Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ ВЕБСИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ КОМАНДНИХ ПРОЄКТІВ ТА ОСОБИСТИХ ЗАВДАНЬ

DEVELOPMENT AND RESEACH OF A WEB SYSTEM FOR ORGANIZING TEAM PROJECTS AND PERSONAL TASKS

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Тимчук В. В.
Керівник:
д.т.н., професор
Андрущак І. Є.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Тимчуку Владиславу Володимировичу

1. Тема кваліфікаційної роботи: Розробка та дослідження вебсистеми для організації командних проєктів та особистих завдань

Керівник роботи: Андрущак Ігор Євгенович, д.т.н., професор

затверджені наказом закладу вищої освіти від «29» березня 2025 р. № 190/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: «04» грудня 2025 р.

3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ.*

4. Зміст розрахунково-пояснювальної записки: *аналіз проблематики сучасного стану систем керування проєктами та менеджменту особистих завдань та вибір методів дослідження, обґрунтування технологій та практичну реалізацію вебсистеми менеджменту завдань та проєктів із застосуванням мультитенантної архітектури, експериментальне дослідження результативності розробленого програмного забезпечення.*

5. Перелік графічного матеріалу: 16 рисунків, 2 таблиці, 7 лістингів коду

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Андрущак І. Є.*		
Теоретичне дослідження та практична реалізація	*Андрущак І. Є.*		
Експериментальне дослідження системи	*Андрущак І. Є.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Андрущак І.Є*		

7. Дата видачі завдання *«02 квітня 2025 р.»*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну модель та архітектуру системи	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ (підпис) Тимчук В. В. (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) Андрущак І. Є. (прізвище, ініціали)

ЗМІСТ

АНОТАЦІЯ

Тимчук В. В. Розробка та дослідження вебсистеми для організації командних проєктів та особистих завдань. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення». Спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел (згідно структури кваліфікаційної роботи, затвердженої кафедрою).

У першому розділі проведено аналіз сучасних підходів до управління проєктами та організації особистого часу, розглянуто існуючі програмні аналоги (таск-менеджери та системи керування проєктами), а також методи та технології для побудови вебсистем. У другому розділі обґрунтовано вибір архітектурного паттерну та стека технологій, реалізовано програмні компоненти серверної та клієнтської частин вебсистеми для менеджменту командних та особистих завдань. У третьому розділі проведено експериментальне дослідження продуктивності розробленої системи при роботі з великими обсягами даних, описано методику тестування. У висновках узагальнено результати теоретичного й практичного дослідження.

Ключові слова: вебсистема, система керування проєктами, менеджер завдань, мультитенантність, Laravel, Inertia, React, Single-Page Application.

ABSTRACT

Tymchuik V. V. Development and Reseach of a Web System for Organizing Team Projects and Personal Tasks. Manuscript.

Master's Qualification Thesis of the Educational Program "Software Engineering" specialty 121 Software Engineering. Lutsk National Technical University. Lutsk, 2025.

The master's thesis consists of an introduction, three chapters, conclusions, and a list of references (according to the structure of the qualification work approved by the department).

The first chapter analyzes modern approaches to project management and personal time organization, reviews existing software analogs (task managers and project management systems), as well as methods and technologies for building web systems. The second chapter justifies the choice of the architectural pattern and technology stack, and implements the software components of the server and client sides of the web system for managing team and personal tasks. The third chapter conducts an experimental study of the developed system's performance when working with large volumes of data and describes the testing methodology. The conclusions summarize the results of the theoretical and practical research.

Keywords: web system, project management system, task manager, multitenancy, Laravel, Inertia, React, Single-Page Application.

ВСТУП

Сучасний етап розвитку цифрових робочих середовищ характеризується переходом від жорстких ієрархічних структур до гнучких проєктних команд, що часто працюють у віддаленому або гібридному форматі. Це стимулювало появу численних платформ для управління проєктами, які стали невід'ємною частиною корпоративної культури. Проте, попри їхню ефективність у командній взаємодії, більшість існуючих рішень ігнорує важливий аспект продуктивності – необхідність інтеграції робочих процесів з особистими планами та цілями працівника.

Вже існуючі системи, такі як Jira або Trello від Atlassian, YouTrack від JetBrains чи Redmine, ефективно організовують командні процеси, але залишають користувача віч-на-віч із проблемою управління власними ресурсами. Працівник змушений використовувати кілька незалежних інструментів: один для корпоративних проєктів, інший для особистих нотаток, третій для довгострокових цілей. Такий фрагментований підхід призводить до перевантаження, втрати фокуса та зниження загальної продуктивності, адже не існує єдиного інструмента який міститиме всі згадані інструменти.

Наукова новизна одержаних результатів полягає в удосконаленні інформаційної технології управління проєктами шляхом розробки та реалізації вебсистеми гібридного типу.

Актуальність теми дослідження зумовлена гострою потребою в інструменті, що забезпечує цілісний підхід до управління завданнями. Створення платформи, яка б об'єднувала командні проєкти та персональні простори користувача, є відповіддю на виклики сучасного робочого середовища. Таке рішення дозволить не лише підвищити індивідуальну ефективність, а й сприятиме покращенню балансу між роботою та особистим життям, що є ключовим фактором запобігання вигоранню.

Об'єктом дослідження є процес організації та управління робочими й особистими завданнями користувача в рамках єдиної цифрової платформи.

Предметом дослідження є методи, моделі та технології для створення інтегрованої платформи управління проєктами, що забезпечує функціональний зв'язок та часткову синхронізацію між командними та персональними робочими просторами.

Метою дослідження є розробка архітектури та вебплатформи, яка надає інструменти для комплексного управління командними проєктами та особистими завданнями через механізм їхньої гнучкої синхронізації, що дозволяє користувачеві формувати єдину, цілісну картину власного навантаження.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз предметної області та існуючих аналогів (систем управління проєктами та таск-менеджерів), виявити їхні архітектурні обмеження та функціональні недоліки в контексті поєднання командної роботи та особистого планування;

– сформулювати та систематизувати вимоги до вебсистеми;

– обґрунтувати вибір стека технологій, для реалізації серверної та клієнтської частин проєкту, підходу для забезпечення зв'язку між ними та СКБД для зберігання даних системи;

– розробити архітектуру програмного забезпечення, що включає проєктування схеми реляційної бази даних, логіку серверної частини та компонентну структуру клієнтського інтерфейсу (SPA);

– реалізувати підсистему розмежування прав доступу та ролей у межах робочих просторів та командних проєктів для забезпечення конфіденційності даних окремих проєктів та учасників команди;

– провести тестування клієнтської та серверної частин вебсистеми на відповідність поставленим вимогам, а також виконати експериментальне дослідження продуктивності роботи системи при різних рівнях навантаження бази даних.

Практична значення роботи полягає в створенні концепції та прототипу інноваційної платформи, яка може бути використана як основа для розробки комерційного продукту. Результати дослідження дозволять запропонувати ринку

інструмент, що підвищує особисту продуктивність користувачів, знижує рівень стресу від управління багатьма потоками завдань завдяки управління ними у межах однієї системи та створює більш комфортне й інтегроване цифрове робоче середовище.

Апробація результатів дослідження. Отримані результати дослідження було опубліковано у науковому журналі «Комп'ютерно-інтегровані технології: освіта, наука, виробництво». Луцьк: Луцький НТУ, 2025. Вип. № 59, 61 [1, 2].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Сучасний етап цифрової трансформації суспільства докорінно змінив підхід до професійної діяльності та організації робочих процесів. Глобалізація, розвиток комунікаційних технологій та, значною мірою, виклики, спричинені пандемією, прискорили перехід від традиційної офісної моделі до гнучких форматів роботи, зокрема віддаленого та гібридного. В цих умовах ключовим фактором успіху стає не стільки фізична присутність працівника, скільки його здатність ефективно функціонувати в рамках злагоджених командних процесів, що часто розподілені географічно. Домінантною парадигмою управління такими командами стали гнучкі методології, серед яких провідні позиції займають Agile та його фреймворки, як-от Scrum та Kanban [3]. На відміну від класичного каскадного підходу (Waterfall), що передбачає жорстку послідовність етапів, Agile робить акцент на ітеративності, адаптивності та постійній комунікації, що дозволяє командам швидше реагувати на зміни та постачати результати роботи замовнику, невеликими, але регулярними ітераціями.

Ця трансформація нерозривно пов'язана зі зростанням залежності від цифрових інструментів. Системи управління проєктами, корпоративні чати у месенджерах, хмарні сховища та засоби для відеоконференцій перетворилися з допоміжних утиліт на основу робочого середовища. Однак це явище має і зворотний бік – проблему інформаційного перевантаження [4]. Сучасний працівник змушений одночасно обробляти величезні потоки даних з різних джерел: завдання в таск-трекері, сповіщення в Slack чи Microsoft Teams, електронні листи, коментарі в документах. Постійне перемикання між цими інструментами створює використання значної кількості зусиль, необхідних для обробки отриманої інформації. Кожне перемикання контексту, тобто перехід від

одного завдання або програми до іншої, вимагає від мозку часу та енергії на те, щоб змістити фокус на новому ресурсі, що в сукупності призводить до зниження концентрації, збільшення кількості помилок та, як наслідок, падіння загальної продуктивності. Таким чином, виникає парадокс: інструменти, покликані підвищувати ефективність, за неправильного використання стають її гальмом.

Для вирішення цих викликів як на рівні команд, так і на індивідуальному рівні, було розроблено низку теоретичних концепцій та практичних методик. Якщо в командній роботі панують згадані Agile-практики, то у сфері персональної продуктивності також існують перевірені часом системи.

Однією з найвпливовіших є методологія Getting Things Done (GTD), розроблена Девідом Алленом. Її центральна ідея полягає у необхідності вивантаження всіх завдань, ідей та зобов'язань із короткочасної пам'яті людини в надійну зовнішню систему [5]. Це дозволяє звільнити розумові ресурси від постійного нагадування про незавершені справи та сфокусуватися безпосередньо на виконанні поточного завдання. Фундаментальним принципом, на якому ґрунтується GTD, є теза, що людський розум ефективний для генерації ідей, але вкрай неефективний для їх зберігання. Незавершені завдання, невиконані обіцянки та невизначені терміни роботи над проєктами створюють так-звані «відкриті цикли» (open loops), які постійно споживають енергію та створюють фоновий стрес, навіть якщо людина не думає про них свідомо.

Для протидії цьому Аллен пропонує структурований процес роботи, що складається з п'яти етапів, з інформацією: збір, обробка, організація, огляд та виконання. На етапі обробки кожен зібраний елемент (вхідні дані) аналізується на предмет того, чи вимагає він якихось дій. Якщо так, ключовим кроком є визначення наступної конкретної фізичної дії [6]. Це принципово відрізняє GTD від звичайних списків справ, які часто містять розмиті цілі (наприклад, «організувати відпустку»), а не чіткі кроки (наприклад, «зателефонувати в турагенцію для консультації»). Завдання, що вимагають більше однієї дії, класифікуються як «проєкти» і відстежуються окремо, слугуючи орієнтиром для визначення наступних кроків. Із таким підходом, «GTD» – це не просто техніка

тайм-менеджменту, а комплексна система управління особистими зобов'язаннями, метою якої є досягнення стану, коли свідомість є спокійною, ясною та готовою адекватно реагувати на будь-які зовнішні подразники, а не відволікатися на внутрішні нагадування.

Іншим фундаментальним інструментом пріоритезації є матриця Ейзенхауера, що пропонує класифікувати завдання за двома критеріями: терміновість та важливість. Цей підхід допомагає відокремити справді значущі справи, що рухають до довгострокових цілей, від рутинної роботи, яка вимагає негайної уваги, але не має стратегічної цінності [7].

Приклад схеми матриці Кові показано на рисунку 1.1.

Рисунок 1.1 – Візуальна презентація матриці Ейзенхауера [8]

Схема включає в себе чотири розподілених категорії [9]:

– важливі, термінові – охоплює кризові ситуації та завдання з критичними дедлайнами, які вимагають негайного виконання для запобігання серйозним негативним наслідкам;

– важливі, не термінові – це стратегічно значущі завдання, такі як планування та розвиток, які не є терміновими, але формують основу для довгострокового успіху та попередження можливих майбутніх криз;

– термінові, не важливі – схема, до якої відносяться завдання, що виглядають терміновими, але не наближають до важливих цілей, тому їх слід делегувати, мінімізувати або автоматизувати, щоб уникнути відволікання;

– не термінові, не важливі – це діяльність з мінімальною цінністю, що не має ані терміновості, ані важливості, і яку слід свідомо ідентифікувати та використовувати як «поглинач» часу.

Метод Pomodoro передбачає роботу короткими, фіксованими інтервалами з регулярними перервами, спрямовані на боротьбу з прокрастинацією та підтримання високого рівня концентрації. Розроблений підхід Франческо Чірілло наприкінці 1980-х років, отримав свою назву від кухонного таймера у формі помідора, який автор використовував у власних інтересах, для покращення продуктивності свого навчання [10].

Класична структура методу складається з чітко визначених циклів: 25 хвилин (іноді 20, в залежності від персональних вподобань) інтенсивної, безперервної роботи над одним завданням (один «помідор»), після яких слідує коротка, зазвичай, 5-хвилинна перерва. Після завершення чотирьох таких циклів рекомендується зробити довшу перерву тривалістю 15-30 хвилин для повноцінного відновлення. Ключовим принципом техніки є неподільність «помідора»: якщо робочий інтервал було перервано, він не зараховується, і його необхідно розпочати знову. Це привчає свідомо захищати свій фокус від зовнішніх чинників відволікання уваги. Для боротьби з раптовими внутрішніми думками чи ідеями, пропонується швидко занотувати їх і негайно повернутися до виконання основного завдання, не перериваючи загальний процес роботи.

