, дизайнерів, аналітиків і користувачів, готових робити внески у розвиток продукту на безкоштовній основі. Публікація коду на відкритих репозиторіях, таких як GitHub, з належною документацією, ліцензією та прикладами розгортання, дозволяє досягти високого рівня прозорості, викликає довіру серед технічної аудиторії та створює умови для незалежного розвитку платформи у різних напрямах. Завдяки цьому проєкт може швидко масштабуватися, отримувати виправлення помилок, нові функціональні модулі, адаптації до різних мов або специфічних середовищ без втручання основної команди розробників.

Open Source-модель також може підтримуватися через донати, спонсорство, участь у грантових програмах, залучення коштів від громадських організацій, міжнародних проєктів цифрової інклюзії або навіть через платні преміум-розширення, які будуть доступні паралельно з основною безкоштовною версією. Такий гібридний підхід дозволяє поєднати соціальну місію з економічною життєздатністю проєкту.

У перспективі відкритий код сприятиме широкому розповсюдженню платформи в освітньому середовищі, особливо серед технічних вишів, де студенти та викладачі зможуть не лише використовувати систему, але й вивчати її структуру, змінювати код, створювати власні розширення або проєкти на її базі. Це, в свою чергу, формуватиме нові генерації розробників, які будуть залучені до її еволюції. Крім того, Open Source дає можливість створювати локалізовані версії для різних країн, які будуть відповідати національним мовним, правовим або технічним особливостям.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі дипломної роботи було здійснено всебічне дослідження етапів перевірки працездатності, ефективності та готовності вебплатформи «Спільні проєкти» до широкого використання. Основна увага була зосереджена на визначенні методики тестування, оцінці функціонального наповнення системи, аналізі зручності інтерфейсу з позиції кінцевого користувача, а також на окресленні стратегічних перспектив розвитку та інтеграції з іншими цифровими сервісами.

Під час розробки методики тестування було враховано ключові принципи сучасного підходу до оцінки вебзастосунків: модульність, повторюваність, покриття основних сценаріїв використання, стресові умови експлуатації, а також аналіз реакції користувача на структуру і логіку інтерфейсу. У результаті цього підходу вдалось виявити потенційні вразливі місця, провести налагодження маршрутів, перевірити надійність системи авторизації, безпечність обробки даних та загальну стійкість серверної частини під навантаженням.

Проведене тестування функціоналу платформи показало, що основні модулі системи працюють узгоджено, без збоїв та затримок. Користувачі мали змогу зареєструватися, створити та редагувати проєкти, приєднуватися до вже існуючих ініціатив, вести спілкування, а також використовувати базові інструменти для управління командною діяльністю. Було підтверджено високу стабільність інтерфейсу при роботі на різних пристроях, завдяки адаптивному дизайну та використанню сучасних фреймворків, таких як TailwindCSS і React.

Оцінка UI/UX-компонентів продемонструвала позитивні результати: більшість користувачів швидко орієнтувалися в навігації, знаходили потрібні функції та з легкістю здійснювали взаємодію з платформою. Це свідчить про вдалу архітектуру інтерфейсу, раціональне компонування візуальних елементів

і загальну логіку побудови взаємодії користувача з системою. За результатами зворотного зв’язку від учасників тестування було внесено низку уточнень, що дозволило ще більше підвищити ергономічність і доступність інтерфейсу.

Значну увагу було приділено питанню подальшого масштабування платформи та її технічної інтеграції зі сторонніми сервісами. Встановлено, що архітектура вебдодатка дозволяє гнучко розширювати функціонал за рахунок модульного підходу, використання REST API та можливості підключення до хмарних сервісів. У перспективі платформа може бути використана як інструмент не лише для командної роботи, а й як основа для SaaS-рішень, освітніх хабів або even-менеджменту. Сумісність із Google Drive, GitHub, Discord, а також можливість підключення державних та наукових API дає змогу адаптувати платформу до різних цільових аудиторій.

Окремо було розглянуто шляхи комерціалізації та перспективи відкритого поширення платформи. Було обґрунтовано доцільність гібридної моделі — поєднання відкритого коду з можливістю пропонувати преміум-функції за підпискою або брендування для окремих організацій. Такий підхід дозволяє зберегти соціальну місію проєкту, забезпечити довіру спільноти, а також створити економічну базу для сталого розвитку та технічного супроводу.

У підсумку, розділ довів, що платформа «Спільні проєкти» не лише функціонально придатна до використання, але й має значний потенціал для масштабування, інтеграції та вдосконалення. Вона відповідає сучасним вимогам цифрової колаборації, поєднує зручність, безпеку, гнучкість і відкритість до майбутніх змін, що є запорукою її життєздатності в умовах стрімко змінного інформаційного середовища.

ВИСНОВКИ

У результаті виконання дипломної роботи на тему розробки вебплатформи для командної співпраці «Спільні проєкти» було успішно реалізовано повний цикл створення сучасного односторінкового вебзастосунку, який охоплює всі ключові аспекти — від концептуального обґрунтування і аналізу потреб користувачів до практичної реалізації, тестування та оцінки можливостей масштабування. Проведене дослідження підтвердило актуальність теми в умовах цифрової трансформації суспільства, розвитку віддаленої роботи, освітніх проєктів та міждисциплінарної колаборації, де ефективна взаємодія між учасниками є критично важливою.

У теоретичній частині було детально розглянуто поняття спільної роботи в мережі, проаналізовано технології веброзробки, описано принципи побудови SPA-додатків на базі React та проблеми безпеки у цифрових середовищах. Аналіз аналогічних рішень дозволив виявити стратегічні прогалини у вже існуючих платформах, що стало основою для формування обґрунтованого технічного завдання з чітко визначеними функціональними та нефункціональними вимогами. Було створено архітектурну модель системи, побудовано структурну схему, UML-діаграми, а також реалізовано взаємодію між основними компонентами системи: клієнтським інтерфейсом, серверною логікою та базою даних.

На практичному етапі було впроваджено ключовий функціонал: реєстрацію та автентифікацію користувачів, створення та пошук проєктів, редагування профілів, внутрішню комунікацію та обмін файлами. Система була успішно розгорнута на хостинговому сервісі з дотриманням вимог до безпеки, продуктивності й масштабованості. Проведене тестування підтвердило стабільність роботи платформи, логічність інтерфейсу, високий рівень користувацької задоволеності та функціональну повноту.

Окрему увагу було приділено перспективам розвитку платформи. З’ясовано, що завдяки відкритій архітектурі, використанню сучасного стеку технологій (React, Node.js, MongoDB, Firebase), можливості REST API, система готова до інтеграції зі сторонніми сервісами, підтримує гнучке налаштування, може масштабуватися та адаптуватися під різні галузі застосування. Було проаналізовано можливі моделі комерціалізації — як SaaS-рішення, а також формат відкритого коду для підтримки ініціатив освітнього, дослідницького чи волонтерського спрямування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Niu, H. (2018). Innovation of scientific research management and data management system in universities under big data environment. *Modern Information Technology*, 2(3), 115–117.
2. Lupenko, S., Lutsyk, N., Yasniy, O., & Sobaszek, Ł. (2018). Statistical analysis of human heart with increased informativeness. *Acta Mechanica et Automatica*, 12(4), 311–315. https://doi.org/10.2478/ama-2018-0047
3. Wang, G., & Wang, Y. (2021). Innovative marketing framework for enterprises using edge-enabled data analysis. *Mobile Information Systems*, 2021, Article ID 6699420.
4. Teslia, I., Shatska, Z., Sliusarenko, N., & Dukhno, O. (2022). Development of the concept of building project management systems in the context of digital transformation of project-oriented companies. *Eastern-European Journal of Enterprise Technologies*, 6(3), 120.
5. Deng, Z., Yan, M., & Xiao, X. (2021). An early risk warning of cross-border e-commerce using BP neural network. *Mobile Information Systems*, 2021, Article ID 5518424.
6. Jo, H., Lee, H., Suh, Y., Kim, J., & Park, Y. (2015). A dynamic feasibility analysis of public investment projects: An integrated approach using system dynamics and agent-based modeling. *International Journal of Project Management*, 33(8), 1863–1876. https://doi.org/10.1016/j.ijproman.2015.07.002
7. Lupenko, S., Orobchuk, O., Stadnik, N., & Zozulya, A. (2018). Modeling and signals processing using cyclic random functions. In *Proceedings of the 13th IEEE International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)* (p. 360). Lviv, Ukraine.

8. Torres, J. P. (2019). System dynamics review and publications 1985–2017: Analysis, synthesis and contributions. *System Dynamics Review*, 35(2), 160–176. https://doi.org/10.1002/sdr.1628
9. Calderon-Tellez, J. A., et al. (2024). Project management and system dynamics modelling: Time to connect with innovation and sustainability. *Systems Research and Behavioral Science*, 41(1), 3–29.
10. van Eck, N. J., & Waltman, L. (2010). Software survey: VOSviewer, a computer program for bibliometric mapping. *Scientometrics*. https://doi.org/10.1093/scipol/scy034
11. Ansari, R. (2019). Dynamic simulation model for project change management policies: Engineering project case. *Journal of Construction Engineering and Management*, 145(7), 965–979.
12. de Wit, A. (1988). Measurement of project success. *International Journal of Project Management*, 6(3), 164–170. https://doi.org/10.1016/0263-7863(88)90043-9
13. Lin, C. Y., Abdel-Hamid, T. K., & Sherif, J. S. (1997). Software engineering process simulation model (SEPS). *Journal of Systems and Software*, 38(3), 263–277. https://doi.org/10.1016/S0164-1212(96)00156-2
14. Association for Project Management. (2019). *APM body of knowledge* (7th ed.). Buckinghamshire: APM.
15. Artto, K., Ahola, T., & Vartiainen, V. (2016). From the front end of projects to the back end of operations: Managing projects for value creation throughout the system lifecycle. *International Journal of Project Management*, 34(2), 258–270. https://doi.org/10.1016/j.ijproman.2015.05.003
16. Winter, M., Smith, C., Morris, P., & Cicmil, S. (2006). Directions for future research in project management: The main findings of a UK government-funded research network. *International Journal of Project Management*, 24(8), 638–649. https://doi.org/10.1016/j.ijproman.2006.08.009

