Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Рейтаровського Олександра Юрійовича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

«Вебзастосунок для організації особистих та командних завдань»
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101086.01.13.ПЗ

Виконав студент IV курсу, група ІПЗ-21-1 Підпис Олександр РЕЙТАРОВСЬКИЙ
Ім'я, ПРІЗВИЩЕ

Керівник асистент
Науковий ступінь, звання
Підпис В'ячеслав БОЙКО
Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент
Підпис Юрій ФОРКУН
Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення
Підпис Леонід БЕДРАТЮК
Ім'я, ПРІЗВИЩЕ

10 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02. 01. 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Рейтаровському Олександру Юрійовичу
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Вебзастосунок для організаціх особистих та командних завдань

Керівник кваліфікаційної роботи Бойко В'ячеслав Олександрович, асистент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23
2. Строк подання студентом роботи на кафедру 01.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проєктування програмного забезпечення, програмна реалізація гри, тестування
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 18 шт.), Контекстна діаграма вебзастосунку, Діаграма декомпозиції 1 рівня, діаграма варіантів використання вебзастосунку, ER-діаграма бази даних

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийн
Нормоконтроль	Форкун Ю. В., канд. техн. наук, доцент	18.04.2025	06.06.2025
Антиплагіат	Форкун Ю. В., канд. техн. наук, доцент	12.01.2025	09.06.2025

7. Дата видачі завдання «02» 01 2025р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 – 31.12.2024	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2025	
3 Проєктування програмного забезпечення	21.02 – 20.03.2025	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2025	
6 Попередній захист КвР	Травень 2025	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2025	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2025	

Студент _____ Рейтаровський О. Ю.
Підпис Ініціали, прізвище

Керівник роботи _____ Бойко В. О.
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Вебзастосунок для організації особистих та комадних завдань».

Автор роботи: Рейтаровський Олександр Юрійович.

Керівник роботи: Бойко В'ячеслав Олександрович.

Пояснювальна записка: 75 с., 31 рис., 3 табл., 2 дод., 27 джерела.

Графічна частина: 18 слайдів.

Метою кваліфікаційної роботи є створення сучасного, зручного у використанні вебзастосунка для організації особистих і командних завдань, який надаватиме користувачам можливість легко створювати, призначати, редагувати й контролювати виконання завдань, вести командну комунікацію, використовувати інтегрований календар, формувати нотатки, аналізувати прогрес і автоматично фіксувати історію змін.

У кваліфікаційній роботі виконана комплексна розробка вебзастосунка, що включає аналіз предметної області, проєктування програмного забезпечення, програмну реалізацію та тестування.

02.01.2025
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документу	Найменування документу	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2101086.01.13.ПЗ	Пояснювальна записка	75		
2	А4		Завдання на кваліфікацій-	1		
			ну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А4	КвРІПЗ.2101086.01.13.Е8	Діаграма варіантів вико-	1		
			ристання вебзастосунку			
5	А4	КвРІПЗ.2101086.01.13.Е8	ER-діаграма бази даних	1		
6	А4	КвРІПЗ.2101086.01.13.Е8	Контекстна діаграма	1		
			вебзастосунку			
7	А4	КвРІПЗ.2101086.01.13.Е8	Діаграма декомпозиції	1		
			1 рівня вебзастосунку			
8	А4		Презентаційні матеріали	18		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2101086.01.13.ВД		
Виконав		Рейтаровський О.Ю		01.06.25	Вебзастосунок для організації особистих та командних завдань	Літ.	Арк. 1 / Акрушів 1
Керівник		Бобко В.О.		01.06.25			
Н. Контр.		Форкун Ю. В.		01.06.25	Відомість документів	ХНУ. ІПЗ-21-1	
Зав. каф.		Бедратюк Л.П		01.06.25			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

БД – База даних

MVC – Model-View-Controller

СУБД – Система управління базами даних

API – Application Programming Interface, програмний інтерфейс додатків

UI – User Interface, інтерфейс користувача

CRUD – Create, Read, Update, Delete; основні операції над даними

SQL – Structured Query Language, структурована мова запитів

ID – Ідентифікатор

HTML – Мова розмітки, HyperText Markup Language

JS – Java Script

IDEF0 – Integration DEFinition for Function Modeling, метод моделювання функцій системи

IDE – Integrated Development Environment, інтегроване середовище розробки

LINQ – Language Integrated Query, інтегровані запити до даних

ВСТУП

У сучасному світі, який характеризується стрімким розвитком інформаційних технологій та дедалі глибшим проникненням цифрових інструментів у всі сфери життєдіяльності людини, питання ефективної організації робочого часу, управління особистими та командними завданнями набувають особливої актуальності. Постійне зростання обсягу інформації, кількості проєктів, різноманітних завдань і дедлайнів вимагає від людини високого рівня самоорганізації, а від команд – чіткої координації, прозорої комунікації та раціонального розподілу ролей і відповідальностей.

В епоху цифрової трансформації дедалі більшу популярність набувають вебзастосунки, які забезпечують користувачам доступ до інструментів для управління особистими і колективними завданнями, проєктами та подіями незалежно від місця перебування. Від простих списків справ до складних корпоративних систем управління проєктами – подібні рішення стають невід'ємною частиною щоденної діяльності людей, сприяють підвищенню їхньої продуктивності та зниженню рівня стресу.

Актуальність теми даної бакалаврської роботи зумовлена тим, що в умовах сучасного ринку праці, де все частіше зустрічаються віддалені та гібридні форми роботи, постає гостра потреба у зручних і гнучких інструментах для організації командної та особистої роботи.

Таким чином, розробка вебзастосунку для організації особистих і командних завдань є надзвичайно актуальним завданням, спрямованим на підвищення ефективності управління часом та ресурсами, покращення командної взаємодії, формування сучасної цифрової культури роботи й особистого розвитку.

Об'єктом дослідження є вебзастосунок для організації особистих та командних завдань, а також процеси управління цими завданнями в індивідуальній та командній діяльності.

Предметом дослідження є методи, інструменти й технології розробки багатофункціонального вебзастосунку для планування, організації та контролю особистих і командних завдань, а також особливості побудови інтерфейсу користувача, інтеграції модулів комунікації, календаря, журналу дій тощо.

Метою бакалаврської роботи є створення сучасного, зручного у використанні вебзастосунка для організації особистих і командних завдань, який надаватиме користувачам можливість легко створювати, призначати, редагувати й контролювати виконання завдань, вести командну комунікацію, використовувати інтегрований календар, формувати нотатки, аналізувати прогрес і автоматично фіксувати історію змін. Застосунок має забезпечити багаторівневий доступ, підтримку ролей, інтуїтивний інтерфейс і масштабовану архітектуру.

Основними завданнями даної роботи є аналіз існуючих рішень для організації завдань та командної роботи, визначення функціональних і нефункціональних вимог до системи, обґрунтування вибору архітектури та технологічного стеку, розробка структури даних, моделей, ролей та взаємодії користувачів, реалізація модулів для управління проєктами, завданнями, календарем, повідомленнями, нагадуваннями та історією дій, проведення тестування розробленого продукту, а також оцінка результатів і перспектив розвитку системи.

Запропонований вебзастосунок поєднує в собі простоту використання з широкими можливостями для планування, делегування, контролю і аналізу завдань як у команді, так і особисто. Система забезпечує інтеграцію ключових модулів, серед яких керування проєктами, завданнями, ролями, чат/повідомлення, календар, історія дій, а також розмежування доступу між особистими та командними проєктами. Впроваджено сучасний підхід до дизайну та користувацького досвіду, що повністю відповідає актуальним вимогам цифрової культури.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Структура предметної області характеризується наявністю двох основних напрямків організації діяльності – індивідуальної (особисті завдання, особисті проєкти) та колективної (командні проєкти, спільні завдання, командна комунікація). У межах кожного напрямку існують свої типи процесів, вимоги до організації даних та особливості взаємодії між користувачами.

Структурні особливості предметної області полягають у чітко визначених основних об’єктах і зв’язках між ними. Центральним елементом системи є користувачі – це фізичні особи, які мають індивідуальний обліковий запис та взаємодіють із системою. Користувачі можуть виконувати як особисті завдання, так і брати участь у командних проєктах, виступаючи членами команди.

Кожен проєкт є логічним об’єднанням завдань і передбачає наявність власника, списку учасників, опису, календаря подій та структурованого переліку задач. Проєкти можуть бути як особистими, так і командними, що обумовлює різний рівень доступу до інформації та організації спільної роботи.

Завдання виступають базовими одиницями роботи, для яких характерними є назва, опис, статус виконання, дедлайни, а також можливість призначення відповідальних користувачів. Це дозволяє організовувати процеси виконання задач більш ефективно та прозоро.

Окрему роль відіграють ролі та рівні доступу, які визначають повноваження користувачів у межах певного проєкту. Завдяки цьому можна гнучко розподіляти обов’язки, обмежувати або розширювати доступ до функціоналу, встановлювати права для адміністраторів, учасників чи гостей.

Для підтримки внутрішньої комунікації впроваджено функціонал повідомлень, загального та приватного чату, що дозволяє обговорювати поточні

питання як у загальних, так і в персональних чатах між учасниками проєкту. Крім цього, календар і система нагадувань забезпечують візуалізацію планування завдань чи подій, дедлайнів, а також автоматичне інформування користувачів про зміни чи наближення важливих термінів.

Журнал дій та історія змін фіксують основні події, операції із завданнями, зміну статусів і дії користувачів, забезпечуючи прозорість виконання процесів і можливість відслідковування історії роботи в системі.

Значна частина предметної області пов’язана з автоматизацією процесів планування та управління завданнями. Автоматизація дозволяє не лише зменшити вплив людського фактора, а й суттєво підвищити продуктивність, знизити ризик пропущених термінів, забезпечити своєчасну комунікацію та облік дій у системі. Використання інформаційних технологій також дозволяє організувати централізоване зберігання й обробку даних, створювати гнучкі моделі доступу до інформації залежно від ролей користувачів, оперативно реагувати на зміни в команді або у вимогах до проєкту.

Попри велику кількість сучасних програмних рішень, у цій предметній області досі існують ряд невирішених проблем. Багато сучасних систем виявляються занадто складними для невеликих команд або особистого використання, що суттєво ускладнює їх адаптацію й повсякденну роботу. Досить часто бракує гнучких механізмів для розмежування особистої й командної діяльності, через що користувачі змушені використовувати кілька різних сервісів для різних цілей. Зокрема, для більшості доступних сервісів інтеграція всіх необхідних функцій, таких як управління завданнями, календар, чат для спілкування, журнал дій та інших, у межах одного зручного інтерфейсу залишається складним питанням.

Кінцевими користувачами системи є як окремі особи, що прагнуть організувати власний час та завдання (студенти, фрілансери, працівники офісів), так і цілі команди, котрі взаємодіють у межах спільних проєктів (робочі групи, стартапи, відділи компаній, студентські колективи тощо).

Функціональні особливості предметної області визначаються потребами користувачів у:

- швидкому та зручному створенні, редагуванні та видаленні завдань;
- можливості управління власними та командними проєктами;
- призначенні завдань іншим учасникам;
- групуванні завдань за проєктами, статусами, пріоритетами;
- можливості комунікації з іншими учасниками;
- можливості фіксації історії дій для аналізу та звітності;
- в безпеці збереження персональних даних;
- в засобах для аналізу виконаних завдань та статистики роботи.

Висновком аналізу предметної області є необхідність розробки комплексного вебзастосунку, який поєднуватиме простоту особистих сервісів та гнучкість командних систем, інтегруватиме інструменти для планування, спілкування й аналізу, а також забезпечуватиме безпеку та зручність для кінцевого користувача.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

У сфері організації особистих і командних завдань, планування проєктів та співпраці існує значна кількість програмних продуктів, що забезпечують різноманітний функціонал для користувачів. Нижче наведено аналіз найбільш популярних і релевантних до теми бакалаврської роботи аналогічних систем, їх призначення, функціональні можливості, а також основні переваги та недоліки з позиції кінцевого користувача.

На сучасному ринку інформаційних технологій існує велика кількість програмних продуктів, що призначені для організації особистих і командних завдань, управління проєктами, планування часу, а також комунікації в команді. До найпопулярніших рішень у цій сфері належать такі системи, як Trello, Jira, Asana, Notion, ClickUp, Microsoft To Do, Todoist та інші. Кожен із цих продуктів має свої

особливості, функціональні можливості та призначений для певної цільової аудиторії.

Trello є одним із найвідоміших інструментів для візуального управління завданнями на основі Kanban-дошок. Його перевагами є простий, інтуїтивно зрозумілий інтерфейс, легкість у налаштуванні під різні потреби, можливість гнучкого керування завданнями та списками. До недоліків Trello можна віднести обмеженість у глибокій структуризації проєктів, недостатню інтеграцію з календарями та відсутність вбудованих засобів для аналітики й управління ролями користувачів.

Jira широко використовується у сфері розробки програмного забезпечення, особливо у великих командах. Вона пропонує потужний функціонал для управління завданнями, баг-трекінгу, гнучких методологій розробки (Scrum, Kanban), аналітики й звітності. Проте Jira має складний інтерфейс, який може бути надмірно перевантаженим для новачків або невеликих команд, а налаштування й підтримка вимагають додаткових знань та часу.

Asana є багатофункціональним сервісом для управління проєктами, що дозволяє створювати завдання, підзадачі, групувати їх за проєктами, відслідковувати прогрес виконання та вести комунікацію всередині завдання. Переваги Asana – це інтуїтивний інтерфейс, гнучкість структурування завдань і гарна інтеграція з іншими сервісами. Водночас, для повноцінного використання багатьох функцій потрібна платна підписка, а функціонал нагадувань і календарів може бути обмеженим у безкоштовній версії.

Notion позиціонується як універсальна платформа для роботи з інформацією, організації особистих та командних робочих процесів, створення баз знань, таблиць, нотаток тощо. Серед її переваг – гнучкість налаштування, багатство форматів даних, можливість створення власних шаблонів. Однак через таку універсальність інтерфейс Notion може здатися перевантаженим, а навчання користувачів вимагає часу. Деяким командам бракує спеціалізованих інструментів для управління завданнями у класичному вигляді.

ClickUp – сучасний хмарний сервіс, що поєднує управління завданнями, документи, цілі, нагадування, чат і аналітику. Його основні переваги – комплексність, можливість налаштування під різні процеси, розвинений модуль аналітики та звітності. Проте багатофункціональність призводить до ускладнення інтерфейсу та появи зайвих для частини користувачів опцій. Також система потребує стабільного інтернет-з'єднання.

Microsoft To Do і Todoist більше орієнтовані на особисте використання, дозволяють швидко створювати списки справ, додавати нагадування, але мають обмежений командний функціонал і невелику кількість інтеграцій.

Загальні переваги існуючого ПЗ:

– велика кількість доступних сервісів для різних потреб;
– багатофункціональність окремих рішень;
– хмарний доступ із різних пристроїв;
– високий рівень надійності та безпеки в популярних системах;
– можливість інтеграції з іншими корпоративними сервісами.

Недоліки існуючого ПЗ:

– недостатня гнучкість для поєднання особистої й командної роботи в одному середовищі;
– обмеження у налаштуванні під специфічні вимоги невеликих команд чи окремих користувачів;
– складність для початківців через перевантаженість функціоналом;
– відсутність локалізації для частини продуктів;
– обмеження безкоштовних версій, висока вартість розширених тарифів;
– недостатня інтеграція між модулями календаря, чату, журналу дій тощо;
– проблеми із захистом персональних даних у деяких сервісах.

Для візуального порівняння основних аналогів можна навести наступну таблицю:

Таблиця 1.1 – Порівняння наявних ПЗ

Продукти	Переваги	Недоліки
Trello	Простота, швидкий старт, інтеграції, легке налаштування	Обмежений функціонал безкошнтовної версії
Jira	Потужний функціонал, звітність, аналітика, гнучкість, інтеграції, журнал дій	Складний інтерфейс, потребує налаштування, дорого, надлишковість функцій
Asana	Зручний інтерфейс, інтеграції, дедлайни, підзадачі, сповіщення	Багато функцій лише платно, обмеження для безкоштовних
Notion	Гнучкість, універсальність, кастомізація	Відсутність чатів, поріг входження
Click Up	Універсальність, гнучкість, аналітика, інтеграції, налаштування під свої процеси	Перевантаженість інтерфейсу, зайве для малих команд, платні модулі
Microsoft To Do	Доступність, простота, інтеграція з Microsoft, безкоштовність, мобільність	Лише персональне використання, мало аналітики, немає журналу дій, простий функціонал
Todoist	Легкість, нагадування, інтеграції, пріоритети, безкоштовна версія	Обмеження для командної роботи, частина функцій лише платно

На основі проведеного аналізу можна зробити висновок, що жоден із існуючих продуктів повною мірою не охоплює поєднання простоти особистого використання, повноцінного модулю командної взаємодії, інтегрованого календаря, аналітики, зручної роботи з ролями та історією дій. Більшість популярних сервісів або обмежені у безкоштовних версіях, або мають складний інтерфейс та високу вартість, або не містять деяких важливих для цільової

аудиторії функцій (наприклад, повноцінного внутрішнього чату).

Саме ці проблеми й стали підставою для створення власного вебзастосунку, що поєднує переваги вищезазначених рішень та уникає їхніх недоліків, орієнтуючись на максимальну зручність, гнучкість та інтегрованість у робочий процес як окремих користувачів, так і команд.

1.3 Визначення функціональних та нефункціональних вимог до ПЗ

Визначення вимог до програмного забезпечення є одним із ключових етапів його розробки, оскільки саме на цій стадії формується уявлення про майбутній функціонал системи, її можливості, обмеження, а також якісні характеристики, яким повинна відповідати розроблювана система. Усі вимоги до вебзастосунку для організації особистих і командних завдань доцільно розділити на функціональні та нефункціональні.

Для кращого візуального сприйняття відмінностей між функціональними та нефункціональними вимогами до програмного забезпечення, на рисунку 1 наведено порівняльну ілюстрацію.

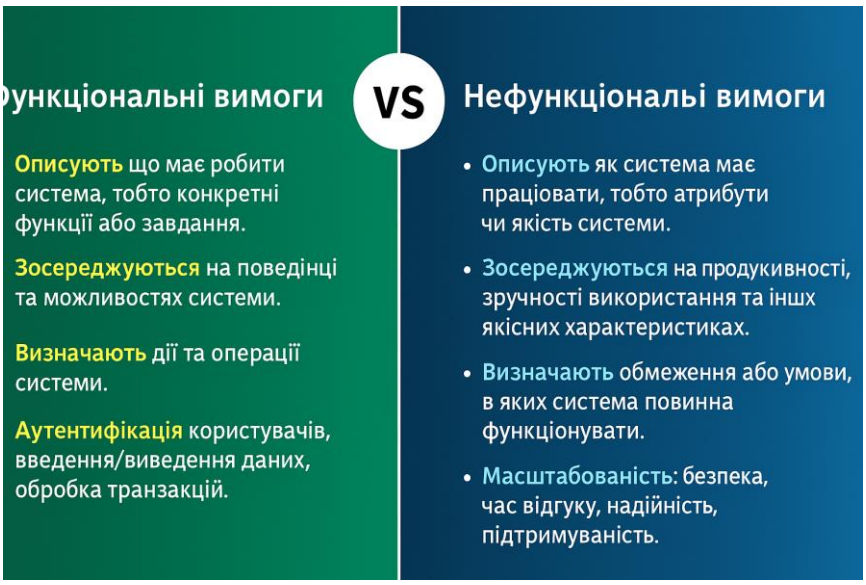


Рисунок 1 – порівняльна ілюстрація вимог [1]

Для ефективного збору, аналізу й подальшого моделювання функціональних вимог доцільно застосовувати сучасні методи та візуальні засоби формалізації, серед яких особливе місце займає мова уніфікованого моделювання UML (Unified Modeling Language). UML-діаграми дозволяють наочно відобразити основні функції, сценарії роботи користувачів із системою та взаємозв'язки між компонентами програмного забезпечення [1].

У першу чергу, для моделювання вимог широко використовуються діаграми варіантів використання (Use Case diagrams), які дозволяють сформувати уявлення про основні функції майбутньої системи, описати взаємодію користувачів із нею, а також встановити межі та контекст роботи програмного забезпечення [2]. Перед побудовою діаграм варіантів використання доцільно скласти короткі описи користувачів (акторів) системи та функцій, які вони виконують, що дає змогу структуровано підійти до визначення функціональних можливостей ПЗ та їх подальшого візуального представлення (див. табл. 1.2).

Таблиця 1.2 – Опис користувачів (Акторів)

Актор	Короткий опис
Власник проекту	Має повний доступ до функціоналу: може створювати і редагувати проєкти, призначати ролі, запрошувати учасників до команди, переглядати журнал дій.
Менеджер	Керує завданнями в межах команди: створює, редагує, призначає відповідальних, планує дедлайни в календарі та веде командну комунікацію.
Виконавець	Працює із завданнями: переглядає завдання, редагує їхній статус, бере участь у комунікації в межах команди.
Гість	Має обмежений доступ до перегляду певних проєктів або завдань (наприклад, тільки читання, без права редагування)

Таблиця 1.3 – Опис варіантів використання

Найменування ВВ	Опис ВВ
Реєстрація та аутентифікація користувачів	Дозволяє новим користувачам створити обліковий запис, а існуючим – увійти до системи із захистом персональних даних.
Керування проектами	Дозволяє користувачу створювати нові проекти, редагувати їхню інформацію та видаляти їх.
Додавання завдання	Дозволяє створити нове завдання, визначити його назву, опис, дедлайн, пріоритетність і пов’язати з певним проєктом.
Редагування завдання	Дозволяє змінювати параметри вже створеного завдання, зокрема опис, статус, терміни виконання, відповідальних користувачів тощо.
Призначення відповідальних користувачів до завдань	Дозволяє вибрати одного або кількох користувачів, відповідальних за виконання конкретного завдання.
Перегляд календаря	Дозволяє користувачам бачити всі заплановані завдання й події у вигляді інтегрованого календаря для зручного планування.
Введення комунікації	Дозволяє користувачам спілкуватися в персональному або загальному чаті у межах команди
Журнулювання дій	Дозволяє автоматично фіксувати й переглядати всі важливі дії користувачів у системі
Призначення ролей учасникам команди	Дозволяє адміністратору або власнику проєкту надавати користувачам різні ролі з відповідними правами доступу.

На основі отриманих даних доцільно перейти до побудови діаграми варіантів використання (див. рис. 1.1), яка візуально відобразить зв’язки між різними ролями й основними функціональними можливостями системи, що значно спрощує розуміння архітектури та сценаріїв роботи вебзастосунку.

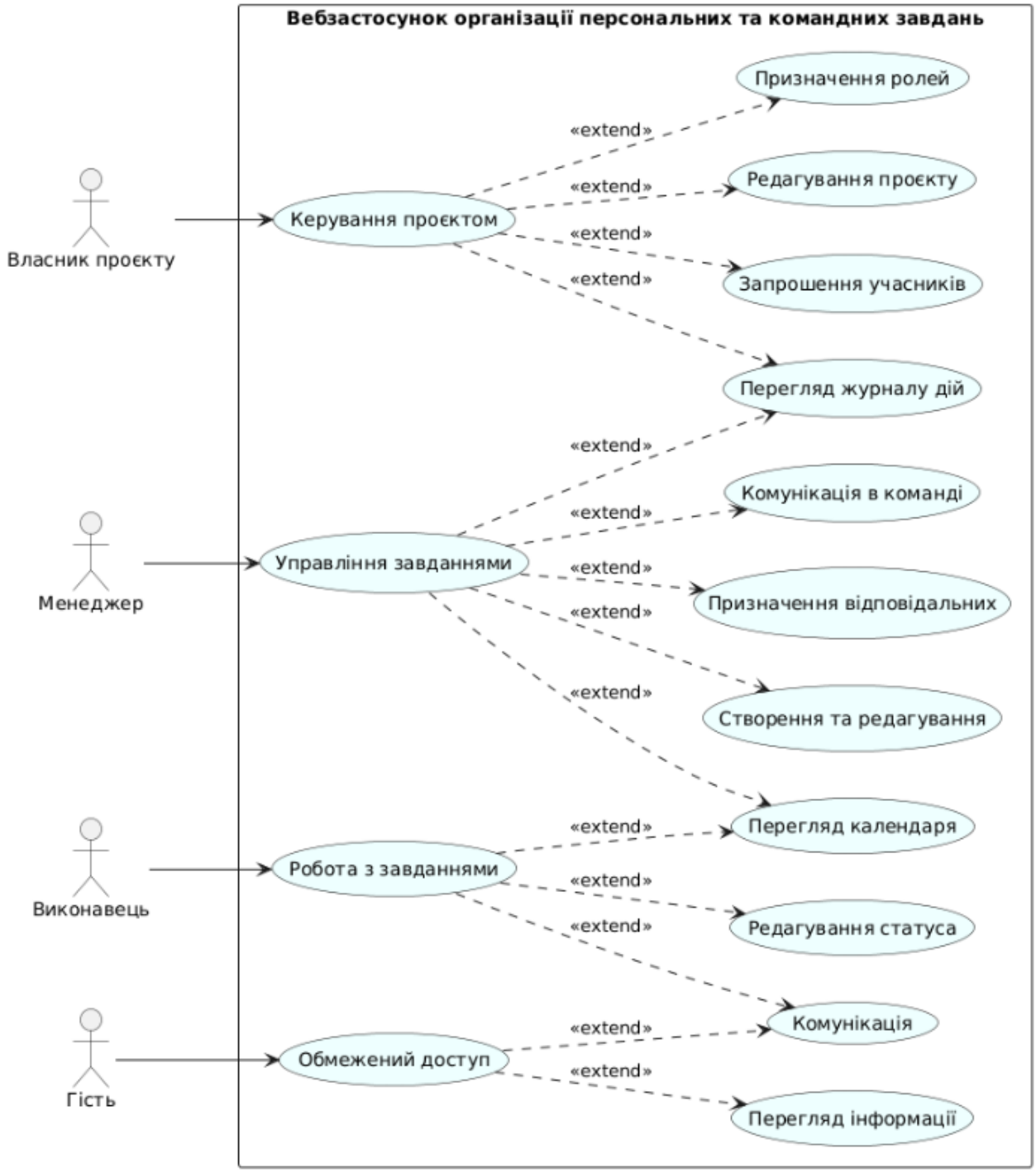


Рисунок 1.1 – діаграма варіантів використання

Окрім функціональних вимог, які визначають, які саме можливості повинна надавати система, надзвичайно важливим є формулювання нефункціональних вимог до програмного забезпечення. Нефункціональні вимоги встановлюють додаткові критерії якості, яким має відповідати система: це такі характеристики, як продуктивність, надійність, безпека, масштабованість, зручність використання, сумісність із різними пристроями та програмними платформами [3]. Саме виконання нефункціональних вимог забезпечує високий рівень комфорту роботи користувачів, стабільність та ефективність функціонування програмного продукту у реальних умовах експлуатації.

До основних нефункціональних вимог, що пред'являються до сучасних інформаційних систем, належить забезпечення високої продуктивності. Система повинна швидко обробляти користувацькі запити, мінімізувати затримки при навігації між різними модулями, а також гарантувати стабільну роботу навіть у періоди підвищеного навантаження.

Не менш важливою є масштабованість, тобто здатність програмного забезпечення коректно функціонувати та обслуговувати зростаючу кількість користувачів і проєктів без погіршення якості роботи. Це дозволяє системі розвиватися разом із потребами організації та не обмежувати її подальше зростання.

Безпека також відіграє ключову роль: повинна бути реалізована надійна система захисту персональних даних користувачів, впроваджене шифрування паролів, а також механізми розмежування прав доступу для різних ролей і категорій користувачів. Особливу увагу слід приділяти захисту інформації від несанкціонованого доступу та можливих кіберзагроз.

Ще однією важливою вимогою є надійність програмного забезпечення. Система повинна працювати безвідмовно, забезпечувати цілісність та збереження даних навіть у разі виникнення критичних збоїв чи аварійних ситуацій. Для цього впроваджуються механізми резервного копіювання даних та відновлення інформації.

Серед нефункціональних вимог не можна оминути й зручність інтерфейсу користувача. Вебзастосунок має бути оснащений сучасним, адаптивним інтерфейсом, що забезпечує інтуїтивну навігацію, легкість використання на різних пристроях – від настільних комп'ютерів до смартфонів і планшетів. Це дозволяє залучати ширше коло користувачів і підвищувати їхню продуктивність.

Також важливою є сумісність, яка передбачає коректну роботу вебзастосунку у сучасних браузерах і на різних операційних системах. Це гарантує доступність сервісу для всіх користувачів незалежно від використовуваного програмного забезпечення чи пристрою.

Загалом, виконання зазначених нефункціональних вимог сприятиме створенню якісного, стабільного та безпечного програмного продукту, який буде здатний успішно функціонувати у реальних умовах експлуатації та відповідати очікуванням кінцевих користувачів.

Щодо інтерфейсу програмного забезпечення, то він має бути сучасним, адаптивним до різних типів пристроїв, забезпечувати інтуїтивну навігацію між основними модулями (проєкти, завдання, календар, повідомлення), дозволяти швидко створювати, редагувати, видаляти завдання, відображати інформацію у вигляді таблиць, списків і календаря. Важливою є можливість пошуку, фільтрації та сортування завдань, а також наявність системи повідомлень про зміни й нагадування для користувачів.

1.4 Постановка задачі

На основі проведеного аналізу предметної області, розгляду існуючих програмних рішень, а також визначених функціональних і нефункціональних вимог, у даній бакалаврській роботі ставиться задача розробки вебзастосунку для організації особистих і командних завдань, який повинен поєднувати в собі простоту використання, багатий функціонал, гнучкість налаштувань та відповідати

сучасним вимогам до цифрових інструментів для продуктивної роботи.

Завдання полягає у створенні комплексного програмного продукту, що дозволить як окремим користувачам, так і командам ефективно планувати, організовувати та контролювати виконання завдань і проєктів, забезпечуючи при цьому високий рівень інтеграції різних модулів. Вебзастосунок повинен надавати інтуїтивно зрозумілий інтерфейс для створення, редагування, перегляду й видалення завдань, їх об’єднання у проєкти, розподілу ролей між учасниками, контролю статусу виконання, а також ведення ефективної командної комунікації. Особливу увагу слід приділити безпековим аспектам, зокрема ідентифікації та аутентифікації користувачів, контролю прав доступу, захисту персональних даних та журналюванню дій у системі.

Задача також включає розробку логічної й фізичної структури даних, моделювання бізнес-процесів, проектування архітектури вебзастосунку з урахуванням можливості подальшого розширення й масштабування системи. Передбачається створення багаторівневої моделі користувачів із гнучкою системою ролей, що дозволить відокремити особисті та командні проєкти, налаштувати індивідуальні сценарії роботи для різних категорій користувачів.

Таким чином, до основних завдань, які необхідно реалізувати у межах системи, належить забезпечення реєстрації та аутентифікації користувачів. Система має підтримувати можливість створення нових облікових записів із перевіркою унікальності імені користувача чи email, а також безпечну аутентифікацію вже зареєстрованих користувачів. Важливо впровадити механізми захисту пароля та підтримку відновлення доступу. Окрему увагу слід приділити реалізації різних рівнів доступу, зокрема адміністратора, учасника чи гостя, для ефективного розмежування прав і можливостей у межах системи.

Ще одним важливим завданням є створення й управління проєктами. Користувачі повинні мати змогу створювати як особисті, так і командні проєкти, зазначаючи їхню назву, опис та інші важливі атрибути. У разі командних проєктів передбачена можливість запрошення інших користувачів, визначення ролей і прав

доступу, формування списку учасників, а також керування членством у проєкті.

Система повинна дозволяти створювати, редагувати й видаляти завдання із зазначенням основних характеристик – назви, опису, дати створення, дедлайну, статусу виконання, а також призначення відповідальних користувачів. Для командних задач має бути підтримка мультивибору виконавців. Важливим є також наявність зручного інтерфейсу для гнучкого управління поточними і завершеними справами.

Необхідно передбачити зручний механізм перегляду та фільтрації завдань. Користувачі повинні мати змогу переглядати як особисті завдання, так і завдання командних проєктів у різних форматах – таблиці чи календаря – з можливістю перемикання між режимами, пошуку та фільтрації за різними критеріями (статус виконання, відповідальний, термін, пріоритет, проєкт).

Для кожного завдання повинна бути реалізована можливість призначення одного або декількох відповідальних користувачів, із подальшою можливістю зміни складу виконавців у процесі роботи над командними проєктами.