Психологічна ефективність методу полягає у тому, що він розбиває великі, часто «страшні» (від об'єму роботи, або недостатніх знань) завдання на невеликі, керовані сегменти, що суттєво знижує поріг для початку роботи та є дієвим інструментом проти прокрастинації. Регулярні перерви, у свою чергу, запобігають відпочинку мозку та допомагають підтримувати стабільно високий рівень продуктивності протягом усього дня. Таким чином, метод Pomodoro є не просто технікою відліку часу, а цілісною системою для тренування уваги та управління енергією, що перетворює абстрактне поняття часу на конкретну, вимірювану одиницю, що стає прямо-пропорційною до кількості докладених зусиль.

Інша техніка, доволі популярна – це використання «Парковки ідей» («Parking lot»). Суть даного підходу полягає в зосередженні на основному завданні: коли в процесі роботи втрачається фокус, а саме появляється якесь додаткове питання, завдання або ідея, яка, фактично, не внесе суттєвих змін (або взагалі не вплине) на реалізацію основної задачі, ця думка вноситься в «паркінг». Раз у певний період, або в спеціально визначений час, користувач переглядає, «припарковані» ідеї, питання або топіки для аналізу, і фільтрує їх, вносячи виділені у основний список завдань для розгляду [11].

Аналіз згаданих методик та інструментів виявляє ключову проблему: методології управління командними проєктами та системи особистої продуктивності розвиваються переважно паралельно, існуючи у двох різних світах. Інструменти, що їх реалізують, рідко пропонують механізми для їхньої гармонійної інтеграції, залишаючи це завдання на розсуд самого користувача.

Для повного розуміння предметної області та обґрунтування необхідності створення нового рішення необхідно провести детальний аналіз існуючих на ринку програмних продуктів. Їх можна поділити на три великі категорії, кожна з яких має свої сильні сторони та фундаментальні недоліки.

Перша категорія – це корпоративні системи управління проєктами, орієнтовані насамперед на командну взаємодію (Team-First Platforms). Яскравими представниками цього сегменту є Jira, Asana, Trello та Redmine.

Вебплатформа Jira, розроблена компанією Atlassian, де-факто є стандартом в індустрії розробки програмного забезпечення. Що не дивно – компанія спеціалізується на продуктах для роботи над різного роду проєктами, розробник відомий своїми програмними рішеннями для менеджменту IT-проєктами, зокрема Confluence, додаток, який використовується більш як 80 тис. компаніями по всьому світу [12]. Проте, компанія Atlassian не стала на місці: функціонал Jira доповнюється багатьма новими рішеннями, які дозволяють впровадити функціонал для зручності управління не лише IT-проєктами, а й проєктами бізнес-плану, просторами для аналітики, дизайну, управління людськими ресурсами тощо. Для Jira пропонується безліч потужних функцій «з нуля»,

зокрема, планування спринтів, управління беклогом, приклад якого показано на рисунку 1.2, відстеження помилок, аналітики робочих процесів, інтеграції з Git (система контролю версій), та ще багато інших інструментів, які можна підключити за допомогою великого спектру доступних доповнень.

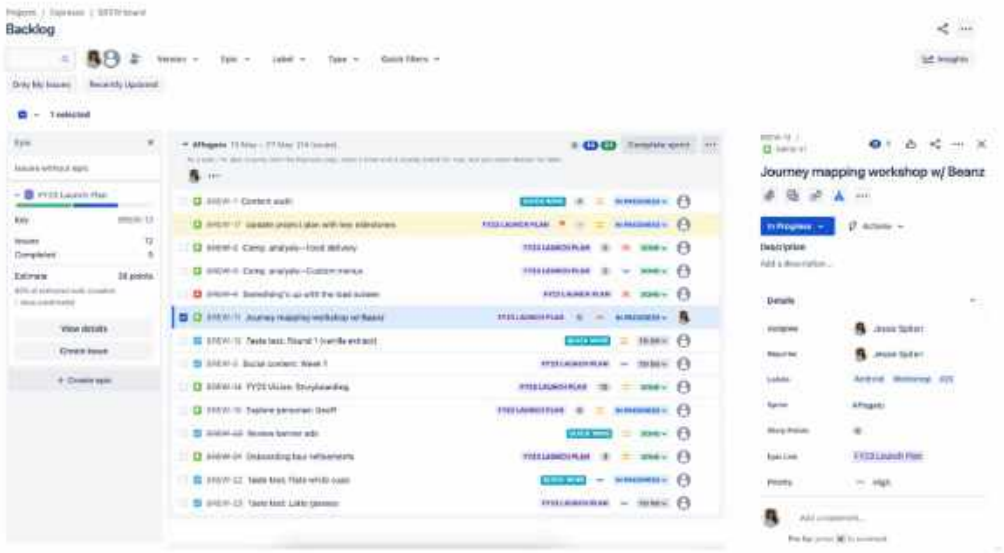

Рисунок 1.2 – Зовнішній вигляд сторінки беклогу проекту [13]

На рисунку 1.3 можна побачити зовнішній вигляд торгового майданчику доповнень для Jira.

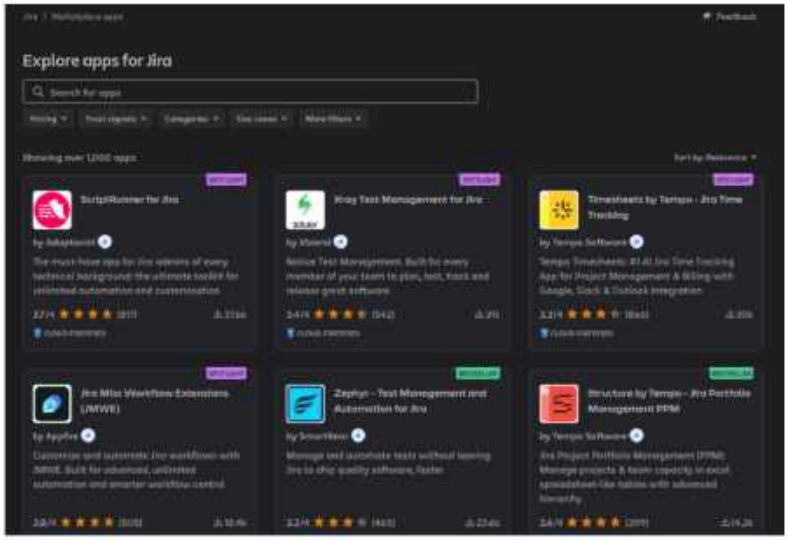

Рисунок 1.3 – Торговий майданчик доповнень для Jira [14]

Asana є більш універсальним рішенням, що дозволяє візуалізувати проєкти у вигляді списків, приклад на рисунку 1.4, дошок Канбан, діаграм Ґанта та календарів, що робить їх популярними серед маркетингових, дизайнерських та бізнес-команд.

Рисунок 1.4 – Зовнішній вигляд сторінки зі списком завдань у Asana [15]

Trello, зі свого боку, завоював популярність завдяки своїй простоті та інтуїтивності, і вважається найкращим інструментом для роботи команд, які використовують систему організації робочого процесу «Канбан» [16].

Redmine є системою управління проєктами з відкритим вихідним кодом, яка відома своєю гнучкістю та потужним функціоналом. Основна перевага Redmine полягає у його універсальності: система може бути легко адаптована для різних типів проєктів і команд завдяки широкому набору модулів та налаштувань. Redmine добре підходить під ситуації, де потрібно кількома проєктами з єдиної платформи, забезпечуючи прозорість та зручність у роботі.

Сильною стороною всіх цих систем є їхня здатність структурувати складні процеси, забезпечувати прозорість щодо статусу завдань та сприяти ефективній колаборації всередині команди. Однак їхній головний недолік полягає в їхній філософії, основаній на менеджменті командної роботи. Вони розглядають користувача виключно в ролі члена команди або виконавця завдань у рамках конкретного проєкту. У цих системах практично немає місця для особистого контексту працівника: його довгострокових кар'єрних цілей, завдань з інших проєктів, особистих справ, що можуть впливати на робочий графік, чи просто

ідей, не пов'язаних із поточною роботою. Таким чином, працівник змушений вести подвійне життя: одне – всередині корпоративної системи, інше – за її межами.

Друга категорія – це персональні таск-менеджери та органайзери (Individual-First Platforms). До них належать такі популярні додатки, як Todoist, Microsoft To Do, Things та TickTick. Ці інструменти створені з однією метою: допомогти людині організувати власне життя. Їхні переваги – це простота, гнучкість, доступність на всіх пристроях та фокус на індивідуальних потребах. Вони дозволяють швидко фіксувати завдання, встановлювати нагадування, створювати повторювані події, відстежувати звички та структурувати особисті проєкти (наприклад, «Ремонт у квартирі» або «Підготовка до марафону»). Однак їхня сила є одночасно і їхньою слабкістю. Вони повністю ізольовані від корпоративного робочого середовища. Критично важливе завдання, призначене користувачеві в Asana з дедлайном «на завтра», не з'явиться автоматично в його Todoist. Користувач змушений вручну дублювати інформацію, що не тільки забирає час, але й створює ризик людської помилки – можна забути перенести завдання, неправильно вказати дедлайн або втратити важливий контекст. Цей інформаційний розрив змушує людину постійно тримати в голові дві окремі ментальні моделі своїх зобов'язань – робочу та особисту, що прямо суперечить принципам GTD.

Третя, відносно нова категорія – це гнучкі «все-в-одному» платформи (All-in-One Toolkits), найвідомішими представниками яких є Notion, ClickUp та Airtable. Ці інструменти позиціонують себе як універсальні робочі простори, що можуть адаптуватися під будь-які потреби. Їхня архітектура, заснована на концепції блоків та баз даних, дозволяє користувачам створювати власні системи для управління знаннями, проєктами, CRM та інші. Теоретично, в Notion чи ClickUp можна збудувати як складну систему для командної роботи, так і персональний таск-менеджер, і навіть пов'язати їх між собою. Приклад одного з шаблонів для створення таск-менеджера в Notion показано на рисунку 1.5.

Рисунок 1.5 – Шаблон таск-менеджера для Notion [17]

Ця неймовірна гнучкість є їхньою головною перевагою. Проте вона ж породжує і ключовий недолік. Ці платформи є «конструкторами», а не готовими рішеннями. Вони вимагають від користувача значних часових інвестицій та високого рівня дисципліни для початкового налаштування та подальшої підтримки створеної системи. Навіть при тому, що є можливість використання шаблонів суспільства – більшість цих шаблонів є платними, і зробленими «для себе». Але найголовніше – в системах даного типу відсутній вбудований, продуманий механізм інтелектуальної синхронізації між командними та особистими просторами. Користувач може вручну налаштувати зв'язки між різними базами даних, але це не буде нативною, інтуїтивно зрозумілою інтеграцією. Це радше «обхід» ситуації, що вимагає глибокого розуміння логіки та нюансів інтеграції платформи. Відсутність спеціалізованого функціоналу для гармонізації робочих та особистих завдань робить ці платформи потужними, але занадто складними для середньостатистичного користувача, який прагне простоти та автоматизації. У результаті, інструмент, покликаний зменшити когнітивне навантаження, парадоксальним чином створює додатковий рівень складності, вимагаючи від користувача бути не лише виконавцем, а й архітектором власного робочого простору

Таким чином, проведений аналіз існуючих програмних рішень дозволяє чітко сформулювати невирішену проблему та дослідницький розрив. Ринок

інструментів продуктивності виявився сильно фрагментованим: він пропонує або потужні, але знеособлені корпоративні системи, або прості, але ізольовані персональні менеджери. Платформи «все-в-одному» хоч і намагаються подолати цей розрив, проте пропонують надмірну складність замість готової методології інтеграції. Отже, ключовий дослідницький розрив полягає у відсутності програмної платформи, яка б нативно та безшовно поєднувала структуроване, орієнтоване на команду середовище, з гнучким, персоналізованим простором користувача через механізм часткової, інтелектуальної та легко налаштовуваної синхронізації завдань. Це є необхідним інструментом, який би дозволив користувачеві бачити завдання з робочого проєкту, такого як у Jira чи Asana, у своєму особистому планувальнику, і не як чужорідний елемент, а як частину єдиної картини своїх зобов'язань, поряд із записом до лікаря чи планом особистого розвитку.

Можна зробити висновок, що існує гостра потреба в розробці нової архітектури для системи управління завданнями, яка б втілювала дещо особливий підхід до продуктивності. Проведений аналіз підтверджує актуальність, наукову новизну та практичну доцільність створення такої інтегрованої платформи. Це дослідження закладає основу для проєктування та розробки системи, яка має на меті заповнити виявлену прогалину на ринку, запропонувавши користувачам інструмент для досягнення справжнього балансу та синергії між професійним та особистим життям.

1.2 Огляд і аналіз методів та засобів розробки вебсистеми для організації командних проєктів та особистих завдань для вирішення проблеми дослідження

Вирішення виявленої проблеми інтеграції командних та особистих завдань потребує ретельного підходу до вибору технологічного стеку для розробки програмного забезпечення. Ефективність, масштабованість, підтримка від спільноти розробників та зручність розробки – всі є критичними факторами, які

визначають успішність реалізації такої складної системи. Сучасний ландшафт веброзробки пропонує широкий спектр фреймворків, бібліотек та систем керування базами даних, кожна з яких має свої переваги та обмеження. Для забезпечення оптимального вибору необхідно провести порівняльний аналіз наявних технологій для розробки та обґрунтувати рішення, які найкращим чином відповідають специфічним вимогам проєкту.