17. Pargar, F., Kujala, J., Aaltonen, K., & Ruutu, S. (2019). Value creation dynamics in a project alliance. *International Journal of Project Management*, 37(5), 716–730. https://doi.org/10.1016/j.ijproman.2018.12.006

ДОДАТОК 1

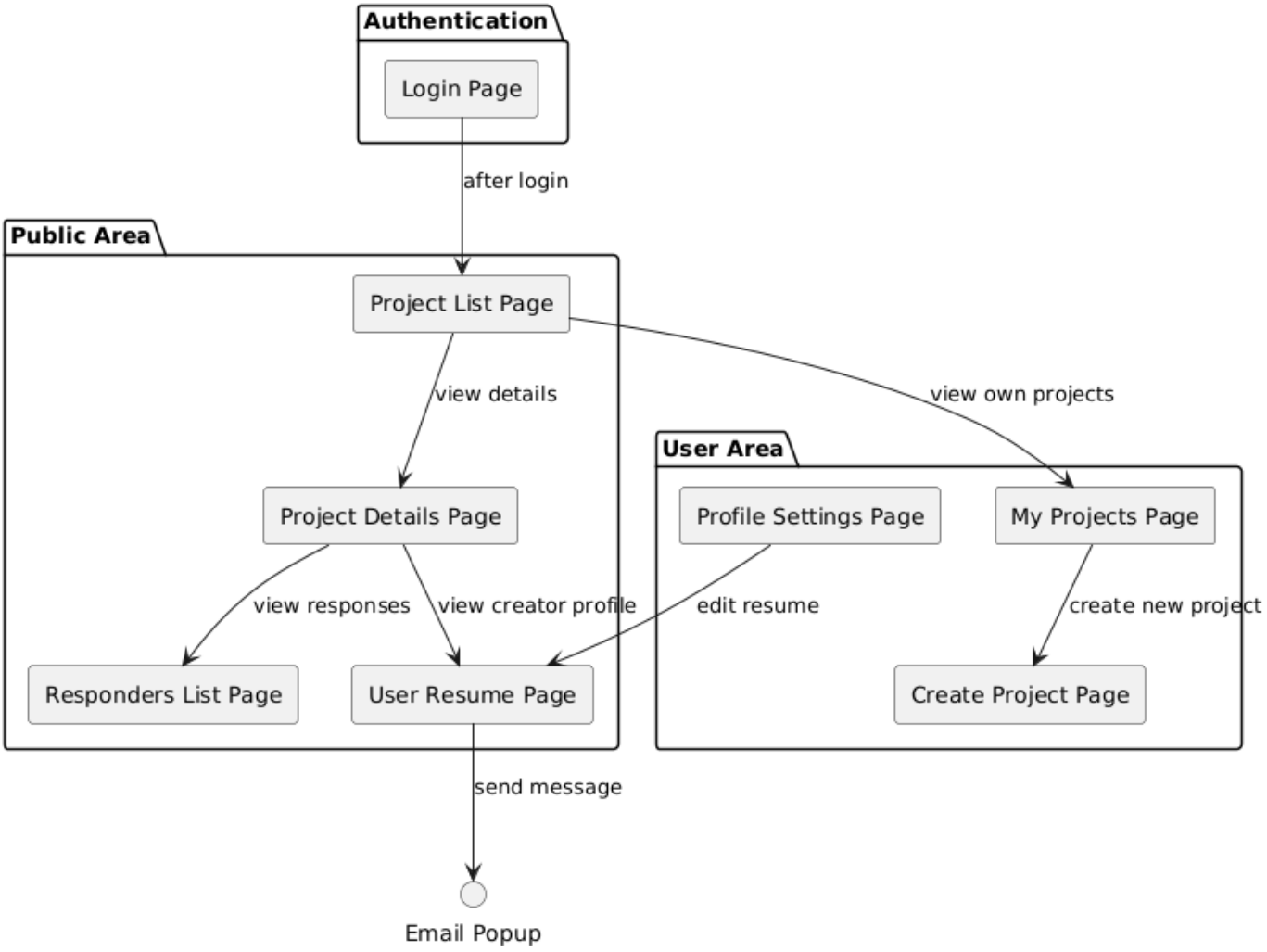
Платформа для пошуку партнерів для спільних проєктів

Структурна схема системи

ІАЛЦ.467100.004 Д1

Аркушів 1

Київ 2025 р



					ІАЛЦ.467100.004 Д1		
		№ докум.	Підпис	Дата			
Розробив		КапраренкоД.М.			**Еволюційні алгоритми глобальної пошукової оптимізації** Структурна схема системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Гордієнко Ю. Г.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-12	
Н. Контр.							
Затвердив							

ДОДАТОК 2

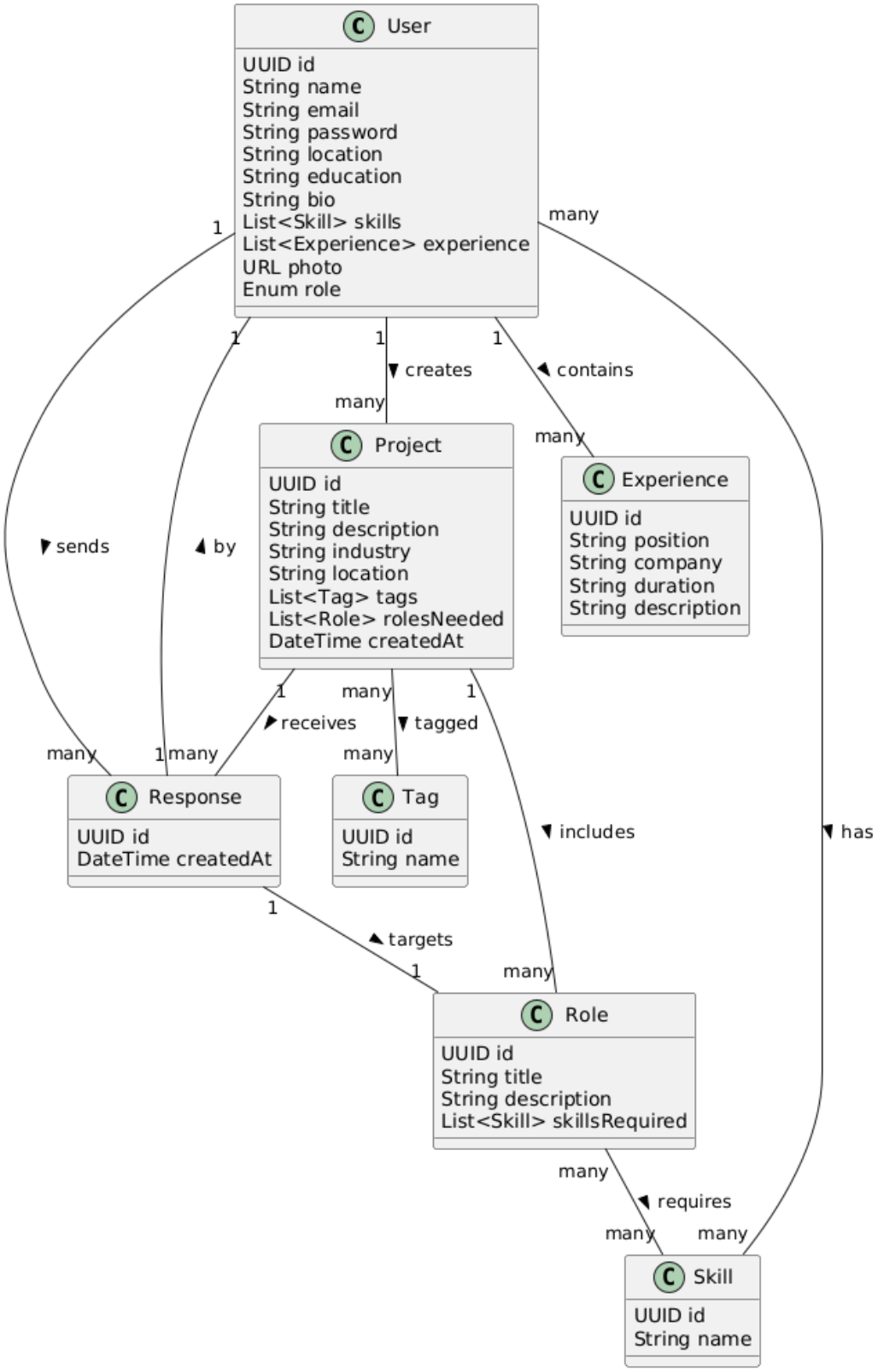
Платформа для пошуку партнерів для спільних проєктів

Функціональна схема (діаграма класів)

ІАЛЦ.467100.005 Д2

Аркушів 1

Київ 2025 р



					ІАЛЦ.467100.006 Д2		
		№ докум.	Підпис	Дата			
Розробив		КапраренкоД.М.			**Еволюційні алгоритми глобальної пошукової оптимізації** Алгоритм дій програмного забезпечення	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Гордієнко Ю. Г.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-12	
Н. Контр.							
Затвердив							