Важливо, щоб користувачі могли додавати до завдань персональні нотатки та коментарі, переглядати історію обговорень, фіксувати інструкції чи зворотний зв’язок, що підвищить ефективність командної взаємодії.

Система повинна дозволяти налаштовувати індивідуальні або командні нагадування про наближення дедлайнів, важливі події чи зміни у завданнях і проєктах, надсилаючи сповіщення у вигляді push- чи email-повідомлень.

Для забезпечення прозорості роботи необхідно впровадити журнал дій, який буде фіксувати всі ключові зміни й операції із завданнями та проєктами, зміну ролей, додавання учасників, коментарі тощо, із зазначенням дати, часу, імені користувача та типу змін.

Система має містити модуль командної комунікації, що забезпечує обмін повідомленнями між учасниками проєкту або конкретного завдання, зберігаючи історію взаємодії та дозволяючи швидко узгоджувати дії без необхідності використання сторонніх месенджерів.

Крім того, важливим завданням є планування й інтеграція з календарем, який дозволяє переглядати завдання, події та дедлайни у вигляді інтегрованої календарної сітки з можливістю навігації між датами та додавання завдань безпосередньо з календаря.

Система повинна забезпечувати зручне й безпечне управління профілем користувача, що включає зміну особистих даних, контактної інформації, аватару, налаштувань інтерфейсу, а також можливість змінювати пароль, здійснювати відновлення доступу та налаштовувати мову або регіональні параметри відповідно до індивідуальних потреб користувача. Всі ці операції мають бути інтуїтивно зрозумілими та доступними навіть для недосвідчених користувачів.

Для контролю доступу до функціоналу системи важливо підтримувати різні ролі користувачів із чітко визначеними правами, а також забезпечити можливість оперативної зміни ролей, делегування прав і контролю за тим, хто має доступ до тієї чи іншої інформації. Це особливо актуально для командних проєктів, де важливо гнучко розподіляти обов’язки між учасниками.

Обов’язковим завданням є впровадження сучасних засобів захисту персональних даних, шифрування паролів, захисту від несанкціонованого доступу та регулярного оновлення програмного забезпечення для усунення можливих вразливостей. Особлива увага повинна приділятися відповідності системи актуальним стандартам безпеки, що є критичним для довіри користувачів та стабільності роботи платформи.

Нарешті, застосунок повинен мати адаптивний інтерфейс, який коректно відображається та функціонує на різних типах пристроїв – настільних комп’ютерах, ноутбуках, планшетах і смартфонах – зберігаючи повноцінний набір функцій та зручність користування незалежно від розміру екрану.

Таким чином, постановка задачі включає цілу сукупність функцій і модулів, кожен з яких спрямований на забезпечення максимальної зручності, ефективності й безпеки як для окремих користувачів, так і для командної роботи у сучасному цифровому середовищі.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз та вибір архітектури вебзастосунку

Архітектура програмного забезпечення відіграє ключову роль у забезпеченні його масштабованості, гнучкості, підтримуваності, розширюваності та ефективності функціонування. На сучасному етапі розвитку інформаційних технологій розробники мають у своєму розпорядженні цілий спектр архітектурних підходів, які відрізняються як структурними особливостями, так і можливістю адаптації під специфічні вимоги проєкту [4]. Отже, виконаємо аналіз архітектур.

Монолітна архітектура. Традиційний підхід, при якому всі модулі, компоненти та функціональні частини програмного забезпечення реалізовані у межах єдиного додатку. Основними перевагами є простота розробки, відсутність складної логіки взаємодії між окремими частинами, зручність розгортання та відносно легке тестування на початкових етапах. Проте така архітектура має суттєві недоліки: складність масштабування, ускладнення процесу внесення змін або додавання нових функцій, високий ризик виникнення залежностей між модулями, що негативно впливає на розширюваність продукту [5].

Клієнт-серверна архітектура. Цей підхід базується на чіткому розділенні клієнтської та серверної частин програмного забезпечення. Сервер відповідає за обробку логіки, зберігання та управління даними, тоді як клієнт забезпечує взаємодію користувача із системою. Клієнт-серверна архітектура забезпечує гнучкість масштабування, зручність підтримки, розподіл навантаження, а також дозволяє реалізувати багаторівневий доступ до ресурсів. Основні недоліки – це потенційна складність синхронізації даних, підвищені вимоги до організації захисту даних під час їх передачі між клієнтом і сервером, а також можливі складнощі з підтримкою розподілених транзакцій [6].

Сервісно-орієнтована архітектура (SOA). Ця модель передбачає розділення функціоналу системи на окремі сервіси, які взаємодіють між собою за допомогою

визначених протоколів. Кожен сервіс виконує конкретну задачу і може бути незалежно розроблений, розгорнутий, оновлений чи масштабований. SOA добре підходить для великих корпоративних систем, дозволяє забезпечити високий рівень гнучкості та повторного використання коду, а також спрощує інтеграцію з іншими зовнішніми системами. Недоліки включають ускладнення адміністрування, підвищені вимоги до інфраструктури, а також потенційні проблеми з продуктивністю у разі некоректного проектування сервісів [7].

Мікросервісна архітектура. Останнім часом цей підхід набув особливої популярності завдяки можливості незалежного розвитку, розгортання та масштабування окремих сервісів. Кожен мікросервіс реалізує власний функціонал і взаємодіє з іншими сервісами через стандартизовані інтерфейси (наприклад, REST API). Основними перевагами є гнучкість розробки, полегшення оновлення окремих частин системи, підвищена стійкість до збоїв. Однак для невеликих чи середніх проєктів цей підхід може бути надмірно складним: підвищуються вимоги до організації DevOps, тестування, адміністрування, забезпечення консистентності даних і моніторингу [8].

Враховуючи специфіку поставленої задачі, обсяги системи, прогнозовану кількість користувачів, а також вимоги до гнучкості, масштабованості та зручності підтримки, найбільш доцільним є використання клієнт-серверної архітектури з виділенням окремих рівнів для бізнес-логіки (архітектурний шаблон MVC). Такий підхід дозволяє:

- розмежувати відповідальність між різними частинами застосунку;
- забезпечити легкість супроводу, розширення та масштабування системи;
- впровадити сучасні механізми контролю доступу й аутентифікації;
- забезпечити ефективну взаємодію між користувачем і сервером;
- спростити тестування й інтеграцію нових функціональних модулів.

У сучасній розробці вебзастосунків архітектурний шаблон Model-View-Controller (MVC) посідає особливе місце завдяки своїй гнучкості, структурованості

та зрозумілості для розробників. Його головна ідея полягає у чіткому розділенні відповідальностей між різними компонентами системи, що істотно полегшує розробку, супровід і масштабування програмного продукту. Такий підхід сприяє покращенню якості коду, спрощує тестування та дозволяє одночасно працювати над різними частинами застосунку різним розробникам [9].

Model (Модель) – це рівень, який відповідає за логіку роботи з даними та відображає предметну область застосунку. Модель містить усю бізнес-логіку, інтерпретує, обробляє і зберігає дані, а також відповідає за взаємодію із системою зберігання даних (базою даних). Саме на цьому рівні реалізуються основні правила, перевірки та зв’язки між різними сутностями (наприклад, користувачі, проєкти, завдання, повідомлення). Модель інкапсулює всі дані та механізми їх обробки, забезпечуючи надійний захист і цілісність інформації, а також централізовану логіку роботи з даними.

View (Відображення) – це компонент, що відповідає за презентацію даних користувачу, тобто відображення інформації у зручному для сприйняття вигляді. View отримує дані від моделі та формує інтерфейс, з яким безпосередньо взаємодіє кінцевий користувач. На цьому рівні реалізуються шаблони сторінок, форми, таблиці, списки завдань, календар, елементи керування, а також динамічні елементи UI. Важливо, що View не містить бізнес-логіки – його завдання обмежується лише відображенням і візуалізацією даних, що гарантує чистоту розділення коду.

Controller (Контролер) – це посередник між моделлю та відображенням. Контролер отримує запити від користувача (наприклад, натискання кнопок, надсилання форм), обробляє їх, викликає необхідну логіку у моделі, а потім повертає результат у View для відображення. Саме контролер містить основні сценарії взаємодії користувача із системою: обробку введених даних, маршрутизацію запитів, керування потоком інформації між іншими компонентами. Таким чином, Controller забезпечує логіку поведінки застосунку, відповідає за прийняття рішень і координацію взаємодії між різними частинами системи.

Завдяки чіткому розділенню відповідальностей між Model, View і Controller забезпечується прозорість архітектури, спрощується тестування окремих компонентів, підвищується гнучкість і масштабованість розробки, а також досягається легкість підтримки й подальшого розвитку застосунку. Саме ці переваги стали вирішальними у виборі шаблону MVC для розробки даної системи.

Щоб візуально побачити як взаємодіють між собою компоненти MVC, можна переглянути схему на рисунку 2.1

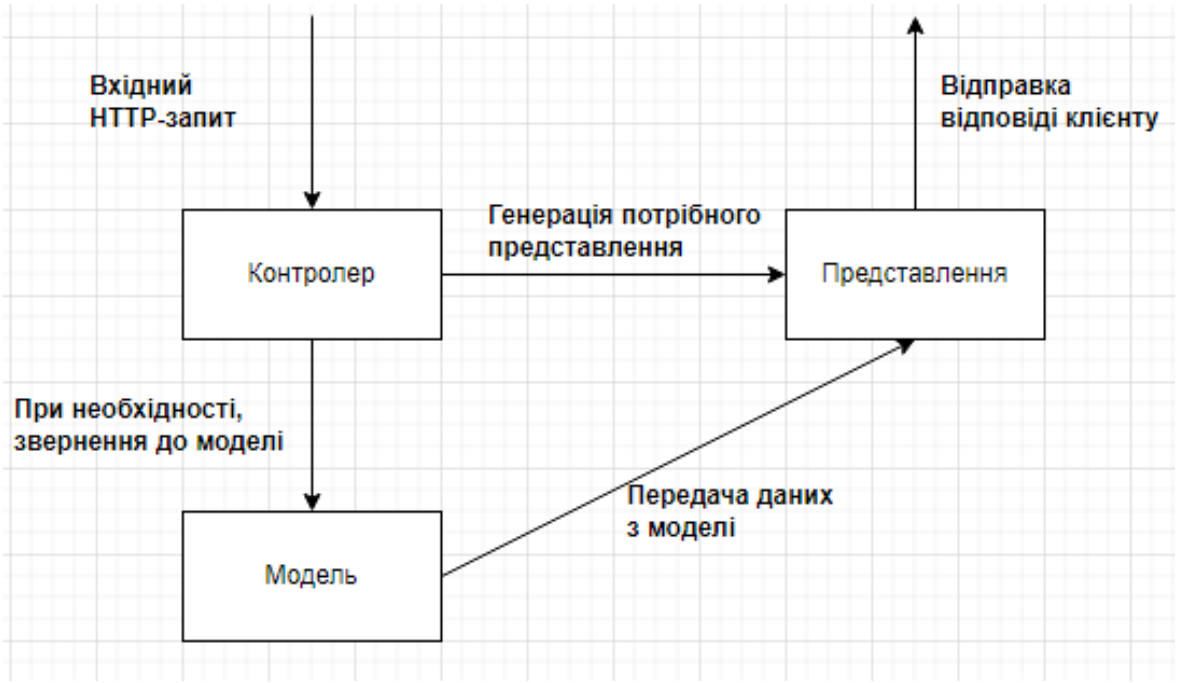


Рисунок 2.1 – архітекстурний шаблон MVC

Підсумовуючи, вибір клієнт-серверної архітектури із впровадженням шаблону MVC та супутніх патернів проектування дозволяє створити ефективний, гнучкий, масштабований і підтримуваний вебзастосунок, що повністю відповідає функціональним і нефункціональним вимогам до розроблюваної системи. Такий підхід забезпечує оптимальний розподіл відповідальностей між компонентами, сприяє спрощенню розробки, полегшує подальшу підтримку й розширення функціоналу, а також дозволяє інтегрувати сучасні технології для підвищення якості та надійності роботи програмного продукту.

2.2 Опис структури даних та моделі бази даних

На сучасному етапі розвитку програмного забезпечення бази даних є невід'ємною складовою більшості інформаційних систем. Вони забезпечують надійне зберігання, організацію та обробку великих обсягів структурованої інформації, дозволяють здійснювати ефективний пошук, фільтрацію, оновлення та аналіз даних у режимі реального часу. Саме завдяки використанню баз даних можна забезпечити цілісність, захищеність і актуальність інформації, а також реалізувати складні бізнес-процеси, які пов'язані з управлінням даними.

У рамках даної бакалаврської роботи для зберігання інформації про користувачів, проєкти, завдання, повідомлення, історію дій та інші об'єкти предметної області використовується реляційна модель бази даних, яка базується на побудові взаємопов'язаних таблиць. Реляційні бази даних характеризуються високим рівнем надійності, підтримкою транзакцій, можливістю використання складних запитів для аналізу та агрегації інформації, а також гнучкістю у моделюванні різноманітних типів зв'язків між сутностями [10].

Для організації процесів особистої й командної роботи у вебзастосунку необхідно визначити основні сутності (таблиці) бази даних та їх взаємозв'язки. Отже, кожен користувач системи має можливість зареєструвати обліковий запис, вказавши особисту інформацію (ім'я, електронна пошта, пароль), і надалі входити до свого профілю. Це передбачає існування сутності Користувач, що зберігає всі необхідні персональні дані, а також параметри для ідентифікації та аутентифікації.

Після реєстрації користувач може створювати нові проєкти. Відповідно, у системі має бути присутня сутність Проєкт, яка містить поля для збереження назви, опису, зображення (логотипу чи заставки), дату створення.

У межах кожного проєкту користувач може створювати різноманітні завдання. Таким чином, виникає необхідність у сутності Завдання, що зберігає такі характеристики, як назва, опис, термін виконання (дедлайн), статус, дата

створення, а також зв’язок із відповідним проєктом та інші атрибути.

Для проєктів важливо, щоб завдання могли бути призначені одному або декільком учасникам. Це означає, що між сутностями Завдання та Користувач повинен існувати зв’язок «багато-до-багатьох», що дозволяє одне завдання призначати кільком користувачам і навпаки – користувач може бути відповідальним за декілька завдань.

Оскільки у кожному проєкті може брати участь декілька користувачів із різними ролями (наприклад, адміністратор, менеджер, виконавець, спостерігач), слід передбачити зв’язок між Користувачем та Проєктом, що визначає роль користувача в конкретному проєкті. Для цього доцільно використовувати проміжну сутність або таблицю.

Додатково, для забезпечення ефективної командної роботи, передбачено можливість обміну повідомленнями у межах проєкту чи конкретного завдання. Це вимагає впровадження сутності Повідомлення, яка містить текст повідомлення, автора, дату й час надсилання, а також посилання на проєкт або завдання, до якого належить це повідомлення.

Щоб забезпечити історичність та можливість відстеження дій користувачів (наприклад, створення, редагування чи видалення завдань), в системі має бути сутність журнал дій, який фіксує тип дії, дату, користувача та об’єкт, до якого дія відноситься. Для повноти функціоналу система повинна підтримувати календарні події, які можуть бути пов’язані з проєктом або завданням.

Таким чином, аналіз основних сценаріїв використання системи приводить до необхідності створення наступних ключових сутностей:

- користувач;
- проєкт;
- завдання;
- повідомлення;
- журнал дій.

Кожна із цих сутностей матиме власний набір атрибутів, що детально визначають їхні властивості, а між сутностями встановлюються відповідні зв’язки, що відображають логіку реального процесу співпраці та організації роботи.

Детальна структура та взаємозв’язки між цими сутностями ілюструються на діаграмі структури бази даних, що наведена на рисунку 2.2.

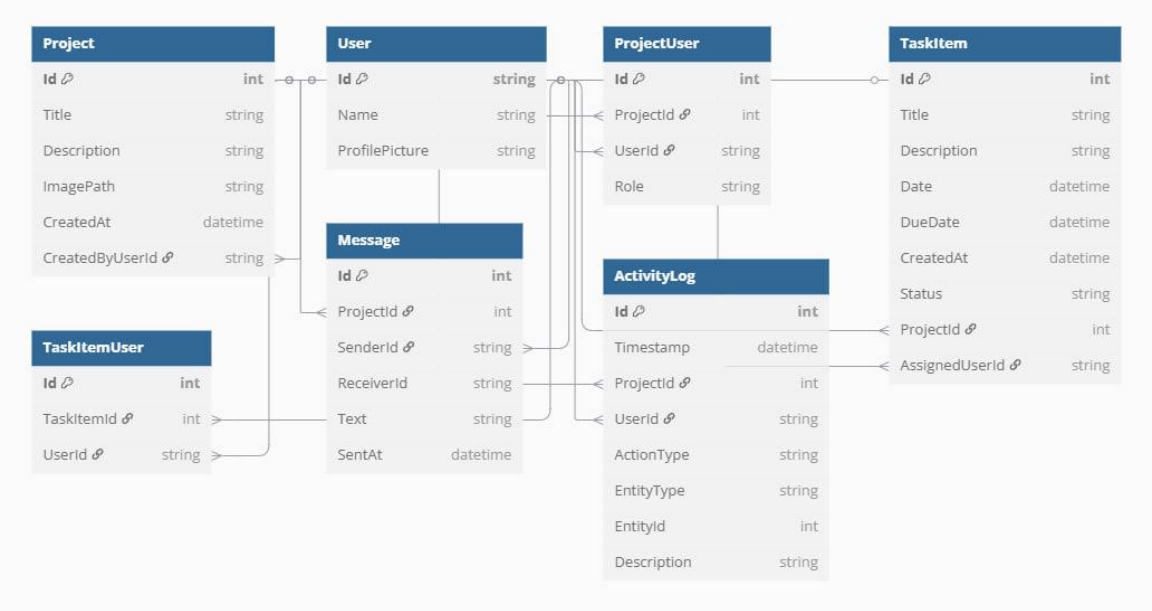


Рисунок 2.2 – діаграма структури таблиць в БД

ER-діаграма, представлена на рисунку 2.2, дозволяє наочно відобразити всі взаємозв’язки між основними сутностями інформаційної системи. Вона слугує важливим інструментом для подальшого проектування, розробки та оптимізації бази даних, оскільки чітко демонструє, як саме організовані дані, які атрибути притаманні кожній таблиці, та яким чином реалізовані зв’язки «один-до-багатьох» і «багато-до-багатьох» між користувачами, проєктами та завданнями [11]. Завдяки такому підходу стає можливим не лише забезпечити цілісність даних, а й легко масштабувати систему у майбутньому, додаючи нові функції або розширюючи існуючі модулі без втрати стабільності та керованості структурою даних.

2.3 Детальне проектування модулів

Детальне проектування модулів є важливим етапом життєвого циклу розробки програмного забезпечення, що дозволяє чітко структурувати систему, визначити функціональні зв’язки між компонентами та забезпечити зрозумілий розподіл відповідальностей. На цьому етапі кожен модуль майбутнього вебзастосунку аналізується окремо, визначаються його цілі, вхідні та вихідні дані, а також способи взаємодії з іншими модулями.

Для формалізації та графічного опису бізнес-процесів у межах системи широко використовується методологія функціонального моделювання IDEF0. Цей підхід базується на створенні ієрархічних діаграм, які розкривають основні функції системи, їхню декомпозицію на підпроцеси та взаємозв’язки між ними. Використання IDEF0 дозволяє наочно побудувати модель предметної області, описати логіку роботи кожного модуля та продемонструвати інформаційні потоки, що проходять через систему [12].

Суттєвою перевагою методології IDEF0 є її гнучкість і масштабованість – вона дозволяє поступово деталізувати окремі функції аж до рівня конкретних операцій і навіть алгоритмів обробки даних [13]. Це особливо корисно при проектуванні складних інформаційних систем, де важливо врахувати не лише внутрішню логіку модулів, а й зовнішні впливи (наприклад, інтеграцію з іншими сервісами, взаємодію з користувачами, впровадження систем захисту та контролю доступу користувачів).

Методологія IDEF0 має низку переваг для проектування складних інформаційних систем, зокрема вебзастосунків для організації завдань та проєктів. Діаграми IDEF0 дозволяють відобразити не лише основні функціональні блоки, але й усі зовнішні впливи, керуючі чинники та необхідні ресурси. Такий підхід спрощує подальший етап реалізації, полегшує інтеграцію та забезпечує можливість швидкої модифікації окремих модулів у майбутньому.

На рисунку 2.3 показано контекстну діаграму роботи вебзастосунку для організації персональних та командних завдань.

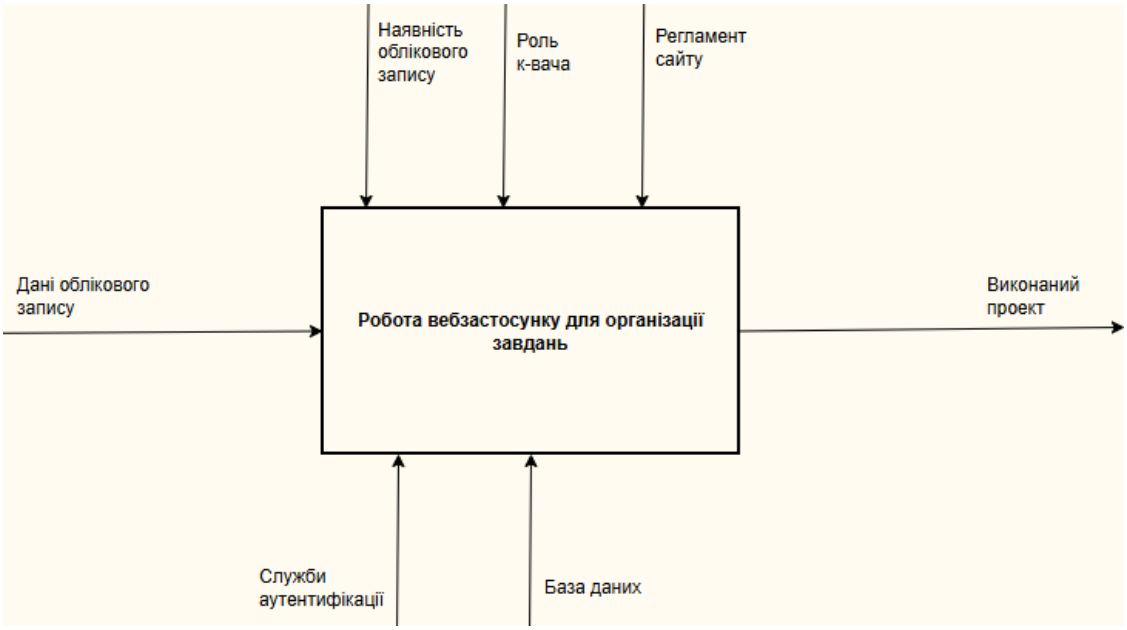


Рисунок 2.3 – контекстна діаграма

Контекстна діаграма наочно демонструє загальні межі та основні взаємозв'язки вебзастосунку для організації завдань у вигляді одного великого функціонального блоку [14]. У центрі розташовано основний процес – «Робота вебзастосунку для організації завдань», який приймає вхідні дані, керуючі впливи та ресурси й формує основний вихід – «Виконаний проект».

Вхідними даними для системи є «Дані облікового запису» користувача, які запускають всі основні процеси платформи. Серед керуючих впливів виділено наявність облікового запису (перевірка автентичності), роль користувача (визначає рівень доступу та можливості), а також регламент сайту (правила, політики і обмеження). До ресурсів, необхідних для функціонування, відносяться служби аутентифікації (механізми реєстрації та входу) та база даних, яка містить усі дані щодо користувачів, проектів, завдань тощо.

Результатом роботи системи є «Виконаний проект», що відображає досягнення мети користувача – організувати, реалізувати та завершити роботу над

персональними чи командними завданнями відповідно до встановлених правил і ролей. Контекстна діаграма формує основу для подальшої декомпозиції процесів, деталізації логіки функціонування та побудови більш детальних схем взаємодії між окремими модулями.

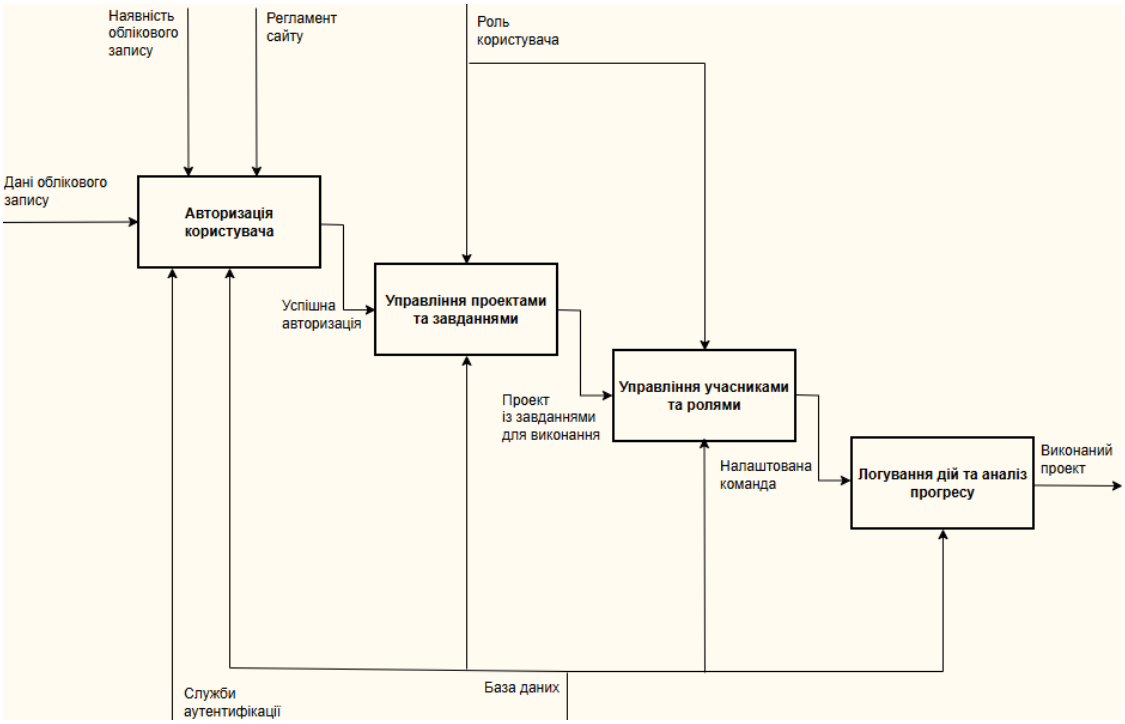


Рисунок 2.4 – діаграма декомпозиції 1 рівня

На діаграмі декомпозиції першого рівня деталізовано основні функціональні блоки вебзастосунку для організації завдань. Від загального процесу, показаного на контекстній діаграмі, система розбивається на ключові бізнес-процеси, які разом забезпечують досягнення кінцевої мети – виконання проекту.

Перший етап – це перевірка ідентифікаційних даних користувача. Система приймає вхідні дані (логін, пароль), перевіряє наявність облікового запису, застосовує регламент (політики безпеки, обмеження) і взаємодіє зі службами автентифікації. Успішна авторизація дає доступ до подальших модулів.

Після авторизації користувач отримує можливість створювати, редагувати або видаляти проекти і завдання. Цей модуль керує усією логікою роботи із

задачами, використовуючи інформацію про роль користувача і зберігаючи зміни в базі даних.

У наступному блоці відбувається налаштування команди – додавання учасників до проекту, призначення ролей та прав доступу. Саме тут визначається, хто і які дії може виконувати в межах певного проекту. Вхідними є проект із завданнями і роль користувача.

Останній модуль відповідає за фіксацію усіх ключових змін та дій користувачів, а також за аналіз виконання завдань. На виході формується статус – виконаний проект. Цей блок тісно взаємодіє із попередніми: отримує інформацію про зміни, стан завдань, налаштування команди, і зберігає історію змін у базі даних.

Таким чином, декомпозиція 1 рівня дозволяє побачити основні модулі системи, їхній взаємозв’язок та роль у досягненні мети – ефективної організації та контролю виконання персональних і командних завдань у вебзастосунку.

Наступним етапом буде декомпозиція кожного з процесів. Розпочнемо з декомпозиції процесу «Авторизація користувача». Побудована діаграма декомпозиції другого рівня показана на рисунку 2.5.

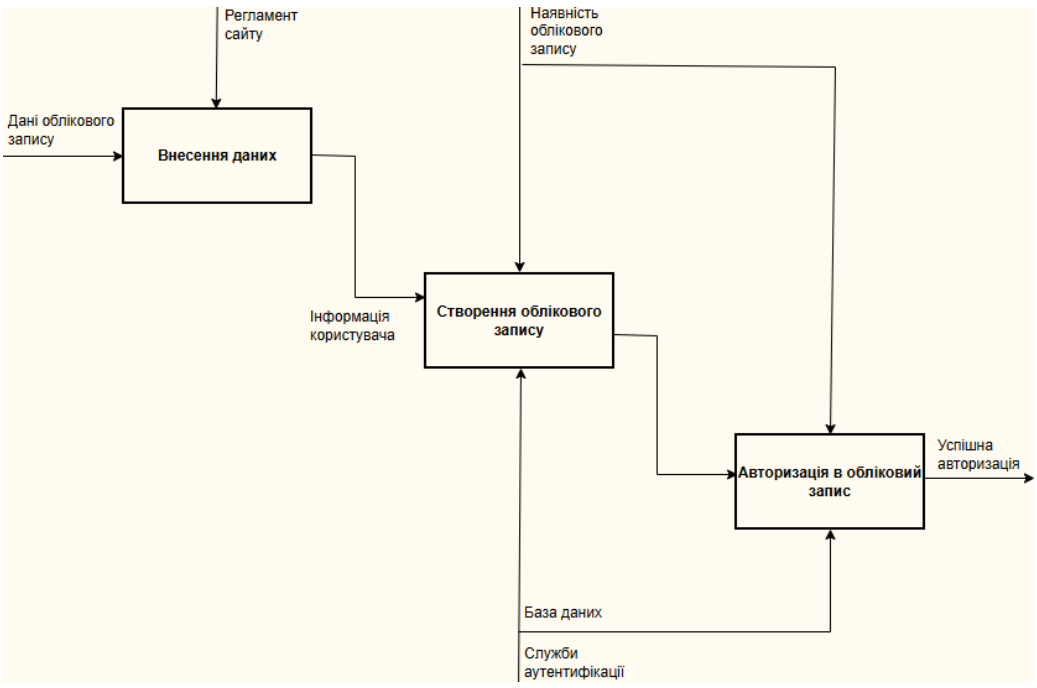


Рисунок 2.5 – Діаграма декомпозиції 2 рівня «Авторизація користувача»

Дана діаграма деталізує процес авторизації користувача у вебзастосунку, що включає послідовне внесення даних, створення нового облікового запису та автентифікацію в системі. На вході процесу – персональні дані користувача та регламент сайту, які перевіряються та обробляються відповідно до встановлених вимог. Після створення облікового запису здійснюється перевірка його наявності й коректності введених даних із залученням бази даних і служб автентифікації. Результатом є підтвердження успішної авторизації, що дозволяє користувачу повноцінно працювати із сервісом. Такий підхід забезпечує захищеність та достовірність ідентифікації, а також підготовку користувача до подальшої роботи в системі.

Далі виконаємо декомпозиція процесу «Управління проектами та завданнями». Побудована діаграма другого рівня зображена на рисунку 2.6.