У сфері серверної розробки (бекенд) існує декілька домінуючих фреймворків, що широко використовуються для створення вебдодатків корпоративного рівня. Серед найпопулярніших можна виділити Laravel (PHP), Django (Python), Spring Boot (Java), Express.js (Node.js) та Ruby on Rails (Ruby). Кожен з цих фреймворків має свою екосистему, філософію розробки та сферу застосування.

Django позиціонує себе як «фреймворк для перфекціоністів з дедлайнами» і відрізняється філософією «batteries included», що означає наявність широкого спектру передбачених вбудованих компонентів без необхідності встановлення додаткових допоміжних пакетів [18]. За даними Python Developers Survey 2023, Django залишається найпопулярнішим вебфреймворком у Python-екосистемі, випереджаючи навіть Flask та FastAPI у сфері повноцінних вебдодатків [19]. Ключовою перевагою Django є його автоматична генерації адміністративної панелі – унікальна можливість, що дозволяє створювати повнофункціональний інтерфейс управління даними без написання жодного рядка коду, що, інколи, зберігає певну кількість ресурсів. Ця панель автоматично генерується на основі визначених моделей в розроблюваному додатку, а це надає можливості для читання, оновлення, створення, та видалення записів (CRUD операції), а також підтримує складні фільтри, пошук та масові операції [20].

Архітектура Django базується на патерні MTV (Model-Template-View), який є адаптацією класичного MVC (Model-View-Controller). Вбудований ORM Django надає потужні можливості для роботи з базами даних, включаючи складні запити з використанням об'єктів-запитів, агрегацію, транзакції та міграції для схеми бази даних. Система міграцій Django особливо елегантна – вона автоматично

відстежує зміни в моделях та генерує відповідні скрипти для запуску в обраній системі керування базами даних, що значно спрощує розгортання та стеження за версіями схеми бази даних [21]. Однак жорстка структура фреймворку може стати обмеженням у проєктах, що вимагають нестандартних архітектурних рішень. Відхилення від передбаченої парадигми потребує значних зусиль та глибокого розуміння внутрішніх механізмів фреймворку. Крім того, монолітний характер Django робить його менш придатним для мікросервісної архітектури, хоча поява FastAPI та інших асинхронних фреймворків поступово змінює цю ситуацію в Python-екосистемі. Варто також зазначити, що Django традиційно орієнтований на синхронну обробку запитів, хоча з версії 3.0 з'явилася часткова підтримка асинхронних види та посередники, що розширює можливості для високонавантажених застосунків.

Spring Boot є частиною більшої екосистеми Spring Framework і призначений для спрощення розробки «enterprise-додатків» на Java. Згідно з JVM Ecosystem Report 2024, Spring Boot використовується у понад 60 % корпоративних Java-проєктів, що робить його де-факто стандартом для розробки бізнес-додатків у Java-екосистемі [22]. Фреймворк побудований на принципі «convention over configuration» і пропонує систему «стартерів» – попередньо налаштованих залежностей, що автоматично конфігурують необхідні компоненти для конкретних завдань (наприклад, залежність «spring boot starter web» для вебдодатків або «spring boot starter data jpa» для роботи з базами даних). Ключовою перевагою Spring Boot є його зрілість та надійність – фреймворк розробляється та підтримується компанією VMware з 2013 року (раніше – компанією Pivotal) та має величезну кількість «загартованих» компонентів для вирішення типових задач «enterprise-розробки» [23].

Архітектура Spring Boot базується на концепції інверсії управління (IoC) та впровадження залежностей (Dependency Injection), що забезпечує високу модульність та зручність у тестуванні коду. Фреймворк надає потужні можливості для створення RESTful API, інтеграції з різноманітними базами даних через Spring Data, реалізації безпеки через Spring Security та побудови

реактивних додатків через Spring WebFlux. Spring Boot сильно виділяється серед інших у сценаріях, що вимагають складної бізнес-логіки, транзакційної цілісності та інтеграції з корпоративними системами. Однак використання Spring Boot має і свої недоліки. По-перше, навіть базові застосунки вимагають значного обсягу пам'яті (memory footprint) – стандартний додаток на основі Spring Boot може споживати 200-400 МБ оперативної пам'яті навіть у стані спокою, що робить його менш придатним для мікросервісів з обмеженими ресурсами. По-друге, час запуску може досягати 10-30 секунд для складних додатків, що ускладнює розробку та тестування. По-третє, крива навчання Spring Boot є досить крутою – для ефективного використання фреймворку необхідне глибоке розуміння Java-екосистеми, шаблонів проєктування та самої архітектури Spring. Варто також зазначити, що багатослівність (verbosity) Java як мови програмування призводить до того, що навіть прості операції вимагають значно більше коду порівняно з сучасними динамічними мовами [24].

Express.js позиціонує себе як швидкий, гнучкий, мінімалістичний вебфреймворк для Node.js, і є найпопулярнішим фреймворком у екосистемі JavaScript для серверної розробки. За даними Stack Overflow 2023, Express.js використовується у понад 70 % Node.js проєктів, що робить його найбільш поширеним інструментом для бекенд розробки на JavaScript [25]. Ключовою особливістю Express.js є його мінімалістичний підхід – фреймворк надає лише базовий функціонал для обробки HTTP-запитів та маршрутизації, залишаючи розробнику повну свободу у виборі архітектури та додаткових компонентів. Це робить Express.js надзвичайно гнучким інструментом, здатним адаптуватися до будь-яких вимог проєкту, від простих API до складних додатків із підтримкою різного функціоналу.

Архітектура Express.js базується на концепції middleware – функцій, що послідовно обробляють запити та можуть модифікувати об'єкти «request» та «response». Ця модель забезпечує високу модульність та можливість повторного використання коду. Express.js особливо ефективний у сценаріях, що вимагають асинхронної обробки великої кількості одночасних з'єднань, завдяки

неблокуючій «event-driven» архітектурі Node.js, що базується на виконанні коду способом «циклу подій», яка дозволяє не блокувати виконання коду зовнішніми чинниками, такими як великі запити до бази даних. Це робить його ідеальним вибором для «real-time» додатків (чати, командні інструменти), API-шлюзів та мікросервісів. Однак мінімалістичний характер Express.js має і свої недоліки. По-перше, відсутність структурованої архітектури «з коробки» вимагає від розробника самостійного прийняття всіх архітектурних рішень – організації файлової структури, вибору ORM або «query builder» для роботи з базами даних, налаштування валідації даних, обробки помилок, логування тощо. Це збільшує час розробки, особливо на початкових етапах проєкту, та створює ризик помилок у архітектурі, якщо розробник не має достатнього досвіду. По-друге, екосистема Node.js відома своєю фрагментованістю – для кожної задачі існує десятки альтернативних рішень, і вибір правильного може бути нетривіальним завданням. По-третє, однопотокова природа Node.js, а точніше, мови JavaScript (навіть з підтримкою async/await) робить його менш придатним для «CPU-intensive» операцій, таких як складні обчислення або обробка зображень. Варто також зазначити, що JavaScript як мова програмування не має статичної типізації (хоча TypeScript частково вирішує цю проблему), що може призводити до помилок типу «runtime» у великих проєктах.

Ruby on Rails (скорочена назва – «Rails») є повнофункціональним вебфреймворком, побудованим на принципах «Convention over Configuration» (CoC) та «Don't Repeat Yourself» (DRY). За даними Ruby on Rails Community Survey 2023, Rails залишається провідним фреймворком у Ruby-екосистемі і широко використовується у стартапах та компаніях середнього розміру, особливо у сфері SaaS-продуктів [26]. Ключовою філософією Rails є максимізація продуктивності розробника (developer productivity) – фреймворк нав'язує певні угоди та структуру проєкту, що дозволяє розробникам зосередитися на бізнес-логіці замість прийняття рутинних рішень щодо організації коду. Rails був піонером багатьох концепцій, що згодом стали стандартом у веброзробці,

включаючи RESTful маршрутизацію, шаблон «активного запису» для ORM та «scaffolding» для автоматичної генерації базового CRUD-функціоналу [27].

Архітектура Rails базується на класичному MVC-патерні (Model-View-Controller) і надає великий набір інструментів «з коробки», включаючи «Active Record ORM», систему міграцій бази даних, вбудовану підтримку тестування (з RSpec або Minitest), «Action Mailer» для роботи з електронною поштою та «Active Job» для фонової обробки завдань. Особливо сильною стороною Rails є його підхід «batteries included» – фреймворк включає рішення для більшості типових завдань веброзробки, від автентифікації (через «гем devise») до файлового зберігання (через «Active Storage»). Rails також має потужну систему генераторів, що дозволяє швидко створювати каркаси, міграції, моделі, контролери та інші компоненти через командний рядок [28]. Однак Rails має і суттєві обмеження. По-перше, продуктивність Ruby, як мови програмування, значно поступається компільованим мовам (Java, Go) та навіть деяким інтерпретованим (Python, з оптимізаціями), що може стати проблемою для високонавантажених систем. По-друге, «магічність» Rails – автоматична конфігурація, угоди іменування, неявні залежності – може ускладнювати розуміння того, що насправді відбувається «під капотом», особливо для початківців. По-третє, монолітна архітектура Rails менш придатна для мікросервісного підходу, хоча останні версії фреймворку (Rails 7+) додали підтримку API-only режиму та покращили інтеграцію з фронтенд фреймворками. Варто також зазначити, що популярність Rails поступово знижується порівняно з піком середини та цінця 2010-х років, хоча фреймворк продовжує активно розвиватися та має лояльну спільноту розробників.

Laravel виділяється серед інших рішень своїм унікальним балансом між потужністю, елегантністю коду та зручністю розробки. За даними Stack Overflow Developer Survey 2024 [29], Laravel залишається одним з найулюбленіших вебфреймворків серед розробників, що свідчить про високу задоволеність його використанням. Ключовою особливістю Laravel є його філософія «developer-friendly» – синтаксис фреймворку спроектований таким чином, щоб

код був максимально виразним та зрозумілим для розробників. Це досягається через використання сучасних можливостей PHP, таких як анонімні та стрілкові функції, трейти та атрибути, а також через впровадження патернів проєктування, що стали стандартом у галузі. Eloquent ORM, вбудована система інтеграції баз даних Laravel, дозволяє працювати з даними через інтуїтивно зрозумілий об'єктно-реляційний інтерфейс, що значно спрощує написання запитів та підвищує читабельність коду [30].

Особливо важливою характеристикою Laravel є активна та відкрита спільнота розробників. На відміну від багатьох інших фреймворків, де розвиток контролюється переважно корпоративними командами, Laravel має модель розвитку, що активно залучає пропозиції від спільноти. Регулярні мінорні оновлення, які виходять кожен тиждень, та глобальні, що виходять кожен рік, не тільки виправляють помилки та покращують безпеку, а й впроваджують інноваційні функції, запропоновані самими користувачами фреймворку. Це створює унікальну екосистему, де інструмент еволюціонує відповідно до реальних потреб розробників. Додатковою перевагою є наявність офіційних «стартер-кітів» (starter kits) – готових шаблонів проєктів, що включають попередньо налаштовану автентифікацію, авторизацію, управління профілями користувачів та базові компоненти користувацького інтерфейсу [31]. Це дозволяє значно скоротити час на налаштування початкової інфраструктури проєкту та сфокусуватися на реалізації унікальної бізнес-логіки. Варто зазначити, що існує також підтримка стартер-кітів, створених спільнотою, що розширює можливості швидкого старту для специфічних типів додатків.

У сфері клієнтської розробки (фронтенд) ландшафт технологій не менш різноманітний. Основними конкурентами є React, Vue.js, Angular.

Angular є повноцінним фреймворком, розробленим та підтримуваним компанією Google, що відрізняється жорстко визначеною архітектурою на основі TypeScript, декораторів та використанням принципу «ін'єкції залежностей» (dependency injection). За даними State of JS 2023, Angular залишається популярним вибором у великих корпоративних проєктах та

«enterprise-додатках», де критичною є передбачуваність, стандартизація кодової бази та можливість роботи великих команд над одним проєктом [32]. Фреймворк нав'язує чітку структуру проєкту з модулями, компонентами, сервісами та директивами, що забезпечує уніфікований підхід до розробки незалежно від розміру команди. Angular включає «з коробки» рішення для маршрутизації (Angular Router), HTTP-клієнта (HttpClient), форм (у тому числі, реактивні Reactive Forms та Template-driven Forms), валідації даних та інших типових завдань фронтенд розробки. Однак специфічний синтаксис Angular, що включає директиви, такі як «*ngFor» та «*ngIf», а також концепції «RxJS» та «Observables», створює високий поріг входу для початківців. Крім того, розмір фінальної збірки може бути значно більше очікуваного, через включення багатьох вбудованих модулів, навіть якщо вони не активно використовуються у конкретному додатку, що негативно впливає на швидкість завантаження, особливо для користувачів з повільним інтернет-з'єднанням або на мобільних пристроях.

Vue.js позиціонує себе як прогресивний фреймворк, що поєднує найкращі ідеї Angular та React, пропонуючи інтуїтивний синтаксис та поступову адаптацію – від простої бібліотеки до повноцінного фреймворку. Vue.js має одну з найвищих показників задоволеності серед розробників і, що цікаво, є особливо популярним у азіатському регіоні та серед фрилансерів [33]. Ключовою перевагою Vue.js є його доступність для початківців завдяки простому «template based» синтаксису, детальній документації та плавній кривій навчання. Vue надає офіційні рішення для маршрутизації (Vue Router) та управління станом (Pinia, раніше Vuex), що забезпечує єдину екосистему без необхідності вибору між різними альтернативами. Однак випуск Vue 3 у 2020 році призвів до фрагментації спільноти через введення «Composition API» як альтернативи традиційному «Options API». Хоча Composition API надає більше можливостей для організації логіки та повторного використання коду (особливо через «composables»), він значно ускладнив синтаксис та вимагає глибшого розуміння реактивності Vue.