ДОДАТОК 3

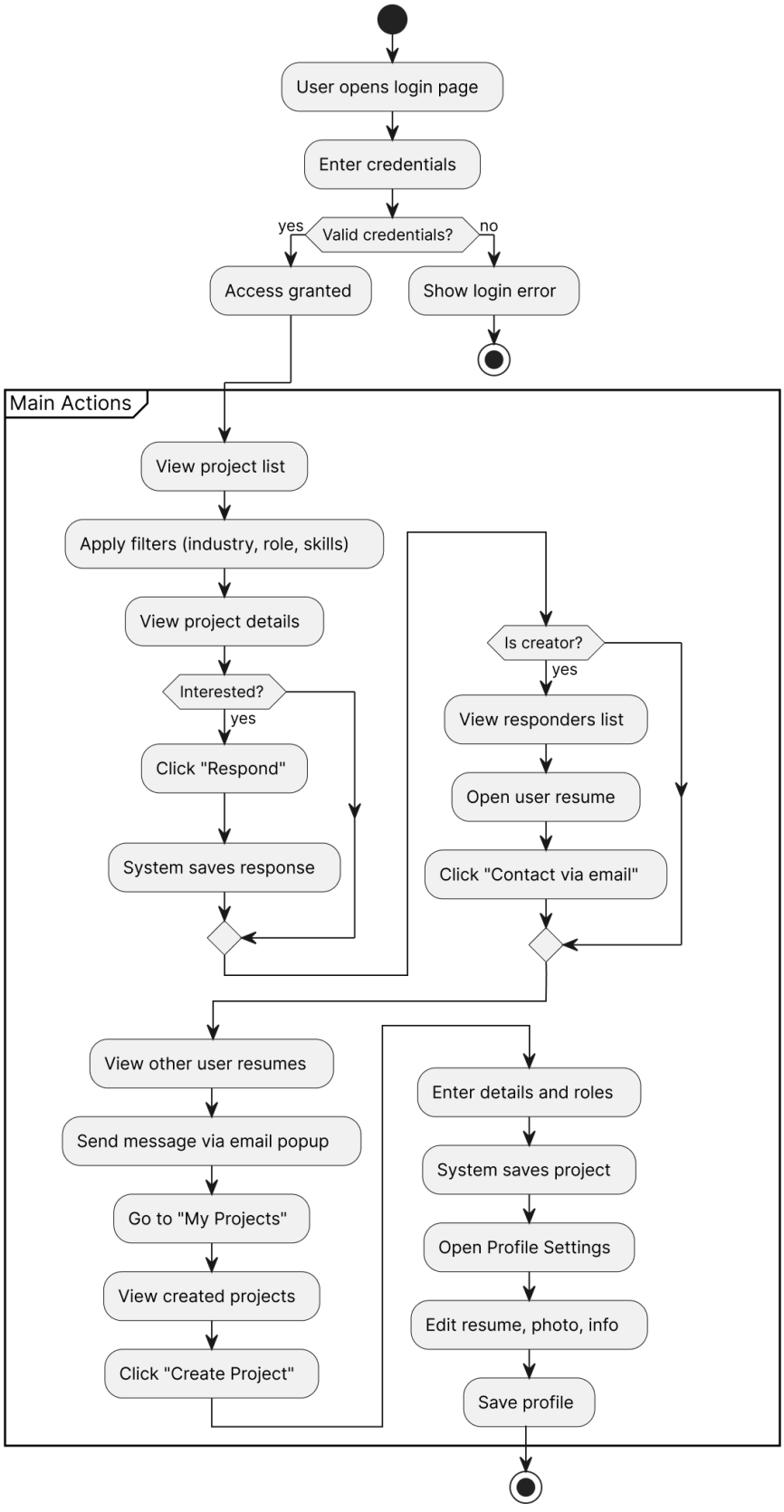
Платформа для пошуку партнерів для спільних проєктів

Алгоритм дій програмного забезпечення

ІАЛЦ.467100.006 Д3

Аркушів 1

Київ 2025 р



					ІАЛЦ.467100.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		КапраренкоД.М.			**Еволюційні алгоритми глобальної пошукової оптимізації** Алгоритм дій програмного забезпечення	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Гордієнко Ю. Г.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-12	
Н. Контр.							
Затвердив							

ДОДАТОК 4

Платформа для пошуку партнерів для спільних проєктів

Текст програмного коду

ІАЛЦ.467100.007 Д4

Аркушів 55

Київ 2025 р

Client Side

app.jsx

```
import { createBrowserRouter, RouterProvider } from "react-router-
dom";
import Login from "./components/auth/Login";
import Signup from "./components/auth/Signup";
import Home from "./components/Home";
import Jobs from "./components/Jobs";
import Browse from "./components/Browse";
import Profile from "./components/Profile";
import JobDescription from "./components/JobDescription";
import Companies from "./components/admin/Companies";
import CompanyCreate from "./components/admin/CompanyCreate";
import CompanySetup from "./components/admin/CompanySetup";
import AdminJobs from "./components/admin/AdminJobs";
import PostJob from "./components/admin/PostJob";
import Applicants from "./components/admin/Applicants";
import ProtectedRoute from "./components/admin/ProtectedRoute";

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <Home />,
  },
  {
    path: "/login",
    element: <Login />,
  },
  {
    path: "/signup",
    element: <Signup />,
  },
  {
    path: "/jobs",
    element: <Jobs />,
```

```
  },
  {
    path: "/description/:id",
    element: <JobDescription />,
  },
  {
    path: "/browse",
    element: <Browse />,
  },
  {
    path: "/profile",
    element: <Profile />,
  },
  // admin ke liye yha se start hoga
  {
    path: "/admin/companies",
    element: (
      <ProtectedRoute>
        <Companies />
      </ProtectedRoute>
    ),
  },
  {
    path: "/admin/companies/create",
    element: (
      <ProtectedRoute>
        <CompanyCreate />
      </ProtectedRoute>
    ),
  },
  {
    path: "/admin/companies/:id",
    element: (
      <ProtectedRoute>
        <CompanySetup />
      </ProtectedRoute>
    ),
  },
  {
    path: "/admin/jobs",
    element: (
      <ProtectedRoute>
```

```
            <AdminJobs />
          </ProtectedRoute>
        ),
      },
      {
        path: "/admin/jobs/create",
        element: (
          <ProtectedRoute>
            <PostJob />
          </ProtectedRoute>
        ),
      },
      {
        path: "/admin/jobs/:id/applicants",
        element: (
          <ProtectedRoute>
            <Applicants />
          </ProtectedRoute>
        ),
      },
    ]);
    function App() {
      return (
        <div>
          <RouterProvider router={appRouter} />
        </div>
      );
    }

    export default App;
```

index.css

```
@tailwind base;
  @tailwind components;
  @tailwind utilities;

  @layer base {
    :root {
      --background: 0 0% 100%;
      --foreground: 222.2 84% 4.9%;

      --card: 0 0% 100%;
      --card-foreground: 222.2 84% 4.9%;
```

```
    --popover: 0 0% 100%;
    --popover-foreground: 222.2 84% 4.9%;

    --primary: 222.2 47.4% 11.2%;
    --primary-foreground: 210 40% 98%;

    --secondary: 210 40% 96.1%;
    --secondary-foreground: 222.2 47.4% 11.2%;

    --muted: 210 40% 96.1%;
    --muted-foreground: 215.4 16.3% 46.9%;

    --accent: 210 40% 96.1%;
    --accent-foreground: 222.2 47.4% 11.2%;

    --destructive: 0 84.2% 60.2%;
    --destructive-foreground: 210 40% 98%;

    --border: 214.3 31.8% 91.4%;
    --input: 214.3 31.8% 91.4%;
    --ring: 222.2 84% 4.9%;

    --radius: 0.5rem;
  }

  .dark {
    --background: 222.2 84% 4.9%;
    --foreground: 210 40% 98%;

    --card: 222.2 84% 4.9%;
    --card-foreground: 210 40% 98%;

    --popover: 222.2 84% 4.9%;
    --popover-foreground: 210 40% 98%;

    --primary: 210 40% 98%;
    --primary-foreground: 222.2 47.4% 11.2%;

    --secondary: 217.2 32.6% 17.5%;
    --secondary-foreground: 210 40% 98%;

    --muted: 217.2 32.6% 17.5%;
```

```
        --muted-foreground: 215 20.2% 65.1%;

        --accent: 217.2 32.6% 17.5%;
        --accent-foreground: 210 40% 98%;

        --destructive: 0 62.8% 30.6%;
        --destructive-foreground: 210 40% 98%;

        --border: 217.2 32.6% 17.5%;
        --input: 217.2 32.6% 17.5%;
        --ring: 212.7 26.8% 83.9%;
      }
    }

    @layer base {
      * {
        @apply border-border;
      }
      body {
        @apply bg-background text-foreground;
      }
    }
```

AppliedJobTable.jsx

```
import {
  Table,
  TableBody,
  TableCaption,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "./ui/table";
import { Badge } from "./ui/badge";
import { useSelector } from "react-redux";

const AppliedJobTable = () => {
  const { allAppliedJobs } = useSelector((store) => store.job);
  return (
    <div>
      <Table>
        <TableCaption>A list of your applied jobs</TableCaption>
        <TableHeader>
```

```
          <TableRow>
            <TableHead>Date</TableHead>
            <TableHead>Job Role</TableHead>
            <TableHead>Project</TableHead>
            <TableHead className="text-right">Status</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {allAppliedJobs.length <= 0 ? (
            <span>You haven't applied any job yet.</span>
          ) : (
            allAppliedJobs.map((appliedJob) => (
              <TableRow key={appliedJob._id}>
                <TableCell>{appliedJob?.createdAt?.split("T")[0]}</TableC
ell>
                <TableCell>{appliedJob.job?.title}</TableCell>
                <TableCell>{appliedJob.job?.company?.name}</TableCell>
                <TableCell className="text-right">
                  <Badge
                    className={`${
                      appliedJob?.status === "rejected"
                        ? "bg-red-400"
                        : appliedJob.status === "pending"
                        ? "bg-gray-400"
                        : "bg-green-400"
                    }`}>
                    {appliedJob.status.toUpperCase()}
                  </Badge>
                </TableCell>
              </TableRow>
            ))
          )}
        </TableBody>
      </Table>
    </div>
  );
};

export default AppliedJobTable;
```