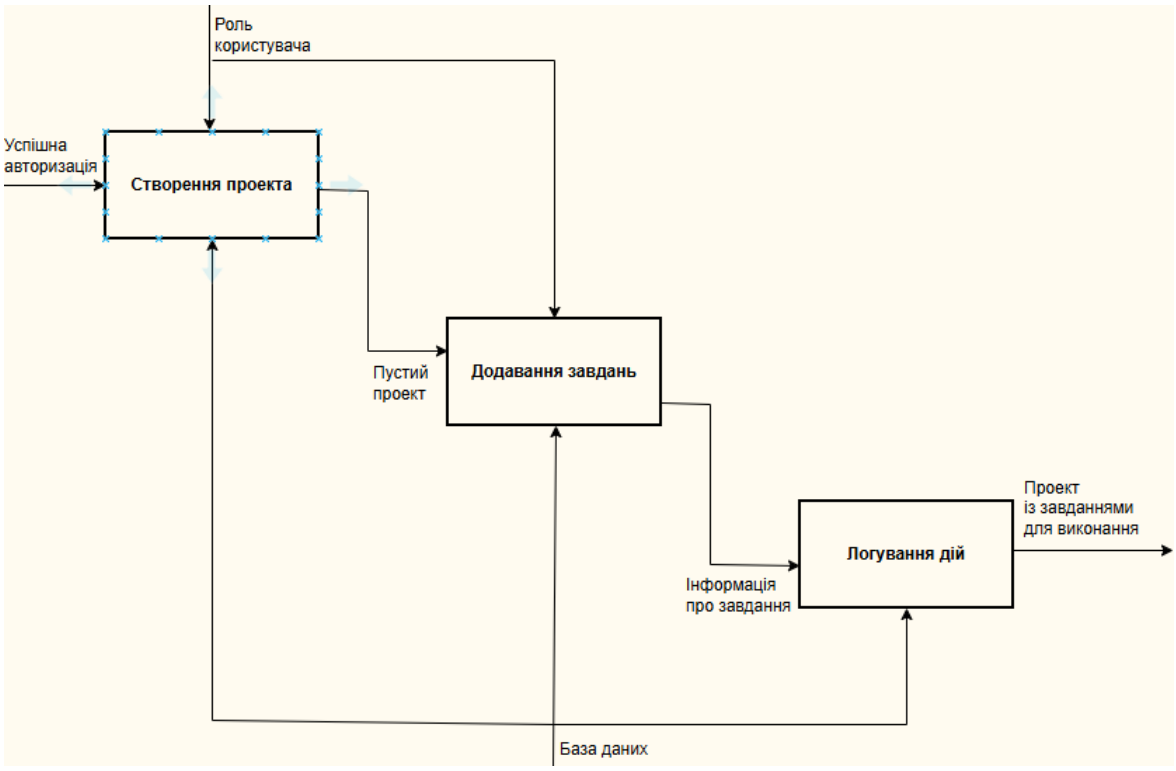


Рисунок 2.6 – Діаграма декомпозиції 2 рівня для процесу «Управління проектами та завданнями»

Після успішної авторизації користувача, з урахуванням його ролі, здійснюється створення нового проекту. Далі, користувач наповнює проект завданнями, інформація про які зберігається у базі даних. Усі дії щодо додавання завдань автоматично фіксуються у журналі змін. В результаті формується проект із завданнями, готовий до виконання командою, а журнал дій забезпечує прозорість і контроль етапів розробки.

Виконаємо наступну декомпозицію, а саме для процесу «Управління учасниками та ролями» (рис. 2.7).

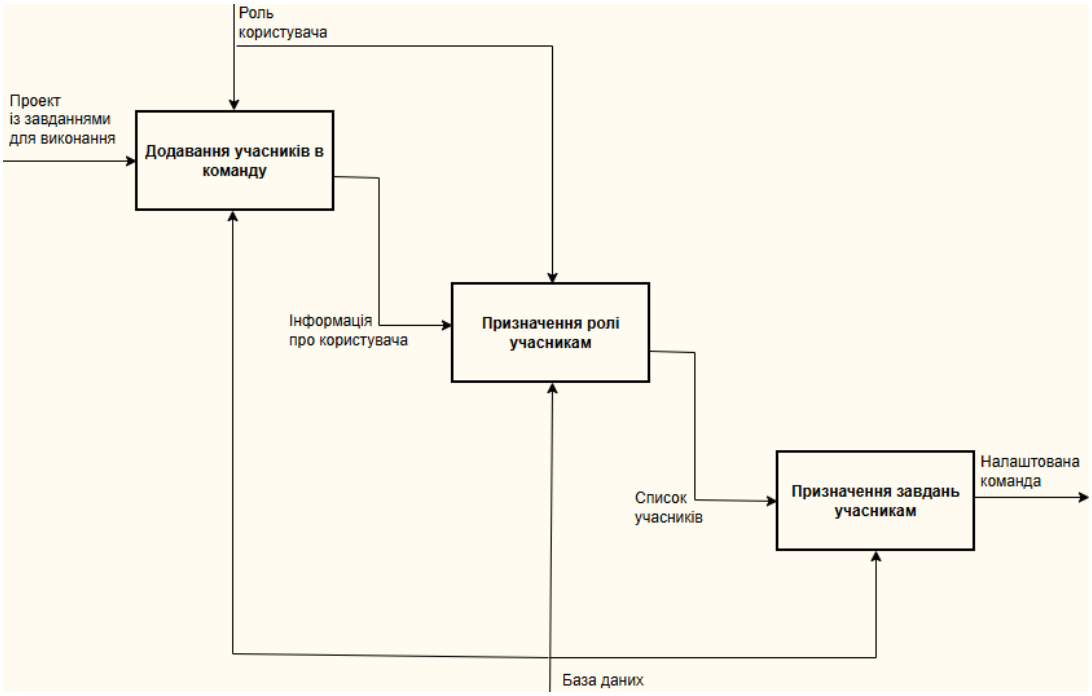


Рисунок 2.7 – Діаграма декомпозиції 2 рівня для процесу «Управління учасниками та ролями»

Діаграма декомпозиції 2 рівня для процесу управління учасниками та ролями відображає послідовність налаштування командної роботи у проекті. Спочатку відбувається додавання учасників у команду на основі вже створеного проекту з завданнями. Далі для кожного учасника визначається роль відповідно до його функцій і прав доступу, що враховує як дані про користувача, так і рольову політику. Після призначення ролей формуються списки учасників, і кожному з них

призначаються конкретні завдання. У результаті створюється налаштована команда, готова до спільної роботи, а дані про ролі та призначення завдань зберігаються у базі даних.

На завершення потрібно виконати декомпозиція процесу «Логування дій та аналіз прогресу» (рис. 2.8).

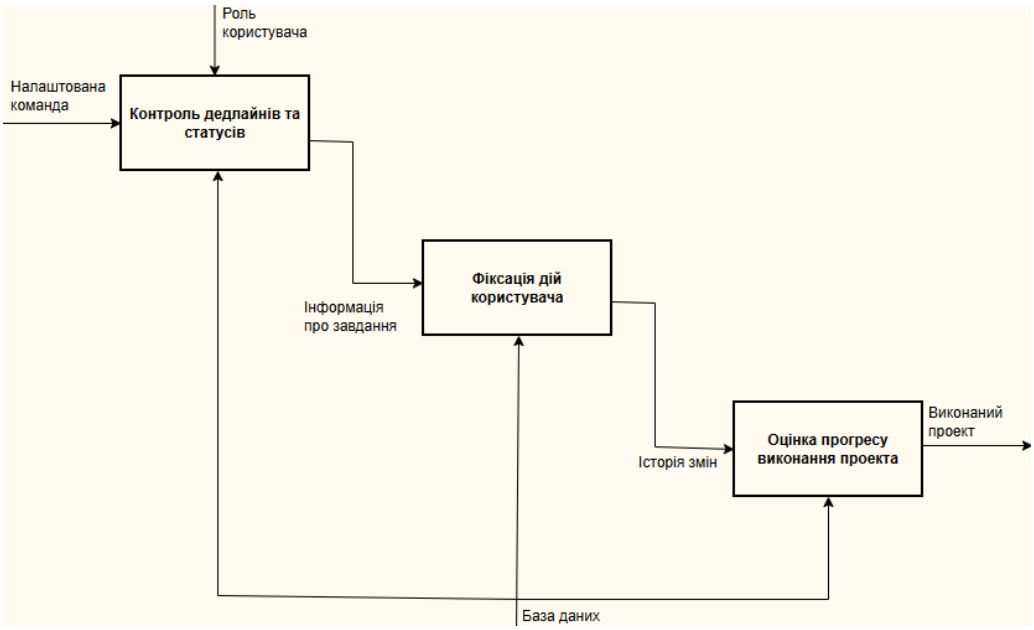


Рисунок 2.8 – Діаграма декомпозиції 2 рівня для процесу «Логування дій та аналіз прогресу»

На останній діаграмі відображається як здійснюється контроль виконання проекту на основі фіксації ключових дій користувачів. Таким чином, в залежності від ролі, учасники команди виконують свої завдання, контролюючи дедлайни і статуси. Далі всі зміни та важливі дії користувачів фіксуються в системі, а інформація про виконання завдань зберігається у базі даних. На основі накопиченої історії змін здійснюється оцінка прогресу виконання проекту, що дозволяє визначити ступінь завершеності та формально завершити проект, якщо всі цілі досягнуті. Таким чином, процес логування і аналізу дозволяє не лише відстежувати всі ключові події, а й забезпечити досягнення кінцевої мети – виконання проекту.

2.4 Проектування інтерфейсу користувача

У процесі проектування інтерфейсу для вебзастосунку організації особистих та командних завдань були створені інтерактивні прототипи основних сторінок системи у Figma. Це дозволило продемонструвати логічну структуру, навігацію та основні елементи дизайну ще до реалізації програмного коду, а також отримати зворотний зв'язок від користувачів та зацікавлених сторін на ранніх етапах розробки системи.

Навігація у системі реалізована у вигляді бічної панелі (sidebar), що забезпечує швидкий доступ до основних розділів: дашборд, список завдань, повідомлення, аналітика, а також до списку проектів. Кожна сторінка має власну структуру й інтерфейсні елементи, що відповідають її функціональному призначенню. Далі подано короткий опис основних прототипів сторінок.

Макет пустого та заповненого дашборду показано на рисунках 2.9 та 2.10.

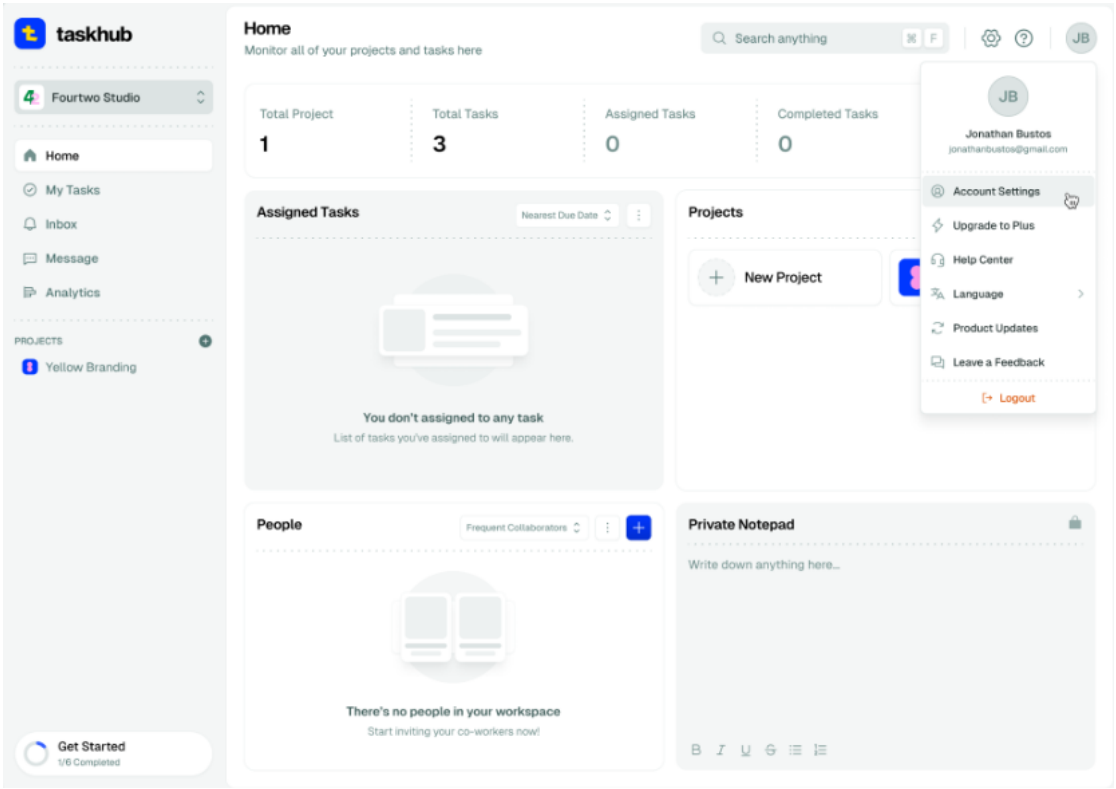


Рисунок 2.9 – Пустий дашборд

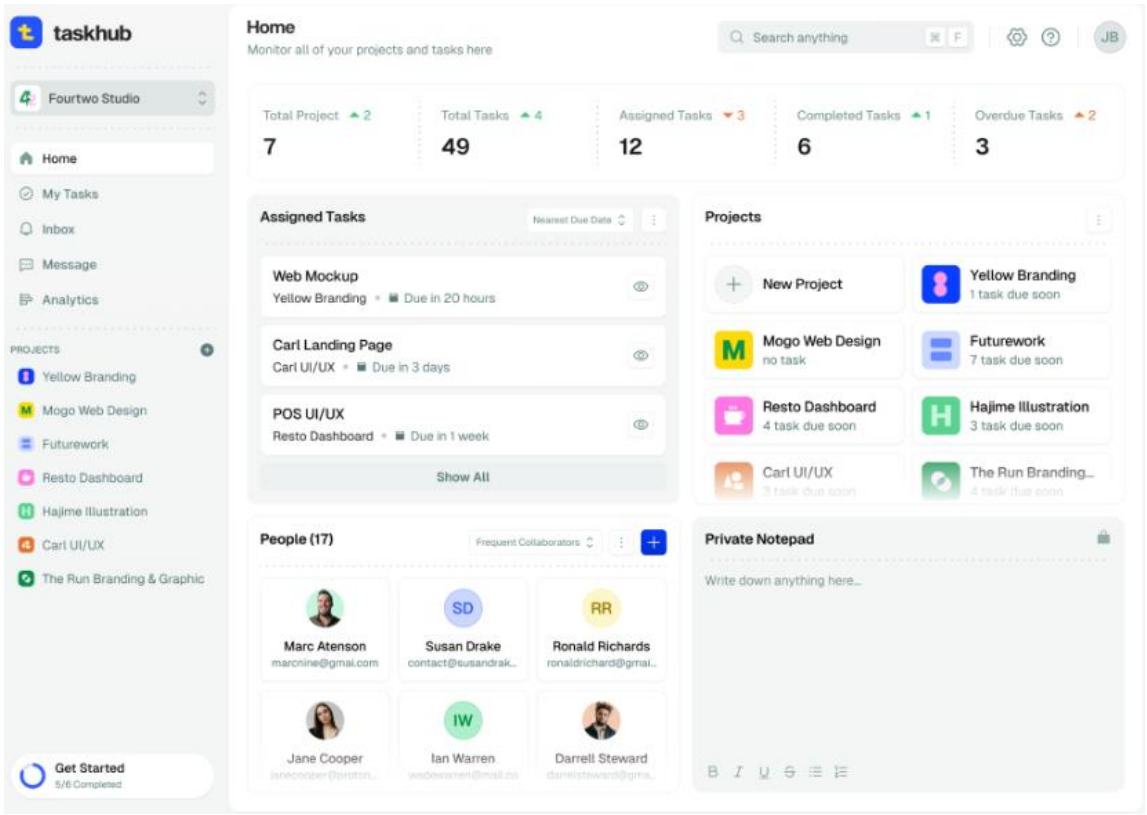


Рисунок 2.10 – Наповнений дашборд

Головна сторінка є дашбордом користувача, де зібрана узагальнена інформація по всіх проектах і завданнях. В одному місці відображається загальна статистика – кількість створених проектів, кількість усіх завдань, виконаних та призначених задач, а також інформація про завершені та прострочені завдання. Це дозволяє користувачу миттєво оцінити рівень завантаженості команди, прогрес виконання задач, а також відслідковувати динаміку змін у роботі.

Така організація головної сторінки робить робочий процес максимально прозорим, ефективним та дозволяє кожному користувачу залишатися в курсі всіх подій у проектній команді.

Макет сторінок із завданнями користувача в табличному та календарному вигляді відображено на рисунку 2.11 та 2.12.

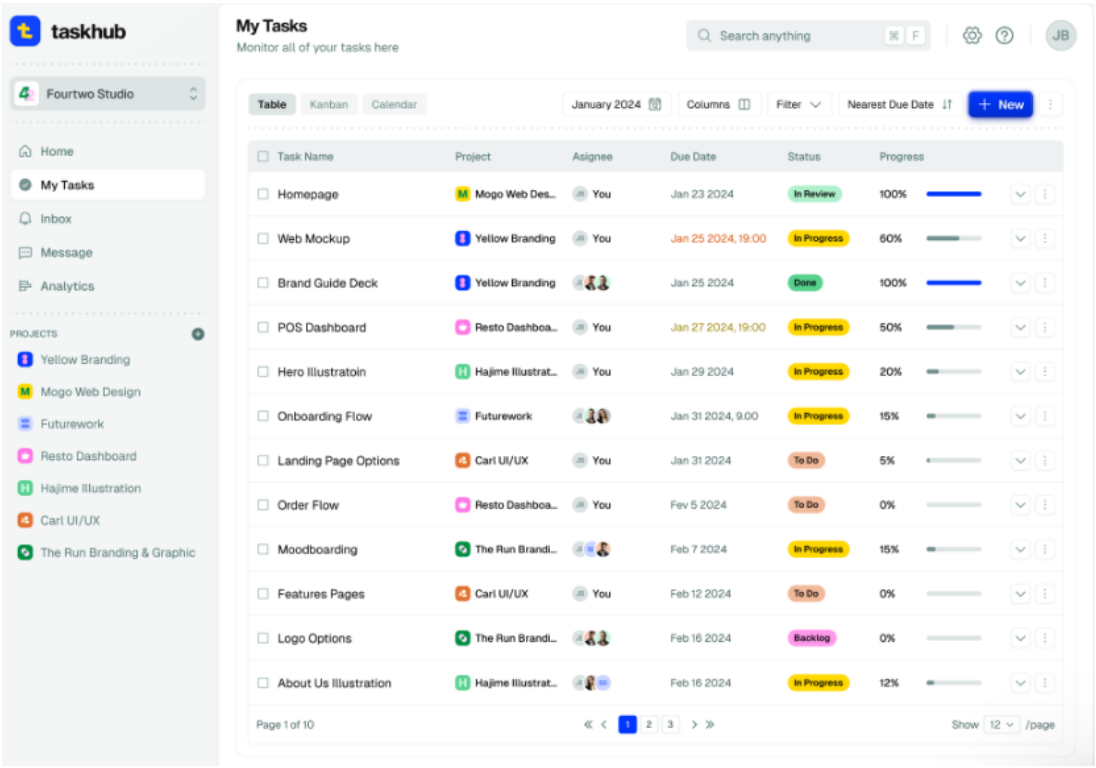


Рисунок 2.11 – Табличний вигляд завдань

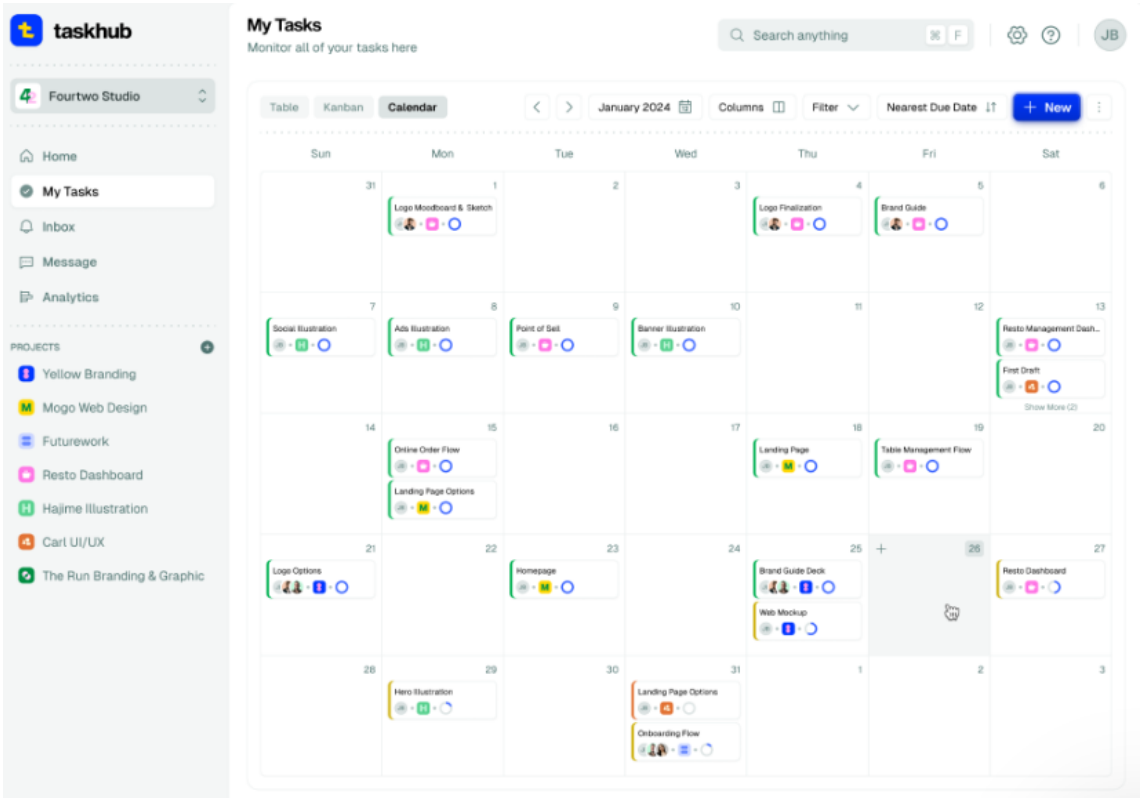


Рисунок 2.12 – Календарний вигляд завдань

Ці сторінки дозволяють користувачу зручно переглядати всі свої завдання як у вигляді таблиці, так і в календарному форматі. Вся ключова інформація відображається компактно, а календарний режим допомагає ефективно планувати час, стежити за дедлайнами й оптимізувати робоче навантаження. Такий підхід забезпечує швидкий доступ до завдань і гнучке управління навіть великою кількістю задач.

Макет сторінки з чатом показано на рисунку 2.13.

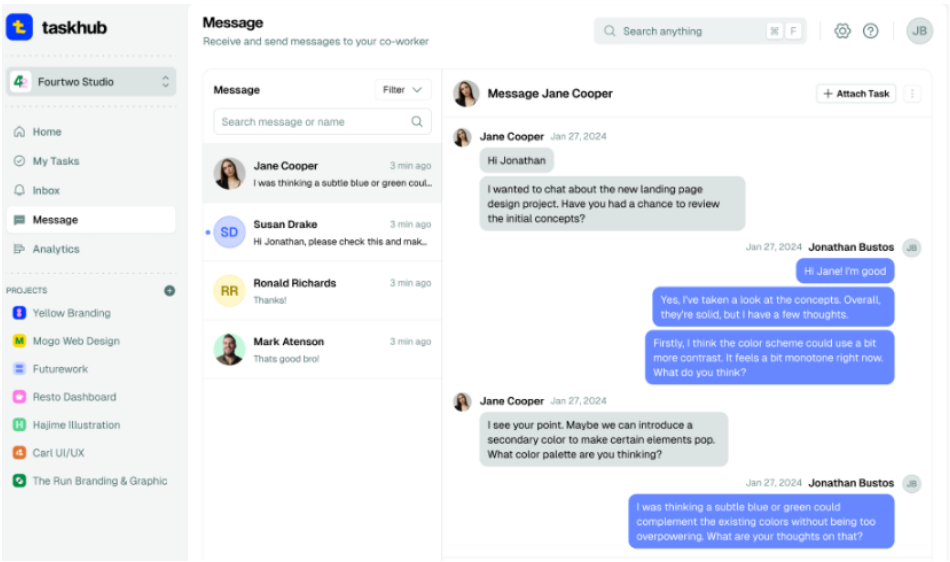


Рисунок 2.13 – Чат

Сторінка чату реалізує можливість особистого й командного спілкування між користувачами системи. Зліва знаходиться список діалогів: командний чат та окремі користувачі. В центральній частині – сама розмова у вигляді переписки з відображенням часу й автора повідомлень. Такий інтерфейс підвищує рівень командної взаємодії та оперативності сповіщень.

Таким чином, було ретельно опрацьовано структуру інтерфейсу та використано сучасні дизайн-підходи, що дозволяє користувачам легко орієнтуватися у системі, оперативно знаходити потрібну інформацію й ефективно взаємодіяти із сервісом у щоденній роботі. Усі елементи виконані в єдиному стилі, що забезпечує цілісність та інтуїтивність інтерфейсу.

2.5 Аналіз та вибір технологій і методів реалізації вебзастосунку

Одним із ключових рішень для ефективної організації роботи розробників стало використання інтегрованого середовища розробки Visual Studio. Це потужний інструмент, який забезпечує зручний процес написання, налагодження та тестування коду, інтеграцію з системами контролю версій, підтримку різноманітних шаблонів і розширень [15].

В основі логіки та структури вебзастосунку лежить мова програмування C#. Простота та зрозумілість синтаксису, багатий набір стандартних бібліотек, підтримка сучасних парадигм програмування і безпека виконання роблять C# ідеальним вибором для розробки складних корпоративних і веб-систем [16].

Вибір саме C# органічно поєднується з використанням фреймворку ASP.NET Core, який відкриває широкі можливості для створення кросплатформених вебзастосунків із високою продуктивністю. ASP.NET Core надає розробнику засоби для впровадження сучасної архітектури, розділення логіки на різні рівні, впровадження залежностей (dependency injection) і захищеної системи маршрутизації. Його гнучкість дозволяє швидко впроваджувати нові модулі, організовувати роботу зі складною бізнес-логікою та ефективно підтримувати великі проєкти [17].

Невід’ємною частиною став Entity Framework Core, який виступає об’єктно-реляційним відображенням (ORM) і значно спрощує взаємодію з базою даних. Entity Framework Core дозволяє працювати з даними на рівні об’єктів, не турбуючись про деталі SQL-запитів, автоматизує міграції, забезпечує цілісність і захищеність інформації, а також сприяє підвищенню продуктивності розробки за рахунок скорочення шаблонного коду [18].

Для реалізації динамічних сторінок і побудови гнучких інтерфейсів у межах ASP.NET Core використовується технологія Razor Pages, що поєднує C# та HTML у єдиному шаблоні. Razor забезпечує чітке розділення бізнес-логіки та візуального

представлення, підтримує двостороннє зв’язування даних, дозволяє будувати адаптивні, сучасні інтерфейси й зручно працювати з формами та валідацією даних. Це сприяє створенню інтерактивних та інтуїтивно зрозумілих користувацьких інтерфейсів, які відповідають вимогам сучасних вебзастосунків [19].

Потужним інструментом для роботи з даними виступає LINQ, що забезпечує гнучкі та зрозумілі засоби для вибірки, фільтрації, групування й агрегації даних безпосередньо в коді C#. Використання LINQ дозволяє розробнику уникати складних SQL-запитів, підвищує читабельність і безпечність коду, а також спрощує підтримку й модифікацію логіки взаємодії з даними впродовж усього життєвого циклу системи [20].

Для досягнення високого рівня інтерактивності на стороні клієнта доцільно використовувати JavaScript та популярну бібліотеку JQuery, що спрощує маніпуляції з елементами DOM, обробку подій, анімації, динамічну зміну вмісту сторінки [21]. У поєднанні з технологією AJAX це дає змогу реалізовувати асинхронний обмін даними між клієнтом і сервером, оновлювати частини сторінки без повного перезавантаження, створювати зручні та сучасні веб-інтерфейси [22].

Усі дані застосунку зберігаються у реляційній базі даних MS SQL Server Express, яка забезпечує надійність, підтримку транзакцій, індексацію, ефективний пошук і високий рівень цілісності даних. Обрана СУБД легко інтегрується з Entity Framework Core, має розвинуті засоби адміністрування та чудово підходить для прототипування, навчальних, тестових і малих промислових рішень, дозволяючи при необхідності масштабувати систему до повнофункціональної комерційної версії SQL Server [23].

Усі інструменти гармонійно поєднуються, забезпечуючи сучасний рівень якості, безпеки та масштабованості програмного продукту, що повністю відповідає вимогам і завданням, поставленим у межах бакалаврської роботи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Програмна реалізація модулів

У процесі розробки вебзастосунку реалізація модулів відбувалася послідовно, починаючи з найважливіших складових. Кожен модуль відповідає за окремий аспект роботи системи – від реєстрації користувачів до обробки даних, від побудови інтерфейсу до допоміжних сервісів.

Першим етапом стала реалізація модуля, пов’язаного з авторизацією та аутентифікацією користувачів. Для цього було використано готовий шаблон аутентифікації “Individual Accounts”, який пропонується інфраструктурою ASP.NET Core під час створення проєкту. Цей підхід дозволив швидко інтегрувати перевірені механізми реєстрації, входу, виходу, зміни пароля, відновлення доступу та підтвердження електронної пошти. Крім того, він гарантує дотримання сучасних стандартів безпеки та захисту персональних даних, надаючи широкі можливості для розширення логіки і налаштування під потреби проєкту.

Наступним кроком стала реалізація модуля для роботи з базою даних. На цьому етапі було створено контекст бази даних (ApplicationDbContext), у якому описані всі основні сутності системи й зв’язки між ними. Контекст було підключено до застосунку через механізм залежностей у файлі Program.cs, а конфігурація підключення здійснювалася за допомогою connection string із файлу налаштувань appsettings.json. Це дозволило централізовано керувати доступом до БД, автоматизувати створення та міграцію таблиць, а також забезпечити прозору взаємодію між усіма модулями вебзастосунку.

Після налаштування базової інфраструктури розпочався етап розробки моделей вебзастосунку. Кожна модель була представлена окремим C#-класом, що містив у собі властивості для зберігання даних, а також анотації для валідації та конфігурування атрибутів сутностей у БД. Наприклад, модель проекту (Project) може виглядати наступним чином:

```
public class Project
{
```

```
    [Key]
    public int Id { get; set; }
    [Required]
    [StringLength(100)]
    public string Title { get; set; } = string.Empty;
    public string? Description { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public ICollection<TaskItem> TaskItems { get; set; } = new
List<TaskItem>();
    // Інші властивості та зв'язки
}
```

Ця модель дозволяє автоматично формувати відповідну таблицю у базі даних, а її властивості – зберігати необхідну інформацію про проект і всі пов’язані із ним завдання.

Далі була розроблена група контролерів – спеціальних класів, які відповідають за логіку обробки HTTP-запитів, взаємодію з моделями й підготовку даних для представлень. Кожен контролер відповідає за окрему частину функціоналу (наприклад, роботу з проектами, завданнями, користувачами). Реалізація CRUD-операцій (Create, Read, Update, Delete) у контролерах забезпечує повний цикл керування сутностями системи. Ось приблизний приклад того, як має виглядати контролер:

```
public class ProjectsController : Controller
{
    private readonly ApplicationDbContext _context;

    public ProjectsController(ApplicationDbContext context)
    {
        _context = context;
    }

    // GET: Projects
    public async Task<IActionResult> Index()
    {
        var projects = await _context.Projects.ToListAsync();
        return View(projects);
    }
    // POST: Projects/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
```

```
    public async Task<IActionResult> Create(Project project)
    {
        if (ModelState.IsValid)
        {
            _context.Projects.Add(project);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(project);
    }


    // POST: Projects/Edit/5
    [HttpPost]
    public async Task<IActionResult> Edit(int id, Project project)
    {
        //редагування проекта
    }

    // POST: Projects/Delete/5
    [HttpPost]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        //видалення проекта
    }
}
```

Аналогічним чином реалізовано інші контролери, які забезпечують повний набір операцій над об’єктами предметної області.

Особливу увагу було приділено модулю представлень (View). Для кожного контролера та операції були створені відповідні файли розмітки на Razor (cshtml), які відповідають за відображення інформації користувачеві, побудову форм, списків, таблиць, інтеграцію елементів управління та налаштування стилів за допомогою CSS. Таким чином, ось так може виглядати розмітка представлення для того, щоб відобразити список проектів:

```
@model IEnumerable<Project>

<h2>Список проектів</h2>

<ul class="list-group">
@foreach (var project in Model)
{
```

```
    <li class="list-group-item">
        <h2>@project.Title</h2>
        <span>@project.Description</span>
        <div>Створено: @project.CreatedAt.ToShortDateString()</div>
        <a asp-action="Edit" asp-route-
id="@project.Id">Редагувати</a>
        <a asp-action="Delete" asp-route-
id="@project.Id">Видалити</a>
    </li>
}
</ul>

<a asp-action="Create">Додати новий проект</a>
```

На окремому етапі відбулась розробка сервісів для реалізації допоміжної функціональності, яка виходить за рамки базової CRUD-логіки. Було розроблено, наприклад, FileService для зберігання та отримання файлів, а також ActivityLogger для ведення журналу дій користувачів. Сервіси підключаються до системи через dependency injection (реєстрація у Program.cs):

```
builder.Services.AddScoped<IFileService, FileService>();
builder.Services.AddScoped<IActivityLogger, ActivityLogger>();
```

Доступ до них здійснюється через конструктори відповідних контролерів:

```
private readonly IActivityLogger _activityLogger;
private readonly IFileService _fileService;

public ProjectsController(IActivityLogger activityLogger,
IFileService fileService)
{
    _activityLogger = activityLogger;
    _fileService = fileService;
}
```

Це дозволяє централізовано використовувати логіку завантаження файлів або логування в різних частинах застосунку, що підвищує гнучкість архітектури.