Це призвело до того, що частина команд, які раніше цінували Vue за простоту, почала розглядати альтернативи, такі як React або Svelte.

Svelte представляє фундаментально інший підхід до побудови користувацьких інтерфейсів, діючи як компілятор, а не як «runtime-based» фреймворк. На відміну від React, Vue та Angular, які використовують віртуальний DOM для оптимізації оновлень інтерфейсу, Svelte компілює компоненти у високооптимізований «ванільний» JavaScript код на етапі збірки проєкту. Svelte має найвищий показник інтересу серед нових фронтенд технологій і стрімко набирає популярності завдяки своїй простоті та продуктивності [34]. Ключові переваги Svelte включають мінімальний розмір фінальної збірки (оскільки не потрібно включати «runtime-бібліотеку» фреймворку), інтуїтивний синтаксис без JSX або шаблонів-директив, та високу продуктивність завдяки відсутності віртуального DOM. Svelte також пропонує вбудовану підтримку анімацій, переходів та реактивності через синтаксис «$:», що робить код більш виразним. Однак екосистема Svelte все ще значно менша порівняно з React або Vue – обмежена кількість готових UI-бібліотек, сторонніх інтеграцій та «community-driven» рішень може створювати труднощі у складних проєктах. Крім того, ринок праці для Svelte-розробників значно менший, що може ускладнити пошук спеціалістів або зміну роботи для розробників, які спеціалізуються виключно на цій технології.

React, розроблений та підтримуваний Meta, протягом останніх років демонструє стрімке зростання популярності. За даними State of JavaScript 2023, React залишається найбільш використовуваною бібліотекою для створення користувацьких інтерфейсів, з показником використання понад 80 % серед опитаних розробників [35]. Важливо зазначити, що React технічно є бібліотекою, а не повноцінним фреймворком, що надає розробникам більшу свободу у виборі архітектурних рішень. Основна сила React полягає в його компонентно-орієнтованому підході та концепції віртуального DOM, що дозволяє ефективно оновлювати лише ті частини інтерфейсу, які дійсно

змінилися. Це призводить до високої продуктивності навіть у складних, динамічних додатках.

Екосистема React є найбільш розвиненою серед усіх фронтенд технологій. Величезна кількість бібліотек, інструментів розробки та готових рішень дозволяє швидко реалізувати практично будь-який функціонал. Особливо цінується наявність таких бібліотек, як «shadcn» – набір компонентів з відкритим вихідним кодом, що пропонує елегантні, доступні та легко кастомізовані елементи інтерфейсу. На відміну від традиційних UI-бібліотек, «shadcn» не встановлюється як залежність, а інтегрується безпосередньо в код проєкту, що дозволяє повністю контролювати та адаптувати компоненти під специфічні потреби дизайн-системи.

Вибір системи керування базами даних (СКБД) є не менш критичним рішенням. Найпопулярнішими реляційними СКБД для вебдодатків є MySQL, PostgreSQL, Microsoft SQL Server та Oracle Database. Для певних типів завдань також розглядаються NoSQL рішення, такі як MongoDB або Redis. PostgreSQL традиційно вважається найбільш функціонально багатою СКБД з відкритим кодом, що пропонує розширену підтримку складних типів даних, повнотекстового пошуку, JSON, геопросторових даних та багато інших можливостей [36]. Його сильною стороною є висока продуктивність операцій запису та модифікації даних, а також краща підтримка паралельних транзакцій. MongoDB, як представник NoSQL бази даних, пропонує гнучкість схеми та горизонтальну масштабованість, що робить його привабливим для проєктів з неструктурованими даними та непередбачуваним зростанням.

MySQL, у свою чергу, залишається оптимальним вибором для певного класу додатків, зокрема систем управління проєктами. Численні бенчмарк-тести, проведені незалежними дослідниками, демонструють, що MySQL має суттєву перевагу в продуктивності операцій читання даних. За результатами тестування, проведеного у 2023 році дослідницькою групою з University of California, MySQL показав на 15-25 % вищу швидкість виконання SELECT запитів порівняно з PostgreSQL при роботі з базами даних середнього розміру (до 100 ГБ) [31]. Це

критично важливо для систем управління проєктами, де переважна більшість операцій – це саме читання інформації: перегляд списків завдань, читання коментарів, відображення статусів, завантаження історії змін. На відміну від систем електронної комерції, соціальних мереж або фінансових платформ, де потік даних є значно більшим і операції запису та оновлення відбуваються з високою інтенсивністю, система управління проєктами характеризується відносно стабільним об'ємом даних та переважанням операцій перегляду.

Важливо зазначити, що MySQL не є ідеальним рішенням для всіх сценаріїв. Його продуктивність може значно знижуватися при роботі з надзвичайно великими базами даних (понад 500 ГБ), а також у випадках з дуже складними аналітичними запитами. PostgreSQL у таких ситуаціях дійсно має переваги завдяки більш розвиненому оптимізатору запитів та ширшому набору вбудованих функцій для роботи з даними. У контексті розроблюваної системи існують операції, такі як складне фільтрування та сортування завдань за множинними критеріями в проєктах чи особистих просторах, де розширені можливості PostgreSQL могли б бути корисними [37]. Однак, враховуючи те, що в типовому проєкті кількість активних завдань рідко перевищує кількість у кілька сотень, переваги від використання складних функцій PostgreSQL не будуть критичними. Натомість, висока швидкість отримання даних MySQL може забезпечити кращий користувацький досвід при повсякденній роботі з системою, де швидкість завантаження інтерфейсу та відгук на дії користувача є пріоритетом, і немає великої кількості одночасних запитів, що може повпливати на швидкість відповідей системи.

1.3 Постановка завдання на кваліфікаційну роботу магістра

На основі аналізу предметної області, інтеграція командної роботи та індивідуального планування сприяє підвищенню ефективності організації часу, забезпечує гнучкість планування та підвищує продуктивність, а створення вебсистеми для такої інтеграції слугуватиме цифровим рішенням, яке усуває

необхідність використання множини розрізнених інструментів. Для ефективної координації різнотипних завдань потрібно створити ресурс, який органічно поєднуватиме функціонал систем керування проєктами з можливостями особистих трекерів завдань, зберігаючи при цьому розмежування доступу між командними та приватними просторами. У свою чергу, аналіз засобів та методів розробки підтверджує можливість технічної реалізації такого рішення завдяки широкому спектру сучасних технологій веброзробки.

На основі проведеного аналізу було визначено мету даного дослідження: розробка та дослідження вебсистеми для організації особистих завдань та командних проєктів. Вебдодаток повинен поєднати робочі та особисті процеси користувача, що дозволить йому ефективніше контролювати та керувати власним тайм-менеджментом у єдиному цифровому просторі.

Для досягнення даної менти було розроблено перелік завдань, які повинні бути вирішені у результаті виконання кваліфікаційної роботи магістра:

– сформулювати та систематизувати вимоги до вебсистеми;

– обґрунтувати вибір стека технологій, для реалізації серверної та клієнтської частин проєкту, підходу для забезпечення зв'язку між ними та СКБД для зберігання даних системи;

– розробити архітектуру програмного забезпечення, що включає проєктування схеми реляційної бази даних, логіку серверної частини та компонентну структуру клієнтського інтерфейсу (SPA);

– реалізувати підсистему розмежування прав доступу та ролей у межах робочих просторів та командних проєктів для забезпечення конфіденційності даних окремих проєктів та учасників команди;

– провести тестування клієнтської та серверної частин вебсистеми на відповідність поставленим вимогам, а також виконати експериментальне дослідження продуктивності роботи системи при різних рівнях навантаження бази даних.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБСИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ КОМАНДНИХ ПРОЄКТІВ ТА ОСОБИСТИХ ЗАВДАНЬ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Результати аналізу предметної області та бази знань щодо існуючих систем показали доцільність використання архітектури, що забезпечує динамічну взаємодію з користувачем, швидкодію інтерфейсу та ефективне управління даними завдань і проєктів. З цією метою було обрано підхід односторінкового застосунку (SPA) із тісною інтеграцією серверної та клієнтської частин. Технічна реалізація базується на поєднанні бекенд логіки та фронтенд компонентів через спеціалізований інструмент, що забезпечує безперервну передачу даних від серверних контролерів до клієнтських компонентів.

Набір обраних технологій, наведено у таблиці 2.1.

Таблиця 2.1 – Стек технологій, що використовується для реалізації

Компонент	Рішення	Фактор
Серверна частина	Laravel	Широкий, «developer-friendly» синтаксис, наявність стартер-кітів, які можуть забезпечити зручний зв'язок з клієнтською частиною застосунку
Міст клієнт-серверної архітектури	Inertia.js	Основне призначення інструменту – тісне поєднання бекенду Laravel із популярними рішеннями фронтенду
Клієнтська частина	React	Велика кількість підтримуваних пакетів та доповнень, зокрема пакетів з широким вибором готових для використання компонентів, наприклад «shadcn»
Система керування базами даних	MySQL	Висока продуктивність зчитування даних порівняно з іншими популярними СКБД різних типів

Ключовою особливістю вибору бекенд фреймворку Laravel була філософія «developer-friendly» – синтаксис, спроектований таким чином, щоб код був максимально виразним та зрозумілим. Це досягається через використання сучасних можливостей мови програмування PHP, таких як анонімні та стрілкові функції, трейти та атрибути, а також через впровадження патернів проєктування, що стали певним стандартом розробки. Вбудована ORM-система дозволяє працювати з даними через інтуїтивно зрозумілий об'єктно-реляційний інтерфейс, що значно спрощує написання запитів та підвищує читабельність коду. Також, важливою характеристикою є активна та відкрита спільнота розробників. Регулярні оновлення не тільки виправляють помилки та покращують безпеку, а й впроваджують інноваційні функції, запропоновані самими користувачами фреймворку. Це створює унікальну екосистему, де інструмент еволюціонує відповідно до реальних потреб розробників. Додатковою перевагою є наявність офіційних стартер-кітів – готових шаблонів проєктів, що включають попередньо налаштовану автентифікацію, авторизацію та базові компоненти користувацького інтерфейсу. Це дозволяє значно скоротити час на налаштування початкової інфраструктури проєкту та сфокусуватися на реалізації унікальної бізнес-логіки.

Критичним елементом обраного технологічного стеку є Inertia.js – інноваційний інструмент, який виступає мостом між серверною та клієнтською частинами додатку. Традиційно, поєднання бекенд фреймворку з сучасною фронтенд бібліотекою вимагає створення RESTful API або GraphQL ендпоінтів, налаштування аутентифікації через токени, вирішення проблем CORS та інших складнощів, пов'язаних з розділенням фронтенду та бекенду. Цей процес є не тільки трудомістким, але й створює додаткові точки потенційних помилок та проблем безпеки, у той час як Inertia.js усуває цю складність, дозволяючи розробникам писати класичні серверні маршрути в Laravel, які повертають не JSON, а повноцінні, популяризовані даними React компоненти. При цьому навігація в додатку відбувається без повного перезавантаження сторінки, зберігаючи всі переваги Single Page Application (SPA).

Особливо цінною є глибока інтеграція Inertia.js з офіційними стартер кітами Laravel, основаними на Vue та React. Ці стартер-кіти надають повністю функціональну систему автентифікації, включаючи реєстрацію, вхід, відновлення пароля, двофакторну аутентифікацію та управління сесіями, реалізовану з використанням React компонентів через Inertia. Це означає, що розробник отримує готове для використання рішення, де складна логіка безпеки вже реалізована та протестована, а інтерфейсні компоненти можуть бути легко адаптовані під дизайн проєкту. Варто також згадати, що існує активна спільнота, яка створює та підтримує власні стартер-кіти для специфічних потреб, розширюючи можливості швидкого старту розробки.

Інтеграція з бібліотекою «shadcn» у контексті стеку Laravel-Inertia-React створює особливо потужну комбінацію. Оскільки «shadcn» надає компоненти у вигляді коду, а не пакету, розробник може легко інтегрувати необхідні елементи інтерфейсу безпосередньо у свої React компоненти, які рендеряться через Inertia. Це дозволяє створювати узгоджений та естетично привабливий інтерфейс з мінімальними зусиллями, при цьому зберігаючи повний контроль над логікою роботи компонентів на стороні сервера. Така архітектура особливо вигідна для проєктів, де потрібна тісна інтеграція між серверною логікою (наприклад, складні правила доступу до проєктів або синхронізація між командними та особистими завданнями) та динамічним користувацьким інтерфейсом.

Аналізуючи обрану технологічну платформу в цілому, можна констатувати, що комбінація Laravel 12, PHP 8.4, React, Inertia.js та MySQL створює оптимальний баланс між продуктивністю, зручністю розробки та можливостями масштабування. Кожен компонент стеку відповідає специфічним вимогам проєкту: Laravel забезпечує надійний, зручний у підтримці бекенд з елегантним кодом; React дозволяє створити сучасний, динамічний інтерфейс; MySQL гарантує високу продуктивність операцій читання; а Inertia.js усуває складність інтеграції між серверною та клієнтською частинами. Це рішення не тільки відповідає поточним потребам розробки, але й забезпечує потенціал для

майбутнього розширення функціоналу та масштабування системи відповідно до зростання користувацької бази та потреб.