Browse.jsx

```
import { useEffect } from "react";
```

```
import Navbar from "./shared/Navbar";
import Job from "./Job";
import { useDispatch, useSelector } from "react-redux";
import { setSearchedQuery } from "@/redux/jobSlice";
import useGetAllJobs from "@/hooks/useGetAllJobs";

// const randomJobs = [1, 2,45];

const Browse = () => {
  useGetAllJobs();
  const { allJobs } = useSelector((store) => store.job);
  const dispatch = useDispatch();
  useEffect(() => {
    return () => {
      dispatch(setSearchedQuery(""));
    };
  }, []);
  return (
    <div>
      <Navbar />
      <div className="max-w-7xl mx-auto my-10">
        <h1 className="font-bold text-xl my-10">
          Search Results ({allJobs.length})
        </h1>
        <div className="grid grid-cols-3 gap-4">
          {allJobs.map((job) => {
            return <Job key={job._id} job={job} />;
          })}
        </div>
      </div>
    </div>
  );
};

export default Browse;
```

CategoryCarousel.jsx

```
import {
  Carousel,
  CarouselContent,
  CarouselItem,
  CarouselNext,
```

```
  CarouselPrevious,
} from "./ui/carousel";
import { Button } from "./ui/button";
import { useDispatch } from "react-redux";
import { useNavigate } from "react-router-dom";
import { setSearchedQuery } from "@/redux/jobSlice";

const category = [
  "Frontend Developer",
  "Backend Developer",
  "Data Science",
  "Graphic Designer",
  "FullStack Developer",
];

const CategoryCarousel = () => {
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const searchJobHandler = (query) => {
    dispatch(setSearchedQuery(query));
    navigate("/browse");
  };

  return (
    <div>
      <Carousel className="w-full max-w-xl mx-auto my-20">
        <CarouselContent>
          {category.map((cat, index) => (
            <CarouselItem className="md:basis-1/2 lg-basis-1/3">
              <Button
                onClick={() => searchJobHandler(cat)}
                variant="outline"
                className="rounded-full">
                {cat}
              </Button>
            </CarouselItem>
          ))}
        </CarouselContent>
        <CarouselPrevious />
        <CarouselNext />
      </Carousel>
    </div>
  );
```

```
};

export default CategoryCarousel;
```

FilterCard.jsx

```
import { useEffect, useState } from "react";
import { RadioGroup, RadioGroupItem } from "./ui/radio-group";
import { Label } from "./ui/label";
import { useDispatch } from "react-redux";
import { setSearchedQuery } from "@/redux/jobSlice";

const fitlerData = [
  {
    fitlerType: "Location",
    array: ["Kyiv", "Lviv", "Vinnytsia", "Kharkiv", "Poltava"],
  },
  {
    fitlerType: "Industry",
    array: ["Frontend Developer", "Backend Developer", "FullStack
Developer"],
  },
  {
    fitlerType: "Salary",
    array: ["0-5k$", "6-10k$", "11-20k$"],
  },
];

const FilterCard = () => {
  const [selectedValue, setSelectedValue] = useState("");
  const dispatch = useDispatch();
  const changeHandler = (value) => {
    setSelectedValue(value);
  };
  useEffect(() => {
    dispatch(setSearchedQuery(selectedValue));
  }, [selectedValue]);
  return (
    <div className="w-full bg-white p-3 rounded-md">
      <h1 className="font-bold text-lg">Filter Jobs</h1>
      <hr className="mt-3" />
      <RadioGroup value={selectedValue} onValueChange={changeHandler}>
        {fitlerData.map((data, index) => (
```

```
          <div>
            <h1 className="font-bold text-lg">{data.fitlerType}</h1>
            {data.array.map((item, idx) => {
              const itemId = `id${index}-${idx}`;
              return (
                <div className="flex items-center space-x-2 my-2">
                  <RadioGroupItem value={item} id={itemId} />
                  <Label htmlFor={itemId}>{item}</Label>
                </div>
              );
            })}
          </div>
        ))}
      </RadioGroup>
    </div>
  );
};

export default FilterCard;
```

HeroSection.jsx

```
import { useState } from "react";
import { Button } from "./ui/button";
import { Search } from "lucide-react";
import { useDispatch } from "react-redux";
import { setSearchedQuery } from "@/redux/jobSlice";
import { useNavigate } from "react-router-dom";

const HeroSection = () => {
  const [query, setQuery] = useState("");
  const dispatch = useDispatch();
  const navigate = useNavigate();

  const searchJobHandler = () => {
    dispatch(setSearchedQuery(query));
    navigate("/browse");
  };

  return (
    <div className="text-center">
      <div className="flex flex-col gap-5 my-10">
        <h1 className="text-5xl font-bold">
```

```
          <span className="text-[#616161]">Let's search for a
project</span>
        </h1>

        <div className="flex w-[40%] shadow-lg border border-gray-200 pl-
3 rounded-full items-center gap-4 mx-auto">
          <input
            type="text"
            placeholder="Project Title"
            onChange={(e) => setQuery(e.target.value)}
            className="outline-none border-none w-full"
          />
          <Button
            onClick={searchJobHandler}
            className="rounded-r-full bg-[#616161]">
            <Search className="h-5 w-5" />
          </Button>
        </div>
      </div>
    </div>
  );
};

export default HeroSection;
```

Home.jsx

```
import { useEffect } from "react";
import Navbar from "./shared/Navbar";
import HeroSection from "./HeroSection";
import CategoryCarousel from "./CategoryCarousel";
import LatestJobs from "./LatestJobs";
import Footer from "./shared/Footer";
import useGetAllJobs from "@/hooks/useGetAllJobs";
import { useSelector } from "react-redux";
import { useNavigate } from "react-router-dom";

const Home = () => {
  useGetAllJobs();
  const { user } = useSelector((store) => store.auth);
  const navigate = useNavigate();
  useEffect(() => {
    if (user?.role === "recruiter") {
```

```
      navigate("/admin/companies");
    }
  }, []);
  return (
    <div>
      <Navbar />
      <HeroSection />
      <CategoryCarousel />
      <LatestJobs />
      <Footer />
    </div>
  );
};

export default Home;
```

Job.jsx

```
import { Button } from "./ui/button";
import { Avatar, AvatarImage } from "./ui/avatar";
import { Badge } from "./ui/badge";
import { useNavigate } from "react-router-dom";

const Job = ({ job }) => {
  const navigate = useNavigate();
  // const jobId = "lsekdhjgdsnfvsdkjf";

  const daysAgoFunction = (mongodbTime) => {
    const createdAt = new Date(mongodbTime);
    const currentTime = new Date();
    const timeDifference = currentTime - createdAt;
    return Math.floor(timeDifference / (1000 * 24 * 60 * 60));
  };

  return (
    <div className="p-5 rounded-md shadow-xl bg-white border border-gray-
100">
      <div className="flex items-center justify-between">
        <p className="text-sm text-gray-500">
          {daysAgoFunction(job?.createdAt) === 0
            ? "Today"
            : `${daysAgoFunction(job?.createdAt)} days ago`}
        </p>
```

```
      </div>

      <div className="flex items-center gap-2 my-2">
        <Button className="p-6" variant="outline" size="icon">
          <Avatar>
            <AvatarImage src={job?.company?.logo} />
          </Avatar>
        </Button>
        <div>
          <h1 className="font-medium text-lg">{job?.company?.name}</h1>
          <p className="text-sm text-gray-500">Ukraine</p>
        </div>
      </div>

      <div>
        <h1 className="font-bold text-lg my-2">{job?.title}</h1>
        <p className="text-sm text-gray-600">{job?.description}</p>
      </div>
      <div className="flex items-center gap-2 mt-4">
        <Badge className={"text-blue-700 font-bold"} variant="ghost">
          {job?.position} Positions
        </Badge>
        <Badge className={"text-[#F83002] font-bold"} variant="ghost">
          {job?.jobType}
        </Badge>
        <Badge className={"text-[#7209b7] font-bold"} variant="ghost">
          {job?.salary}$
        </Badge>
      </div>
      <div className="flex items-center gap-4 mt-4">
        <Button
          onClick={() => navigate(`/description/${job?._id}`)}
          variant="outline">
          Details
        </Button>
      </div>
    </div>
  );
};

export default Job;
```

JobDescription.jsx

```
import { useEffect, useState } from "react";
import { Badge } from "./ui/badge";
import { Button } from "./ui/button";
import { useParams } from "react-router-dom";
import axios from "axios";
import { APPLICATION_API_END_POINT, JOB_API_END_POINT } from
"@/utils/constant";
import { setSingleJob } from "@/redux/jobSlice";
import { useDispatch, useSelector } from "react-redux";
import { toast } from "sonner";

const JobDescription = () => {
  const { singleJob } = useSelector((store) => store.job);
  const { user } = useSelector((store) => store.auth);
  const isIntiallyApplied =
    singleJob?.applications?.some(
      (application) => application.applicant === user?._id
    ) || false;
  const [isApplied, setIsApplied] = useState(isIntiallyApplied);

  const params = useParams();
  const jobId = params.id;
  const dispatch = useDispatch();

  const applyJobHandler = async () => {
    try {
      const res = await axios.get(
        `${APPLICATION_API_END_POINT}/apply/${jobId}`,
        { withCredentials: true }
      );

      if (res.data.success) {
        setIsApplied(true); // Update the local state
        const updatedSingleJob = {
          ...singleJob,
          applications: [...singleJob.applications, { applicant:
user?._id }],
        };
        dispatch(setSingleJob(updatedSingleJob)); // helps us to real
time UI update
        toast.success(res.data.message);
      }
```

```
    } catch (error) {
      console.log(error);
      toast.error(error.response.data.message);
    }
  };

  useEffect(() => {
    const fetchSingleJob = async () => {
      try {
        const res = await axios.get(`${JOB_API_END_POINT}/get/${jobId}`,
{
          withCredentials: true,
        });
        if (res.data.success) {
          dispatch(setSingleJob(res.data.job));
          setIsApplied(
            res.data.job.applications.some(
              (application) => application.applicant === user?._id
            )
          ); // Ensure the state is in sync with fetched data
        }
      } catch (error) {
        console.log(error);
      }
    };
    fetchSingleJob();
  }, [jobId, dispatch, user?._id]);

  return (
    <div className="max-w-7xl mx-auto my-10">
      <div className="flex items-center justify-between">
        <div>
          <h1 className="font-bold text-xl">{singleJob?.title}</h1>
          <div className="flex items-center gap-2 mt-4">
            <Badge className={"text-blue-700 font-bold"} variant="ghost">
              {singleJob?.postion} Positions
            </Badge>
            <Badge className={"text-[#F83002] font-bold"}
variant="ghost">
              {singleJob?.jobType}
            </Badge>
            <Badge className={"text-[#7209b7] font-bold"}
variant="ghost">
```

```
          {singleJob?.salary}$
        </Badge>
      </div>
    </div>
    <Button
      onClick={isApplied ? null : applyJobHandler}
      disabled={isApplied}
      className={`rounded-lg ${
        isApplied
          ? "bg-gray-600 cursor-not-allowed"
          : "bg-[#7209b7] hover:bg-[#5f32ad]"
      }`}>
      {isApplied ? "Already Applied" : "Apply Now"}
    </Button>
  </div>
  <h1 className="border-b-2 border-b-gray-300 font-medium py-4">
    Job Description
  </h1>
  <div className="my-4">
    <h1 className="font-bold my-1">
      Role:{" "}
      <span className="pl-4 font-normal text-gray-800">
        {singleJob?.title}
      </span>
    </h1>
    <h1 className="font-bold my-1">
      Location:{" "}
      <span className="pl-4 font-normal text-gray-800">
        {singleJob?.location}
      </span>
    </h1>
    <h1 className="font-bold my-1">
      Description:{" "}
      <span className="pl-4 font-normal text-gray-800">
        {singleJob?.description}
      </span>
    </h1>
    <h1 className="font-bold my-1">
      Experience:{" "}
      <span className="pl-4 font-normal text-gray-800">
        {singleJob?.experience} yrs
      </span>
    </h1>
```

```
        <h1 className="font-bold my-1">
          Salary:{" "}
          <span className="pl-4 font-normal text-gray-800">
            {singleJob?.salary}$
          </span>
        </h1>
        <h1 className="font-bold my-1">
          Total Applicants:{" "}
          <span className="pl-4 font-normal text-gray-800">
            {singleJob?.applications?.length}
          </span>
        </h1>
        <h1 className="font-bold my-1">
          Posted Date:{" "}
          <span className="pl-4 font-normal text-gray-800">
            {singleJob?.createdAt.split("T")[0]}
          </span>
        </h1>
      </div>
    </div>
  );
};

export default JobDescription;
```