Завдяки такій послідовній реалізації модулів вдалося створити добре структурований, масштабований і зручний для подальшого розвитку

вебзастосунок, що відповідає вимогам сучасної розробки і забезпечує користувачам надійний інструмент для організації особистих і командних завдань.

3.2 Розроблення бази даних

У процесі розробки сучасних вебзастосунків проектування та впровадження бази даних відіграє ключову роль, оскільки саме база даних забезпечує централізоване, структуроване й надійне зберігання всієї інформації, необхідної для функціонування системи. Від правильної організації бази даних залежать такі важливі параметри, як швидкість доступу до даних, цілісність, захист інформації та можливість масштабування програмного продукту. В рамках цієї роботи для реалізації зберігання даних обрано реляційну модель, що відповідає предметній області та архітектурі застосунку.

Особливістю реалізації бази даних у даному проекті є використання сучасної технології Entity Framework Core та підходу Code First. На відміну від класичного підходу, коли база даних створюється вручну в середовищі керування базами даних, Code First передбачає спочатку опис усіх сутностей і зв’язків у вигляді C#-класів (моделей) у програмному коді. Після цього, на основі створених моделей, Entity Framework автоматично генерує структуру бази даних, таблиці та зв’язки між ними. Це дозволяє підтримувати єдину модель даних на всіх етапах розробки, спрощує внесення змін у схему БД, зменшує кількість помилок і полегшує супровід системи [24].

Процес створення бази даних за методом Code First складається з кількох основних етапів.

По-перше, здійснюється створення так званого контексту бази даних – спеціального класу, що наслідує від DbContext і визначає набір властивостей типу DbSet<T> для кожної сутності (таблиці) у базі даних. Контекст відповідає за

керування з’єднанням із БД, відстеження змін, виконання транзакцій і реалізацію запитів. Наприклад, базовий контекст може виглядати так:

```
public class ApplicationDbContext : DbContext
{
    public DbSet<Project> Projects { get; set; }
    public DbSet<TaskItem> TaskItems { get; set; }
    // Інші DbSet для сутностей
    public
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

Далі у проєкті створюються моделі сутностей – класи, які описують структуру майбутніх таблиць. У моделях вказуються всі необхідні атрибути (властивості класу), а також анотації (data annotations) чи конфігурації для коректного формування стовпців і зв’язків. Наприклад, модель користувача:

```
public class Project
{
    [Key]
    public int Id { get; set; }

    [Required]
    [StringLength(100)]
    public string Title { get; set; } = string.Empty;

    [StringLength(500)]
    public string? Description { get; set; }

    public string? ImagePath { get; set; }

    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    [Required]
    public string CreatedByUserId { get; set; } = string.Empty;

    public bool IsPersonal { get; set; } = false;

    public string? Status { get; set; }

    [ForeignKey("CreatedByUserId")]
```

```
    public User? CreatedByUser { get; set; }

    public ICollection<ProjectUser> ProjectUsers { get; set; } = new
List<ProjectUser>();
    public ICollection<TaskItem> TaskItems { get; set; } = new
List<TaskItem>();
}
```

У цій моделі атрибут [Key] визначає первинний ключ проекту, [Required] і [StringLength] встановлюють обов'язковість і обмеження довжини для відповідних полів. Поле CreatedByUserId містить ідентифікатор користувача, який створив проект, і пов'язане з навігаційною властивістю CreatedByUser, що дозволяє отримати об'єкт користувача-автора. Колекції ProjectUsers і TaskItems є навігаційними полями, які забезпечують доступ до всіх користувачів, залучених у проект, та до всіх завдань, пов'язаних із цим проектом, що значно спрощує роботу з пов'язаними сутностями у запитах і логіці застосунку.

Аналогічно описуються інші сутності, такі як Project, TaskItem, ProjectUser, TaskItemUser тощо. Завдяки цьому формується цілісна й структурована модель даних, яка відображає логіку предметної області.

Після опису всіх моделей і налаштування контексту, наступним етапом є створення та застосування міграцій. Міграції – це механізм Entity Framework, що дозволяє автоматично синхронізувати структуру БД із моделями, зберігати історію змін та безболісно оновлювати БД у разі розвитку застосунку.

Для створення першої міграції у консолі диспетчера пакетів Visual Studio виконується команда:

```
add-migration [name]
```

Ця команда генерує спеціальні файли, що містять опис усіх таблиць, стовпців і зв'язків згідно з поточними моделями. Після цього для створення (або оновлення) бази даних застосовується команда:

```
update-database
```

В результаті у вибраній СУБД (MS SQL Server Express) створюються всі таблиці з відповідними атрибутами й зовнішніми ключами, а також автоматично застосовуються всі визначені правила та обмеження.

Далі, у процесі розробки, будь-які зміни до моделей (наприклад, додавання нового поля або сутності) знову фіксуються через створення нової міграції та її застосування. Це забезпечує гнучкість і контроль над версіями структури БД.

Процес підключення бази даних до модулів ПЗ здійснюється через впровадження залежностей: контекст БД реєструється у контейнері сервісів (Program.cs):

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
```

Після цього вже відповідні сервіси чи контролери отримують доступ до нього через конструктор або ін’єкцію залежностей.

Таким чином, створення та впровадження бази даних за підходом Code First у поєднанні з Entity Framework Core забезпечує структурованість, масштабованість і легкість підтримки всієї інфраструктури зберігання даних, а також гнучкість подальшого розвитку вебзастосунку.

3.3 Налагодження та тестування програмного забезпечення

Якість програмного забезпечення значною мірою залежить від коректності його роботи, стабільності і здатності обробляти нештатні ситуації. Тому важливим етапом життєвого циклу розробки вебзастосунку є налагодження (debugging) та тестування, що дає змогу виявити помилки й недоліки ще до впровадження продукту в експлуатацію. Основна мета тестування – переконатися, що всі модулі

працюють відповідно до вимог програмного забезпечення, а також забезпечити цілісність, надійність і безпеку системи.

Серед підходів до тестування найбільш поширеним і ефективним для серверної логіки є юніт-тестування (unit testing). Юніт-тести дозволяють перевірити роботу окремих модулів, класів, методів або сервісів у відриві від решти системи, ідеально – ізольовано від зовнішніх залежностей, таких як база даних чи сторонні API [25]. Для цього активно використовуються бібліотеки для створення так званих «моків» (mock objects) – підставних об'єктів, що імітують поведінку реальних залежностей [26].

В екосистемі .NET стандартом де-факто для написання юніт-тестів є фреймворки xUnit, NUnit або MSTest. Всі вони дозволяють створювати автоматизовані тести, організовувати їх у тестові класи, фіксувати перевірки (assertion), та отримувати звітність. Для імітації зовнішніх залежностей, як-от бази даних чи сторонні сервіси, використовується бібліотека Moq – зручний інструмент для створення і налаштування мок-об’єктів, перевірки викликів і перевірки взаємодії із залежностями [27].

У цьому проекті юніт-тестування використовувалось для перевірки коректності роботи основних моделей, сервісів та контролерів, що забезпечують базову логіку роботи із задачами, проектами та логуванням дій. Зокрема, були протестовані такі аспекти:

– створення задачі (TaskItem) та перевірка правильності та коректності ініціалізації її властивостей;

– призначення користувачів до задач;

– додавання задач та учасників у проект (Project);

– правильність створення та наповнення логів активності (ActivityLog);

– робота контролерів: створення задачі, логування дій, редагування, видалення задачі тощо;

Тести були написані як для окремих моделей, так і для методів контролерів

із використанням Moq для створення підставних версій контексту бази даних та наборів даних (DbSet). Наприклад, перевірялося, що при створенні задачі вона додається до контексту, а призначення користувачів здійснюється коректно; при редагуванні задачі змінюються відповідні поля, а при видаленні – об’єкт коректно вилучається. Окремо перевірявся сервіс логування активності, який фіксує дії користувачів.

Проведене тестування дозволило переконатися у правильності логіки додавання, редагування, видалення задач, формування логів активності та базових взаємодій між моделями. Усі основні юніт-тести були виконані успішно, що підтверджує стабільність і коректність реалізованих ключових функцій вебзастосунку. Результати тестування показані на рисунку 3.1.

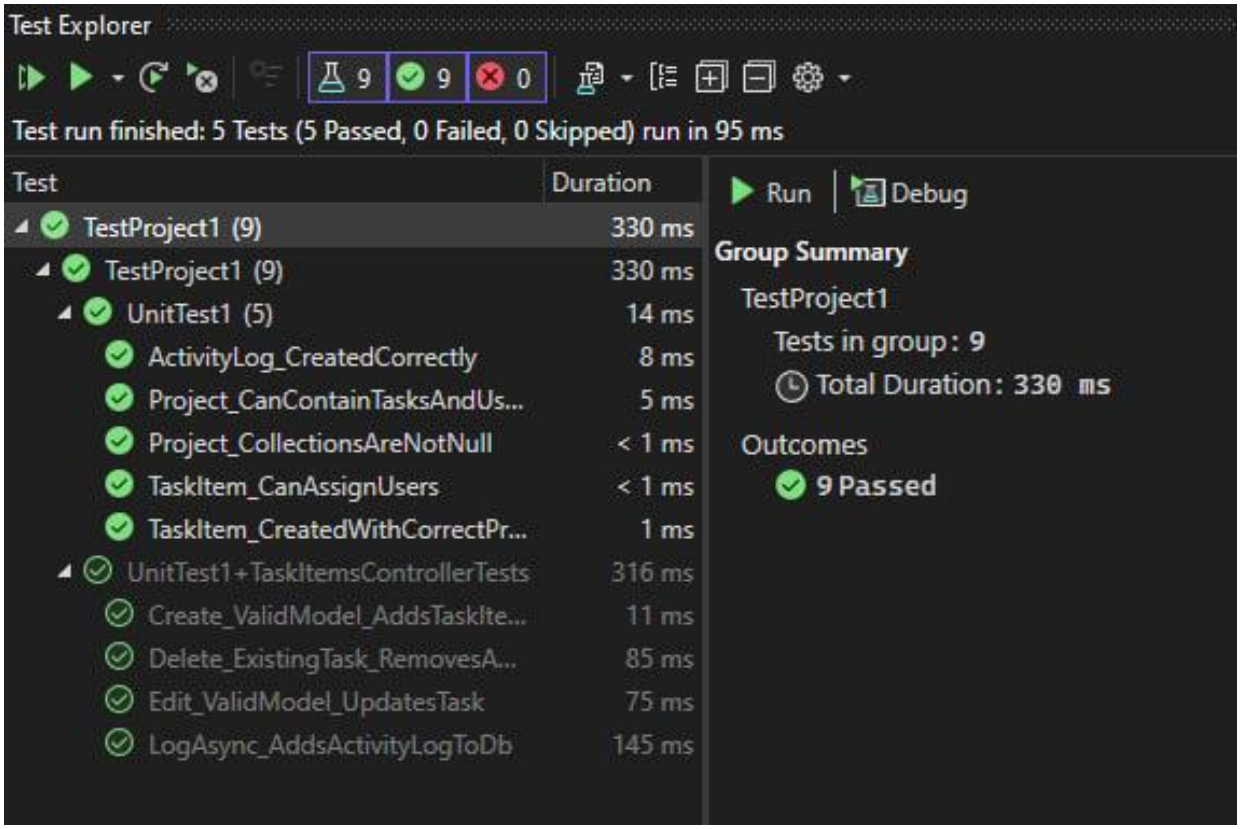


Рисунок 3.1 – Результати тестування

3.4 Інструкція користувача

Розглянемо покрокову інструкцію для користувача, яка охоплює всі основні функції вебзастосунку та демонструє результати виконання кожної дії.

На початку роботи користувачу необхідно пройти авторизацію у системі (рис. 3.2). Для цього потрібно ввести свою електронну адресу та пароль у відповідні поля форми входу, після чого натиснути кнопку “Log in”. Якщо облікового запису ще немає, можна скористатися посиланням “Register as a new user” для створення нового профілю.

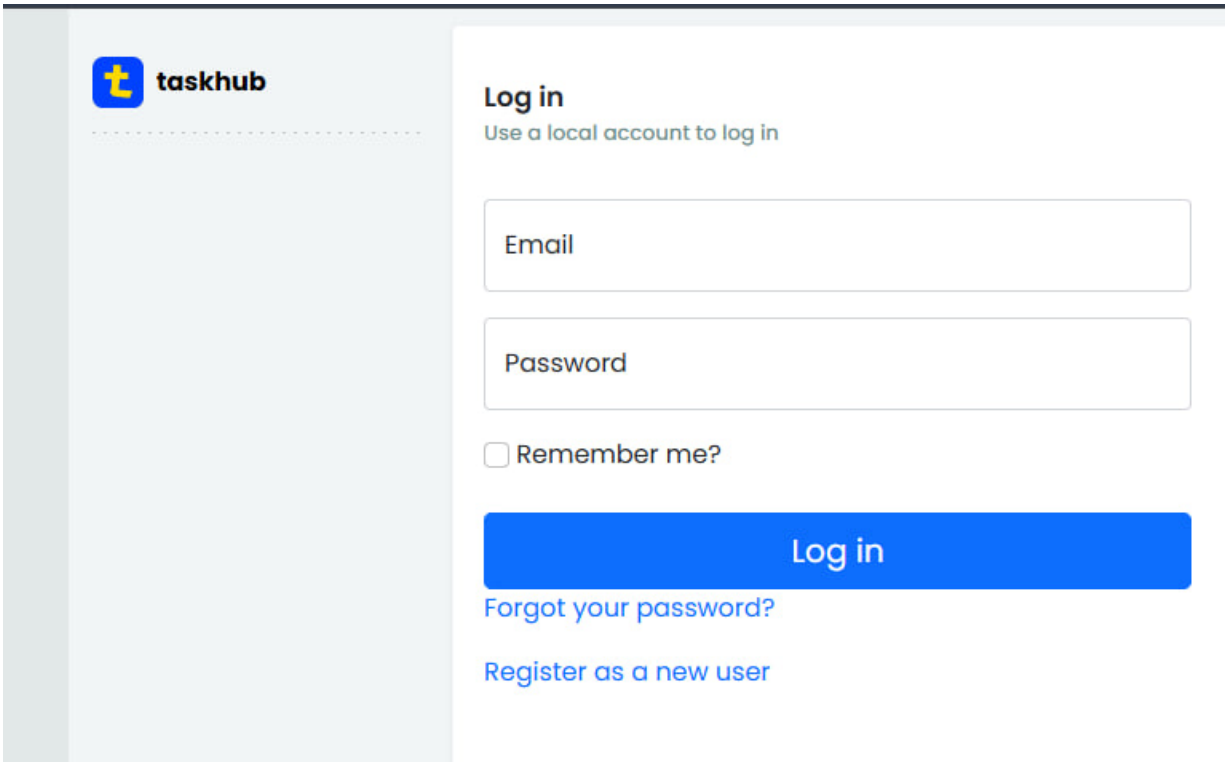


Рисунок 3.2 – Сторінка авторизації

Після успішної авторизації користувача система автоматично перенаправляє на головну сторінку – дашборд. На початковому етапі, коли ще не створено власних проєктів і завдань, дашборд виглядає порожнім та містить лише стартовий проект, який служить орієнтиром для початку роботи.

На цій сторінці користувач бачить основні панелі, які в майбутньому відображатимуть загальну статистику по проєктах і завданнях: загальна кількість проєктів, завдань, призначених та виконаних задач, а також завдання з простроченими дедлайнами. Спеціальні блоки дозволяють швидко переходити до створення нового проекту або ознайомитися з поточними завданнями, які ще не призначені жодному користувачу.

Також на дашборді відображаються додаткові секції для списку учасників робочого простору, приватного нотатника та інших важливих елементів, які поки що порожні й наповнюються у процесі активної роботи.

Нижче наведено вигляд стартового дашборду після входу до системи:

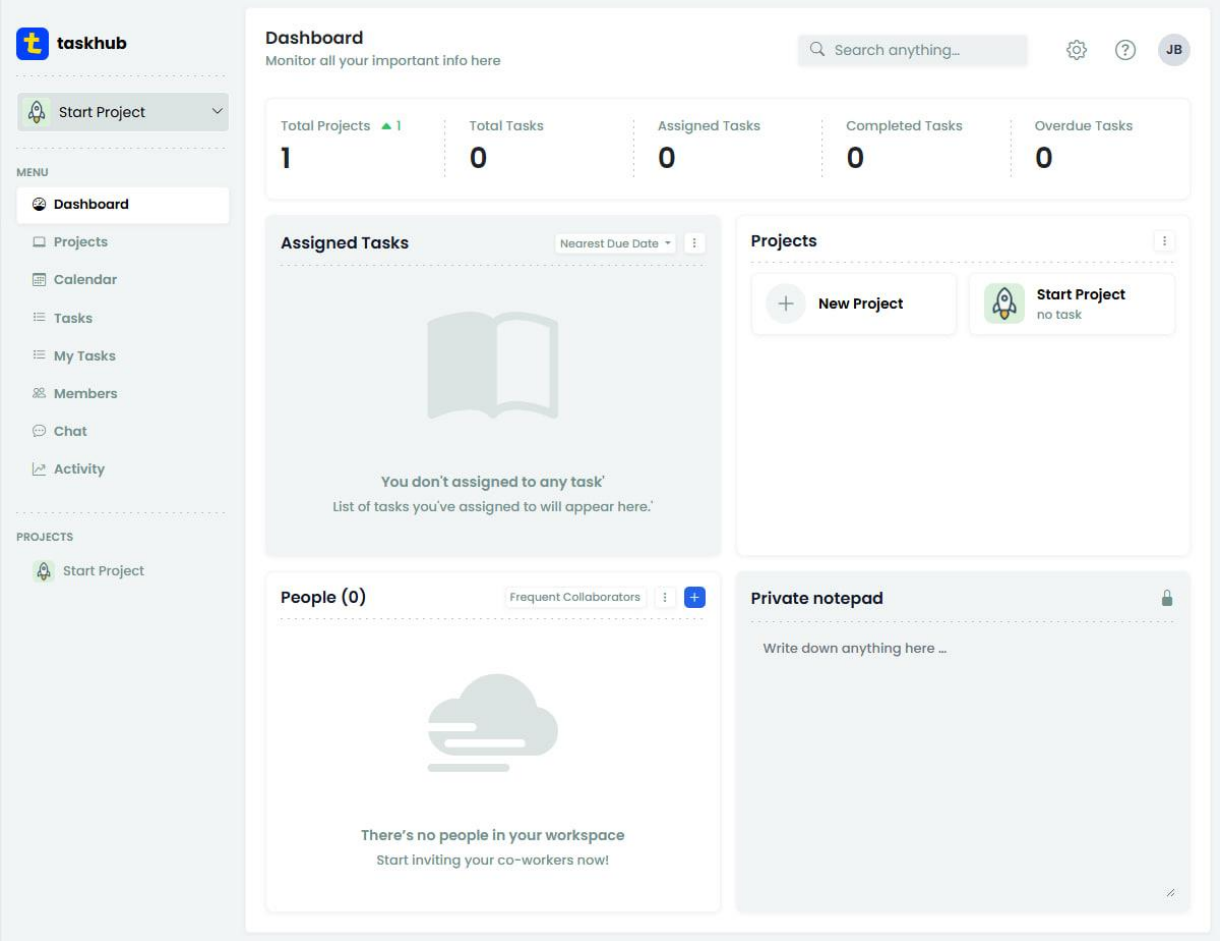


Рисунок 3.3 – Стартовий пустий дашборд

Далі користувач може приступати до роботи й створювати власні проєкти, використовуючи відповідну функцію на дашборді. Для цього достатньо натиснути кнопку “New Project” або перейти в пункт “Project” в навігаційній панелі зліва і вже там продовжити створення, після чого відкривається спеціальна форма для створення нового проєкту. У вікні створення необхідно вказати назву проєкту, додати короткий опис, а також за бажанням завантажити обкладинку (cover image), щоб зробити проєкт більш впізнаваним у загальному списку.

Після заповнення усіх необхідних полів потрібно натиснути кнопку “Confirm”, і новий проєкт буде доданий у робочий простір користувача. У подальшому він може редагувати, видаляти чи відкривати проєкт для детального перегляду та управління завданнями й учасниками.

Нижче представлено вигляд вікна створення нового проєкту:

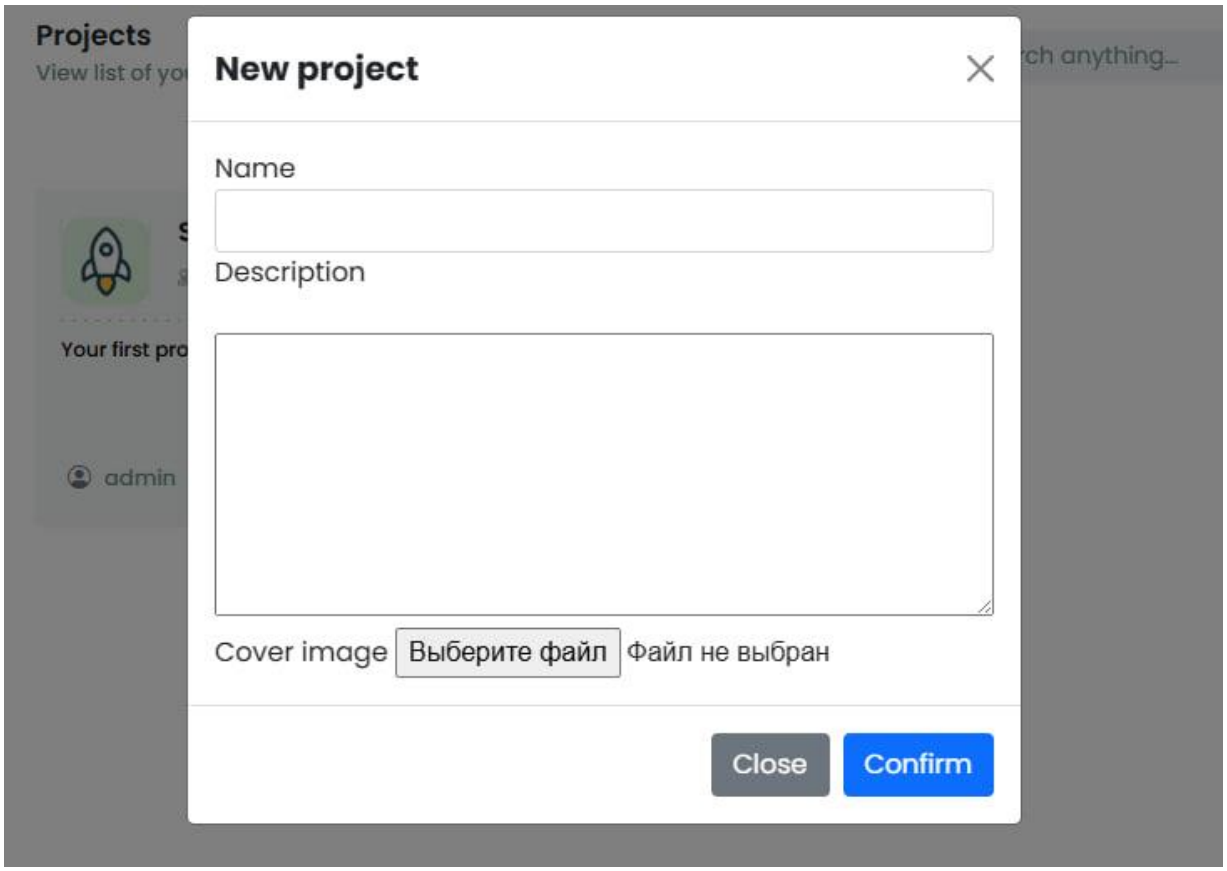


Рисунок 3.4 – Створення нового проєкту

Отже, після створення кількох проєктів на сторінці “Projects” відображається зручний перелік усіх доступних проєктів у вигляді карток. Кожна картка містить всю необхідну інформацію: назву проєкту, короткий опис, іконку або обкладинку, дату створення, ім’я адміністратора, а також кількість учасників і завдань у межах цього проєкту. Такий формат дозволяє швидко орієнтуватися у наявних проєктах та обирати потрібний для подальшої роботи.

Окрім основної інформації, на кожній картці розташовані функціональні кнопки – наприклад, “Details” для перегляду детальної інформації про проєкт, а також меню додаткових дій (редагування, видалення тощо). Це значно спрощує навігацію та управління проєктами, роблячи інтерфейс максимально зручним та інтуїтивним для користувача.

Нижче показано вигляд сторінки зі списком проєктів та приклади інформації, яка відображається на картках:

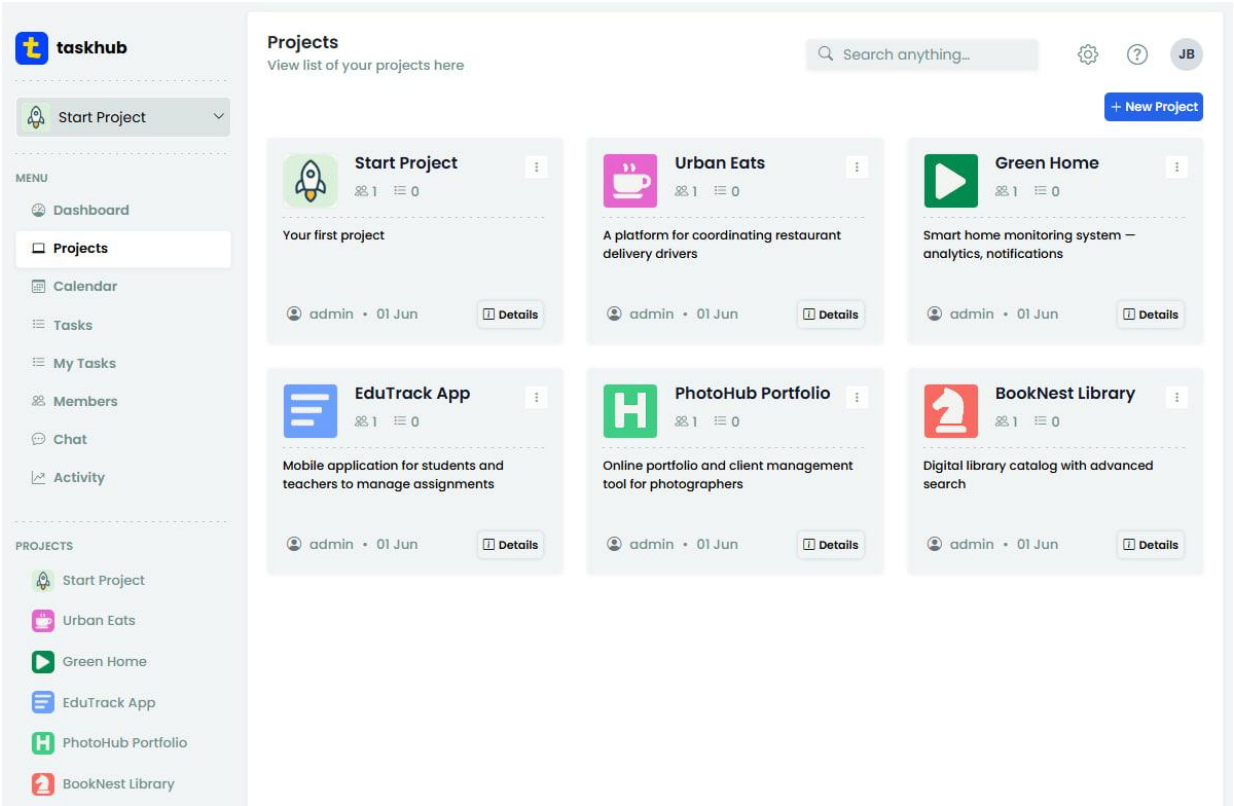


Рисунок 3.5 – Список проєктів

Для ефективного виконання проєкту доцільно створити команду, запросивши нових учасників, які братимуть участь у розподілі й виконанні завдань. За це відповідає спеціальна секція на дашборді, де відображається перелік учасників обраного проєкту. На даний момент список учасників порожній, про що свідчить відповідне повідомлення в інтерфейсі. Саме в цій секції згодом з'являться всі запрошені співробітники, які приєднаються до робочого простору. Нижче наведено вигляд цієї панелі в інтерфейсі застосунку:

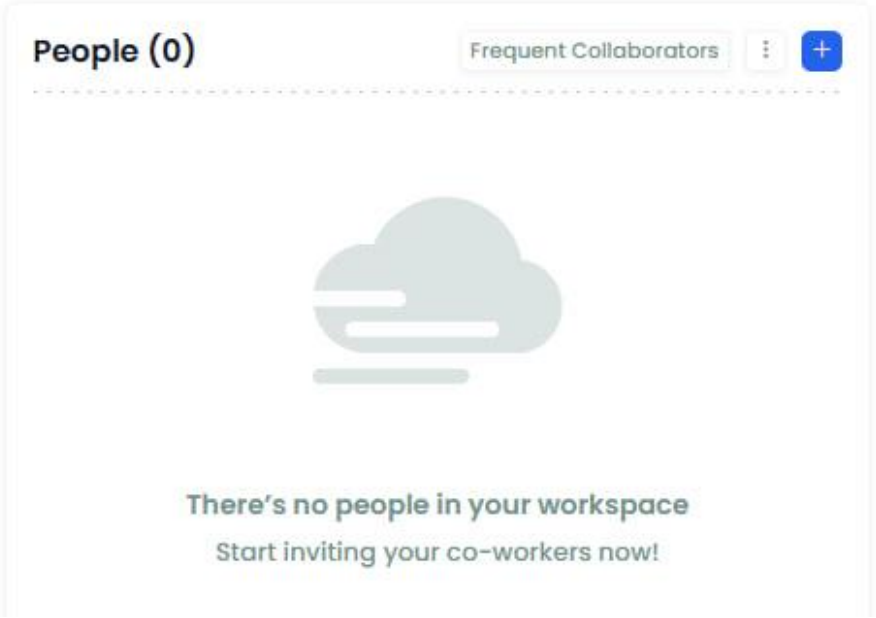


Рисунок 3.6 – Список учасників команди

Таким чином, запросимо декілька нових учасників. Щоб це зробити, достатньо натиснути синю кнопку з плюсом у правому верхньому куті цієї секції.

Після натискання кнопки, відкривається спеціальне вікно для запрошення нового користувача. У цьому діалоговому вікні необхідно ввести електронну адресу майбутнього учасника. Для зручності реалізовано автоматичний пошук: у міру введення частини адреси система підказує всі відповідні варіанти, показуючи користувачів зі схожими email у списку. Достатньо вибрати потрібну адресу та натиснути кнопку "Invite".

На зображенні нижче наведено вигляд цього вікна та підказок системи:

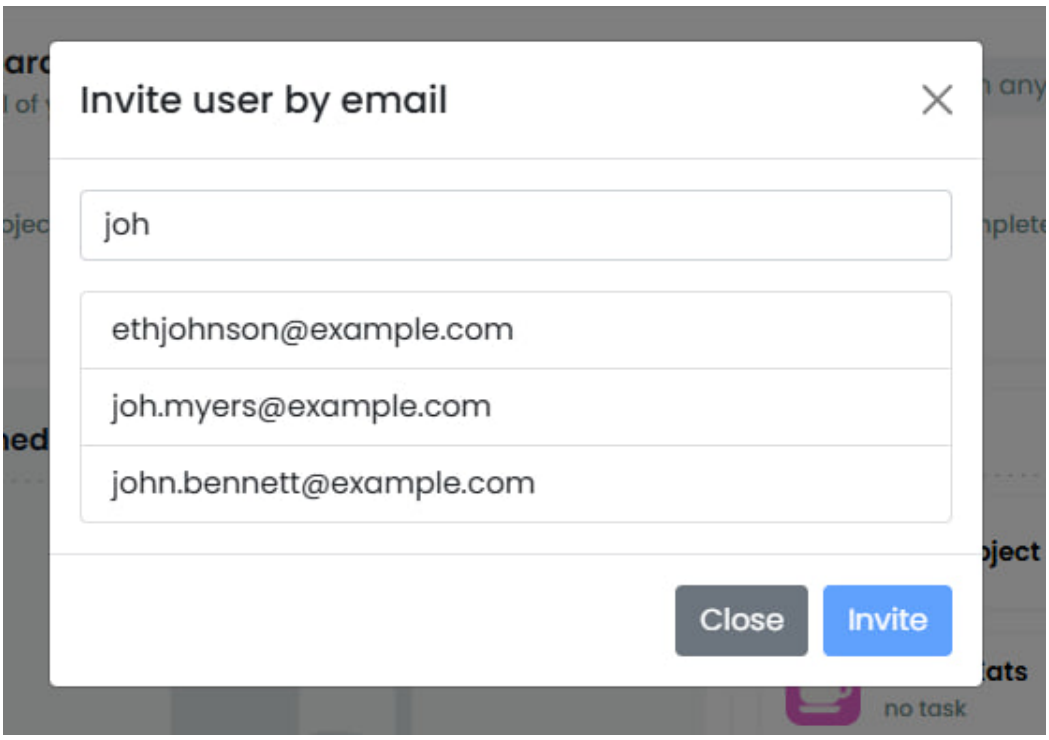


Рисунок 3.7 – Запрошення користувачів

Таким чином, після додавання нових учасників, отримуємо ось такий список, зображений на рисунку ?. Тут можна побачити імена, електронні адреси та аватари всіх учасників робочого простору, що значно спрощує управління командою та розподіл завдань.