2.2 Практична реалізація об'єкта проектування

Практична реалізація вебсистеми проводилася поетапно, і першим етапом було визначення вимог та проектування архітектури системи.

Система керування проектами (СКП) – це один із двох основ, які були взяті у систему. У контексті СКП різні команди або підрозділи організації працюють над незалежними ініціативами, які не повинні перетинатися на рівні доступу до даних. Ізоляція проєктів та завдань забезпечує чітке розмежування відповідальності, унеможливлює випадкові або навмисні доступи до перегляду або модифікації даних сторонніми особами та підтримує організаційну структуру з різними рівнями доступу. Це особливо критично у середовищах, де одна система обслуговує множину незалежних проєктів або клієнтів. Для реалізації такої ізоляції часто використовують архітектуру мультитенантності. Вона передбачає собою існування батьківської сутності – тенанта, яка може бути доступна лише певним користувачам, та ні у якому випадку не перетинатися з іншою сутністю того ж типу. У системі керування проєктами, зазвичай використовується саме такий підхід – наявність сутності, на рівень вищої за проєкти, наприклад – модель організація, у межах якої є свої співробітники, і свої проєкти. Саме таку структуру і було використано як основу частини СКП у системі.

Наступним етапом було створити схему бази даних. Було сформовану наступну структуру даних:

– користувачі, як основа кожної багатокористувацької платформи;

– робочі середовища, «workspaces» – як модель-тенант у мультитенантній архітектурі системи;

– «UserWorkspace» – поєднувальна модель відношення «багато до багатьох» між користувачами та робочими середовищами, яка також визначає роль користувача у моделі-тенанті;

– проєкти – використовуються як для колаборативної частини системи так і як особисті простори для персональних завдань користувачів, батьківська модель визначається морфним зв'язком, який дозволяє використовувати два або більше типи моделей як батьківську, у розроблюваній системі це є робочий простір «workspace» (визначає, що проєкт є командним) або користувач (визначає, що сутність є особистим простором);

– «ProjectUser» – це ще одна поєднувальна модель відношення багато-до-багатьох, яка пов'язує між собою користувачів та проєкти, визначаючи роль учасника проєкту та інші допоміжні дані;

– статуси, які прив'язуються до проєкту (або особистого простору) та можуть бути прив'язаними до завдань у межах того ж проєкту. Кожен статус відповідає одному з типів: «відкритий», «у роботі», «закритий», кожен тип відповідає за візуальне відображення статусу завдання та структуру робочого процесу;

– завдання, над якими працюють користувачі системи;

– логи аудиту, для трекінгу активності у завданнях;

– коментарі, для обговорень завдань (або для додаткового контексту, нотаток, тощо);

– наглядачі – поєднувальна модель, яка дозволяє прив'язувати різних користувачів до завдання, щоб вони отримували сповіщення у разі активності у завданні;

– сповіщення;

– відношення завдань, для можливості клонування та створення інших відношень між завданнями;

– «IssueStatusTransition», для можливості налаштування робочого процесу у проєктах;

– часові логи, для трекінгу витраченого часу над завданнями.

Після створення схеми даних у системі – було розпочато саму роботу над проєктом. Було створено пустий проєкт Laravel, через CLI-інструмент Laravel Installator, вигляд якого у процесі створення нового проєкту Laravel показано на рисунку 2.1.

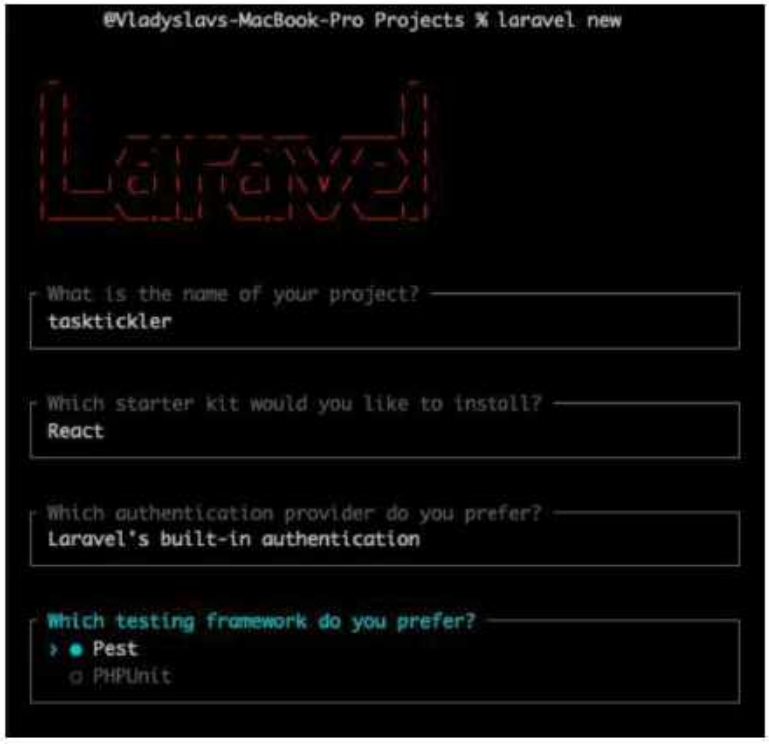

Рисунок 2.1 – зовнішній вигляд CLI-інструменту Laravel Installator

За допомогою даного інструменту, під час створення проєкту можна вибрати спеціальні налаштування, які відразу будуть застосовані та готові до роботи у новоствореному проєкті:

– назва проєкту;

– стартер-кіт, який застосується (надається вибір із підтримуваних бібліотек або фреймворків фронтенду, як: React, Vue.js, Svelte, або не використовувати любий з даного вибору);

– тип та спосіб організації автентифікації користувачів;

– фреймворк для автоматичних тестів (Pest або PHPUnit).

Далі проводилося налаштування різних значень для локальної розгортки проєкту, для можливості роботи над ним. Для цього було використано інструмент Docker. Він дозволяє розгортати проєкт у закритих контейнерах, незалежних один від одного. Зокрема, були створені контейнери:

– «app» («php-fpm-alpine»), для роботи з кодовою базою бекенду;

– «webserver» (на основі «nginx») – для доступу до проєкту у браузері;

– «database» – СКБД MySQL.

Для роботи з контейнерами використовувалися інструменти «docker-cli» та програмний застосунок Docker Desktop, зовнішній вигляд інтерфейсу якого показано на рисунку 2.2.

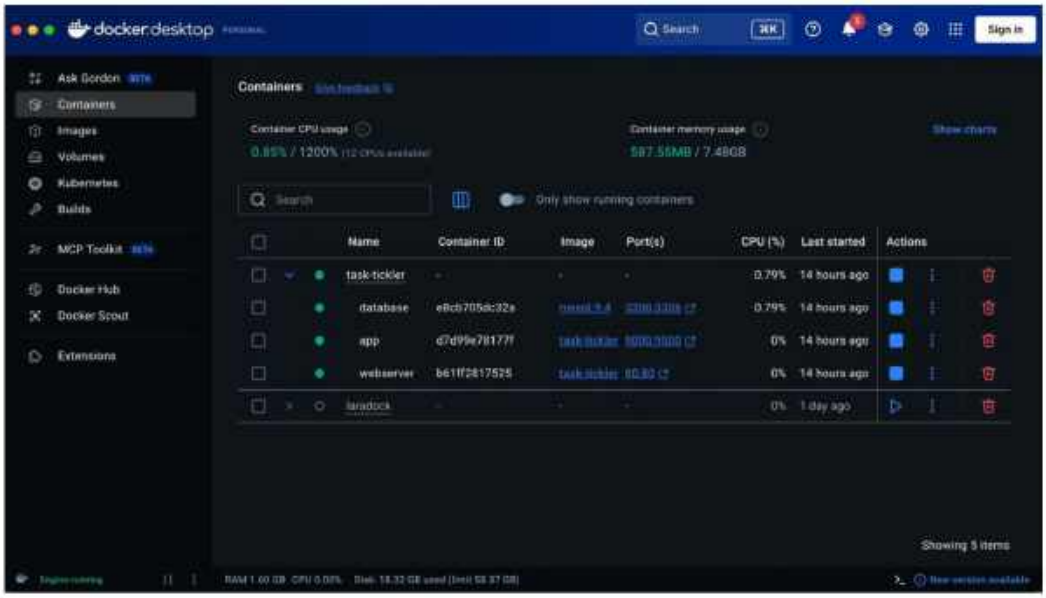

Рисунок 2.2 – Зовнішній вигляд інтерфейсу Docker Desktop

Далі було створено всі потрібні міграції для втілення схеми даних у роботу. Було створено 18 міграцій (разом із стандартними від фреймворку), які створюють 22 таблиці у базі даних.

Для роботи з даними у Laravel використовуються класи, які унаслідують тип Model. Загалом було створено набір моделей для основних сутностей, таких як «User» чи «Issue», окрім того, було створено два модель-класи для поєднувальних сутностей «ProjectUser» та «UserWorkspace».

Класи-моделі використовуються Eloquent ORM – інструментом об'єктно-реляційного відображення, який може використовуватися як посередник між фреймворком та базою даних. Він використовується для полегшення маніпуляції з даними у БД, які використовуються застосунком, що розробляється. За допомогою нього можна створювати як прості так і складні запити до бази даних у зрозумілому вигляді навіть для розробника, який не має високого рівня знань роботи з маніпуляцією даних з БД. У класі-моделі, у свою чергу, можуть бути записаний різний допоміжний функціонал для роботи з відповідною моделлю. Для демонстрації, детальніше розглянемо клас-модель сутності статусу завдання, «IssueStatus».

Перш за все, в тілі класу моделі зустрічається оголошення використання допоміжних класів – Trait, або «трейтів». Трейти використовуються як «вставка» блоку коду у інший клас. Наприклад, у моделі статусу завдань за замовчуванням використовується трейт «HasFactory», він впроваджує у клас моделі статичні методи для роботи з фабриками. Фабрика – це клас, який використовується зазвичай у контексті автоматичного тестування – він дозволяє заповнити БД фейковими даними.

Далі, у моделі «IssueStatus» було перезаписано наступних два конфігураційних масиви:

– «$fillable» – даний масив вказує фреймворку, які поля можуть бути змінені при масованому записі даних – процес привласнення значень багатьох полів до екземпляру класу моделі одночасно;

– «$casts» – асоціативний масив, у якому можна показати фреймворку, яке поле потрібно і за яким правилом перетворити, коли воно витягується з об'єкту запиту (об'єкт «Request» – набір даних, які передаються при запиті від клієнтської частини на серверну з циклу «request-response») або при витягнені з бази даних, та у зворотньому випадку – коли дані передаються у відповідь на клієнтську частину системи, або іде запис у таблицю БД.

У лістингу 2.1 показано присвоєння значень цих масивів у моделі «IssueStatus».

Лістинг 2.1 – Присвоєння значень масивів «$fillable» та «$casts» у класі моделі «IssueStatus»

```
protected $fillable = [
    'name',
    'category',
];

protected $casts = [
    'category' => IssueStatusCategory::class,
];
```

Кінець лістингу 2.1

Правила у асоціативному масиві «$casts» можуть показувати різні процеси перетворення, за різними принципами. Є як передбачені правила самим фреймворком Laravel, так і можна писати власні правила перетворення. Зокрема, як правило перетворення оброблюваних даних можна використати структуру даних enum. Такий спосіб перетворення, зокрема, застосовано у моделі «IssueStatus» – там використовується enum-структура «IssueStatusCategory», код якої показано в лістингу 2.2.

Лістинг 2.2 – Код enum-структури «IssueStatusCategory»

```
<?php

namespace App\Enums;

enum IssueStatusCategory: string
{
    case TODO = 'to_do';
    case IN_PROGRESS = 'in_progress';
    case DONE = 'done';
    public function label(): string
    {
        return ucwords(str_replace('_', ' ', $this->value));
    }
}
```

Кінець лістингу 2.2

Разом з конфігураційними масивами «$fillable» та «$casts» є багато інших, передбачених фреймворком, але у розробці вебсистеми інші не використовувалися.

Окрім використання трейтів та конфігураційних масивів, у моделях також оголошуються відношення до інших моделей, які можна пізніше використати у коді. Зокрема, у моделі «IssueStatus» оголошено відношення «BelongsTo» із моделлю «Project», та відношення «HasMany» з моделлю «Issue» (ліст. 2.3).

Лістинг 2.3 – Оголошення відношень «project» та «issues» у моделі «IssueStatus»

```
public function project(): BelongsTo
{
    return $this->belongsTo(Project::class);
}

public function issues(): HasMany
{
    return $this->hasMany(Issue::class, 'status_id');
}
```

Кінець лістингу 2.3

Відношення «BelongsTo» вказує на те, що у моделі (у даному випадку – «IssueStatus») є поле, яке зберігає значення ідентифікатора якогось із записів моделі, зв'язок з якою оголошується (у поточному прикладі – моделі «Project»).

Відношення «HasMany» – є протилежним до «BelongsTo». Воно передбачає, що у моделі, до якої відношення оголошується (у даному прикладі – у моделі «Issue»), є відповідне поле, яке зберігає значення ідентифікатора поточної моделі.

Фреймворком передбачені конвенції назв відношень, класів, зокрема моделей, та ін. Вони допомагають у багатьох випадках уникати явного вказування на різні назви. Але неможливо уникнути явного вказування різних назв у разі не підтримування конвенцій, і у такому випадку, наприклад, у оголошенні відношення «issues()» у моделі статусу завдань – як другий параметр

функції відношення вручну вказано назву поля, на яке слід дивитися, коли з екземпляру класу «IssueStatus» код пробує витягнути зв'язані завдання, через функцію «issues()».