Jobs.jsx

```
import { useEffect, useState } from "react";
import Navbar from "./shared/Navbar";
import FilterCard from "./FilterCard";
import Job from "./Job";
import { useSelector } from "react-redux";
import { motion } from "framer-motion";

// const jobsArray = [1, 2, 3, 4, 5, 6, 7, 8];

const Jobs = () => {
  const { allJobs, searchedQuery } = useSelector((store) => store.job);
  const [filterJobs, setFilterJobs] = useState(allJobs);

  useEffect(() => {
    if (searchedQuery) {
      const filteredJobs = allJobs.filter((job) => {
```

```
        return (
          job.title.toLowerCase().includes(searchedQuery.toLowerCase())
||
          job.description.toLowerCase().includes(searchedQuery.toLowerCas
e()) ||
          job.location.toLowerCase().includes(searchedQuery.toLowerCase()
)
        );
      });
      setFilterJobs(filteredJobs);
    } else {
      setFilterJobs(allJobs);
    }
  }, [allJobs, searchedQuery]);

  return (
    <div>
      <Navbar />
      <div className="max-w-7xl mx-auto mt-5">
        <div className="flex gap-5">
          <div className="w-20%">
            <FilterCard />
          </div>
          {filterJobs.length <= 0 ? (
            <span>Job not found</span>
          ) : (
            <div className="flex-1 h-[88vh] overflow-y-auto pb-5">
              <div className="grid grid-cols-3 gap-4">
                {filterJobs.map((job) => (
                  <motion.div
                    initial={{ opacity: 0, x: 100 }}
                    animate={{ opacity: 1, x: 0 }}
                    exit={{ opacity: 0, x: -100 }}
                    transition={{ duration: 0.3 }}
                    key={job?._id}>
                    <Job job={job} />
                  </motion.div>
                ))}
              </div>
            </div>
          )}
        </div>
      </div>
```

```
    </div>
  );
};

export default Jobs;
```

LatestJobCards.jsx

```
import { Badge } from "./ui/badge";
import { useNavigate } from "react-router-dom";

const LatestJobCards = ({ job }) => {
  const navigate = useNavigate();
  return (
    <div
      onClick={() => navigate(`/description/${job._id}`)}
      className="p-5 rounded-md shadow-xl bg-white border border-gray-100
cursor-pointer">
      <div>
        <h1 className="font-medium text-lg">{job?.company?.name}</h1>
        <p className="text-sm text-gray-500">Ukraine</p>
      </div>
      <div>
        <h1 className="font-bold text-lg my-2">{job?.title}</h1>
        <p className="text-sm text-gray-600">{job?.description}</p>
      </div>
      <div className="flex items-center gap-2 mt-4">
        <Badge className={"text-blue-700 font-bold"} variant="ghost">
          {job?.position} Positions
        </Badge>
        <Badge className={"text-[#F83002] font-bold"} variant="ghost">
          {job?.jobType}
        </Badge>
        <Badge className={"text-[#7209b7] font-bold"} variant="ghost">
          {job?.salary}$
        </Badge>
      </div>
    </div>
  );
};

export default LatestJobCards;
```

LatestJobs.jsx

```
import LatestJobCards from "./LatestJobCards";
import { useSelector } from "react-redux";

// const randomJobs = [1, 2, 3, 4, 5, 6, 7, 8];

const LatestJobs = () => {
  const { allJobs } = useSelector((store) => store.job);

  return (
    <div className="max-w-7xl mx-auto my-20">
      <h1 className="text-4xl font-bold">
        <span className="text-[#616161]">Latest & Top </span> Job
Openings
      </h1>
      <div className="grid grid-cols-3 gap-4 my-5">
        {allJobs.length <= 0 ? (
          <span>No Job Available</span>
        ) : (
          allJobs
            ?.slice(0, 6)
            .map((job) => <LatestJobCards key={job._id} job={job} />)
        )}
      </div>
    </div>
  );
};

export default LatestJobs;
```

Profile.jsx

```
import { useState } from "react";
import Navbar from "./shared/Navbar";
import { Avatar, AvatarImage } from "./ui/avatar";
import { Button } from "./ui/button";
import { Contact, Mail, Pen } from "lucide-react";
import { Badge } from "./ui/badge";
import { Label } from "./ui/label";
import AppliedJobTable from "./AppliedJobTable";
import UpdateProfileDialog from "./UpdateProfileDialog";
import { useSelector } from "react-redux";
import useGetAppliedJobs from "@/hooks/useGetAppliedJobs";
```

```
// const skills = ["Html", "Css", "Javascript", "Reactjs"]
const isResume = true;

const Profile = () => {
  useGetAppliedJobs();
  const [open, setOpen] = useState(false);
  const { user } = useSelector((store) => store.auth);

  return (
    <div>
      <Navbar />
      <div className="max-w-4xl mx-auto bg-white border border-gray-200
rounded-2xl my-5 p-8">
        <div className="flex justify-between">
          <div className="flex items-center gap-4">
            <Avatar className="h-24 w-24">
              <AvatarImage
                src="https://www.shutterstock.com/image-vector/circle-
line-simple-design-logo-600nw-2174926871.jpg"
                alt="profile"
              />
            </Avatar>
            <div>
              <h1 className="font-medium text-xl">{user?.fullname}</h1>
              <p>{user?.profile?.bio}</p>
            </div>
          </div>
          <Button
            onClick={() => setOpen(true)}
            className="text-right"
            variant="outline">
            <Pen />
          </Button>
        </div>
        <div className="my-5">
          <div className="flex items-center gap-3 my-2">
            <Mail />
            <span>{user?.email}</span>
          </div>
          <div className="flex items-center gap-3 my-2">
            <Contact />
            <span>{user?.phoneNumber}</span>
```

```
          </div>
        </div>
        <div className="my-5">
          <h1>Skills</h1>
          <div className="flex items-center gap-1">
            {user?.profile?.skills.length !== 0 ? (
              user?.profile?.skills.map((item, index) => (
                <Badge key={index}>{item}</Badge>
              ))
            ) : (
              <span>NA</span>
            )}
          </div>
        </div>
        <div className="grid w-full max-w-sm items-center gap-1.5">
          <Label className="text-md font-bold">Resume</Label>
          {isResume ? (
            <a
              target="blank"
              href={user?.profile?.resume}
              className="text-blue-500 w-full hover:underline cursor-
pointer">
              {user?.profile?.resumeOriginalName}
            </a>
          ) : (
            <span>NA</span>
          )}
        </div>
      </div>
      <div className="max-w-4xl mx-auto bg-white rounded-2xl">
        <h1 className="font-bold text-lg my-5">Applied Jobs</h1>
        {/* Applied Job Table   */}
        <AppliedJobTable />
      </div>
      <UpdateProfileDialog open={open} setOpen={setOpen} />
    </div>
  );
};

export default Profile;
```