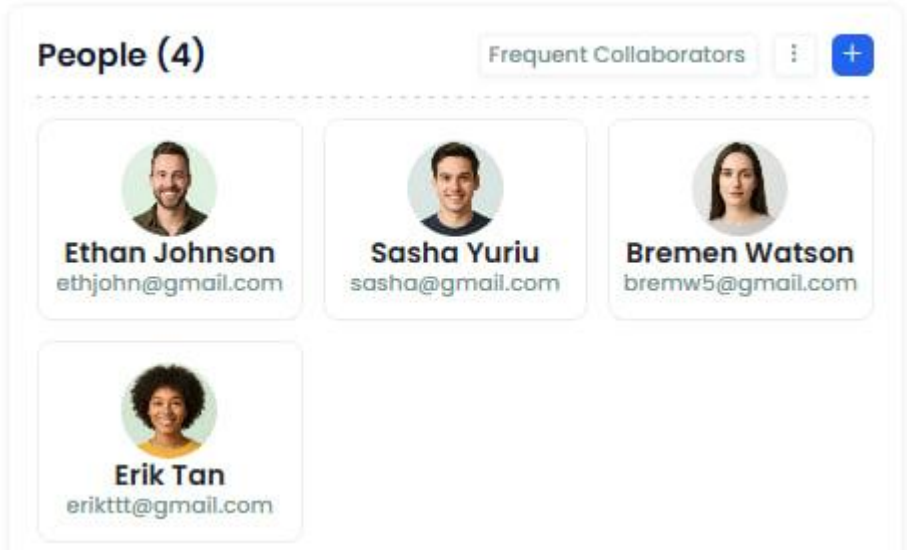


Рисунок 3.8 – Заповнений список учасників

Далі можна переходити до етапу створення завдань. Для цього необхідно відкрити пункт меню “Tasks” і натиснути кнопку для створення нового завдання. З’явиться вікно, в якому користувачеві потрібно заповнити основну інформацію про завдання: дату призначення, назву, опис, бажану дату виконання та статус. При необхідності можна призначити відповідальних виконавців, обравши їх зі списку учасників проєкту. Після заповнення всіх полів слід натиснути кнопку “Створити”, і нове завдання з’явиться у списку завдань назначених користувачів.

Нижче наведено приклади вигляду форми створення завдання:

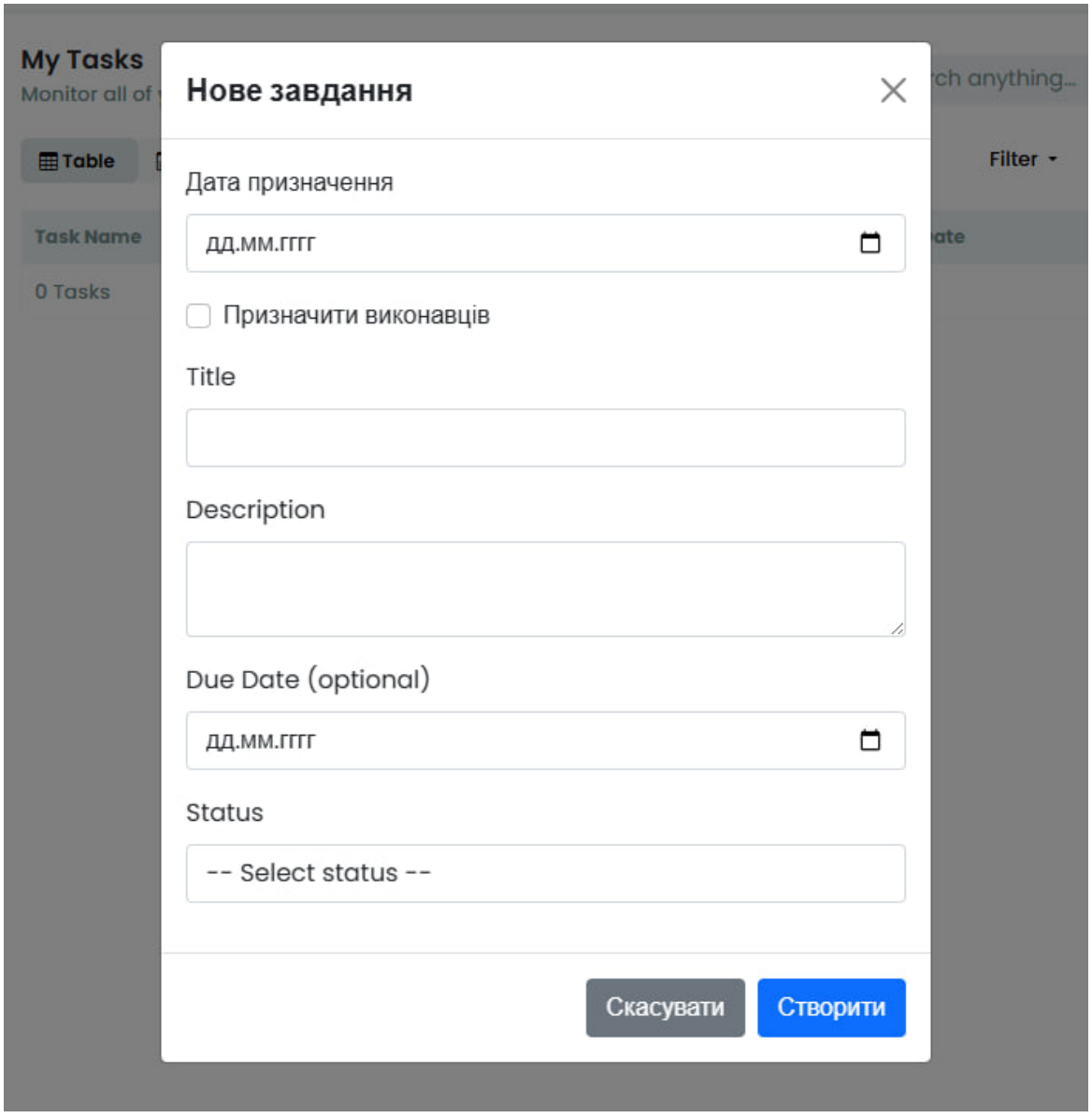


Рисунок 3.9 – Додавання нового завдання

Отже, після успішного створення завдань, можна переглянути їх у застосунку в двох основних форматах.

Перший – табличний вигляд, де всі завдання відображаються у вигляді структурованої таблиці зі стовпцями: назва задачі, проект, відповідальні виконавці, дата виконання та поточний статус.

Другий – календарний вигляд, який дає змогу візуально планувати роботу, відслідковувати дедлайни та розподіляти завдання у часі. Завдання автоматично відображаються у календарі на відповідних датах, що особливо зручно для командної роботи та ефективного менеджменту часу.

Нижче наведено приклади обох форматів перегляду завдань:

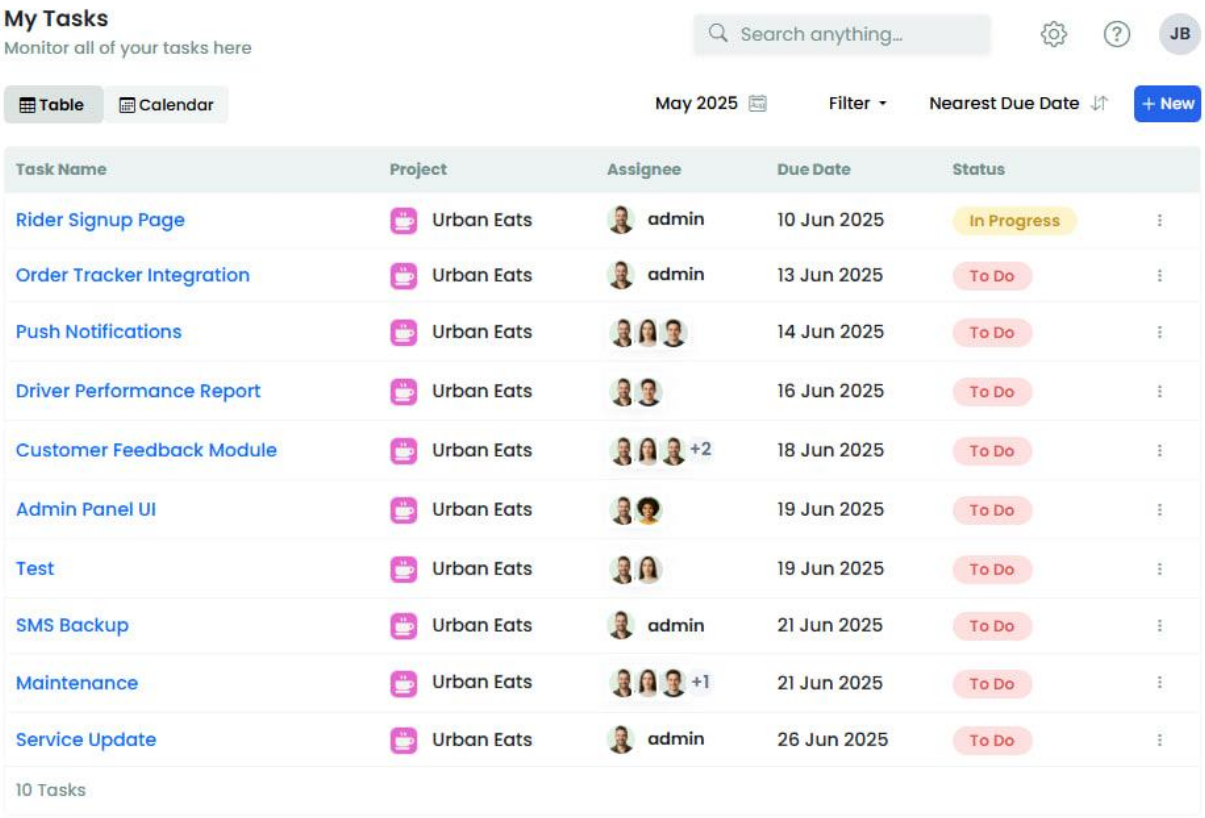


Рисунок 3.10 – Табличний вигляд завдань користувача

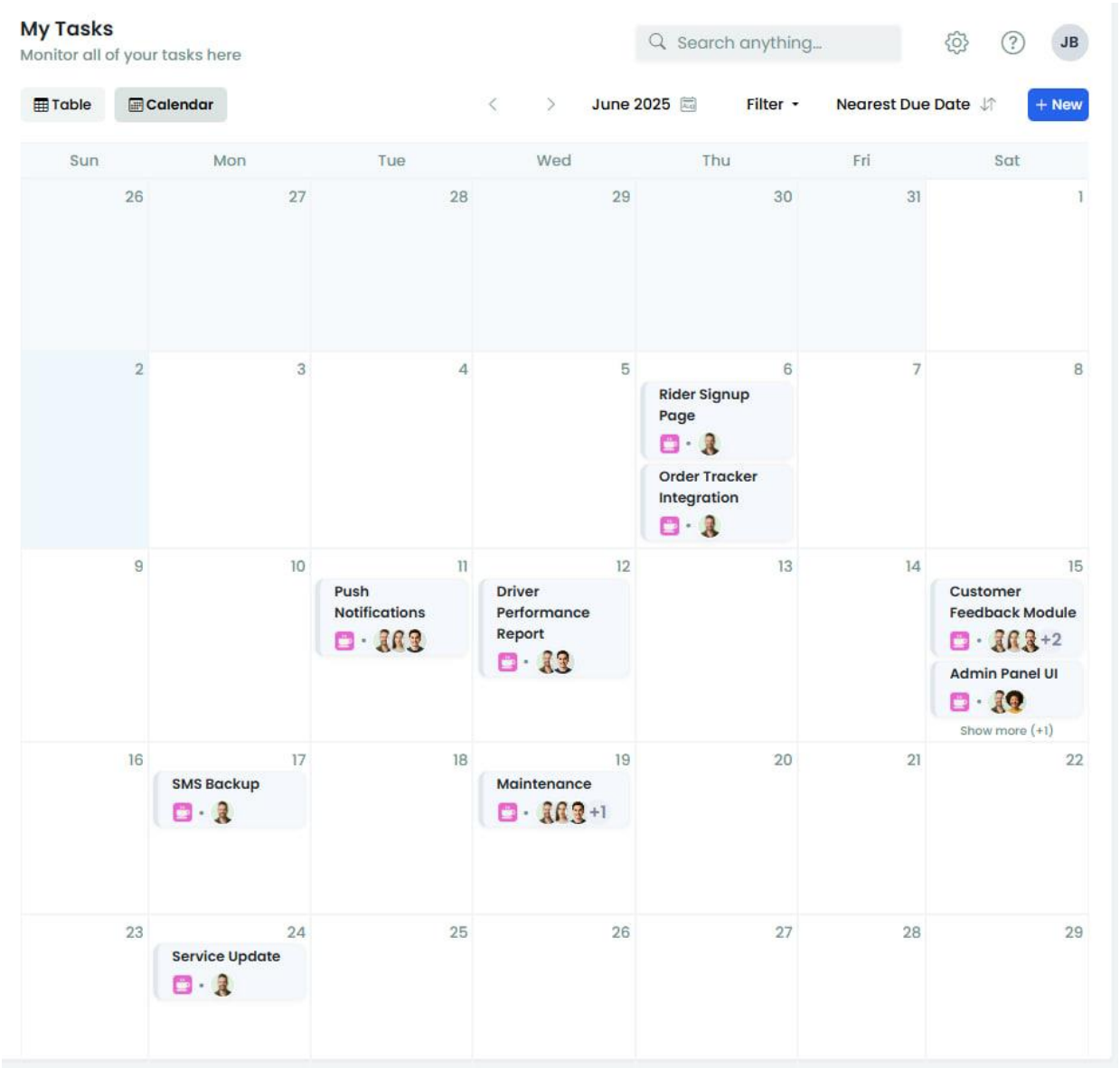


Рисунок 3.10 – Календарний вигляд завдань користувача

Також при перегляді списку завдань можна помітити, що назви завдань є інтерактивними посиланнями. При натисканні на назву користувач переходить на окрему сторінку завдання, де представлена вся детальна інформація, що стосується обраної задачі.

На цій сторінці відображаються основні дані: дати призначення та дедлайну, перелік відповідальних користувачів, а також зручні функціональні кнопки для редагування, видалення чи перепризначення виконавців.

Окремий блок містить інформацію про всіх призначених виконавців, де для

кожного учасника можна переглянути детальнішу інформацію, натиснувши на кнопку “Info”. Це дозволяє швидко зорієнтуватися у складі команди, побачити контактні дані чи ролі учасників, а також забезпечує повний контроль над процесом виконання задачі.

Нижче наведено приклад вигляду сторінки детального перегляду завдання:

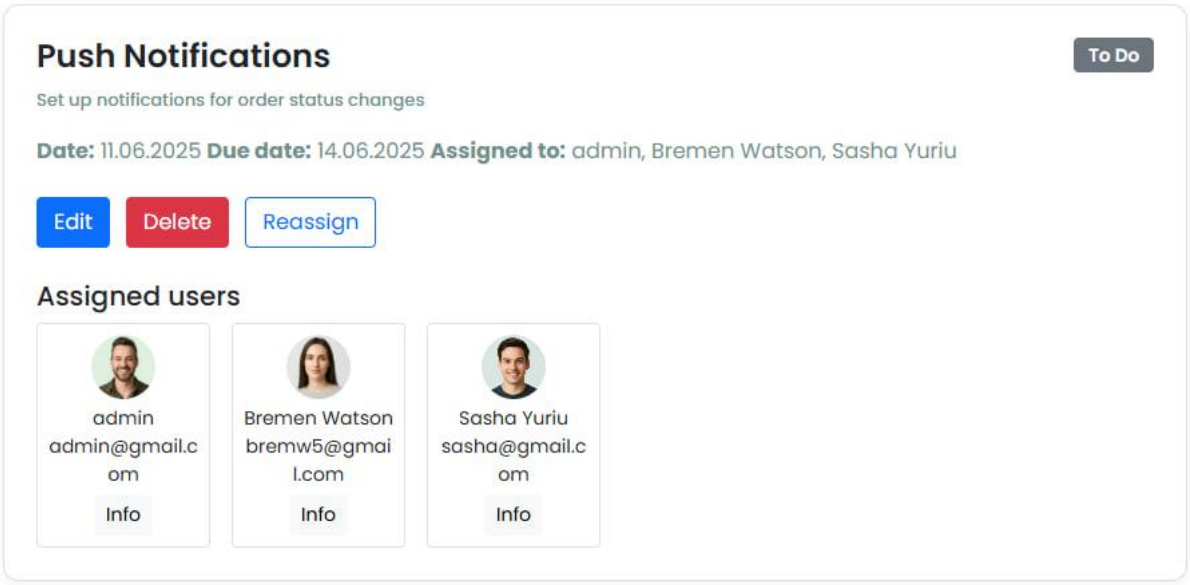


Рисунок 3.11 – Сторінка завдання

Перейшовши на сторінку учасника команди, користувач отримує доступ до детальної інформації про конкретного члена проєкту. Тут можна побачити його електронну адресу, поточну роль у команді (наприклад, «Member»), а також за потреби змінити роль за допомогою відповідної кнопки “Change Role”.

На цій сторінці також представлений перелік усіх завдань, які наразі призначені цьому учаснику. Це дозволяє швидко оцінити ступінь його залученості у виконання різних задач проєкту та переглянути статус кожного завдання. Крім того, у нижній частині сторінки передбачена кнопка для видалення користувача з проєкту (“Remove from Project”), що дозволяє адміністратору оперативно управляти складом команди у разі необхідності.

Нижче представлено приклад вигляду сторінки користувача:

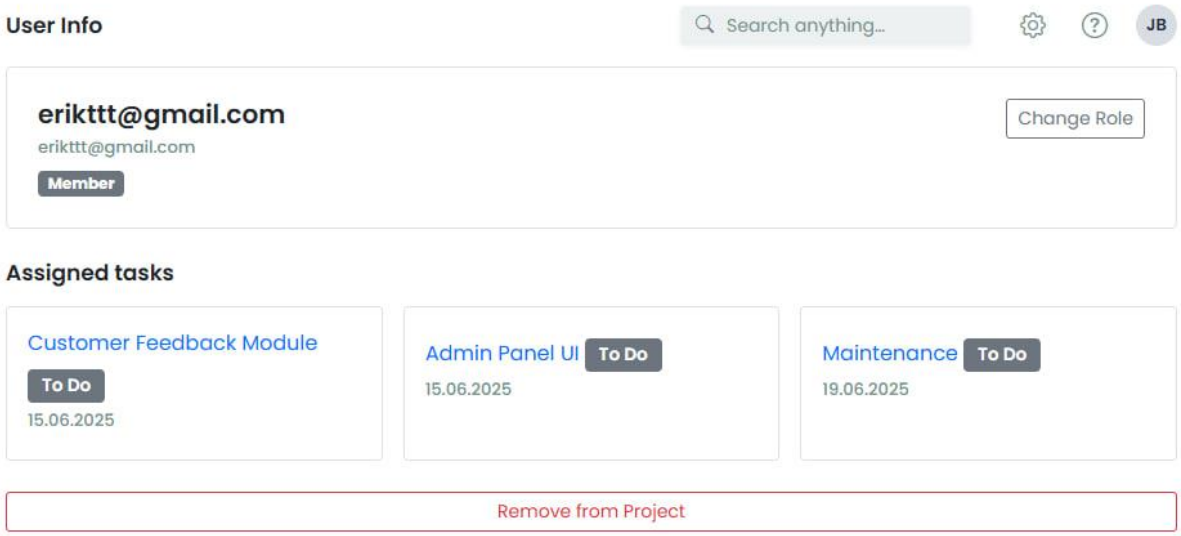


Рисунок 3.12 – Сторінка учасника команди

Якщо ж одному з учасників команди потрібно швидко передати інформацію, поставити запитання чи обговорити робочі моменти, для цього у системі передбачено вбудований чат (пункт меню “Chat”). Цей модуль є важливою складовою для підтримки ефективної командної взаємодії, адже дозволяє не лише спілкуватися з усією командою в загальному чаті, а й вести індивідуальні приватні розмови. Для цього достатньо обрати потрібного учасника зі списку контактів, який розміщений у лівій частині інтерфейсу, після чого відкрити відповідний діалог.

Всі повідомлення відображаються у реальному часі, що дозволяє оперативно координувати дії, передавати важливу інформацію та підтримувати командну взаємодію на високому рівні. Інтерфейс чату простий і зручний, а для надсилання повідомлення достатньо ввести текст та натиснути кнопку «Send».

Нижче зображено вигляд загального чату команди, що дозволяє ознайомитися зі структурою вікна, списком контактів та прикладом повідомлень у процесі командної комунікації.

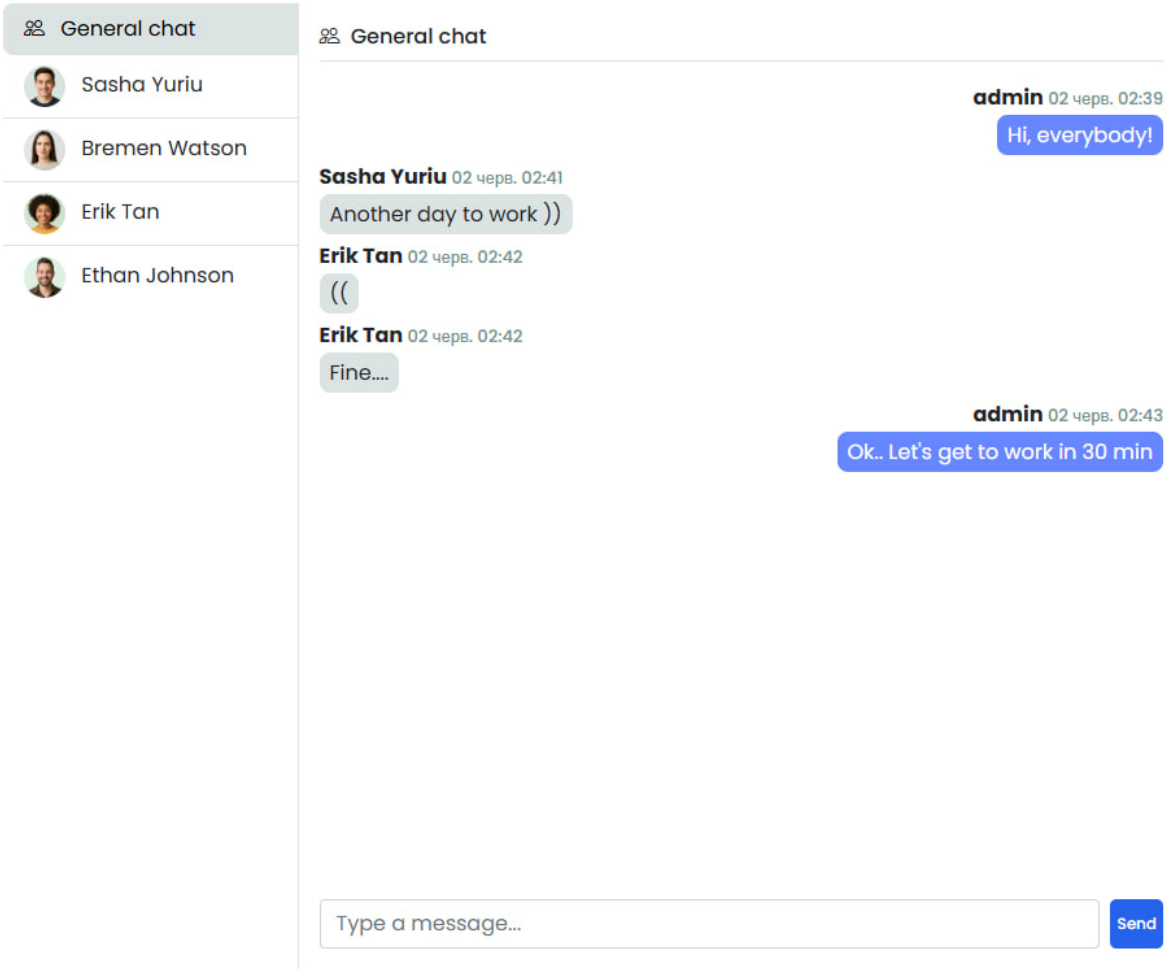


Рисунок 3.12 – Сторінка з чатом

При необхідності користувач має можливість переглянути детальну історію всіх активностей у проєкті. Для цього достатньо перейти до розділу меню “Activity”, де відображається повний журнал дій, що містить хронологічний перелік усіх важливих подій у рамках конкретного проєкту або навіть всієї системи. Журнал дій є важливим інструментом для аналізу командної роботи, контролю виконання завдань та виявлення можливих проблем чи неточностей у процесі співпраці. Завдяки цьому підходу можна підвищити рівень відповідальності учасників, мінімізувати ризики втрати важливої інформації та забезпечити аналітику для подальшого вдосконалення процесів управління проєктом.

Нижче на рисунку 3.12 показано приклад вигляду журналу дій.

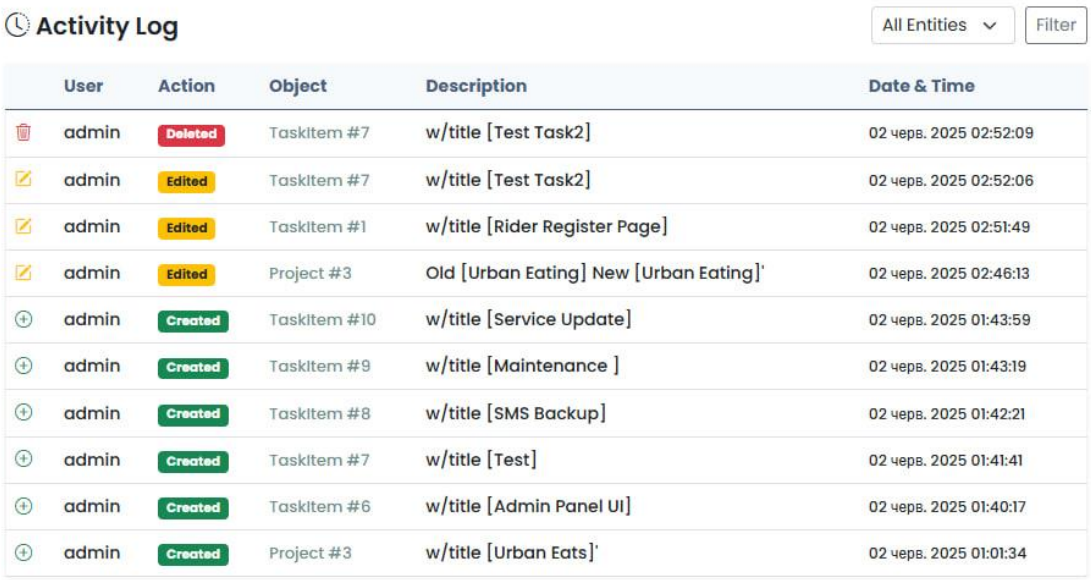


Рисунок 3.12 – Журнал дій

На завершення, користувач має можливість повноцінно керувати своїм обліковим записом через спеціальний розділ налаштувань. Тут він може змінити ім’я, електронну адресу, номер телефону, пароль, а також оновити фотографію профілю, завантаживши нове зображення.

Нижче представлено вигляду сторінки управління обліковим записом:

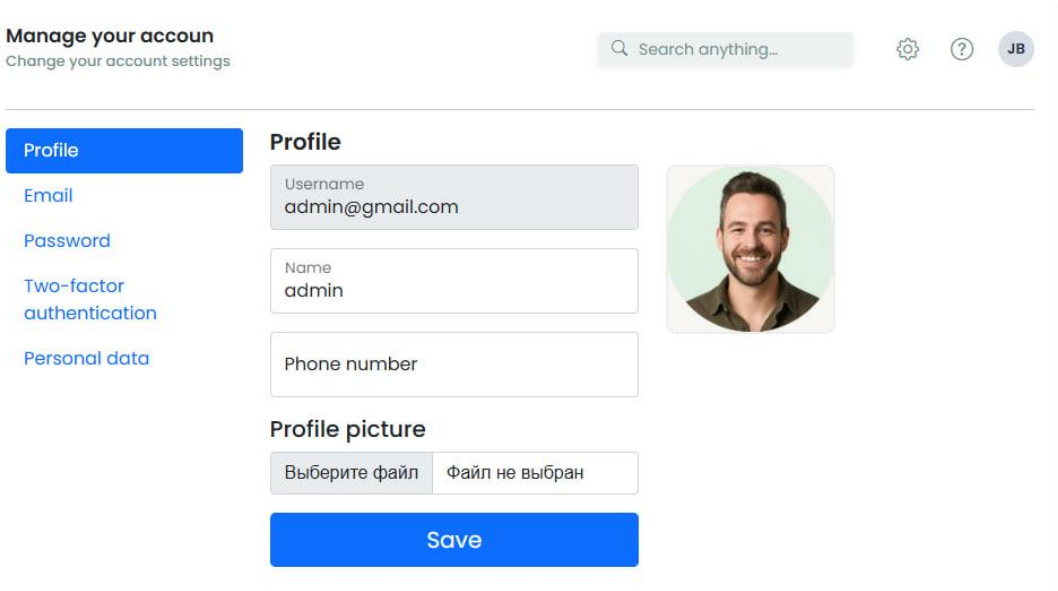


Рисунок 3.12 – Сторінка управління обліковим записом

3.5 Вимоги до технічних засобів

Успішне функціонування розробленого вебзастосунку можливе лише за умови наявності відповідних апаратних та програмних засобів, що забезпечують належний рівень продуктивності, стабільності та безпеки. Вимоги до технічного та програмного забезпечення стосуються як серверної, так і клієнтської частини системи.

Для серверної частини, на якій розміщується вебзастосунок, доцільно використовувати комп’ютер або віртуальний сервер із такими мінімальними технічними характеристиками:

– процесор не менше 2-х ядер із тактовою частотою від 2 ГГц (рекомендується Intel Core i3 або AMD Ryzen 3 і вище);

– оперативна пам’ять (RAM) не менше 4 ГБ (оптимально 8 ГБ і більше для забезпечення стійкої роботи серверу з кількома підключеннями);

– вільне місце на жорсткому диску (SSD або HDD) мінімум 10 ГБ для встановлення операційної системи, бази даних і збереження службових та користувацьких даних;

– мережевий інтерфейс підключення до локальної мережі або Інтернету зі швидкістю не менше 10 Мбіт/с для забезпечення стабільного обміну даними з клієнтами;

– резервне живлення (UPS) для забезпечення безперервної роботи у разі перебоїв із електропостачанням (рекомендується для промислового розгортання).

Для клієнтської частини (комп’ютер або мобільний пристрій кінцевого користувача) вимоги є більш лояльними, оскільки основна логіка виконується на сервері:

– процесор з 1 ядром, 1.5 ГГц і вище;

– оперативна пам’ять від 2 ГБ;

– вільний простір 100 МБ;

– мережевий доступ з постійним підключення до Інтернету;

– операційна система Windows, MacOS, Linux, Android, iOS (будь-яка сучасна версія);

– веб-браузер Google Chrome, Microsoft Edge, Mozilla Firefox, Safari – версії не старіше останніх двох років із підтримкою JavaScript, HTML5, CSS3;

– додаткове ПЗ не вимагається;

Завдяки застосуванню сучасних вебтехнологій і адаптивного дизайну, вебзастосунок не вимагає встановлення жодного спеціального клієнтського програмного забезпечення або додаткових модулів. Єдиною необхідною умовою є наявність сучасного браузера з підтримкою основних стандартів інтернету та стабільного мережевого підключення.