За таким прикладом коду моделі «IssueStatus», були заповнені всі інші моделі, які використовуються у системі.

Далі було записано оголошення всіх передбачених маршрутів у межах вебсистеми. Для цього у Laravel, у папці «routes» є передбачений файл «web.php». У даному файлі було оголошено маршрути для обох частин системи: персональних та робочих просторів. У лістингу 2.4 показано оголошені маршрути, які відповідають за інформаційні сторінки робочих просторів, а саме:

– про вибраний робочий простір;
– проєкти вибраного робочого простору;
– користувачі, які мають доступ до вибраного робочого простору;
– налаштування вибраного робочого простору.

Лістинг 2.4 – Оголошення маршрутів робочих просторів у вебсистемі

```
    Route::group(['prefix' => 'ws/{workspace:slug}', 'as' => 'ws.',
'middleware' => 'auth'], function () {
        Route::get('/about', [WorkspaceController::class,
    'about'])->name('about');
        Route::get('/members', [WorkspaceController::class,
    'members'])->name('members');
        Route::get('/issues', [WorkspaceController::class,
    'issues'])->name('issues.index');
        Route::get('/settings', [WorkspaceController::class,
    'settings'])->name('settings');
    }
```

Кінець лістингу 2.4

У даному лістингу можна побачити багаторазове звернення до класу «WorkspaceController». Заповнення класів цього типу, контролерів, було наступним кроком. Контролер – це клас, який відповідає за обробку запитів користувачів системи та видачу оброблених відповідей. У процесі роботи над вебсистемою, класи цього типу були розділені на дві групи: персональні

контролери та контролери робочих середовищ. Зазвичай, кожен контролер відповідає за певну сутність, зокрема, у системі існують такі контролери, як «WorkspaceController», «ProjectController», «IssueController» та ін., з назв яких можна зрозуміти, за який тип сутності (або модель) має відповідати кожен контролер.

У контролері методи можуть відповідати за любу дію, зокрема з набору «CRUD» – абревіатура назв операцій створення, оновлення, читання та видалення. Також, існує конвенція назв методів у контролері (яку зазвичай дотримуються, окрім деяких випадків). Зокрема, використовуються назви «index» – зазвичай цей метод означає, що він має повернути колекцію моделей сутності, за яку відповідає цей контролер. Наприклад, у лістингу 2.5 показано код методу «index», який був описаний в контролері «WorkspaceController».

Лістинг 2.5 – Оголошення маршрутів робочих просторів у вебсистемі

```
    public function index(Request $request)
    {
        $workspaces = $request->user()
            ->workspaces()
            ->withPivot('role')
            ->withCount('projects')
            ->withSum('projects', 'issue_counter')
            ->get();

        return Inertia::render('workspaces/index', ['workspaces' =>
WorkspaceCollection::make($workspaces)]);
    }
```

Кінець лістингу 2.5

Коли даний метод викликається, він автоматично отримує, через внутрішній функціонал фреймворку, за допомогою принципу «Dependency Injection», об’єкт типу Request. Він містить всі дані, які описують запит від користувача, це можуть бути значення Header, любі значення текстових полів з сайту, та багато іншої інформації, зокрема, Laravel автоматично, через метод «user()» з цього об’єкту витягує об’єкт моделі користувача, який зробив запит, якщо цей користувач був авторизований. Даний метод, в оголошенні маршрутів

захищений посередником («Middleware») «Authenticate», який і дозволяє витягувати з об'єкту класу «Request» в методі контролера, об'єкт авторизованого користувача. Якщо запит був зроблений не авторизованим користувачем, або якщо термін авторизації вийшов – зазвичай користувача переносить (робить «редірект») на сторінку входу в систему, але можна налаштувати, щоб приходила відповідь з HTTP кодом «401».

Після того, через оголошене в моделі «User» відношення із назвою «workspace», створюється об'єкт «QueryBuilder», що сворює шаблон для запиту SQL, до якого застосовуються три умови:

– «withPivot('role')», яка каже інструменту Eloquent ORM, який обробляє наш об'єкт, що потрібно також витягнути значення «role» із зв'язної таблиці (Pivot table);

– «withCount('projects')», яка рахує кількість проєктів, які зв'язані з кожним робочим простором, присвоює відповідне значення до кожного «workspace»;

– «withSum('projects', 'issue_counter')», яка обраховує суму кількості проєктів у кожному робочому просторі та кількість завдань на всі проєкти у кожному екземплярі «workspace», і зберігає відповідні значення у відповідні об'єкти.

Після цього викликається метод «get()», який запускає виконання запиту в базі даних, і за мить отримує всі потрібні значення, який вставляє в компонент «workspaces/index» клієнтської частини вебсистеми, через інструмент Inertia, робить рендер цієї сторінки, та відправляє її у відповідь, кінцевому користувачеві.

У деяких методах контролерів були використані допоміжні сервіс-класи. Класи цього типу використовуються для того, щоб зняти «текстове навантаження» з контролера, виконуючи всі тяжкі завдання по обробці даних, і повертаючи якесь готове значення. Сервіси можуть бути вставлені в контролер як через принцип «Dependency Injection», так і напряму через виклик конструктора відповідного сервісу. У кодовій базі вебсистеми використовувалися

обидва підходи, оскільки немає строгих конвенцій по використанню сервісів, створених вручну розробниками. Приклад використання сервісу через виклик його конструктора показано у лістингу 2.6.

Лістинг 2.6 – Приклад використання сервісу через виклик його конструктора

```
    (new ProjectService())->createWorkspaceProject($workspace,
$baseUser, [
        'name' => $ucUsername . ' as PM Project',
        'slug' => $username . '-pm',
        'issue_key' => 'PM',
    ]);
```

Кінець лістингу 2.6

У даному випадку використовується метод «createWorkspaceProject» екземпляра сервісу «ProjectService». Внутрішній код даного методу показано в лістингу 2.7.

Лістинг 2.7 – Тіло методу «createWorkspaceProject» екземпляра сервісу «ProjectService»

```
    public function createWorkspaceProject(Workspace $workspace, User
$user, array $fields): Project
    {
        return DB::transaction(function () use ($workspace, $user,
$fields) {
            $project = $workspace->projects()->create($fields);
            $this->createBaseStatuses($project);

            $workspaceOwner = $workspace->owner();
            if ($user->isNot($workspaceOwner)) {
                $this->attachUserToProject($project, $user,
ProjectRole::ProjectManager);
            }
            $this->attachUserToProject($project, $workspaceOwner,
ProjectRole::Manager);

            return $project;
        });
    }
```

Кінець лістингу 2.7

Даний метод передбачає створення нового проєкту, у якому відразу додаються три стандартних статуси «До Виконання», «В Роботі» і «Виконано», і присвоюється значення стандартного статусу для новостворених завдань у проєкті – статус «До Виконання». Останній – використовується у іншому сервісі, який відповідає за процес створення нового завдання у відповідному проєкті, там і присвоюється цей стандартний статус. Також, даний метод відразу перевіряє, чи є користувач, який створює проєкт, власником робочого простору, у якому проходить процес створення, якщо ні – то він прив'язує активного користувача до щойно доданого проєкту, і присвоює йому роль «Проєктний Менеджер», і незалежно від цієї умови, завжди додає власника робочого простору із присвоєнням ролі «Менеджер».

Загалом, цей метод приймає три параметри: екземпляр моделі робочого простору, у якому потрібно створити новий проєкт, користувача, який робить цю дію та параметри полів для екземпляру моделі проєкту, що створюється.

Останнім кроком підготовки бекенду було створення «сідерів» – класів-скриптів, які використовуються для заповнення БД різними даними. Вони можуть бути використаними для різних потреб, зокрема для тестування, демонстрації а також для заповнення потрібними сирими даними БД як на локальному, так і на «продакшн» середовищі. Було створено два сідери:

– «BaseSeeder», для зручної роботи над проєктом;

– «DemoDataSeeder», для демонстрації функціоналу проєкту можливим майбутнім користувачам.

Після реалізації бекенду, почався крок реалізації клієнтської частини, фронтенду. В основі коду фронтенду використовувався запропонований використаним стартер-кітом підхід з використанням «TypeScript», який передбачає замість стандартних файлів-шаблонів типу «.jsx» використання файлів типу «.tsx», а також використання проголошеної строгої типізації.

Спочатку були внесені мінімальні правки в запропонований навігаційний «Layout» встановленим стартер-кітом. Зокрема, відразу було змінено тип з навігації в «хедері» на навігацію через бокову панель та змінено назву на назву

вебсистеми. Також, в бокову панель були добавлені тимчасові посилання, які несли допоміжний функціонал у навігації клієнтської частини під час розробки. Отриманий вигляд бічної панелі можна побачити на рисунку 2.3.

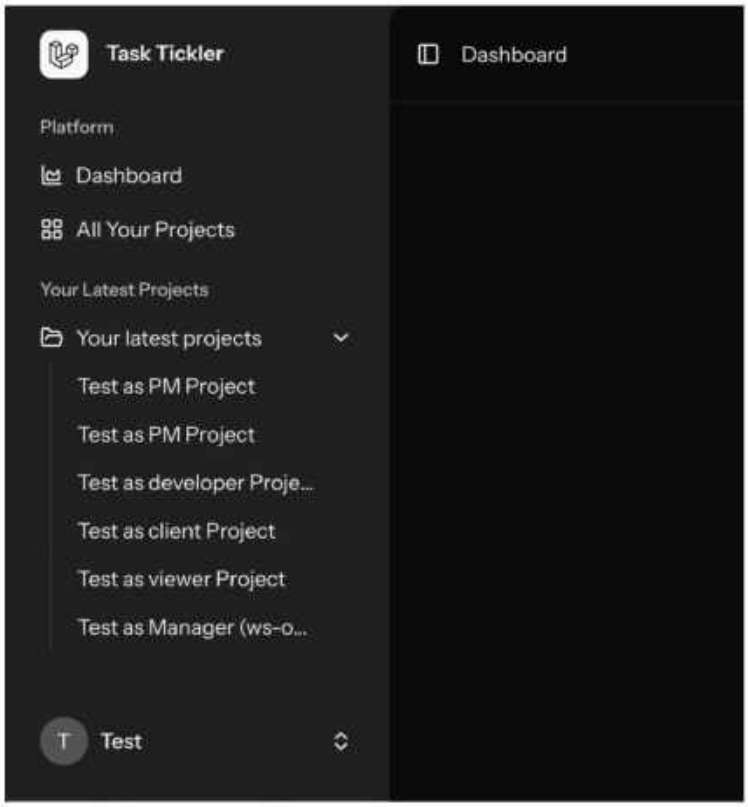

Рисунок 2.3 – Зовнішній вигляд першої версії навігації по вебсистемі

Далі в процесі роботи була головна сторінка (панель) для користувача, з набором базової статистики користувача по системі, зокрема:

– кількість особистих просторів користувача, та у скількох командних проєктах він є учасником;

– до скількох робочих просторів користувач був приєднаний;

– загальна статистика по завданнях: всього завдань, в яких виконавцем був даний користувач, загальна кількість закритих завдань, кількість завдань, у яких користувач робив любу активність нещодавно (останніх 30 днів), та кількість завдань, у яких стік термін виконання, і вони все ще не закриті;

– список завдань, у яких нещодавно було щось зроблено даним користувачем;

– список останніх змін у завданнях, які призначені на нього, або в яких він є відмічений як наглядач. Більшість елементів є клікабельними, які ведуть на відповідні списки, або ж відповідні сторінки, наприклад, можна перейти на сторінку.

Зовнішній вигляд отриманої сторінки показано на рисунку 2.4.

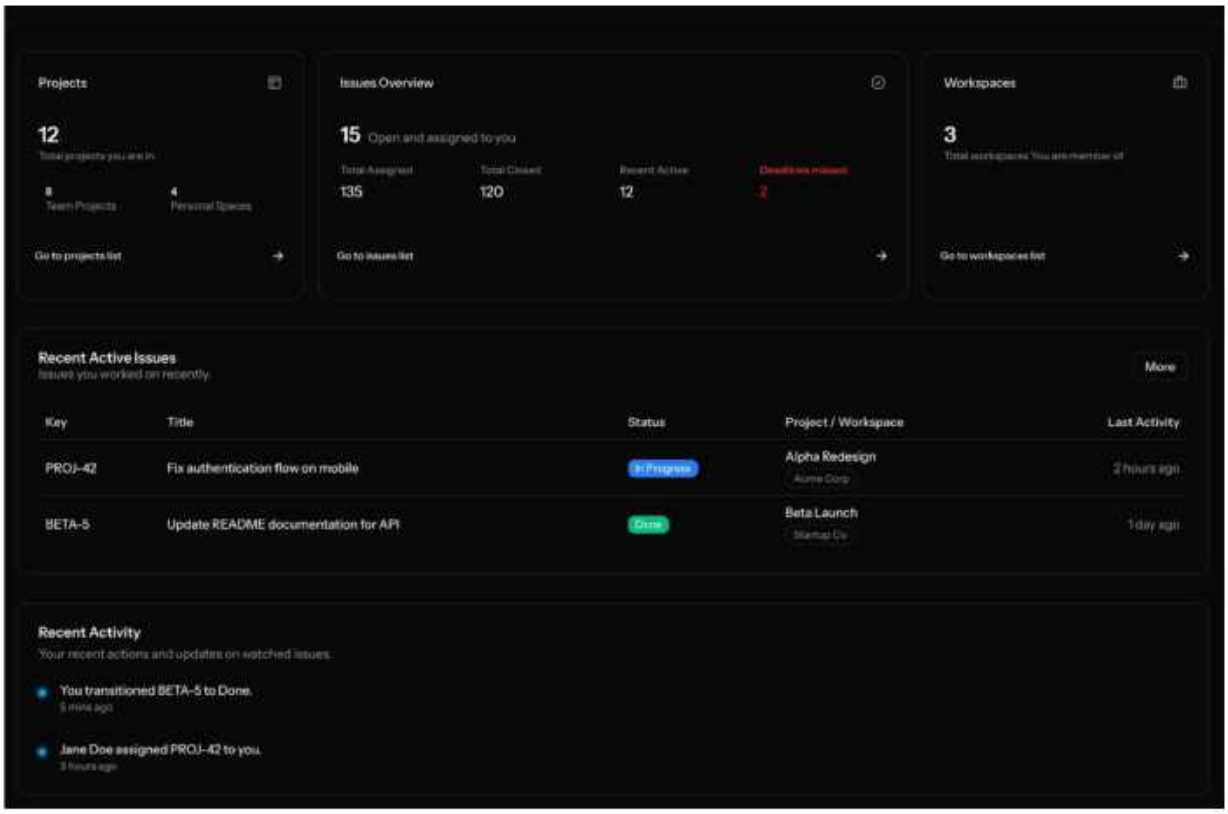

Рисунок 2.4 – Зовнішній вигляд отриманої панелі користувача

Після готової панелі користувача, розроблялися сторінки для робочих середовищ. Зокрема, сторінка з набором середовищ, у яких користувач є добавленим. Було вирішено зробити два вигляди сторінки: списком та картками. Щоб зробити сторінку універсальною і вона мала вигляд і коли користувач є у багатьох робочих просторах, і коли він є учасником, наприклад, трьох просторів.