UpdateProfileDialog.jsx

```jsx
import { useState } from "react";
import {
  Dialog,
  DialogContent,
  DialogFooter,
  DialogHeader,
  DialogTitle,
} from "./ui/dialog";
import { Label } from "./ui/label";
import { Input } from "./ui/input";
import { Button } from "./ui/button";
import { Loader2 } from "lucide-react";
import { useDispatch, useSelector } from "react-redux";
import axios from "axios";
import { USER_API_END_POINT } from "@/utils/constant";
import { setUser } from "@/redux/authSlice";
import { toast } from "sonner";

const UpdateProfileDialog = ({ open, setOpen }) => {
  const [loading, setLoading] = useState(false);
  const { user } = useSelector((store) => store.auth);

  const [input, setInput] = useState({
    fullname: user?.fullname || "",
    email: user?.email || "",
    phoneNumber: user?.phoneNumber || "",
    bio: user?.profile?.bio || "",
    skills: user?.profile?.skills?.map((skill) => skill) || "",
    file: user?.profile?.resume || "",
  });
  const dispatch = useDispatch();

  const changeEventHandler = (e) => {
    setInput({ ...input, [e.target.name]: e.target.value });
  };

  const fileChangeHandler = (e) => {
    const file = e.target.files?.[0];
    setInput({ ...input, file });
  };

  const submitHandler = async (e) => {
    e.preventDefault();
```

```
    const formData = new FormData();
    formData.append("fullname", input.fullname);
    formData.append("email", input.email);
    formData.append("phoneNumber", input.phoneNumber);
    formData.append("bio", input.bio);
    formData.append("skills", input.skills);
    if (input.file) {
      formData.append("file", input.file);
    }
    try {
      setLoading(true);
      const res = await axios.post(
        `${USER_API_END_POINT}/profile/update`,
        formData,
        {
          headers: {
            "Content-Type": "multipart/form-data",
          },
          withCredentials: true,
        }
      );
      if (res.data.success) {
        dispatch(setUser(res.data.user));
        toast.success(res.data.message);
      }
    } catch (error) {
      console.log(error);
      toast.error(error.response.data.message);
    } finally {
      setLoading(false);
    }
    setOpen(false);
    console.log(input);
  };

  return (
    <div>
      <Dialog open={open}>
        <DialogContent
          className="sm:max-w-[425px]"
          onInteractOutside={() => setOpen(false)}>
          <DialogHeader>
            <DialogTitle>Update Profile</DialogTitle>
```

```
</DialogHeader>
<form onSubmit={submitHandler}>
  <div className="grid gap-4 py-4">
    <div className="grid grid-cols-4 items-center gap-4">
      <Label htmlFor="name" className="text-right">
        Name
      </Label>
      <Input
        id="name"
        name="name"
        type="text"
        value={input.fullname}
        onChange={changeEventHandler}
        className="col-span-3"
      />
    </div>
    <div className="grid grid-cols-4 items-center gap-4">
      <Label htmlFor="email" className="text-right">
        Email
      </Label>
      <Input
        id="email"
        name="email"
        type="email"
        value={input.email}
        onChange={changeEventHandler}
        className="col-span-3"
      />
    </div>
    <div className="grid grid-cols-4 items-center gap-4">
      <Label htmlFor="number" className="text-right">
        Number
      </Label>
      <Input
        id="number"
        name="number"
        value={input.phoneNumber}
        onChange={changeEventHandler}
        className="col-span-3"
      />
    </div>
    <div className="grid grid-cols-4 items-center gap-4">
      <Label htmlFor="bio" className="text-right">
```

```
            Bio
          </Label>
          <Input
            id="bio"
            name="bio"
            value={input.bio}
            onChange={changeEventHandler}
            className="col-span-3"
          />
        </div>
        <div className="grid grid-cols-4 items-center gap-4">
          <Label htmlFor="skills" className="text-right">
            Skills
          </Label>
          <Input
            id="skills"
            name="skills"
            value={input.skills}
            onChange={changeEventHandler}
            className="col-span-3"
          />
        </div>
        <div className="grid grid-cols-4 items-center gap-4">
          <Label htmlFor="file" className="text-right">
            Resume
          </Label>
          <Input
            id="file"
            name="file"
            type="file"
            accept="application/pdf"
            onChange={fileChangeHandler}
            className="col-span-3"
          />
        </div>
      </div>
      <DialogFooter>
        {loading ? (
          <Button className="w-full my-4">
            {" "}
```

```
                <Loader2 className="mr-2 h-4 w-4 animate-spin" />
Please wait{" "}
                </Button>
              ) : (
                <Button type="submit" className="w-full my-4">
                  Update
                </Button>
              )}
            </DialogFooter>
          </form>
        </DialogContent>
      </Dialog>
    </div>
  );
};

export default UpdateProfileDialog;
```

useGetAllAdminJobs.jsx

```
import { setAllAdminJobs } from '@/redux/jobSlice'
import { JOB_API_END_POINT } from '@/utils/constant'
import axios from 'axios'
import { useEffect } from 'react'
import { useDispatch } from 'react-redux'

const useGetAllAdminJobs = () => {
    const dispatch = useDispatch();
    useEffect(()=>{
        const fetchAllAdminJobs = async () => {
            try {
                const res = await
axios.get(`${JOB_API_END_POINT}/getadminjobs`,{withCredentials:true});
                if(res.data.success){
                    dispatch(setAllAdminJobs(res.data.jobs));
                }
            } catch (error) {
                console.log(error);
            }
        }
        fetchAllAdminJobs();
    },[])
}
```

```
export default useGetAllAdminJobs
```

useGetAllCompanies.jsx

```
import { setCompanies} from '@/redux/companySlice'
import { COMPANY_API_END_POINT} from '@/utils/constant'
import axios from 'axios'
import { useEffect } from 'react'
import { useDispatch } from 'react-redux'

const useGetAllCompanies = () => {
    const dispatch = useDispatch();
    useEffect(()=>{
        const fetchCompanies = async () => {
            try {
                const res = await
axios.get(`${COMPANY_API_END_POINT}/get`,{withCredentials:true});
                console.log('called');
                if(res.data.success){
                    dispatch(setCompanies(res.data.companies));
                }
            } catch (error) {
                console.log(error);
            }
        }
        fetchCompanies();
    },[])
}

export default useGetAllCompanies
```

useGetAllJobs.jsx

```
import { setAllJobs } from '@/redux/jobSlice'
import { JOB_API_END_POINT } from '@/utils/constant'
import axios from 'axios'
import { useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'

const useGetAllJobs = () => {
    const dispatch = useDispatch();
    const {searchedQuery} = useSelector(store=>store.job);
    useEffect(()=>{
```

```
        const fetchAllJobs = async () => {
            try {
                const res = await
axios.get(`${JOB_API_END_POINT}/get?keyword=${searchedQuery}`,{withCreden
tials:true});
                if(res.data.success){
                    dispatch(setAllJobs(res.data.jobs));
                }
            } catch (error) {
                console.log(error);
            }
        }
        fetchAllJobs();
    },[])
}

export default useGetAllJobs
```

useGetAppliedJobs.jsx

```
import { setAllAppliedJobs } from "@/redux/jobSlice";
import { APPLICATION_API_END_POINT } from "@/utils/constant";
import axios from "axios"
import { useEffect } from "react"
import { useDispatch } from "react-redux"

const useGetAppliedJobs = () => {
    const dispatch = useDispatch();

    useEffect(()=>{
        const fetchAppliedJobs = async () => {
            try {
                const res = await
axios.get(`${APPLICATION_API_END_POINT}/get`, {withCredentials:true});
                console.log(res.data);
                if(res.data.success){
                    dispatch(setAllAppliedJobs(res.data.application));
                }
            } catch (error) {
                console.log(error);
            }
        }
        fetchAppliedJobs();
    },[])
```

```
};
export default useGetAppliedJobs;
```

useGetCompanyById.jsx

```
import { setSingleCompany } from "@/redux/companySlice";
import { COMPANY_API_END_POINT } from "@/utils/constant";
import axios from "axios";
import { useEffect } from "react";
import { useDispatch } from "react-redux";

const useGetCompanyById = (companyId) => {
  const dispatch = useDispatch();
  useEffect(() => {
    const fetchSingleCompany = async () => {
      try {
        const res = await axios.get(
          `${COMPANY_API_END_POINT}/get/${companyId}`,
          { withCredentials: true }
        );
        console.log(res.data.company);
        if (res.data.success) {
          dispatch(setSingleCompany(res.data.company));
        }
      } catch (error) {
        console.log(error);
      }
    };
    fetchSingleCompany();
  }, [companyId, dispatch]);
};

export default useGetCompanyById;
```

tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  darkMode: ["class"],
  content: [
    './pages/**/*.{js,jsx}',
    './components/**/*.{js,jsx}',
```

```
    './app/**/*.{js,jsx}',
    './src/**/*.{js,jsx}',
  ],
  prefix: "",
  theme: {
    container: {
      center: true,
      padding: "2rem",
      screens: {
        "2xl": "1400px",
      },
    },
    extend: {
      colors: {
        border: "hsl(var(--border))",
        input: "hsl(var(--input))",
        ring: "hsl(var(--ring))",
        background: "hsl(var(--background))",
        foreground: "hsl(var(--foreground))",
        primary: {
          DEFAULT: "hsl(var(--primary))",
          foreground: "hsl(var(--primary-foreground))",
        },
        secondary: {
          DEFAULT: "hsl(var(--secondary))",
          foreground: "hsl(var(--secondary-foreground))",
        },
        destructive: {
          DEFAULT: "hsl(var(--destructive))",
          foreground: "hsl(var(--destructive-foreground))",
        },
        muted: {
          DEFAULT: "hsl(var(--muted))",
          foreground: "hsl(var(--muted-foreground))",
        },
        accent: {
          DEFAULT: "hsl(var(--accent))",
          foreground: "hsl(var(--accent-foreground))",
        },
        popover: {
          DEFAULT: "hsl(var(--popover))",
          foreground: "hsl(var(--popover-foreground))",
```

```
        },
        card: {
          DEFAULT: "hsl(var(--card))",
          foreground: "hsl(var(--card-foreground))",
        },
      },
      borderRadius: {
        lg: "var(--radius)",
        md: "calc(var(--radius) - 2px)",
        sm: "calc(var(--radius) - 4px)",
      },
      keyframes: {
        "accordion-down": {
          from: { height: "0" },
          to: { height: "var(--radix-accordion-content-height)" },
        },
        "accordion-up": {
          from: { height: "var(--radix-accordion-content-height)" },
          to: { height: "0" },
        },
      },
      animation: {
        "accordion-down": "accordion-down 0.2s ease-out",
        "accordion-up": "accordion-up 0.2s ease-out",
      },
    },
  },
  plugins: [require("tailwindcss-animate")],
}
```

package.json

```
{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --
max-warnings 0",
    "preview": "vite preview"
  },
```