Усі вищенаведені вимоги забезпечують коректну, безпечну й ефективну роботу розробленого програмного продукту як у тестовому середовищі, так і при промисловому розгортанні для підтримки кількох десятків чи сотень активних користувачів одночасно. У випадку розширення функціоналу або масштабування системи доцільно відповідно підвищувати апаратні ресурси серверної інфраструктури та регулярно оновлювати програмні компоненти для гарантії безпеки та продуктивності.

ВИСНОВКИ

У результаті виконання бакалаврської роботи було реалізовано повний цикл створення сучасного вебзастосунку для організації особистих і командних завдань. На першому етапі детально проаналізовано предметну область, з'ясовано основні проблеми, з якими стикаються команди та окремі користувачі під час планування, контролю та виконання різноманітних задач. Проведено ґрунтовний огляд існуючих рішень, які представлені на ринку, здійснено їх порівняння за функціональними можливостями, інтерфейсом користувача, технологічною базою, перевагами і недоліками. На основі проведеного аналізу було зроблено висновок щодо доцільності створення власного програмного продукту, здатного забезпечити розширену підтримку персональних та командних робочих процесів, інтеграцію інструментів для співпраці, гнучке керування проектами, завданнями, повідомленнями та ролями учасників.

На етапі формалізації вимог до системи визначено чіткі функціональні та нефункціональні вимоги. Це дозволило закласти основи майбутнього застосунку: підтримка багатокористувацького режиму, забезпечення належного рівня безпеки (авторизація, аутентифікація, розмежування прав доступу), можливість інтеграції різних сервісів, гнучка організація робочих просторів, простота адміністрування, інтуїтивно зрозумілий і сучасний користувацький інтерфейс. Також було визначено вимоги до масштабованості системи, її адаптації під майбутні розширення функціоналу та легкості інтеграції з іншими корпоративними інструментами.

У процесі проектування архітектури було проведено порівняльний аналіз можливих підходів, після чого обґрунтовано вибір архітектури MVC як оптимальної для створення масштабованого, підтримуваного і структурованого вебзастосунку. В результаті розроблено комплексну структуру даних, описано моделі сутностей і їх зв'язки на рівні бази даних, що забезпечило узгоджене зберігання та обробку інформації. Додатково розроблено діаграми класів, що

відобразили логіку взаємодії між основними модулями та об’єктами системи.

Особливу увагу приділено деталізації проектування модулів – від механізмів авторизації й управління користувачами до організації проектів, завдань, календаря, чатів, історії активності, файлового менеджменту. Створені ViewModel забезпечили зручний зв’язок між даними та представленням, дозволяючи швидко і надійно відображати необхідну інформацію на сторінках інтерфейсу.

Під час вибору технологічного стеку проаналізовано найсучасніші інструменти для розробки вебдодатків. Для реалізації серверної логіки обрано мову програмування C# і платформу ASP.NET Core MVC, що забезпечують високу продуктивність, безпеку та гнучкість розробки. Для роботи з базою даних використано Entity Framework Core, що дозволило ефективно впровадити підхід Code First, здійснювати міграції структури даних та централізовано керувати змінами моделі. Для створення адаптивного та привабливого інтерфейсу застосовано Razor Pages, CSS, інтеграцію із дизайн-референсами з Figma, а також сучасні інструменти JavaScript – зокрема, jQuery та AJAX для підвищення інтерактивності.

На етапі програмної реалізації було впроваджено ключові модулі системи, описано їхню структуру, забезпечено взаємодію з базою даних, розроблено контролери для реалізації CRUD-операцій і додаткових сценаріїв використання. Окрему увагу приділено розробці та підключенню сервісів, що відповідальні за обробку файлів і логування дій користувачів. У межах реалізації було створено власний Activity Log для прозорості змін у системі, що позитивно впливає на безпеку та контроль над роботою команди.

Ретельно проведено процес налаштування та тестування програмного забезпечення, що дало змогу переконатися у працездатності й коректності реалізованого функціоналу. Завдяки юніт-тестам та покроковому проходженню основних сценаріїв роботи було ідентифіковано та усунуто всі критичні недоліки, а сам застосунок отримав стабільну і надійну основу для подальшого розвитку.

У межах інструкції користувача детально розглянуто основні можливості

системи, наведено послідовність дій, необхідних для виконання базових операцій: від реєстрації й налаштування профілю до створення проектів, призначення завдань, управління учасниками, спілкування у чаті та перегляду історії активностей. Це значно полегшує освоєння функціоналу навіть для нових користувачів і мінімізує ймовірність виникнення труднощів під час експлуатації.

Визначено ключові технічні та програмні вимоги для розгортання системи, що дозволяє інтегрувати застосунок у різні інфраструктури без втрати продуктивності чи функціональності.

Таким чином, у ході виконання цієї бакалаврської роботи було створено сучасний, функціональний, інтуїтивно зрозумілий і масштабований вебзастосунок для ефективної організації роботи як окремих користувачів, так і команд. Система відповідає усім актуальним вимогам до якості програмного продукту, а також закладає надійний фундамент для подальшого розширення можливостей і адаптації до потреб користувачів у динамічному цифровому середовищі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Як будувати UML діаграми. URL: https://dou.ua/forums/topic/40575/ (дата звернення 02.01.2025).

2. Що таке діаграма варіантів використання. URL: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/ (дата звернення 05.01.2025).

3. Functional vs. Non Functional Requirements. URL: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/ (дата звернення 10.01.2025).

4. Архітектура програмного забезпечення: все що треба знати. URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya (дата звернення 20.01.2025).

5. Монолітна архітектура ПЗ. URL: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/ (дата звернення 20.01.2025).

6. Клієнт-серверна архітектура. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення 20.01.2025).

7. Навіщо потрібна сервіс-орієнтована архітектура. URL: https://robotdreams.cc/uk/blog/268-zachem-nuzhna-servis-orientirovannaya-arhitektura (дата звернення 20.01.2025).

8. Мікросервісна архітектура: плюси та мінуси. URL: https://itedu.center/ua/blog/articles/microservices-architecture-advantages-and-disadventages/ (дата звернення 20.01.2025).

9. Model-View-Controller (MVC). URL: https://www.techtarget.com/whatis/definition/model-view-controller-MVC (дата звернення 03.02.2025).

10. Реляційні бази даних. URL: https://alexhost.com/uk/faq/relyaczijni-bazy-danyh-shho-cze-take-i-yak-vony-praczyuyut/ (дата звернення 06.02.2025).

11. Introduction of ER Model. URL: https://www.geeksforgeeks.org/introduction-of-er-model/ (дата звернення 06.02.2025).

12. Методологія IDEF0. URL: https://stud.com.ua/87184/-ekonomika/metodologiya_idef0 (дата звернення 11.02.2025).

13. Features and Benefits of the IDEF0. URL: https://www.informit.com/articles/article.aspx?p=2123714&seqNum=3 (дата звернення 11.02.2025).

14. What are context diagrams? URL: https://miro.com/blog/context-diagram/ (дата звернення 11.02.2025).

15. What is Visual Studio? URL: https://learn.microsoft.com/uk-ua/visualstudio/get-started/visual-studio-ide?view=vs-2022&viewFallbackFrom=vs-2022%20 (дата звернення 23.02.2025).

16. Why C#. URL: https://dotnet.microsoft.com/en-us/languages/csharp (дата звернення 23.02.2025).

17. ASP.NET Core. URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення 25.02.2025).

18. Entity Framework Core. URL: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx (дата звернення 28.02.2025).

19. Razor syntax reference for ASP.NET. URL: https://learn.microsoft.com/uk-ua/aspnet/core/mvc/views/razor?view=aspnetcore-9.0&viewFallbackFrom=aspnetcore-7.0%20 (дата звернення 02.03.2025).

20. What is LINQ? URL: https://www.tutorialsteacher.com/linq/what-is-linq (дата звернення 05.03.2025).

21. Що таке JQuery? URL: http://yoip.com.ua/shho-take-jquery/ (дата звернення 09.03.2025).

22. AJAX Introduction. URL: https://www.w3schools.com/Js/js_ajax_intro.asp (дата звернення 12.03.2025).

23. Using SQL Server Express. URL: https://blog.tooljet.ai/using-sql-server-express-a-comprehensive-guide/ (дата звернення 12.03.2025).

24. What is Code-First? URL: https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx (дата звернення 17.03.2025).

25. Що таке юніт-тестування. URL: https://pmtips.com.ua/qna/shcho-take-iunit-testuvannia-unit-testing (дата звернення 25.03.2025).

26. Moq – Unit Test in .NET Core App Using Mock Object. URL: https://www.c-sharpcorner.com/article/moq-unit-test-net-core-app-using-mock-object/ (дата звернення 26.03.2025).

27. How to use Moq and xUnit for Unit Testing Controllers in ASP.NET Core. URL: https://www.hosting.work/aspnet-core-moq-xunit-unit-testing/ (дата звернення 26.03.2025).

Додаток А

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

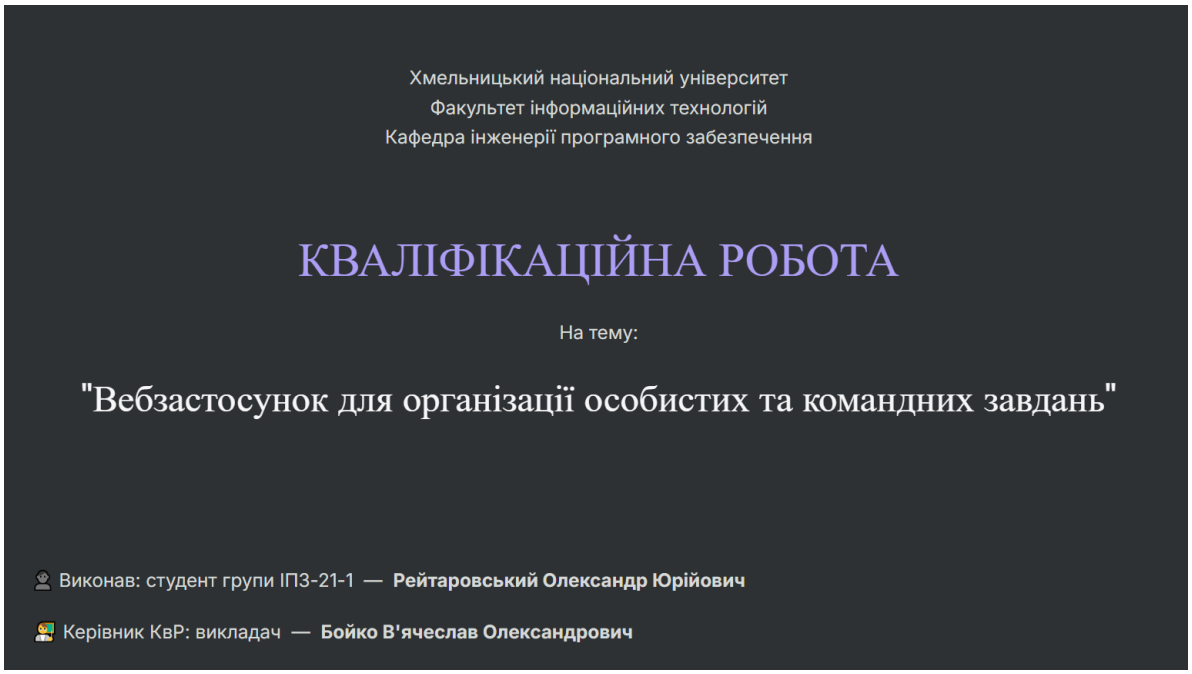


Рисунок А.1 – Слайд 1

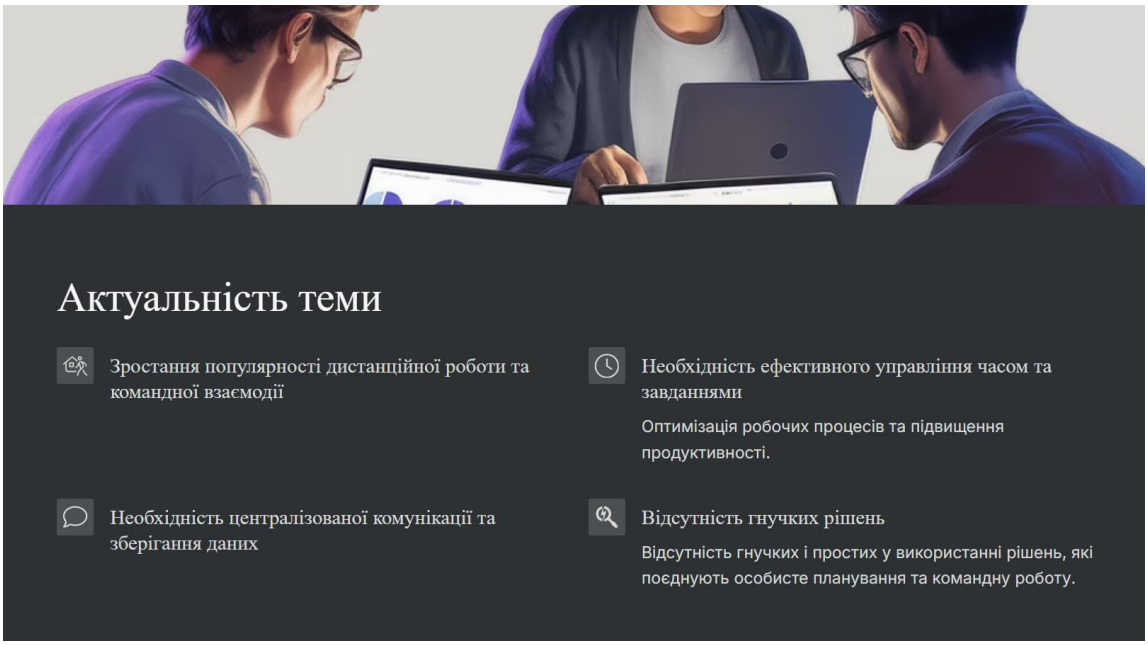


Рисунок А.2 – Слайд 2

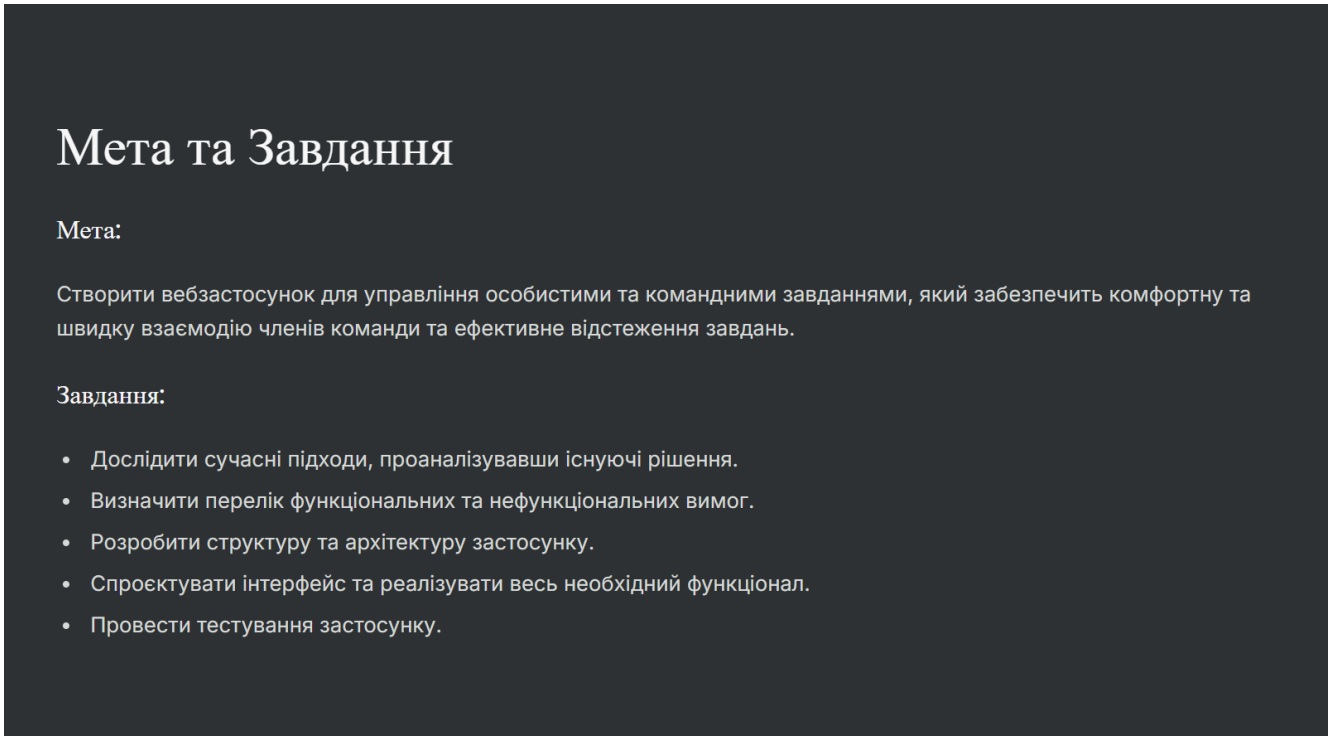


Рисунок А.3 – Слайд 3

Рисунок А.4 – Слайд 4

Аналіз наявного ПЗ

Критерії	Trello	Jira	Notion	Наш застосунок
Особисті задачі	✓ Є	× Немає	✓ Є	✓ Є
Командна взаємодія	✓ Є	✓ Є	✓ Є	✓ Є
Календар завдань	○ Обмежений	✓ Є	○ Обмежений	✓ Є
Комунікація в команді	× Немає	✓ Є	× Немає	✓ Є
Простота інтерфейсу	✓ Простий	× Складний	○ Середній	✓ Простий та інтуїтивний

Рисунок А.5 – Слайд 5

Вимоги до ПЗ

Функціональні:

- Повноцінна робота з обліковими записами
- Можливість створення та редагування завдань і проектів
- Інструменти взаємодії та комунікації з командою
- Система ролей та прав доступу
- Система нагадувань та нотифікацій
- Наявність особистого дашборда користувача

Нефункціональні:

- Інтуїтивно зрозумілий користувацький інтерфейс
- Швидкодія та стабільність роботи
- Можливість подальшого масштабування
- Забезпечення безпеки та конфіденційності даних користувачів