Вигляд отриманих представлень показано на рисунках 2.5 та 2.6.

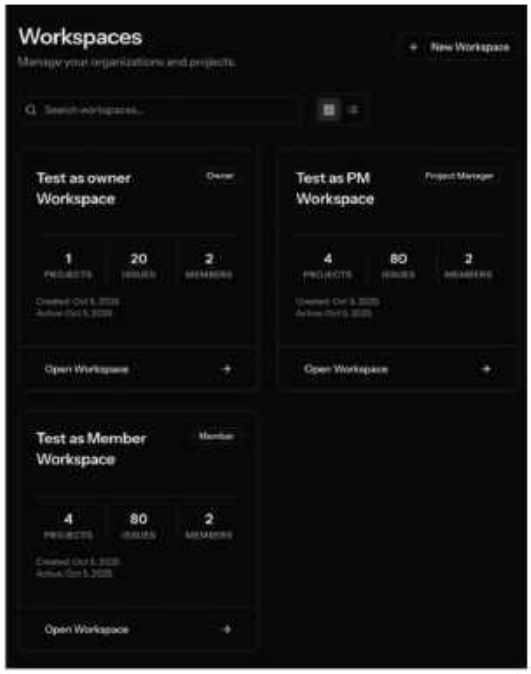

Рисунок 2.5 – Зовнішній вигляд сторінки списку робочих середовищ користувача, у режимі карток

Також, фактором добавлення режиму карток були те, що у режимі карток можна вивести трохи більше інформації, іноді критичної, за рахунок більшого форм-фактору елементів, що показуються.

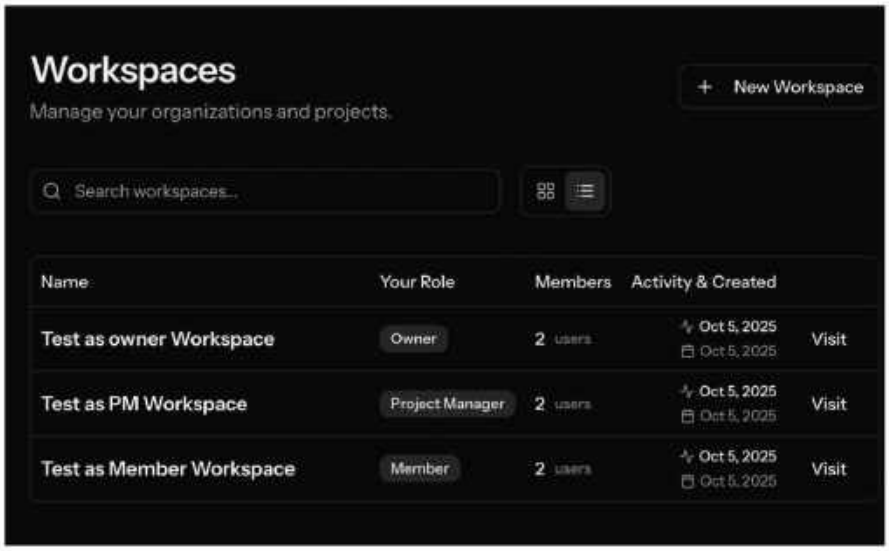

Рисунок 2.6 – Зовнішній вигляд сторінки списку робочих середовищ користувача, у режимі карток

Щодо сторінок окремого робочого простору – було прийняте рішення, що в них буде вигляд вкладок – воно не вирізнятиметься з іншими сторінками, а нестиме вагомий функціонал, оскільки на робочий простір передбачено сторінки:

– «Про Робочий простір» – з основною інформацією про робочий простір;

– «Проєкти» – список проєктів робочого простору;

– «Учасники» – список добавлених користувачів у робочий простір;

– «Завдання» – список завдань із проєктів робочого простору;

– «Налаштування» – сторінка з полями для налаштування назви і опису робочого простору.

Сторінки «Про Робочий простір» та «Налаштування» – це звичайні, примітивні сторінки для відображення або редагування інформації. На інших, згаданих сторінках, для подачі інформації використовувалися таблиці з подачею лише основних даних, і можливістю переходу на відповідні сторінки. На рисунку 2.7 показано зовнішній вигляд основної частини отриманої сторінки проєктів робочого простору. Як було згадано раніше – використано для основної інформації лише деякі дані проєкту, зокрема: назви, ролі поточного користувача у відповідних проєктах, загальна кількість користувачів та дата крайньої активності і створення проєктів. Також, можна побачити вкладки, про які згадувалося раніше.

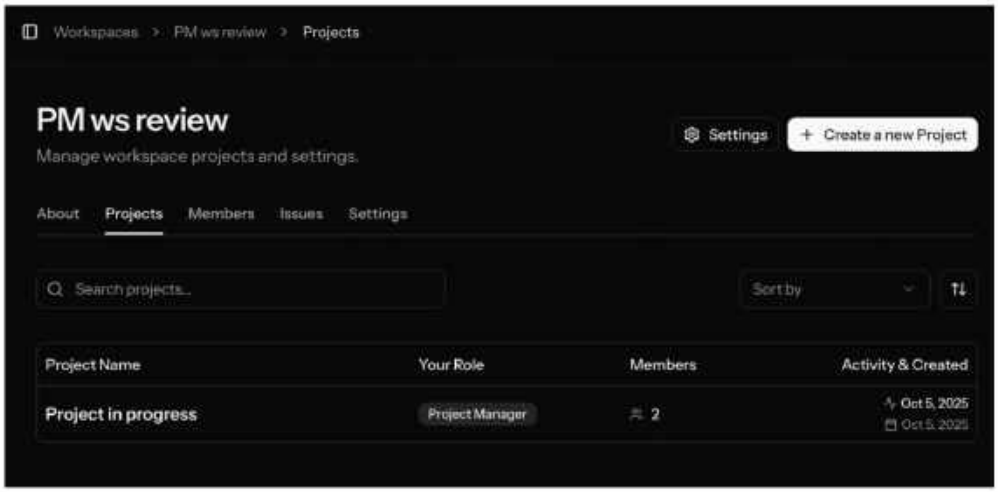

Рисунок 2.7 – Основна частина сторінки проєктів робочого простору

Для реалізації сторінок проєкту було використано таку саму структуру, як у сторінках робочого середовища. Було застосовано підхід з використанням вкладок основного контенту, а саме наступний набір:

– «Завдання» – для списку завдань у проєкті;

– «Команда» – список користувачів, у котрих є доступ до проєкту (менеджери можуть керувати учасниками та змінювати їх ролі);

– «Витрачений час» – сторінка з списком записаного витраченого часу користувачем;

– «Налаштування» – для управління інформацією про проєкт та статусами завдань і робочим процесом у межах проєкту.

На рисунку 2.8 показано вигляд отриманої сторінки завдань проєкту.

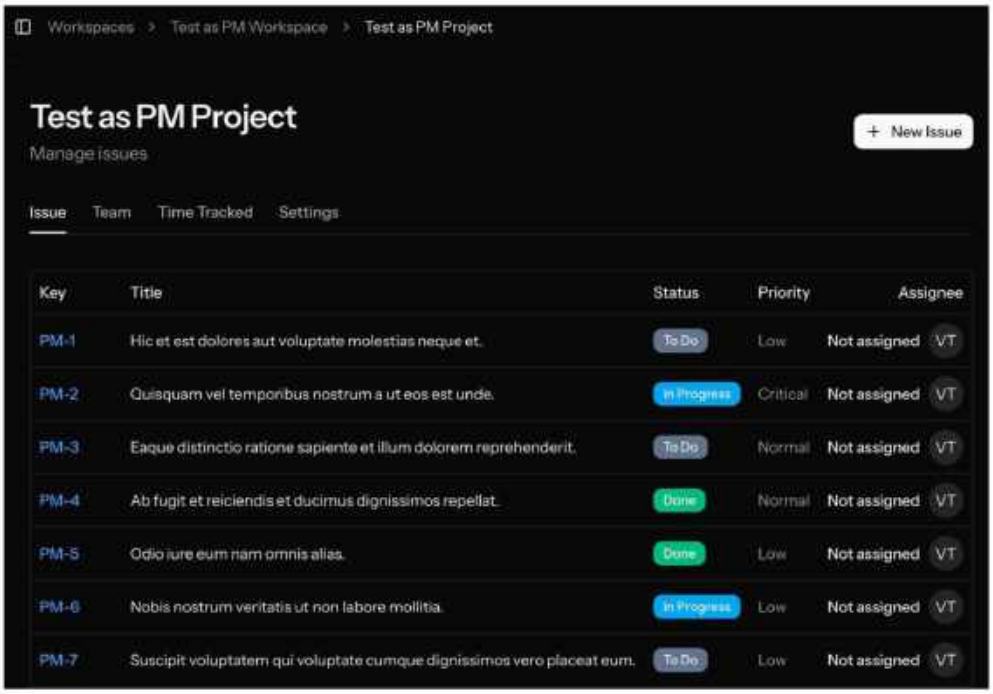

Рисунок 2.8 – Зовнішній вигляд сторінки списку створених завдань у проєкті

На усіх сторінках у межах вебсистеми, де виводиться інформація про завдання, застосовано особливу стилізацію для статусів завдань, яка залежить від типу статусу. Загалом було передбачено три типи:

– «завдання відкрите» – сірий колір, приклад статусу: «До виконання»;

– «завдання у процесі» – синій колір, приклад статусу: «В роботі»;

– «завдання закрите» – зелений колір, приклад завдання: «Виконано».

Для налаштування відношення статусів до відповідних типів, а також налаштування можливих переведень завдань між статусами було добавлено відповідний блок «Налаштування робочого процесу» на сторінці налаштування проєкту. Зовнішній вигляд даного блоку показано на рисунку 2.9.

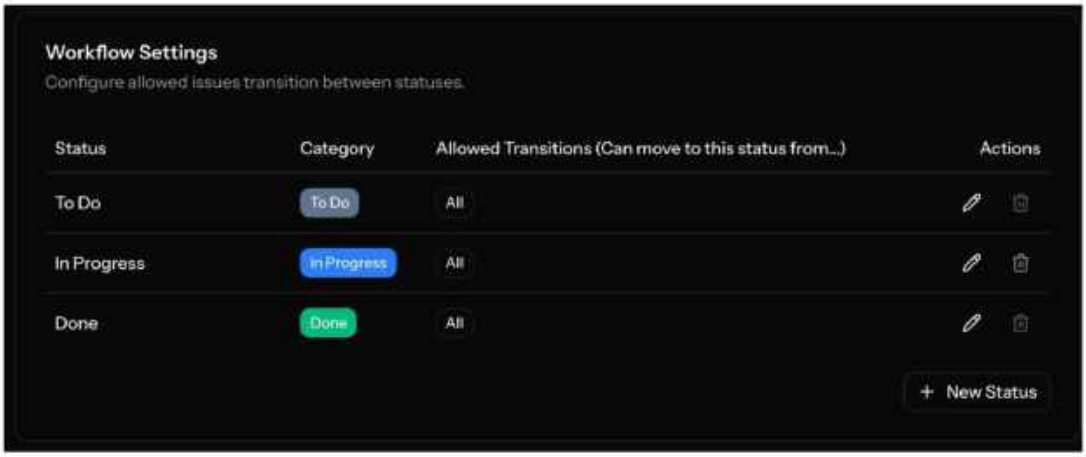

Рисунок 2.9 – Зовнішній вигляд блоку для налаштування робочого процесу у проєкті

Одне із полів таблиці являє собою перелік, з яких статусів можна перевести завдання у кожен відповідний статус. Зокрема, коли можна перейти з усіх статусів, замість переліку кожного статусу, доступного у проєкті, виводиться значення «Зі всіх». Це рішення є зручним з боку UX (User Experience), оскільки воно знижає навантаження інтерфейсу. Також, передбачені кнопки редагування та видалення відповідних статусів. Кнопка видалення стає неактивною, у випадку, якщо статус є єдиним доступним у проєкті із типом «Відкритий» або «Закритий». При видаленні статусів – система перевіряє, чи немає завдань, у яких виставлений відповідний статус, і запитує у який інший перевести усі завдання з статусом, що видаляється з проєкту. При натисканні на кнопку «Редагувати» – на екран користувача виводиться модальне вікно, у якому можна змінити назву, тип статусу, а також вибрати, чи дозволити у проєкті перехід з любого статусу у той,

що редагується, і якщо забрати даний прапорець – стають доступні для вибору прапорці усіх інших статусів, які визначають можливість переходу із відповідних статусів. Зовнішній вигляд модального вікна показано на рисунку 2.10.