```
  "dependencies": {
    "@radix-ui/react-avatar": "^1.0.4",
    "@radix-ui/react-dialog": "^1.1.1",
    "@radix-ui/react-label": "^2.0.2",
    "@radix-ui/react-popover": "^1.0.7",
    "@radix-ui/react-radio-group": "^1.2.0",
    "@radix-ui/react-select": "^2.1.1",
    "@radix-ui/react-slot": "^1.1.0",
    "@reduxjs/toolkit": "^2.2.6",
    "axios": "^1.7.2",
    "class-variance-authority": "^0.7.0",
    "clsx": "^2.1.1",
    "embla-carousel-react": "^8.1.6",
    "framer-motion": "^11.3.7",
    "lucide-react": "^0.395.0",
    "next-themes": "^0.3.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-redux": "^9.1.2",
    "react-router-dom": "^6.23.1",
    "redux-persist": "^6.0.0",
    "sonner": "^1.5.0",
    "tailwind-merge": "^2.3.0",
    "tailwindcss-animate": "^1.0.7"
  },
  "devDependencies": {
    "@types/react": "^18.2.66",
    "@types/react-dom": "^18.2.22",
    "@vitejs/plugin-react": "^4.2.1",
    "autoprefixer": "^10.4.19",
    "eslint": "^8.57.0",
    "eslint-plugin-react": "^7.34.1",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.6",
    "postcss": "^8.4.38",
    "tailwindcss": "^3.4.4",
    "vite": "^5.2.0"
  }
}
```

Server Side

index.js

```
import express from "express";
import cookieParser from "cookie-parser";
import cors from "cors";
import dotenv from "dotenv";
import connectDB from "./utils/db.js";
import userRoute from "./routes/user.route.js";
import companyRoute from "./routes/company.route.js";
import jobRoute from "./routes/job.route.js";
import applicationRoute from "./routes/application.route.js";

dotenv.config({});

const app = express();

// middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
const corsOptions = {
  origin: "http://localhost:5173",
  credentials: true,
};

app.use(cors(corsOptions));

const PORT = process.env.PORT || 8000;

// api's
app.use("/api/v1/user", userRoute);
app.use("/api/v1/company", companyRoute);
app.use("/api/v1/job", jobRoute);
app.use("/api/v1/application", applicationRoute);


app.listen(PORT, () => {
  connectDB();
  console.log(`Server running at port ${PORT}`);
});
```

application.controller.js

```
import { Application } from "../models/application.model.js";
import { Job } from "../models/job.model.js";

export const applyJob = async (req, res) => {
    try {
        const userId = req.id;
        const jobId = req.params.id;
        if (!jobId) {
            return res.status(400).json({
                message: "Job id is required.",
                success: false
            })
        };
        // check if the user has already applied for the job
        const existingApplication = await Application.findOne({ job:
jobId, applicant: userId });

        if (existingApplication) {
            return res.status(400).json({
                message: "You have already applied for this jobs",
                success: false
            });
        }

        // check if the jobs exists
        const job = await Job.findById(jobId);
        if (!job) {
            return res.status(404).json({
                message: "Job not found",
                success: false
            })
        }
        // create a new application
        const newApplication = await Application.create({
            job:jobId,
            applicant:userId,
        });

        job.applications.push(newApplication._id);
        await job.save();
        return res.status(201).json({
```

```
            message:"Job applied successfully.",
            success:true
        })
    } catch (error) {
        console.log(error);
    }
};
export const getAppliedJobs = async (req,res) => {
    try {
        const userId = req.id;
        const application = await
Application.find({applicant:userId}).sort({createdAt:-1}).populate({
            path:'job',
            options:{sort:{createdAt:-1}},
            populate:{
                path:'company',
                options:{sort:{createdAt:-1}},
            }
        });
        if(!application){
            return res.status(404).json({
                message:"No Applications",
                success:false
            })
        };
        return res.status(200).json({
            application,
            success:true
        })
    } catch (error) {
        console.log(error);
    }
}
// admin dekhega kitna user ne apply kiya hai
export const getApplicants = async (req,res) => {
    try {
        const jobId = req.params.id;
        const job = await Job.findById(jobId).populate({
            path:'applications',
            options:{sort:{createdAt:-1}},
            populate:{
```

```
                path:'applicant'
            }
        });
        if(!job){
            return res.status(404).json({
                message:'Job not found.',
                success:false
            })
        };
        return res.status(200).json({
            job,
            succees:true
        });
    } catch (error) {
        console.log(error);
    }
}
export const updateStatus = async (req,res) => {
    try {
        const {status} = req.body;
        const applicationId = req.params.id;
        if(!status){
            return res.status(400).json({
                message:'status is required',
                success:false
            })
        };

        // find the application by applicantion id
        const application = await
Application.findOne({_id:applicationId});
        if(!application){
            return res.status(404).json({
                message:"Application not found.",
                success:false
            })
        };

        // update the status
        application.status = status.toLowerCase();
        await application.save();
```

```
        return res.status(200).json({
            message:"Status updated successfully.",
            success:true
        });

    } catch (error) {
        console.log(error);
    }
}
```

company.controller.js

```
import { Company } from "../models/company.model.js";
import getDataUri from "../utils/datauri.js";
import cloudinary from "../utils/cloudinary.js";

export const registerCompany = async (req, res) => {
    try {
        const { companyName } = req.body;
        if (!companyName) {
            return res.status(400).json({
                message: "Company name is required.",
                success: false
            });
        }
        let company = await Company.findOne({ name: companyName });
        if (company) {
            return res.status(400).json({
                message: "You can't register same company.",
                success: false
            })
        };
        company = await Company.create({
            name: companyName,
            userId: req.id
        });

        return res.status(201).json({
            message: "Company registered successfully.",
            company,
            success: true
        })
    } catch (error) {
        console.log(error);
```

```
    }
}
export const getCompany = async (req, res) => {
    try {
        const userId = req.id; // logged in user id
        const companies = await Company.find({ userId });
        if (!companies) {
            return res.status(404).json({
                message: "Companies not found.",
                success: false
            })
        }
        return res.status(200).json({
            companies,
            success:true
        })
    } catch (error) {
        console.log(error);
    }
}
// get company by id
export const getCompanyById = async (req, res) => {
    try {
        const companyId = req.params.id;
        const company = await Company.findById(companyId);
        if (!company) {
            return res.status(404).json({
                message: "Company not found.",
                success: false
            })
        }
        return res.status(200).json({
            company,
            success: true
        })
    } catch (error) {
        console.log(error);
    }
```

```
}
export const updateCompany = async (req, res) => {
    try {
        const { name, description, website, location } = req.body;

        const file = req.file;
        // idhar cloudinary ayega
        const fileUri = getDataUri(file);
        const cloudResponse = await
cloudinary.uploader.upload(fileUri.content);
        const logo = cloudResponse.secure_url;

        const updateData = { name, description, website, location, logo
};

        const company = await Company.findByIdAndUpdate(req.params.id,
updateData, { new: true });

        if (!company) {
            return res.status(404).json({
                message: "Company not found.",
                success: false
            })
        }
        return res.status(200).json({
            message:"Company information updated.",
            success:true
        })

    } catch (error) {
        console.log(error);
    }
}
```

job.controller.js

```
import { Job } from "../models/job.model.js";

// admin post krega job
export const postJob = async (req, res) => {
    try {
        const { title, description, requirements, salary, location,
jobType, experience, position, companyId } = req.body;
        const userId = req.id;
```

```
        if (!title || !description || !requirements || !salary ||
!location || !jobType || !experience || !position || !companyId) {
            return res.status(400).json({
                message: "Somethin is missing.",
                success: false
            })
        };
        const job = await Job.create({
            title,
            description,
            requirements: requirements.split(","),
            salary: Number(salary),
            location,
            jobType,
            experienceLevel: experience,
            position,
            company: companyId,
            created_by: userId
        });
        return res.status(201).json({
            message: "New job created successfully.",
            job,
            success: true
        });
    } catch (error) {
        console.log(error);
    }
}
// student k liye
export const getAllJobs = async (req, res) => {
    try {
        const keyword = req.query.keyword || "";
        const query = {
            $or: [
                { title: { $regex: keyword, $options: "i" } },
                { description: { $regex: keyword, $options: "i" } },
            ]
        };
        const jobs = await Job.find(query).populate({
            path: "company"
        }).sort({ createdAt: -1 });
        if (!jobs) {
```

```
            return res.status(404).json({
                message: "Jobs not found.",
                success: false
            })
        };
        return res.status(200).json({
            jobs,
            success: true
        })
    } catch (error) {
        console.log(error);
    }
}
// student
export const getJobById = async (req, res) => {
    try {
        const jobId = req.params.id;
        const job = await Job.findById(jobId).populate({
            path:"applications"
        });
        if (!job) {
            return res.status(404).json({
                message: "Jobs not found.",
                success: false
            })
        };
        return res.status(200).json({ job, success: true });
    } catch (error) {
        console.log(error);
    }
}
// admin kitne job create kra hai abhi tk
export const getAdminJobs = async (req, res) => {
    try {
        const adminId = req.id;
        const jobs = await Job.find({ created_by: adminId }).populate({
            path:'company',
            createdAt:-1
        });
        if (!jobs) {
            return res.status(404).json({
                message: "Jobs not found.",
                success: false
```

```
            })
        };
        return res.status(200).json({
            jobs,
            success: true
        })
    } catch (error) {
        console.log(error);
    }
}
```

user.controller.js

```
import { User } from "../models/user.model.js";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
import getDataUri from "../utils/datauri.js";
import cloudinary from "../utils/cloudinary.js";

export const register = async (req, res) => {
    try {
        const { fullname, email, phoneNumber, password, role } =
req.body;

        if (!fullname || !email || !phoneNumber || !password || !role) {
            return res.status(400).json({
                message: "Something is missing",
                success: false
            });
        };
        const file = req.file;
        const fileUri = getDataUri(file);
        const cloudResponse = await
cloudinary.uploader.upload(fileUri.content);

        const user = await User.findOne({ email });
        if (user) {
            return res.status(400).json({
                message: 'User already exist with this email.',
                success: false,
            })
        }
        const hashedPassword = await bcrypt.hash(password, 10);
```