Рисунок А.6 – Слайд 6

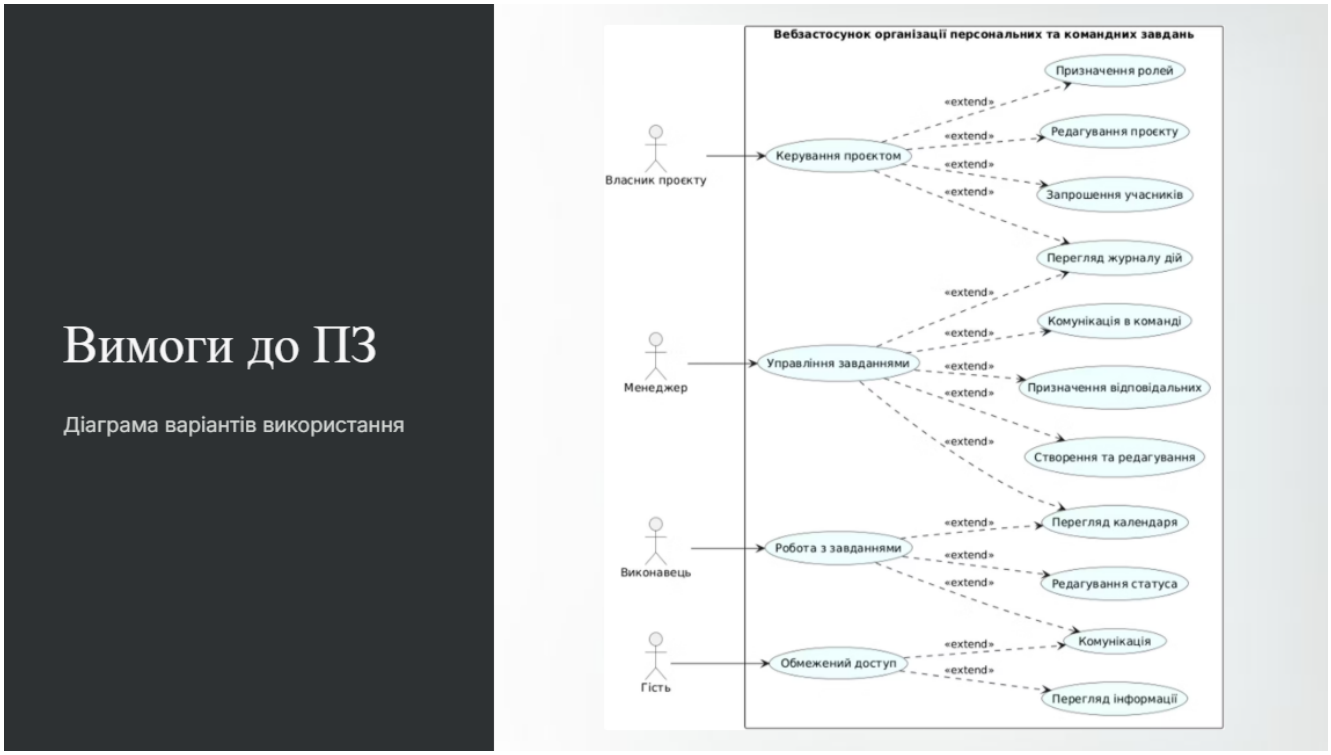


Рисунок А.7 – Слайд 7

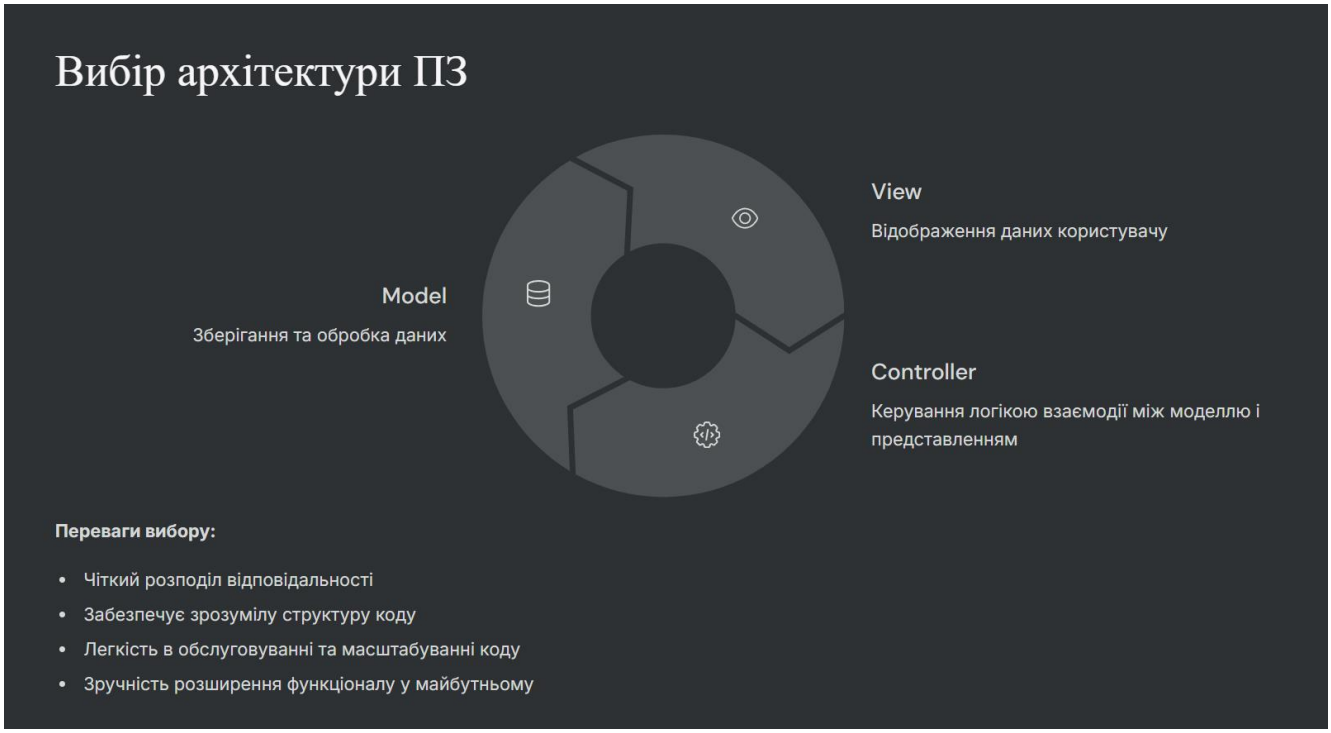


Рисунок А.8 – Слайд 8

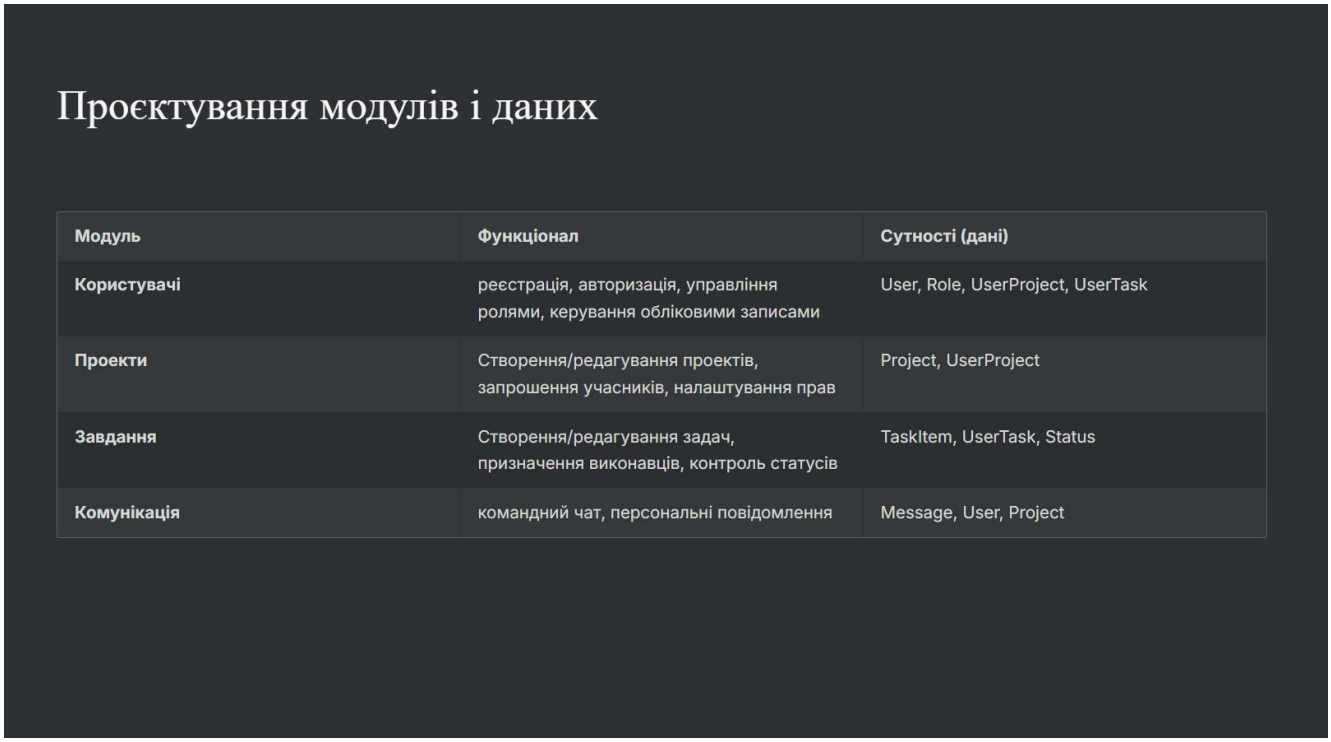


Рисунок А.9 – Слайд 9

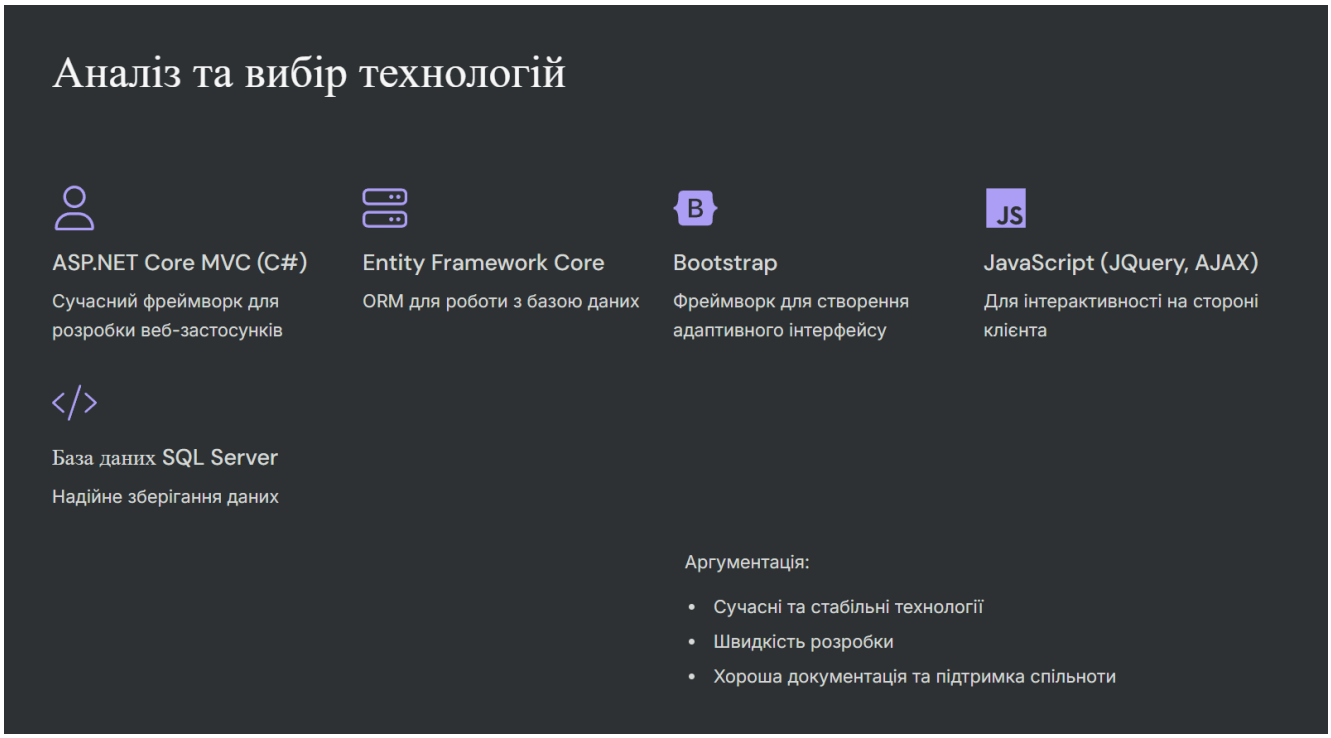


Рисунок А.10 – Слайд 10

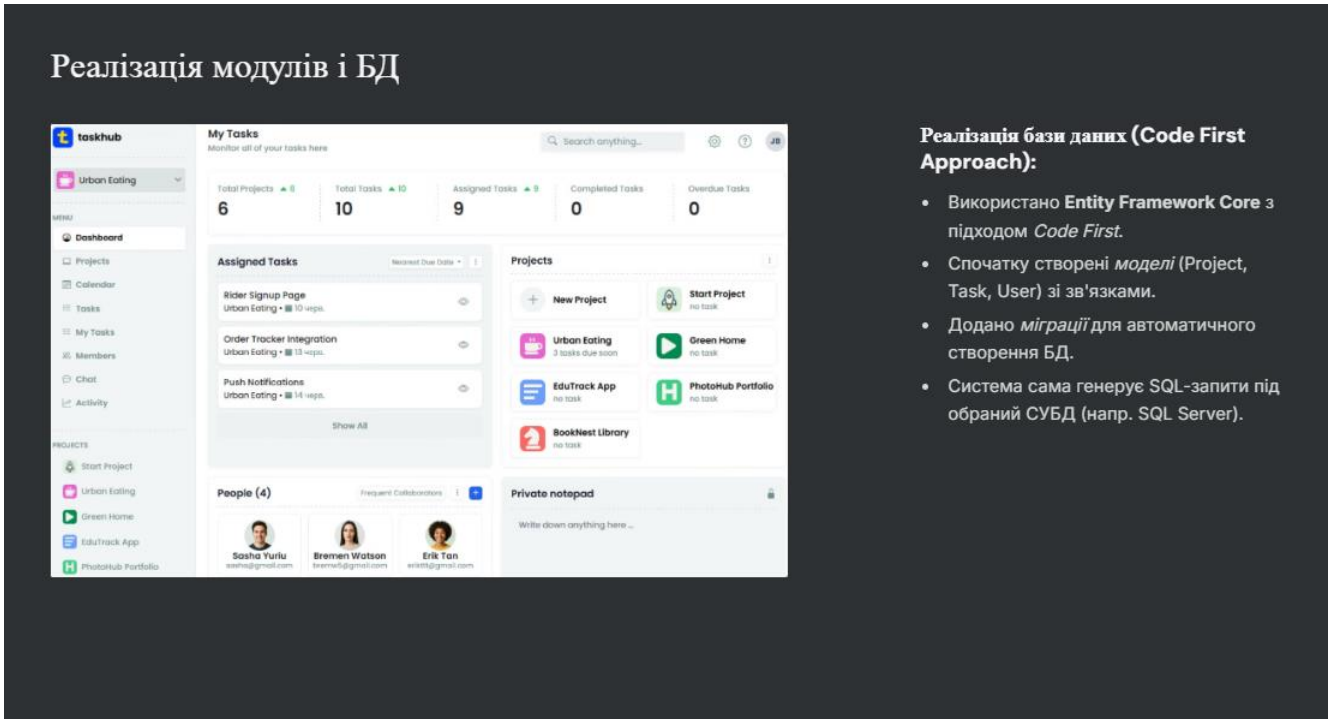


Рисунок А.11 – Слайд 11

Рисунок А.12 – Слайд 12

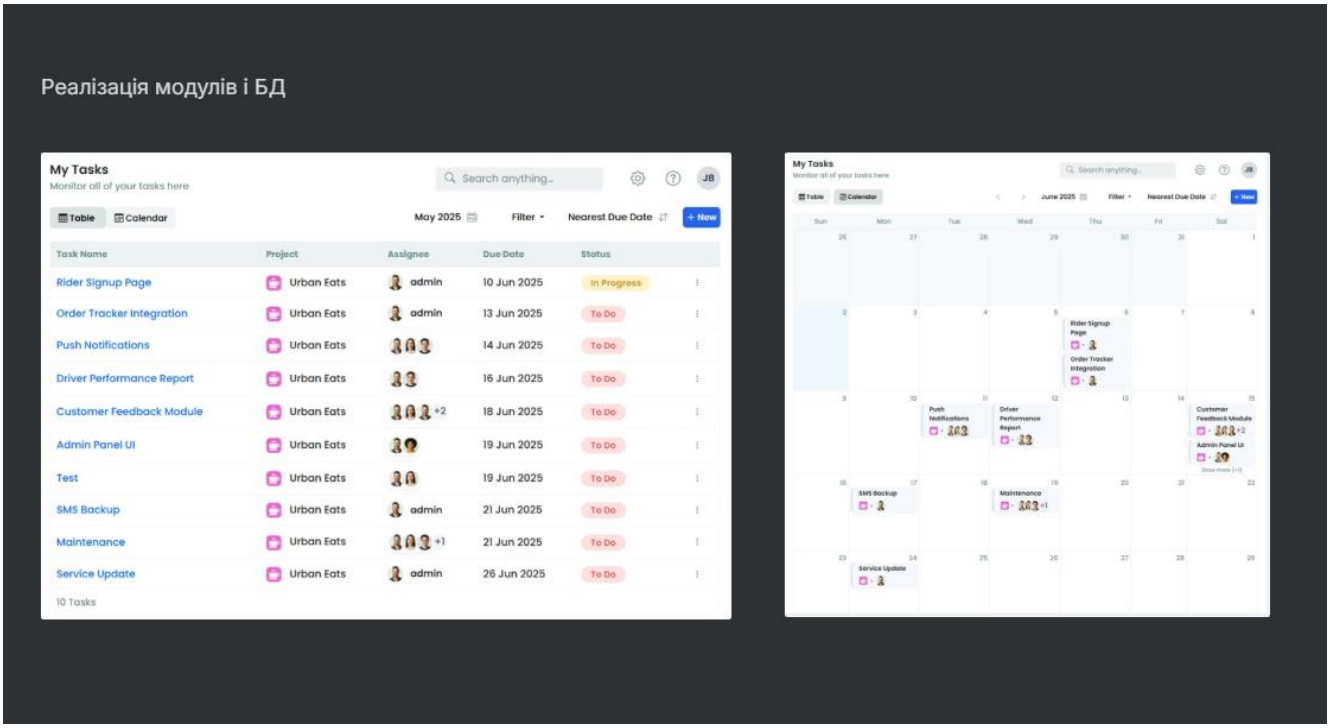


Рисунок А.13 – Слайд 13

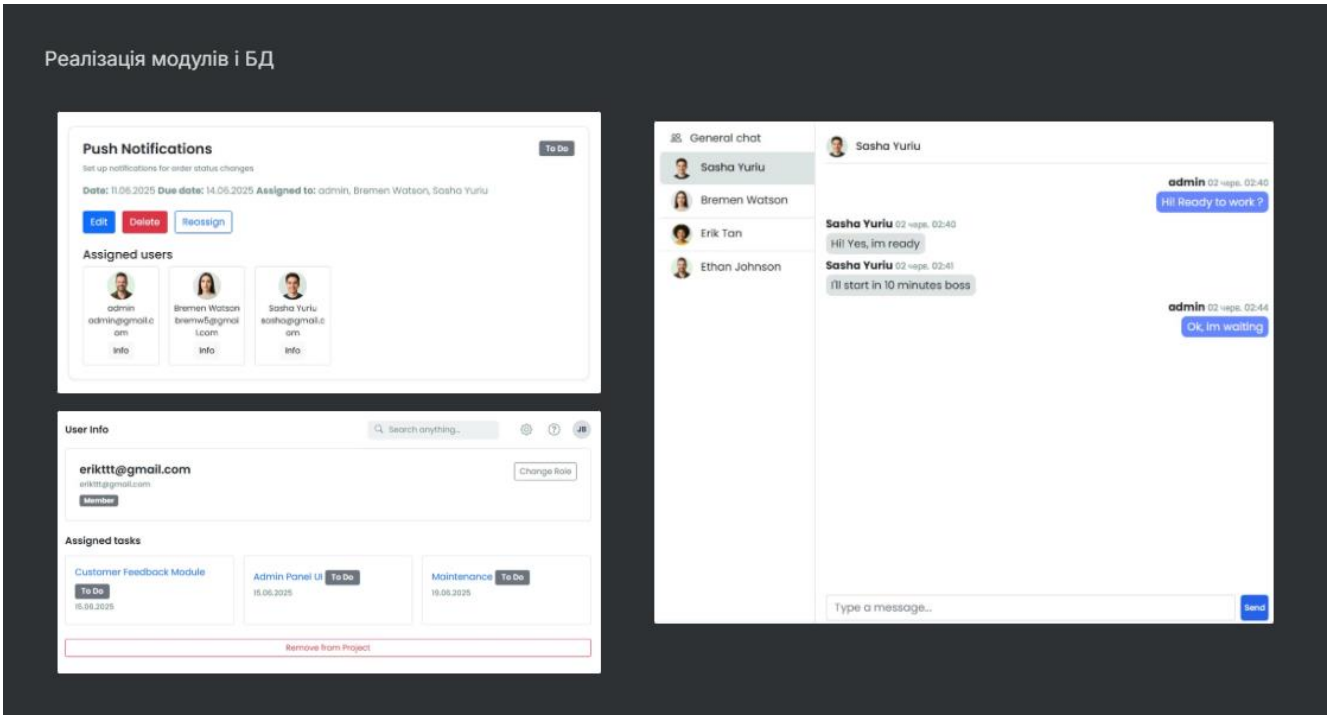


Рисунок А.14 – Слайд 14

Вимоги до технічного та програмного забезпечення

Вимога	Мінімальні характеристики	Рекомендовані характеристики
Операційна система	Windows 10, macOS 12+, Linux (з підтримкою .NET)	Windows 11, macOS 13+
Браузер	Chrome 100+, Firefox 100+, Edge 100+	Останні версії браузерів
Дисплей	1366×768 пікселів	1920×1080 або вище
Процесор	2-ядерний, 2.0 ГГц	4-ядерний, 2.5 ГГц або вище
Оперативна пам'ять	4 ГБ	8 ГБ для великих проектів
Місце на диску	500 МБ (для кешу/офлайн-роботи)	1 ГБ
Інтернет	Стабільне з'єднання (3G+)	

Рисунок А.15 – Слайд 15

Тестування ПЗ

У ході роботи було проведено як ручне, так і автоматизоване тестування ключових сценаріїв системи. Автоматичні тести охоплюють основну бізнес-логіку та перевіряють коректність роботи окремих модулів. Реалізоване логування дій дозволяє швидко ідентифікувати та виправляти можливі проблеми. За результатами тестування всі критичні функції працюють стабільно, що підтверджується успішним проходженням усіх тестів.

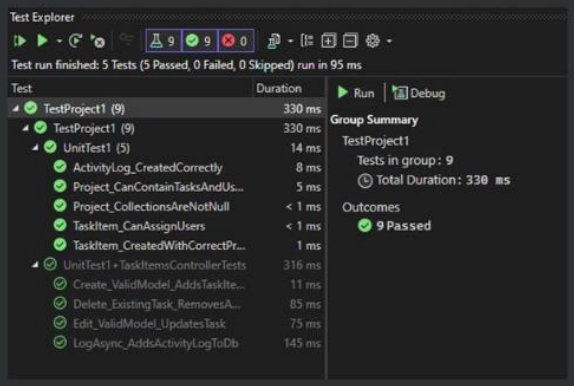


Рисунок А.16 – Слайд 16

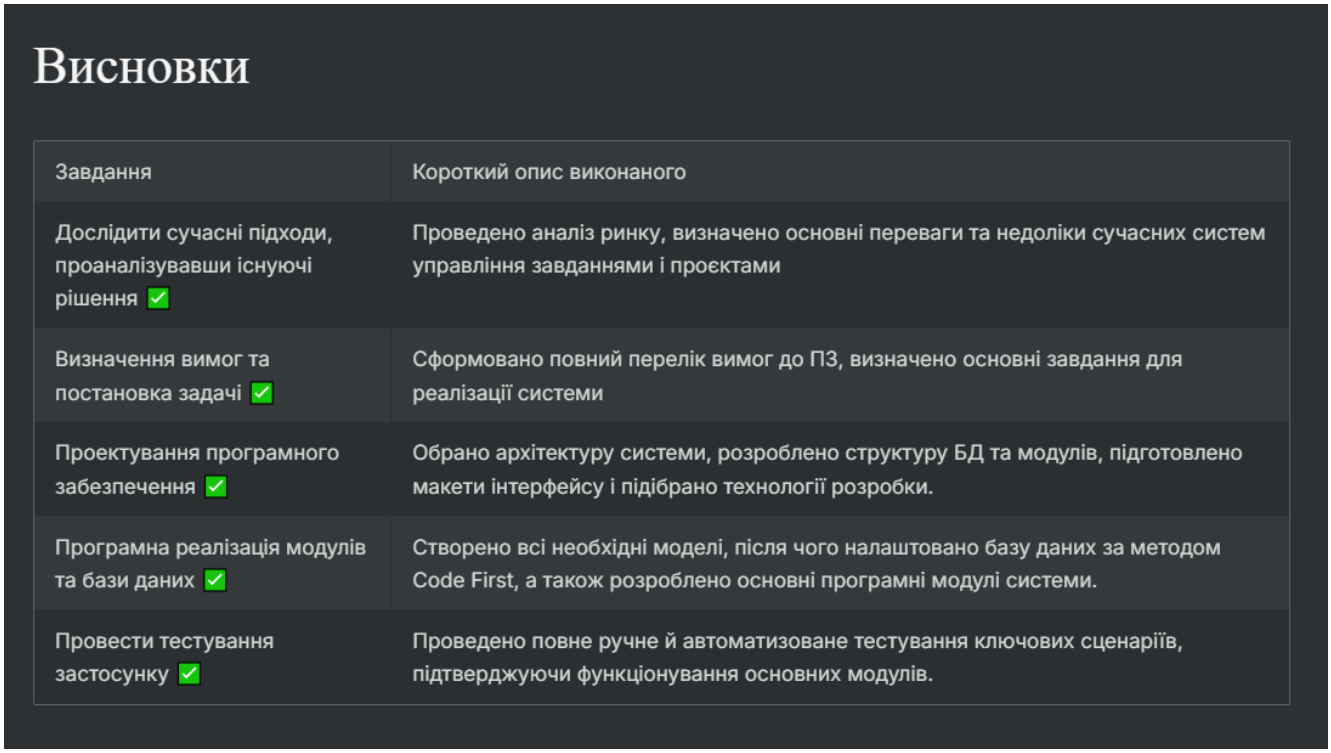


Рисунок А.17 – Слайд 17

Рисунок А.18 – Слайд 18

Додаток Б

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

Б.1 Код конфігурації Program.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using realexam.Data;
using realexam.Helpers;
using realexam.Models;
using realexam.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddIdentity<User, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultUI()
    .AddDefaultTokenProviders();

builder.Services.AddControllersWithViews();

builder.Services.AddScoped<IFileService, FileService>();
builder.Services.AddScoped<IActivityLogger, ActivityLogger>();
builder.Services.AddScoped<IAuthorizationHandler, ProjectRoleHandler>();

builder.Services.AddAuthorization(opts =>
{
    opts.AddPolicy("ProjectAdmin", p => p.Requirements.Add(new
ProjectRoleRequirement(ProjectRole.Admin)));
    opts.AddPolicy("ProjectManager", p => p.Requirements.Add(new
ProjectRoleRequirement(ProjectRole.Manager)));
    opts.AddPolicy("ProjectMember", p => p.Requirements.Add(new
ProjectRoleRequirement(ProjectRole.Member)));
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseMigrationsEndPoint();
```

```
}
else
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for
production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.MapRazorPages();

using (var scope = app.Services.CreateScope())
{
    await DbSeeder.SeedRolesAndAdminAsync(scope.ServiceProvider);
}


app.Run();
```

Б.2 Код контекста бази даних ApplicationDbContext.cs

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using realexam.Models;
using System.Reflection.Emit;

namespace realexam.Data
{
    public class ApplicationDbContext : IdentityDbContext<User>
    {
        public DbSet<Project> Projects { get; set; }
        public DbSet<ProjectUser> ProjectUsers { get; set; }
        public DbSet<TaskItem> TaskItems { get; set; }
        public DbSet<TaskItemUser> TaskItemUsers { get; set; }
        public DbSet<Message> Messages { get; set; }
        public DbSet<ActivityLog> ActivityLogs { get; set; }

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        protected override void OnModelCreating(ModelBuilder builder)
```

```
        {
            base.OnModelCreating(builder);

            builder.Entity<Project>()
                .HasOne(p => p.CreatedByUser)
                .WithMany()
                .HasForeignKey(p => p.CreatedByUserId)
                .OnDelete(DeleteBehavior.Restrict);

            builder.Entity<TaskItemUser>()
                .HasKey(tu => new { tu.TaskItemId, tu.UserId });

            builder.Entity<TaskItemUser>()
                .HasOne(tu => tu.TaskItem)
                .WithMany(t => t.TaskItemUsers)
                .HasForeignKey(tu => tu.TaskItemId);

            builder.Entity<TaskItemUser>()
                .HasOne(tu => tu.User)
                .WithMany(u => u.TaskItemUsers)
                .HasForeignKey(tu => tu.UserId);

            builder.Entity<ActivityLog>()
    .HasOne(a => a.Project)
    .WithMany(p => p.ActivityLogs)
    .HasForeignKey(a => a.ProjectId)
    .OnDelete(DeleteBehavior.Cascade);

        }
    }
}
```

Б.3 Код моделі User.cs

```
using Microsoft.AspNetCore.Identity;

namespace realexam.Models
{
    public class User : IdentityUser
    {
        public string? Name { get; set; }
        public string? ProfilePicture { get; set; }

        public ICollection<ProjectUser> ProjectUsers { get; set; } = new
List<ProjectUser>();
        public ICollection<TaskItemUser> TaskItemUsers { get; set; } = new
List<TaskItemUser>();

    }
}
```

Б.4 Код моделі Project.cs

```
using Microsoft.AspNetCore.Mvc.ModelBinding;
using System.ComponentModel.DataAnnotations.Schema;

namespace realexam.Models
{
    public class Project
    {
        public int Id { get; set; }
        public string Title { get; set; } = string.Empty;
        public string? Description { get; set; }
        public string? ImagePath { get; set; }
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
        public string CreatedByUserId { get; set; } = string.Empty;

        [ForeignKey("CreatedByUserId")]
        public User? CreatedByUser { get; set; }

        public ICollection<ProjectUser> ProjectUsers { get; set; } = new
List<ProjectUser>();
        public ICollection<TaskItem> TaskItems { get; set; } = new
List<TaskItem>();
        public ICollection<ActivityLog> ActivityLogs { get; set; } = new
List<ActivityLog>();
    }

}
```

Б.5 Код моделі TaskItem.cs

```
namespace realexam.Models
{
    public class TaskItem
    {
        public int Id { get; set; }
        public string Title { get; set; } = string.Empty;
        public string? Description { get; set; }

        public DateTime Date { get; set; }
        public DateTime? DueDate { get; set; }
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public string? Status { get; set; }

        public int ProjectId { get; set; }
        public Project? Project { get; set; }

        public string? AssignedUserId { get; set; }
        public User? AssignedUser { get; set; }
```

```
        public ICollection<TaskItemUser> TaskItemUsers { get; set; } = new
List<TaskItemUser>();
    }}
```

Б.6 Код контролера ProjectsController.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Newtonsoft.Json.Linq;
using realexam.Data;
using realexam.Helpers;
using realexam.Models;
using realexam.Services;

namespace realexam.Controllers
{
    public class ProjectsController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly UserManager<User> _userManager;
        private readonly IWebHostEnvironment _webHostEnvironment;
        private readonly IAuthorizationService _authorizationService;
        private readonly IActivityLogger _activityLogger;

        public ProjectsController(ApplicationDbContext context,
            UserManager<User> userManager, IWebHostEnvironment env,
IAuthorizationService auth, IActivityLogger activityLogger)
        {
            _context = context;
            _userManager = userManager;
            _webHostEnvironment = env;
            _authorizationService = auth;
            _activityLogger = activityLogger;
        }

        // GET: Projects
        public async Task<IActionResult> Index()
        {
            var userId = _userManager.GetUserId(User);

            var projects = await _context.Projects
                .Include(p => p.CreatedByUser)
                .Include(p => p.TaskItems)
                .Include(p => p.ProjectUsers)
                .Where(p => p.ProjectUsers.Any(pu => pu.UserId == userId))
                .ToListAsync();

            return View(projects);
        }
```

```
        public async Task<IActionResult> Details(int id)
        {
            // 1) Поточний проєкт
            var project = await _context.Projects
                .Include(p => p.ProjectUsers).ThenInclude(pu => pu.User)
                .FirstOrDefaultAsync(p => p.Id == id);
            if (project == null) return NotFound();

            // 2) Всі таски цього проєкту
            var allTasks = await _context.TaskItems
                .Where(t => t.ProjectId == id)
                .Include(t => t.TaskItemUsers)
                .ToListAsync();

            // 3) Мої три найважливіші таски
            var userId = _userManager.GetUserId(User);
            var myTasks = allTasks
                .Where(t => t.TaskItemUsers.Any(tu => tu.UserId == userId))
                .OrderBy(t => t.DueDate ?? DateTime.MaxValue)
                .Take(3)
                .ToList();

            // 4) Проєкти, в яких я учасник (для сайдбару)
            var userProjects = await _context.Projects
                .Where(p => p.ProjectUsers.Any(pu => pu.UserId == userId))
                .ToListAsync();

            // 5) Учасники поточного проєкту
            var members = project.ProjectUsers.Select(pu => pu.User!).ToList();

            // 6) Робимо дати для порівняння
            var today = DateTime.UtcNow.Date;
            var yesterday = today.AddDays(-1);

            // 7) Розрахунок по тасках
            int tasksToday = allTasks.Count(t => t.CreatedAt.Date == today);
            int tasksYesterday = allTasks.Count(t => t.CreatedAt.Date ==
yesterday);
            int deltaTasks = tasksToday - tasksYesterday;

            int assignedToday = allTasks.Count(t => t.TaskItemUsers.Any()
                                            && t.CreatedAt.Date == today);
            int assignedYesterday = allTasks.Count(t => t.TaskItemUsers.Any()
                                           && t.CreatedAt.Date == yesterday);
            int deltaAssigned = assignedToday - assignedYesterday;

            int completedToday = allTasks.Count(t => t.Status == "Done"
                                           && t.CreatedAt.Date == today);
            int completedYesterday = allTasks.Count(t => t.Status == "Done"
                                           && t.CreatedAt.Date == yesterday);
            int deltaCompleted = completedToday - completedYesterday;

            int overdueToday = allTasks.Count(t => t.DueDate < today
                                            && t.Status != "Done"
                                            && t.CreatedAt.Date == today);
            int overdueYesterday = allTasks.Count(t => t.DueDate < yesterday
                                           && t.Status != "Done"
                                           && t.CreatedAt.Date == yesterday);
            int deltaOverdue = overdueToday - overdueYesterday;

            // 8) Розрахунок по проєктах (за CreatedAt)
```

```
            int projectsToday = await _context.Projects.CountAsync(p =>
p.CreatedAt.Date == today);
            int projectsYesterday = await _context.Projects.CountAsync(p =>
p.CreatedAt.Date == yesterday);
            int deltaProjects = projectsToday - projectsYesterday;

            // 9) Складаємо список карток
            var stats = new List<StatCardVm>
            {
                new StatCardVm("Total Projects",    userProjects.Count,
deltaProjects),
                new StatCardVm("Total Tasks",       allTasks.Count,
deltaTasks),
                new StatCardVm("Assigned Tasks",
allTasks.Count(t=>t.TaskItemUsers.Any()), deltaAssigned),
                new StatCardVm("Completed Tasks",
allTasks.Count(t=>t.Status=="Done"),       deltaCompleted),
                new StatCardVm("Overdue Tasks",     allTasks.Count(t=>t.DueDate <
today && t.Status!="Done"), deltaOverdue),
            };

            // 10) Готуємо VM
            var vm = new DashboardViewModel
            {
                CurrentProject = project,
                AllTasks = allTasks,
                MyTasks = myTasks,
                UserProjects = userProjects,
                Members = members,
                Stats = stats
            };

            // 11) Зберігаємо cookie та ViewBag для сайдбару
            Response.Cookies.Append("currentProjectId", id.ToString());
            ViewBag.ProjectId = id;

            return View(vm);
        }

        [HttpPost]
        public async Task<IActionResult> Create(Project project, IFormFile?
ImageFile)
        {
            if (ModelState.IsValid)
            {
                if (ImageFile != null && ImageFile.Length > 0)
                {
                    var uploadsFolder =
Path.Combine(_webHostEnvironment.WebRootPath, "uploads");
                    Directory.CreateDirectory(uploadsFolder);

                    var uniqueFileName = Guid.NewGuid().ToString() +
Path.GetExtension(ImageFile.FileName);
                    var filePath = Path.Combine(uploadsFolder, uniqueFileName);

                    using (var fileStream = new FileStream(filePath,
FileMode.Create))
                    {
                        await ImageFile.CopyToAsync(fileStream);
                    }
```

```
                    project.ImagePath = "/uploads/" + uniqueFileName;
                }

                var user = await _userManager.GetUserAsync(User);

                project.CreatedAt = DateTime.UtcNow;
                project.CreatedByUserId = user.Id;
                project.CreatedByUser = user;

                _context.Projects.Add(project);
                await _context.SaveChangesAsync();

                // Додаємо зв'язок ProjectUser
                var projectUser = new ProjectUser
                {
                    ProjectId = project.Id,
                    UserId = user.Id,
                    Role = ProjectRole.Admin,
                };

                _context.ProjectUsers.Add(projectUser);
                await _context.SaveChangesAsync();
                await _activityLogger.LogAsync(user.Id, project.Id, "Created",
"Project", project.Id, $"w/title [{project.Title}]'");
            }

            return RedirectToAction(nameof(Index));
        }

        // GET: Projects/Edit/5
        public async Task<IActionResult> Edit(int id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var project = await _context.Projects.FindAsync(id);
            if (project == null)
            {
                return NotFound();
            }

            return View(project);
        }

        [HttpPost]
        public async Task<IActionResult> Edit(int id, Project project, IFormFile?
ImageFile)
        {
            if (id != project.Id) return NotFound();

            var existingProject = await _context.Projects.FindAsync(id);
            if (existingProject == null) return NotFound();

            if (ModelState.IsValid)
            {
                existingProject.Title = project.Title;
                existingProject.Description = project.Description;

                if (ImageFile != null)
```

```
                {
                    // логіка збереження нового зображення
                    var fileName = $"{Guid.NewGuid()}_{ImageFile.FileName}";
                    var filePath = Path.Combine("wwwroot/Uploads", fileName);
                    using var stream = new FileStream(filePath, FileMode.Create);
                    await ImageFile.CopyToAsync(stream);

                    existingProject.ImagePath = $"/Uploads/{fileName}";
                }

                _context.Update(existingProject);
                await _context.SaveChangesAsync();

                var userId = _userManager.GetUserId(User);
                await _activityLogger.LogAsync(userId, id, "Edited", "Project",
id, $"Old [{existingProject.Title}] New [{project.Title}]'");
                return RedirectToAction("Details", new { id = project.Id });
            }

            return View(project);
        }

        [HttpPost]
        public async Task<IActionResult> Delete(int id)
        {
            var project = await _context.Projects.FindAsync(id);
            if (project == null)
            {
                return NotFound();
            }

            _context.Projects.Remove(project);
            await _context.SaveChangesAsync();

            return RedirectToAction("Index");
        }

        private bool ProjectExists(int id)
        {
            return _context.Projects.Any(e => e.Id == id);
        }

        //Calendar
        public IActionResult Calendar(int id)
        {
            // Отримання дат через допоміжний метод
            DateTime currentDate = GetDateFromCookie("currentDate");
            DateTime selectedDate = GetDateFromCookie("selectedDate");

            var dayModels = RenderCalendar(currentDate, selectedDate, id);

            ViewBag.ProjectId = id;
            //додаткові дані для відображення місяця та року
            ViewBag.DayName = DateTime.Now.ToString("dddd", new CultureInfo("en-
US"));
            ViewBag.MonthName = currentDate.ToString("MMMM", new CultureInfo("en-
US"));
            ViewBag.Year = currentDate.Year.ToString();
            ViewBag.HeaderTitle = "Calendar";
            ViewBag.HeaderDescription = "Simple and convenient tool for those who
want to plan their time effectively";
```

```
            return View(dayModels);
        }
        private List<DayModel> RenderCalendar(DateTime currentDate, DateTime
selectedDate, int projectId)
        {
            DateTime startDate = DateUtils.StartOfWeek(new
DateTime(currentDate.Year, currentDate.Month, 1), DayOfWeek.Monday);
            DateTime date = startDate;
            DateTime endDate = startDate.AddDays(7 * 5);

            List<DateTime> dates = new List<DateTime>();
            while (date < endDate)
            {
                dates.Add(date);
                date = date.AddDays(1);
            }

            var userId = _userManager.GetUserId(User);

            var allTasks = _context.TaskItems
                .Where(t => t.ProjectId == projectId
                            && t.Date.Date >= dates.First()
                            && t.Date.Date <= dates.Last()
                            && (t.AssignedUserId == userId
                                || t.TaskItemUsers.Any(tu => tu.UserId ==
userId))) // якщо обидва варіанти
                .Include(t => t.TaskItemUsers)
                    .ThenInclude(tu => tu.User)
                .Include(t => t.Project)
                .ToList();


            var dayModels = new List<DayModel>();
            foreach (var d in dates)
            {
                var tasksOnDay = allTasks.Where(t => t.Date.Date ==
d.Date).OrderBy(t => t.DueDate ?? t.Date).ToList();

                dayModels.Add(new DayModel
                {
                    Date = d,
                    Styles = "day"
                             + (DateUtils.IsSameMonth(d, currentDate) ? " active"
: " inactive")
                             + (DateUtils.IsSameDay(d, selectedDate) ? " selected"
: "")
                             + (DateUtils.IsSameDay(d, DateTime.Now) ? " today" :
""),
                    Tasks = tasksOnDay
                });
            }

            return dayModels;
        }

        public IActionResult ChangeMonth(int id, int direction)
        {
            DateTime currentDate = GetDateFromCookie("currentDate");
            Response.Cookies.Delete("currentDate");
```

```
            currentDate = currentDate.AddMonths(direction);
            Response.Cookies.Append("currentDate",
currentDate.Date.ToString("yyyy-MM-dd", CultureInfo.InvariantCulture));

            return RedirectToAction("Calendar", new { id });
        }

        private DateTime GetDateFromCookie(string cookieKey)
        {
            DateTime date = DateTime.Now;
            if (Request.Cookies.ContainsKey(cookieKey))
            {
                string? value = Request.Cookies[cookieKey];
                if (!string.IsNullOrEmpty(value) &&
                    DateTime.TryParseExact(value, "yyyy-MM-dd",
CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime parsedDate))
                {
                    date = parsedDate;
                }
            }
            return date;
        }

        public IActionResult DateView(int id, DateTime date)
        {
            var project = _context.Projects.Include(p => p.TaskItems)
                                          .FirstOrDefault(p => p.Id == id);

            if (project == null)
                return NotFound();

            var projectUsers = _context.ProjectUsers
                .Include(pu => pu.User)
                .Where(pu => pu.ProjectId == project.Id)
                .Select(pu => new SelectListItem
                {
                    Value = pu.UserId,
                    Text = pu.User.Name // або Name
                }).ToList();


            var viewModel = new ProjectDateViewModel
            {
                Project = project,
                SelectedDate = date,
                ProjectUsers = projectUsers,
                TaskItems = _context.TaskItems
                                .Include(t => t.TaskItemUsers)
                                    .ThenInclude(tu => tu.User)
                                .Where(t => t.ProjectId == id && t.Date.Date ==
date.Date)
                                .ToList()
            };

            ViewBag.ProjectId = project.Id;

            return View(viewModel);


        }
```

```
        [HttpGet]
        public IActionResult SearchUsers(string term)
        {
            if (string.IsNullOrWhiteSpace(term) || term.Length < 3)
                return Json(Array.Empty<object>());

            var users = _context.Users
                .Where(u => u.UserName.Contains(term))
                .Select(u => new SelectListItem
                {
                    Value = u.Id,
                    Text = u.UserName
                }).Take(10).ToList();

            return Json(users);
        }

        [HttpPost]
        public async Task<IActionResult> InviteUser(int projectId, string userId)
        {
            var result = await _authorizationService.AuthorizeAsync(User,
projectId, "ProjectAdmin");
            if (!result.Succeeded) return Forbid();


            var alreadyExists = _context.ProjectUsers
                .Any(pu => pu.ProjectId == projectId && pu.UserId == userId);

            if (!alreadyExists)
            {
                var pu = new ProjectUser
                {
                    ProjectId = projectId,
                    UserId = userId,
                    Role = ProjectRole.Member // за замовчуванням
                };
                _context.ProjectUsers.Add(pu);
                _context.SaveChanges();
            }

            return Ok();
        }

        [HttpGet]
        public async Task<IActionResult> MemberDetails(int id /*projectId*/,
string userId)
        {
            // Перевірка прав
            var auth = await _authorizationService.AuthorizeAsync(User, id,
"ProjectManager");
            if (!auth.Succeeded) return Forbid();

            // Знайти record у ProjectUsers
            var pu = await _context.ProjectUsers
                .Include(x => x.User)
                .FirstOrDefaultAsync(x => x.ProjectId == id && x.UserId ==
userId);
            if (pu == null) return NotFound();

            // Дістати таски, призначені цьому юзеру в цьому проекті
            var tasks = await _context.TaskItems
```

```
                .Where(t => t.ProjectId == id
                        && t.TaskItemUsers.Any(tu => tu.UserId == userId))
                .ToListAsync();

            var project = await _context.Projects.FindAsync(id);

            var vm = new ProjectMemberViewModel
            {
                ProjectId = id,
                UserId = userId,
                UserName = pu.User.UserName,
                UserEmail = pu.User.Email,
                Role = pu.Role,
                AssignedTasks = tasks,
                CreatedByUserId = project.CreatedByUserId
            };

            ViewBag.ProjectId = project.Id;
            return View(vm);
        }

        [HttpPost]
        public async Task<IActionResult> UpdateMemberRole(int projectId, string
userId, ProjectRole newRole)
        {
            var auth = await _authorizationService.AuthorizeAsync(User, projectId,
"ProjectAdmin");
            if (!auth.Succeeded) return Forbid();

            var pu = await _context.ProjectUsers
                .FirstOrDefaultAsync(x => x.ProjectId == projectId && x.UserId ==
userId);
            if (pu == null) return NotFound();

            pu.Role = newRole;
            await _context.SaveChangesAsync();

            return RedirectToAction("MemberDetails", new { id = projectId, userId
});
        }

        [HttpPost]
        public async Task<IActionResult> RemoveMember(int projectId, string
userId)
        {
            var auth = await _authorizationService.AuthorizeAsync(User, projectId,
"ProjectAdmin");
            if (!auth.Succeeded) return Forbid();

            var pu = await _context.ProjectUsers
                .FirstOrDefaultAsync(x => x.ProjectId == projectId && x.UserId ==
userId);
            if (pu == null) return NotFound();

            _context.ProjectUsers.Remove(pu);
            await _context.SaveChangesAsync();

            return RedirectToAction("Details", new { id = projectId });
        }

        public IActionResult Tasks(
```

```
            int id,
            string? sortBy,
            string? statusFilter,
            DateTime? dateFrom,
            DateTime? dateTo,
            string? assignedFilter)
        {
            var project = _context.Projects
                .Include(p => p.TaskItems)
                    .ThenInclude(t => t.TaskItemUsers)
                        .ThenInclude(tu => tu.User)  // ← саме тут
                .FirstOrDefault(p => p.Id == id);


            if (project == null) return NotFound();

            var tasks = project.TaskItems.AsQueryable();

            // — Фільтр по статусу
            if (!string.IsNullOrEmpty(statusFilter))
            {
                if (statusFilter == "Overdue")
                {
                    var today = DateTime.Today;
                    tasks = tasks.Where(t =>
                        t.DueDate.HasValue &&
                        t.DueDate.Value.Date < today &&
                        t.Status != "Done");
                }
                else
                {
                    tasks = tasks.Where(t => t.Status == statusFilter);
                }
            }

            // — Фільтр по діапазону дат (поля .Date прив'язки)
            if (dateFrom.HasValue && dateTo.HasValue)
            {
                var from = dateFrom.Value.Date;
                var to = dateTo.Value.Date;
                tasks = tasks.Where(t =>
                    t.Date.Date >= from &&
                    t.Date.Date <= to);
            }

            // — Фільтр по призначеним / непризначеним
            if (assignedFilter == "assigned")
                tasks = tasks.Where(t => t.TaskItemUsers.Any());
            else if (assignedFilter == "unassigned")
                tasks = tasks.Where(t => !t.TaskItemUsers.Any());

            // — Сортування
            tasks = sortBy switch
            {
                "status" => tasks.OrderBy(t => t.Status),
                "date" => tasks.OrderBy(t => t.Date),
                "assignments" => tasks.OrderByDescending(t =>
t.TaskItemUsers.Count),
                _ => tasks.OrderBy(t => t.Id)
            };
```

```
            // Прокидуємо поточні значення фільтрів у ViewBag, щоб вони
"заповнилися" у формі
            ViewBag.CurrentSort = sortBy;
            ViewBag.StatusFilter = statusFilter;
            ViewBag.DateFrom = dateFrom?.ToString("yyyy-MM-dd");
            ViewBag.DateTo = dateTo?.ToString("yyyy-MM-dd");
            ViewBag.AssignedFilter = assignedFilter;

            ViewBag.ProjectId = project.Id;
            return View(tasks.ToList());
        }

        public async Task<IActionResult> MyTasks(int id)
        {
            var userId = _userManager.GetUserId(User);

            var project = await _context.Projects
                .Include(p => p.TaskItems)
                .ThenInclude(t => t.TaskItemUsers)
                .ThenInclude(tu => tu.User)
                .Include(p => p.ProjectUsers)
                .ThenInclude(pu => pu.User)
                .FirstOrDefaultAsync(p => p.Id == id);

            if (project == null)
                return NotFound();

            // Список тільки тих завдань, де user є виконавцем
            var userTasks = project.TaskItems
                .Where(t => t.TaskItemUsers.Any(tiu => tiu.UserId == userId))
                .ToList();

            var projectUsers = project.ProjectUsers
                .Select(pu => new SelectListItem
                {
                    Value = pu.UserId,
                    Text = pu.User.Name
                }).ToList();

            var viewModel = new ProjectDateViewModel
            {
                Project = project,
                SelectedDate = DateTime.Today, // або за потребою
                ProjectUsers = projectUsers,
                TaskItems = userTasks
            };

            ViewBag.ProjectId = project.Id;

            return View(viewModel);
        }

        // GET: /Projects/Members/5?roleFilter=Manager
        public async Task<IActionResult> Members(int id, ProjectRole? roleFilter)
        {
            // 1) Підтягнути проект (щоб дістати CreatedByUserId, якщо потрібно)
            var project = await _context.Projects.FindAsync(id);
            if (project == null) return NotFound();

            // 2) Базовий запит по всіх ProjectUsers у цьому проекті
            var q = _context.ProjectUsers
```

```
                .Include(pu => pu.User)
                .Where(pu => pu.ProjectId == id)
                .AsQueryable();

            // 3) Застосувати фільтр по ролі, якщо передали roleFilter
            if (roleFilter.HasValue)
                q = q.Where(pu => pu.Role == roleFilter.Value);

            // 4) Віддати у ViewModel чи прямо у ViewBag
            ViewBag.RoleFilter = roleFilter?.ToString() ?? "";
            ViewBag.AvailableRoles = Enum.GetValues<ProjectRole>();

            var list = await q.ToListAsync();

            ViewBag.ProjectId = project.Id;
            return View(list);
        }

        public async Task<IActionResult> ActivityLog(int id)
        {
            IQueryable<ActivityLog> logs = _context.ActivityLogs
                .Include(l => l.User)
            .Include(l => l.Project)
            .Where(l => l.ProjectId == id);


            var logList = await logs
                .OrderByDescending(l => l.Timestamp)
            .ToListAsync();

            ViewBag.ProjectId = id;
            return View(logList);
        }
    }
}
```

Б.7 Код сервісу ActivityLogger.cs

```
using realexam.Data;
using realexam.Models;

namespace realexam.Services
{
    public class ActivityLogger : IActivityLogger
    {
        private readonly ApplicationDbContext _context;
        public ActivityLogger(ApplicationDbContext context)
        {
            _context = context;
        }
        public async Task LogAsync(string userId, int projectId, string
actionType, string entityType, int? entityId, string description)
        {
            var log = new ActivityLog
            {
                UserId = userId,
                ProjectId = projectId,
```

```
                ActionType = actionType,
                EntityType = entityType,
                EntityId = entityId,
                Description = description,
                Timestamp = DateTime.UtcNow
            };
            _context.ActivityLogs.Add(log);
            await _context.SaveChangesAsync();
        }
    }
}
```

Б.8 Розмітка представлення /Projects/Index.cs

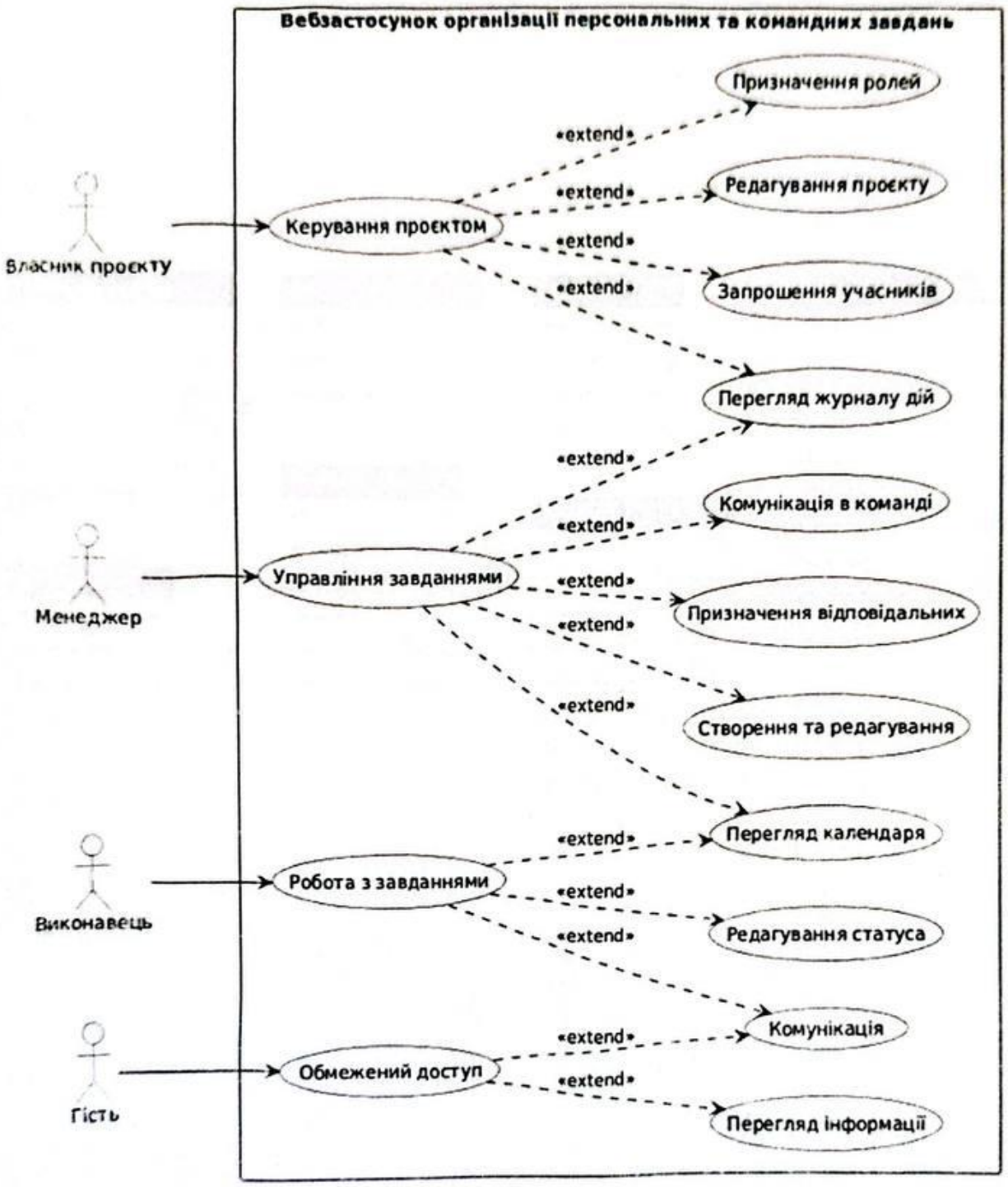
```
using Microsoft.AspNetCore.Authorization;
@model IEnumerable<realexam.Models.Project>
@using System.Security.Claims

@{
    ViewData["Title"] = "Projects";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="container p-0">
    <div class="d-flex justify-content-between align-items-center mb-3">
        <h2 class="fw-semibold"></h2>
        <button id="project-create" type="button" class="btn-main-control" data-
bs-toggle="modal" data-bs-target="#modalProjectCreate">
            <i class="bi bi-plus-lg"></i>
            New Project
        </button>
    </div>

    <div class="row">
        @foreach (var project in Model)
        {
            <div class="col-lg-4 col-md-6 mb-4">
                <div class="project-list-card h-100 border-shadow radius">
                    <div class="d-flex align-items-start mb-2">
                        <img src="@(project.ImagePath ?? "/Uploads/default.png")"
class="rounded-3 project-list-logo me-3" alt="Logo" width="54" height="54">
                        <div class="flex-grow-1">
                            <div class="d-flex justify-content-between align-
items-start">
                                <h5 class="dash-card-title">@project.Title</h5>
                                <!-- Dropdown with actions -->
                                <div class="dropdown">
                                    <button class="dash-card-controls"
type="button" data-bs-toggle="dropdown" aria-expanded="false">
                                        <i class="bi bi-three-dots-vertical"></i>
                                    </button>
                                    <ul class="dropdown-menu">
                                        <li>
                                            <a class="dropdown-item" href="#"
data-bs-toggle="modal" data-bs-target="#modalProjectEdit-@project.Id">
                                                <i class="bi bi-pencil-square me-
1"></i> Edit
```

```
                                        </a>
                                    </li>
                                    <li>
                                        <a class="dropdown-item text-danger"
href="#" data-bs-toggle="modal" data-bs-target="#modalDeleteProject-@project.Id">
                                            <i class="bi bi-trash3 me-1"></i>
Delete
                                        </a>
                                    </li>
                                </ul>
                            </div>
                        </div>
                        <div class="d-flex gap-3 align-items-center mb-1">
                            <div class="d-flex align-items-center gap-1 text-
secondary text-muted">
                                <i class="bi bi-people"></i>
                                <span>@(project.ProjectUsers?.Count() ??
1)</span>
                            </div>
                            <div class="d-flex align-items-center gap-1 text-
secondary text-muted">
                                <i class="bi bi-list-task"></i>
                                <span>@(project.TaskItems?.Count() ??
0)</span>
                            </div>
                        </div>
                    </div>
                </div>
                <div class="divider-dots"></div>
                <div class="project-list-desc text-body-secondary mb-2 mt-2"
style="min-height:40px;">
                    @(!string.IsNullOrWhiteSpace(project.Description) ?
project.Description : "No description yet.")
                </div>
                <div class="d-flex justify-content-between align-items-center
mt-auto px-1">
                    <div class="d-flex align-items-center gap-2 text-secondary
small">
                        <i class="bi bi-person-circle"></i>
                        <small>@(project.CreatedByUser?.Name ??
"User")</small>
                        <small>•</small>
                        <small>@project.CreatedAt.ToString("dd MMM",
System.Globalization.CultureInfo.InvariantCulture)</small>

                    </div>
                    <a asp-controller="Projects" asp-action="Details" asp-
route-id="@project.Id" class="btn-lvl2">
                        <i class="bi bi-info-square"></i>
                        Details
                    </a>
                </div>
            </div>

            <!-- Modal: Edit Project -->
            <form asp-action="Edit" asp-controller="Projects"
enctype="multipart/form-data" method="post">
                <div class="modal fade" id="modalProjectEdit-@project.Id"
data-bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-
labelledby="modalTitleLabelEdit-@project.Id" aria-hidden="true">
                    <div class="modal-dialog">
```

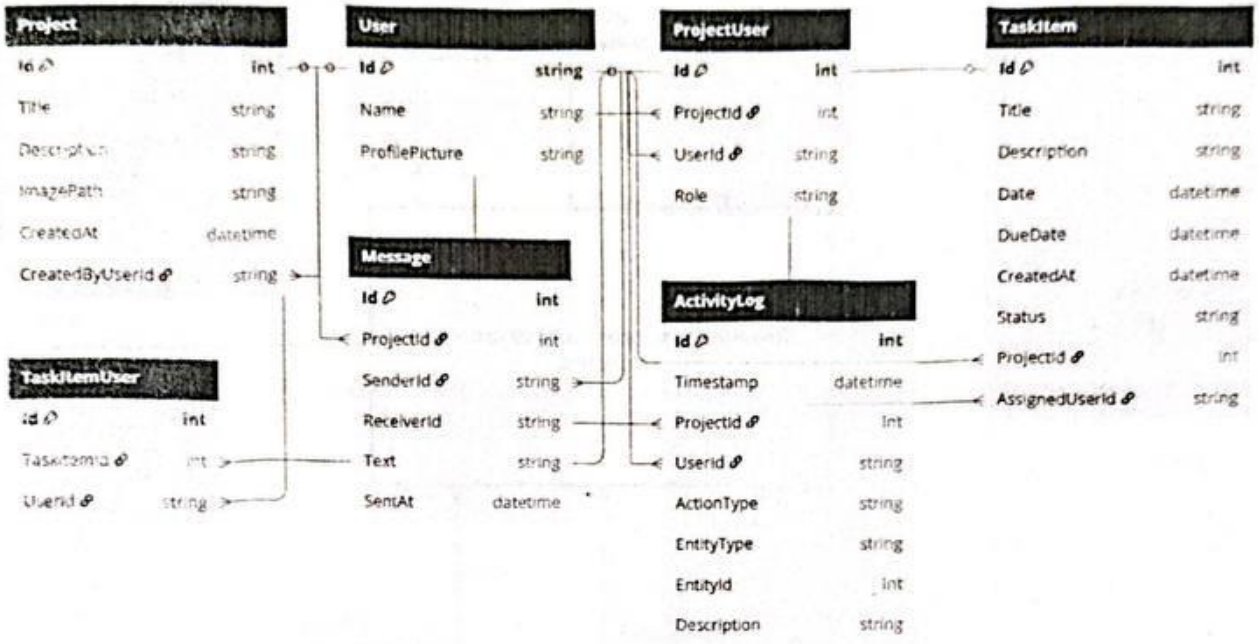
```
                            <div class="modal-content">
                                <div class="modal-header">
                                    <h1 class="modal-title fs-5"
id="modalTitleLabelEdit-@project.Id">Edit Project</h1>
                                    <button type="button" class="btn-close" data-
bs-dismiss="modal" aria-label="Close"></button>
                                </div>
                                <div class="modal-body">
                                    <input type="hidden" name="Id"
value="@project.Id" />
                                    <partial
name="~/Views/Projects/_CreatePartial.cshtml" model="project" />
                                </div>
                                <div class="modal-footer">
                                    <button type="button" class="btn btn-
secondary" data-bs-dismiss="modal">Cancel</button>
                                    <button type="submit" class="btn btn-
primary">Save Changes</button>
                                </div>
                            </div>
                        </div>
                    </div>
                </form>

                <!-- Modal: Delete Project -->
                <div class="modal fade" id="modalDeleteProject-@project.Id" data-
bs-backdrop="static" data-bs-keyboard="false" tabindex="-1" aria-
labelledby="modalDeleteProjectLabel-@project.Id" aria-hidden="true">
                    <div class="modal-dialog modal-dialog-centered">
                        <div class="modal-content">
                            <div class="modal-header">
                                <h1 class="modal-title fs-5"
id="modalDeleteProjectLabel-@project.Id">Delete Confirmation</h1>
                                <button type="button" class="btn-close" data-bs-
dismiss="modal" aria-label="Close"></button>
                            </div>
                            <form asp-action="Delete" asp-controller="Projects"
method="post">
                                <div class="modal-body">
                                    <input type="hidden" name="Id"
value="@project.Id" />
                                    <p>Are you sure you want to delete this
project?</p>
                                </div>
                                <div class="modal-footer">
                                    <button type="button" class="btn btn-
secondary" data-bs-dismiss="modal">Cancel</button>
                                    <button type="submit" class="btn btn-
danger">Delete</button>
                                </div>
                            </form>
                        </div>
                    </div>
                </div>

            </div>
        }
    </div>

</div>
```

```
<!-- Modal Create Project -->
<form asp-action="Create" asp-controller="Projects" enctype="multipart/form-data"
method="post">
    <div class="modal fade" id="modalProjectCreate" data-bs-backdrop="static"
data-bs-keyboard="false" tabindex="-1" aria-labelledby="modalTitleLabelCreate"
aria-hidden="true">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <h1 class="modal-title fs-5" id="modalTitleLabelCreate">New
project</h1>
                    <button type="button" class="btn-close" data-bs-
dismiss="modal" aria-label="Close"></button>
                </div>
                <div class="modal-body">
                    <input type="hidden" name="CreatedByUserId"
value="@User.FindFirstValue(ClaimTypes.NameIdentifier)" />
                    <partial name="~/Views/Projects/_CreatePartial.cshtml"
model="@new Project()"></partial>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Close</button>
                    <button type="submit" class="btn btn-primary">Confirm</button>
                </div>
            </div>
        </div>
    </div>
</form>
```

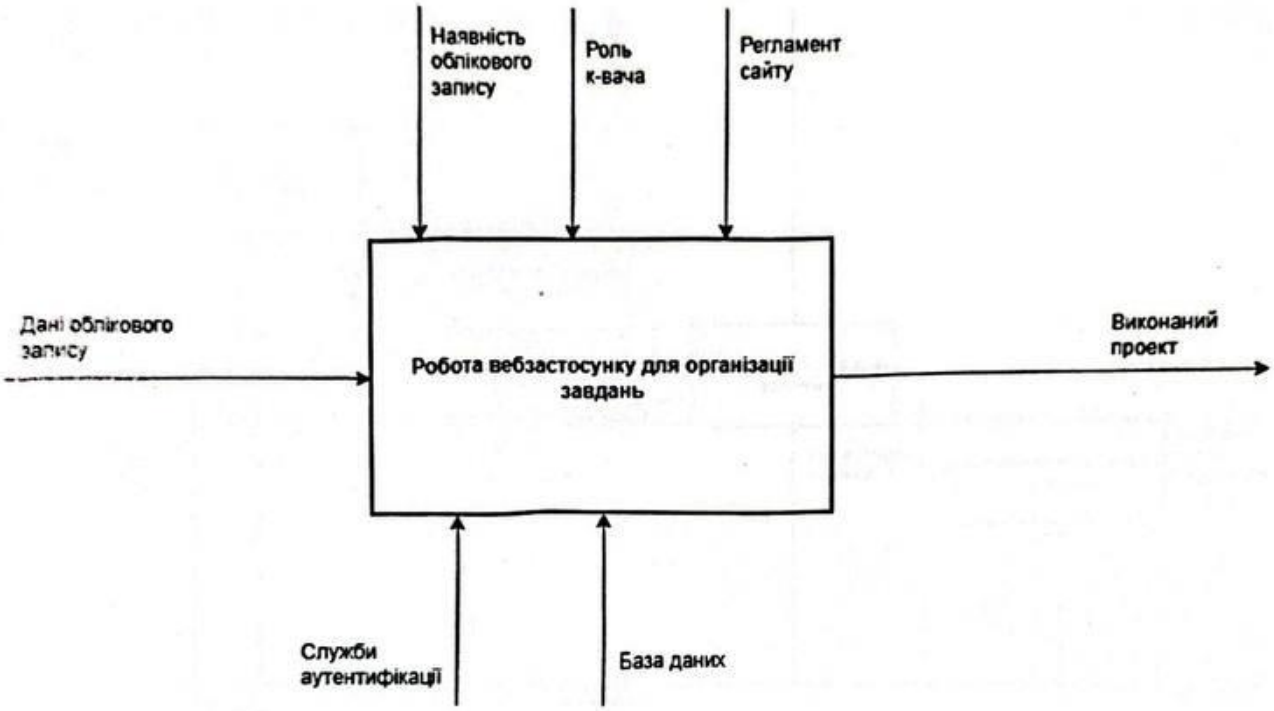
ГРАФІЧНІ МАТЕРІАЛИ



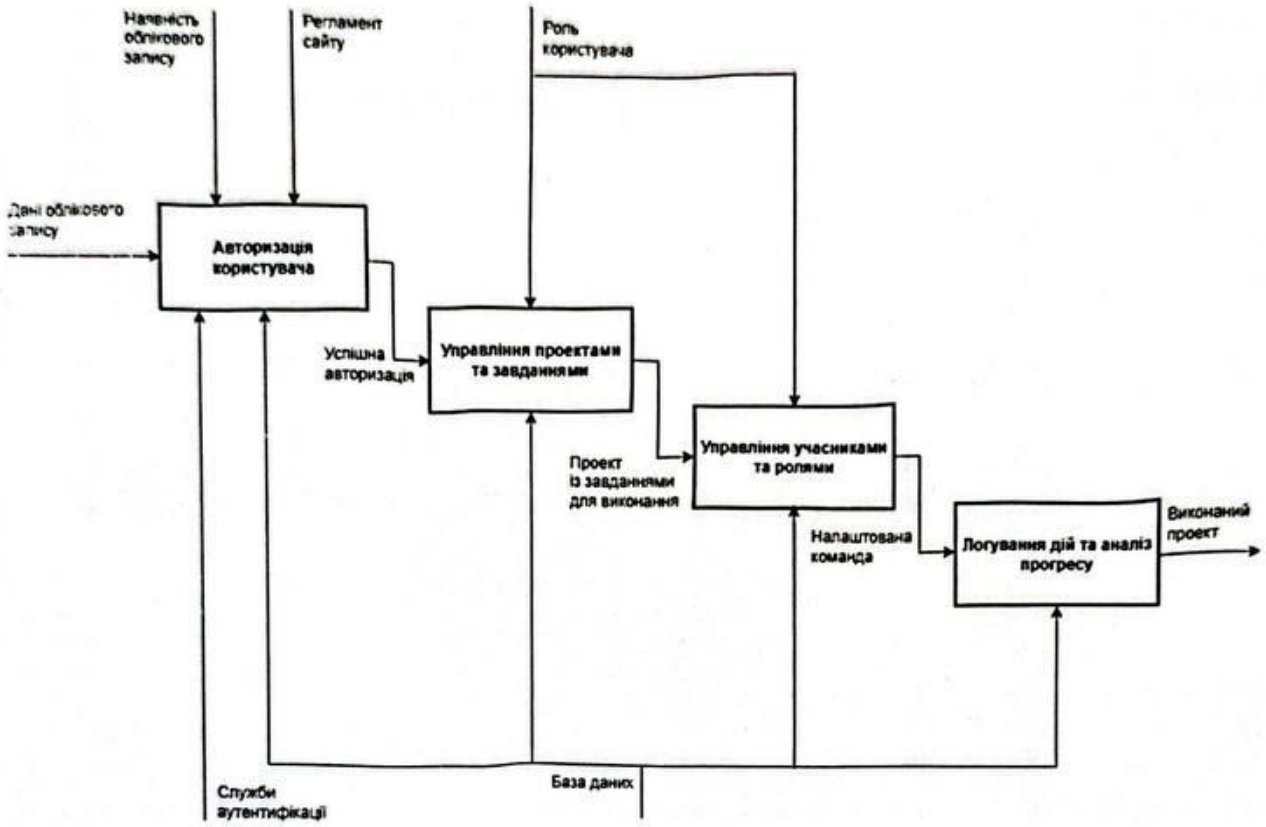
					КвРІПЗ.2101086.01.13.Е8	
Змн.	Арк.	№ докум.	Підпис	Дата	Діаграма варіантів використання вебзастосунку	Літ. / Маса. / Масштаб
Розробив		Рейтаровський О. Ю				
Керівник		Бойко В. О.				Аркуш 1 / Аркушів 4
Консульт.						
Н. Контр.		Форкун Ю. В.				ХНУ, ІПЗ-21-1
Зав. каф.		Бедратюк Л. П.				



					КвРІПЗ.2101086.01.13.Е8	
Змн.	Арк.	№ докум.	Підпис	Дата	ER-діаграма бази даних	Літ. / Маса. / Масштаб
Розробив		Рейтаровський О. Ю.		03.06.25		
Керівник		Бойко В. О.		03.06.25		Аркуш 2 / Аркушів 4
Консульт.						
Н. Контр.		Форкун Ю. В.		03.06.25		ХНУ, ІПЗ-21-1
Зав. каф.		Бедратюк Л. П.		03.06.25		



					КвРІПЗ.2101086.01.13.Е8		
Змн.	Арк.	№ докум.	Підпис	Дата	Контекстна діаграма вебзастосунку	Літ.	Маса.	Масштаб
Розробив		Рейтаровський О. Ю		13.06.25				
Керівник		Бойко В. О.		01.06.25		Аркуш 3	Аркушів 4	
Консульт.								
Н. Контр.		Форкун Ю. В.		09.06.25		ХНУ, ІПЗ-21-1		
Зав. каф.		Бедратюк Л. П.		09.06.25				



					КвРІПЗ.2101086.01.13.Е8		
Змн.	Арк.	№ докум.	Підпис	Дата	Діаграма декомпозиції 1 рівня вебзастосунку	Літ.	Маса.	Масштаб
Розробив		Рейтаровський О. Ю						
Керівник		Бойко В. О.				Аркуш 4	Аркушів 4	
Консульт.								
Н. Контр.		Форкун Ю. В.				ХНУ, ІПЗ-21-1		
Зав. каф.		Бедратюк Л. П.						

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Рейтаровського Олександра Юрійовича
факультет ІТ, IV курс, група ІПЗ-21-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.01.2025
дата

підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 4.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 11%**

ID: 244350 Title: БКР_Вебзастосунок для організації особистих та командних завдань_ Added in a DB: 2025-06-09 Authors: Рейтаровський Олександр Heads: БОЙКО В'ячеслав Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	80272	1184	4980 (6%)	65 (5%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Рейтаровський Олександр

Співавтор:

Назва: БКР_Вебзастосунок для організації особистих та командних завдань

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:5.9%

Коефіцієнт подібності 2:1.6%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 173

Дата створення звіту: 2025-06-07 13:20:29.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обґрунтування:

Дата 07.06.2025 експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Рейтаровський Олександр Юрійович

Тема Вебзастосунок для організації особистих та командних завдань

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 4 ; кількість сторінок записки 75

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено і проаналізовано предметну область організації особистої та командної роботи з використанням сучасних вебзастосунків. Визначено всі функціональні та нефункціональні вимоги до системи. Проведено детальний аналіз існуючих програмних продуктів на ринку, розглянуто їхні переваги та недоліки, що підтвердило актуальність створення нового програмного забезпечення для керування проєктами і завданнями. Обрано оптимальну архітектуру та технологічний стек для реалізації проєкту на основі ASP.NET Core, з використанням Entity Framework Core та MS SQL Server Express. У результаті проведення досліджень та розробки створено вебзастосунок, що дозволяє ефективно організовувати проєкти, розподіляти ролі, керувати завданнями, комунікувати у команді та контролювати виконання задач. Проведено тестування функціоналу, результати якого засвідчили, що розроблене програмне забезпечення працює коректно, відповідає заданим вимогам і готове до впровадження в експлуатацію.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, сформульовано мету та завдання роботи. У першому розділі проведено аналіз предметної області, огляд існуючих рішень і визначено вимоги до майбутньої системи. У другому розділі розглянуто архітектурні підходи, обґрунтовано вибір архітектури та технологій, розроблено структуру даних і основні модулі із використанням сучасних принципів проєктування. У третьому розділі описано реалізацію вебзастосунку, ключові етапи створення бази даних і модулів, наведено застосовані технології та засоби тестування. Підтверджено відповідність функціональним вимогам і готовність системи до подальшого використання.

4. Позитивні сторони роботи Тема кваліфікаційної роботи є актуальною, оскільки питання ефективної організації управління проєктами та завданнями набуває все більшого значення як у професійному, так і в навчальному середовищі. Розроблений вебзастосунок враховує сучасні вимоги до цифрових інструментів співпраці, забезпечує простоту використання, гнучкість налаштувань та широкі можливості для командної взаємодії. Для

реалізації системи застосовано новітні інструменти та бібліотеки, що підвищує якість, продуктивність та безпеку роботи застосунку.

5. Негативні сторони роботи У роботі відсутні деякі детальні опрацювання, зокрема не реалізовано повноцінні фільтри на дашбордах і ряді сторінок. Немає системи сповіщень або нагадувань для користувачів. Також чат не підтримує закріплення повідомлень та надсилання медіаконтенту (файлів, емодзі тощо).

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота виконана на високому рівні та заслуговує позитивної оцінки. Пояснювальна записка структурована, логічна й доступна для сприйняття. Викладені рішення є сучасними й відповідають актуальним вимогам галузі ІТ. Графічні матеріали допомагають чітко уявити структуру та функціонування системи, підвищуючи якість подачі роботи командної роботи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Кобоз Юрій, К.Т.Н., завідувач кафедри кібербезпеки, ХНУ

"10" 06 2025 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.

Назва кваліфікаційної роботи: «Вебзастосунок для організації особистих та командних завдань»

Автор: Рейтаровський Олександр Юрійович

Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»

Спеціальність: 121 – Інженерія програмного забезпечення

Науковий керівник: Бойко В'ячеслав Олександрович, асистент

Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі, єзаконними і не є плагіатом, оскільки:

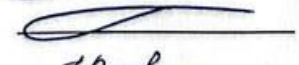
1) у тексті кваліфікаційної роботи системою перевірки на плагіат виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання:

2) запозичення, виявлені у тексті роботи, є фрагментарними.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає **4.0%**. Обсяг запозичень, визначений системою StrikePlagiarism для виявлення ідентичності/подібності, становить **5.9%** за першим коефіцієнтом і **1.6%** за другим коефіцієнтом. Під час аналізу не виявлено мікроповторів, замін букв чи надмірної кількості інтервалів, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 09.06.2024 р.

Завідувач кафедри  Леонід БЕДРАТЮК

Гарант освітньої програми  Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи  В'ячеслав БОЙКО