```
        await User.create({
            fullname,
            email,
            phoneNumber,
            password: hashedPassword,
            role,
            profile:{
                profilePhoto:cloudResponse.secure_url,
            }
        });

        return res.status(201).json({
            message: "Account created successfully.",
            success: true
        });
    } catch (error) {
        console.log(error);
    }
}
export const login = async (req, res) => {
    try {
        const { email, password, role } = req.body;

        if (!email || !password || !role) {
            return res.status(400).json({
                message: "Something is missing",
                success: false
            });
        };
        let user = await User.findOne({ email });
        if (!user) {
            return res.status(400).json({
                message: "Incorrect email or password.",
                success: false,
            })
        }
        const isPasswordMatch = await bcrypt.compare(password,
user.password);
        if (!isPasswordMatch) {
            return res.status(400).json({
                message: "Incorrect email or password.",
                success: false,
```

```
            })
        };
        // check role is correct or not
        if (role !== user.role) {
            return res.status(400).json({
                message: "Account doesn't exist with current role.",
                success: false
            })
        };

        const tokenData = {
            userId: user._id
        }
        const token = await jwt.sign(tokenData, process.env.SECRET_KEY, {
expiresIn: '1d' });

        user = {
            _id: user._id,
            fullname: user.fullname,
            email: user.email,
            phoneNumber: user.phoneNumber,
            role: user.role,
            profile: user.profile
        }

        return res.status(200).cookie("token", token, { maxAge: 1 * 24 *
60 * 60 * 1000, httpsOnly: true, sameSite: 'strict' }).json({
            message: `Welcome back ${user.fullname}`,
            user,
            success: true
        })
    } catch (error) {
        console.log(error);
    }
}
export const logout = async (req, res) => {
    try {
        return res.status(200).cookie("token", "", { maxAge: 0 }).json({
            message: "Logged out successfully.",
            success: true
        })
    } catch (error) {
        console.log(error);
```

```
    }
}
export const updateProfile = async (req, res) => {
    try {
        const { fullname, email, phoneNumber, bio, skills } = req.body;

        const file = req.file;
        // cloudinary ayega idhar
        const fileUri = getDataUri(file);
        const cloudResponse = await
cloudinary.uploader.upload(fileUri.content);



        let skillsArray;
        if(skills){
            skillsArray = skills.split(",");
        }
        const userId = req.id; // middleware authentication
        let user = await User.findById(userId);

        if (!user) {
            return res.status(400).json({
                message: "User not found.",
                success: false
            })
        }
        // updating data
        if(fullname) user.fullname = fullname
        if(email) user.email = email
        if(phoneNumber)  user.phoneNumber = phoneNumber
        if(bio) user.profile.bio = bio
        if(skills) user.profile.skills = skillsArray

        // resume comes later here...
        if(cloudResponse){
            user.profile.resume = cloudResponse.secure_url // save the
cloudinary url
            user.profile.resumeOriginalName = file.originalname // Save
the original file name
        }


        await user.save();
```

```
        user = {
            _id: user._id,
            fullname: user.fullname,
            email: user.email,
            phoneNumber: user.phoneNumber,
            role: user.role,
            profile: user.profile
        }

        return res.status(200).json({
            message:"Profile updated successfully.",
            user,
            success:true
        })
    } catch (error) {
        console.log(error);
    }
}
```

isAuthenticated.js

```
import jwt from "jsonwebtoken";

const isAuthenticated = async (req, res, next) => {
    try {
        const token = req.cookies.token;
        if (!token) {
            return res.status(401).json({
                message: "User not authenticated",
                success: false,
            })
        }
        const decode = await jwt.verify(token, process.env.SECRET_KEY);
        if(!decode){
            return res.status(401).json({
                message:"Invalid token",
                success:false
            })
        };
        req.id = decode.userId;
        next();
    } catch (error) {
        console.log(error);
```

```
    }
}
export default isAuthenticated;
```

mutler.js

```
import multer from "multer";

const storage = multer.memoryStorage();
export const singleUpload = multer({storage}).single("file");
```

application.model.js

```
import mongoose from "mongoose";

const applicationSchema = new mongoose.Schema({
    job:{
        type:mongoose.Schema.Types.ObjectId,
        ref:'Job',
        required:true
    },
    applicant:{
        type:mongoose.Schema.Types.ObjectId,
        ref:'User',
        required:true
    },
    status:{
        type:String,
        enum:['pending', 'accepted', 'rejected'],
        default:'pending'
    }
},{timestamps:true});
export const Application  = mongoose.model("Application",
applicationSchema);
```

company.model.js

```
import mongoose from "mongoose";

const companySchema = new mongoose.Schema({
    name:{
        type:String,
        required:true,
        unique:true
    },
    description:{
```

```
        type:String,
    },
    website:{
        type:String
    },
    location:{
        type:String
    },
    logo:{
        type:String // URL to company logo
    },
    userId:{
        type:mongoose.Schema.Types.ObjectId,
        ref:'User',
        required:true
    }
},{timestamps:true})
export const Company = mongoose.model("Company", companySchema);
```

job.model.js

```
import mongoose from "mongoose";

const jobSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true
    },
    description: {
        type: String,
        required: true
    },
    requirements: [{
        type: String
    }],
    salary: {
        type: Number,
        required: true
    },
    experienceLevel:{
        type:Number,
        required:true,
    },
    location: {
```

```
        type: String,
        required: true
    },
    jobType: {
        type: String,
        required: true
    },
    position: {
        type: Number,
        required: true
    },
    company: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Company',
        required: true
    },
    created_by: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true
    },
    applications: [
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Application',
        }
    ]
},{timestamps:true});
export const Job = mongoose.model("Job", jobSchema);
```

user.model.js

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
    fullname: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
```

```
    phoneNumber: {
        type: Number,
        required: true
    },
    password:{
        type:String,
        required:true,
    },
    role:{
        type:String,
        enum:['student','recruiter'],
        required:true
    },
    profile:{
        bio:{type:String},
        skills:[{type:String}],
        resume:{type:String}, // URL to resume file
        resumeOriginalName:{type:String},
        company:{type:mongoose.Schema.Types.ObjectId, ref:'Company'},
        profilePhoto:{
            type:String,
            default:""
        }
    },
},{timestamps:true});
export const User = mongoose.model('User', userSchema);
```

application.route.js

```
import express from "express";
import isAuthenticated from "../middlewares/isAuthenticated.js";
import { applyJob, getApplicants, getAppliedJobs, updateStatus } from
"../controllers/application.controller.js";

const router = express.Router();

router.route("/apply/:id").get(isAuthenticated, applyJob);
router.route("/get").get(isAuthenticated, getAppliedJobs);
router.route("/:id/applicants").get(isAuthenticated, getApplicants);
router.route("/status/:id/update").post(isAuthenticated, updateStatus);
```

```
export default router;
```

company.route.js

```
import express from "express";
import isAuthenticated from "../middlewares/isAuthenticated.js";
import { getCompany, getCompanyById, registerCompany, updateCompany }
from "../controllers/company.controller.js";
import { singleUpload } from "../middlewares/mutler.js";

const router = express.Router();

router.route("/register").post(isAuthenticated,registerCompany);
router.route("/get").get(isAuthenticated,getCompany);
router.route("/get/:id").get(isAuthenticated,getCompanyById);
router.route("/update/:id").put(isAuthenticated,singleUpload,
updateCompany);

export default router;
```

job.route.js

```
import express from "express";
import isAuthenticated from "../middlewares/isAuthenticated.js";
import { getAdminJobs, getAllJobs, getJobById, postJob } from
"../controllers/job.controller.js";

const router = express.Router();

router.route("/post").post(isAuthenticated, postJob);
router.route("/get").get(isAuthenticated, getAllJobs);
router.route("/getadminjobs").get(isAuthenticated, getAdminJobs);
router.route("/get/:id").get(isAuthenticated, getJobById);

export default router;
```

user.route.js

```
import express from "express";
import { login, logout, register, updateProfile } from
"../controllers/user.controller.js";
import isAuthenticated from "../middlewares/isAuthenticated.js";
import { singleUpload } from "../middlewares/mutler.js";

const router = express.Router();

router.route("/register").post(singleUpload,register);
router.route("/login").post(login);
router.route("/logout").get(logout);
router.route("/profile/update").post(isAuthenticated,singleUpload,updateP
rofile);

export default router;
```

cloudinary.js

```
import {v2 as cloudinary} from "cloudinary";
import dotenv from "dotenv";
dotenv.config();

cloudinary.config({
    cloud_name:process.env.CLOUD_NAME,
    api_key:process.env.API_KEY,
    api_secret:process.env.API_SECRET
});
export default cloudinary;
```

datauri.js

```
import DataUriParser from "datauri/parser.js"

import path from "path";

const getDataUri = (file) => {
    const parser = new DataUriParser();
    const extName = path.extname(file.originalname).toString();
    return parser.format(extName, file.buffer);
}

export default getDataUri;
```

db.js

```
import mongoose from "mongoose";

    const connectDB = async () => {
        try {
            await mongoose.connect(process.env.MONGO_URI);
            console.log('mongodb connected successfully');
        } catch (error) {
            console.log(error);
        }
    }
    export default connectDB;
```

package.json

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
"scripts": {
  "dev": "nodemon index.js",
  "build": "echo 'No build step required for plain Node.js'",
  "start": "node index.js",
  "test": "echo \"Error: no test specified\" && exit 1"
},

  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cloudinary": "^2.3.0",
    "cookie-parser": "^1.4.6",
    "cors": "^2.8.5",
    "datauri": "^4.1.0",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.4.1",
    "multer": "^1.4.5-lts.1",
    "nodemon": "^3.1.3"
  },
  "devDependencies": {
```

```
    "@types/bcryptjs": "^2.4.6",
    "@types/cookie-parser": "^1.4.7",
    "@types/express": "^4.17.21",
    "@types/jsonwebtoken": "^9.0.6"
  }
}
```