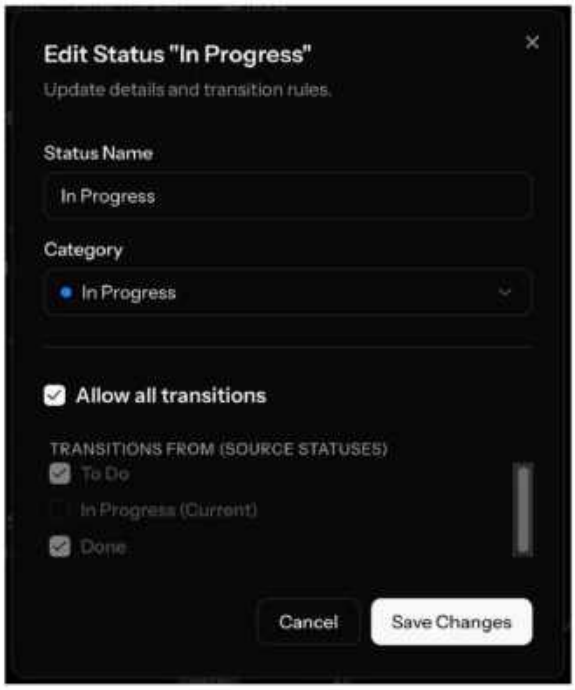

Рисунок 2.10 – Зовнішній вигляд модального вікна редагування статусу у проєкті

Далі було реалізовано сторінку завдання. Дана сторінка вивести користувачеві якнайбільше інформації про завдання, і тому у ній присутня найбільша кількість різних блоків, зокрема:

– блок з назвою та ключем завдання;
– блок з описом (тілом) завдання;
– блок з деталями (для таких даних, як: статус, до кого призначена, тощо);
– блок з списком наглядачів;
– блок з батьківським завданням (якщо є таке прив'язане завдання);
– блок з дочірніми завданнями (якщо такі завдання є);
– блок з прив'язаними завданнями (якщо такі зв'язки є);

– блок з можливістю переключення між списком коментарів, логом проведених змін у завданні та списком витраченого користувачем часу на завдання.

На блоках опису та деталей про завдання передбачено кнопки для редагування, які з'являються при наведенні на відповідний блок, так само як на блоках з прив'язаними завданнями (батьківські, дочірні та прив'язані з іншим типом відношення).

Зовнішній вигляд отриманої сторінки зображено на рисунку 2.11.

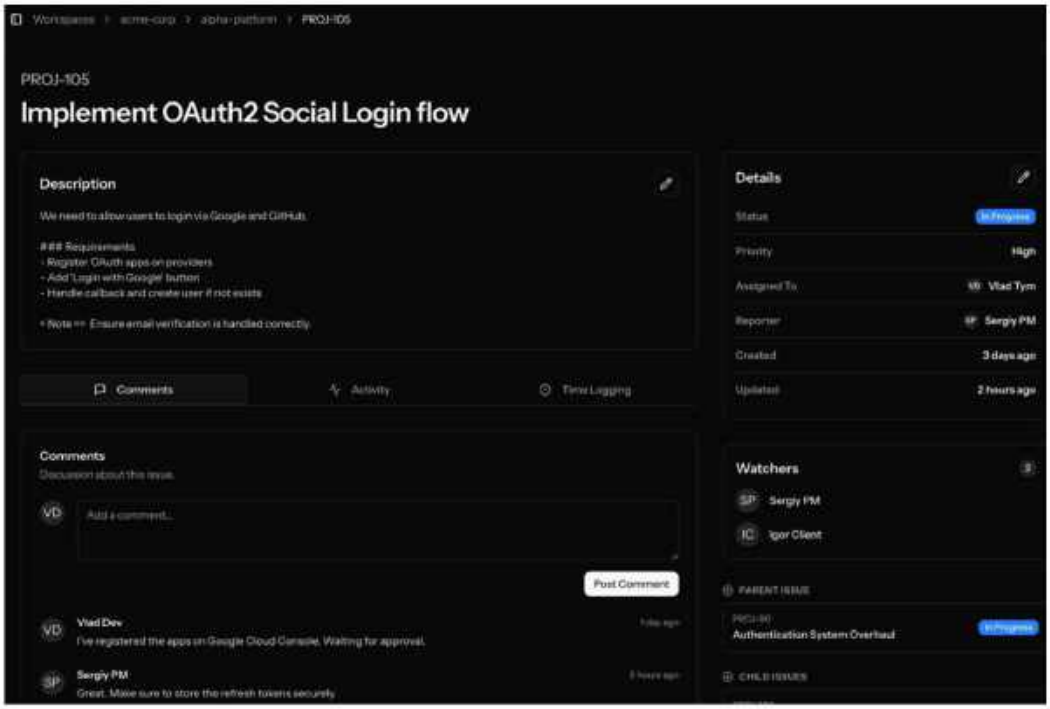

Рисунок 2.11 – Зовнішній вигляд сторінки з інформацією про завдання

Незважаючи на велику кількість блоків з інформацією, кінцевий вигляд сторінки не виглядає перевантаженим через лаконічне розташування елементів на ній, одночасно дозволяючи помістити більшість інформації про завдання без потреби гортання сторінки користувачем.

Із завершенням роботи над сторінками «командної» частини вебсистеми, для реалізації другого етапу, модуля «особистих просторів», було використано ті ж ресурси для особистих проєктів (просторів), завдань, та усіх допоміжних

сторінок, окрім таких як, наприклад, списку команди проєкту – сторінки, яка відповідає саме за командну роботу, а не менеджмент особистого часу і завдань.

На інших компонентах, таких як сторінці зі списком завдань, було теж видалені компоненти, зв'язані з командною частиною програми.

В результаті, в рамках практичної розробки спроєктовано та реалізовано сучасну вебсистему, що поєднує інструментарій для командної роботи над проєктами з гнучкістю персональних списків завдань. Головним результатом етапу стало створення серверної та клієнтської частин, які забезпечують швидку та зручну взаємодію з даними системи.

Зокрема, було реалізовано наступні ключові модулі системи:

– робочі простори (workspaces) – механізм організації проєктів, що дозволяє користувачам розділяти командну діяльність та особисті завдання, з можливістю швидкого перемикання між ними;

– панель основної інформації (dashboard) – сторінка із статистичними даними, які надають миттєвий огляд активності користувача, стану поточних завдань та прогресу виконання проєктів;

– Управління життєвим циклом завдань (issue tracking) – розроблено детальну картку завдання, що підтримує ієрархічні зв'язки (батьківські/дочірні задачі), історію змін, коментарі та облік затраченого часу на виконання;

– гнучкі налаштування – впроваджено систему кастомізації статусів та робочих процесів (workflows), що дозволяє адаптувати систему під специфічні потреби конкретної команди.

Створений програмний продукт відповідає сучасним вимогам до систем управління проєктами, забезпечуючи інтуїтивно зрозумілий інтерфейс завдяки використанню сучасних стандартів.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ РОБОТИ ВЕБСИСТЕМИ УПРАВЛІННЯ ПРОЄКТАМИ

3.1 Методика проведення дослідження

Метою експериментального дослідження була перевірка стійкості архітектури розробленої вебсистеми, оцінка часових характеристик обробки запитів та визначення меж масштабованості при роботі з великими масивами даних.

Враховуючи, що система поєднує функціонал командного менеджменту та персонального планувальника, критично важливим показником є швидкість відгуку інтерфейсу (Response Time) при формуванні аналітичних зведень на сторінці «Dashboard» та при фільтрації списків завдань («Issues List»).

Дослідження проводилося у ізольованому середовищі для мінімізації впливу зовнішніх факторів. Використовувалася наступна конфігурація:

– серверна частина: PHP 8.4, Laravel Framework 12 (запущено через Docker-контейнер «PHP-FPM», та забезпечено зв'язок із серверною частиною через додатково розгорнутий контейнер з вебсервером «nginx»);

– СКБД: MySQL 9.0 (InnoDB engine);

– клієнтська частина: браузер Google Chrome, інструменти розробника для заміру TTFB (Time To First Byte);

– апаратна платформа: процесор Intel Core i7-8800H (6 ядер), 16 ГБ ОЗУ, SSD NVMe накопичувач.

Для забезпечення достовірності результатів було використано вбудований механізм «DatabaseSeeder» та бібліотеку «Faker» фреймворку Laravel. Було згенеровано чотири тестових набори даних, що імітують різні етапи життєвого циклу вебсистеми:

– набір «А» (щойно розгорнутий, демонстраційний проєкт): 100 завдань, 2 проєкти, 5 користувачів;

– набір «B» (рівень малого бізнесу): 1000 завдань, 10 проєктів, 20 користувачів;

– набір «C» (корпоративний рівень): 100 тис. завдань, 50 проєктів, 100 користувачів;

– набір «D» (система з високим навантаженням): 10 млн. завдань, 500 проєктів, 1000 користувачів.

Для кожного набору даних проводився замір продуктивності за трьома ключовими сценаріями, які є найбільш ресурсоємними через складні SQL-запити (JOIN, GROUP BY, COUNT).

Перший сценарій – завантаження сторінки інформаційної панелі користувача. Система виконує агрегацію даних: підрахунок активних проєктів, завдань зі статусом «Відкрите», завдань з пропущеним дедлайном та статистики за останні 30 днів.

Другий сценарій передбачав тестування завантаження списку завдань у проєкті. Вибірка даних з пагінацією (по 20 записів), сортуванням за датою створення та фільтрацією за статусом.

Останній, третій сценарій – це виконання повнотекстового пошуку по базі даних, зокрема, пошук завдання по назві у межах проєкту.

Усі три сценарії повторюють реальні випадки, з якими стикаються користувачі під час взаємодії з вебсистемою.

3.2 Обробка та аналіз отриманих результатів

У ході експерименту було отримано часові показники генерації сторінок сервером. Для отримання статистично значущих результатів кожен тест виконувався 10 разів, після чого розраховувалося середнє арифметичне значення часу виконання.

Результати вимірювань для всіх сценаріїв зведено у таблицю 3.1.

Таблиця 3.1 – Результати дослідження швидкодії системи

Кількість записів (завдань)	Сценарій 1	Сценарій 2	Сценарій 3	Використання пам'яті (RAM)
100	45 мс	38 мс	25 мс	12 МБ
1000	62 мс	42 мс	30 мс	14 МБ
100 тис.	315 мс	110 мс	180 мс	28 МБ
10 млн.	3850 мс	450 мс	5200 мс	102 МБ

При малих обсягах даних (до 1000 записів) час завантаження інформаційної панелі є миттєвим (до 62 мс). Проте при досягненні 10 мільйонів записів спостерігається різке зростання часу відповіді до 3,85 с.

Це пояснюється специфікою роботи рушія InnoDB у MySQL: запити типу «SELECT COUNT» для великих таблиць вимагають сканування значної частини індексу або таблиці, що є ресурсоємною операцією. Зазвичай, коли системи сягають такого рівня популярності, робиться розподіл бази даних, іншими словами – розподіл БД між серверами, або сутностями СКБД.

Сторінка списку завдань показала найкращу стійкість до масштабування. Навіть при 10 млн. записів час завантаження склав 450 мс, що є прийнятним показником для веб-додатків.

Ефективність досягнута завдяки використанню механізму пагінації на рівні SQL-запиту. Система не завантажує всі дані в оперативну пам'ять, а вибирає лише необхідний зріз. Незначне зростання часу (з 38 мс до 450 мс). зумовлене збільшенням глибини індексів типу бінарного дерева у базі даних.

Пошук показав лінійну залежність часу виконання від обсягу даних. При 10 мільйонах записів, використання стандартного оператора «LIKE» призвело до затримок понад 5 секунд, що є критичним для UX. Це вказує на те, що для великих баз даних необхідно використовувати спеціалізовані індекси для

повнотекстового пошуку «Full-Text Search Index», або зовнішні пошукові рушії, такі як Elasticsearch.

На основі проведеного аналізу було виявлено вразливі місця у продуктивності при наявних великих обсягах даних. Для забезпечення стабільної роботи системи у промисловому середовищі було розроблено та впроваджено ряд оптимізацій, які склали практичну цінність роботи. Наприклад, було додано складені індекси для таблиці «issues». Зокрема, індекс з полів: «project_id», «status_id», та «created_at». Він дозволив прискорити фільтрацію списків у другому сценарії на 40 % для великих наборів даних.

ВИСНОВКИ

Розробка вебсистеми для організації командних проєктів та особистих завдань є складним завданням, яке вимагає ретельного аналізу та обґрунтованого вибору технологічного стеку. На основі проведеного огляду методів і засобів розробки було визначено, що оптимальним рішенням для реалізації проєкту є використання комбінації Laravel, PHP 8.4, React, MySQL та Inertia.js.

Глибокий аналіз серверних фреймворків, таких як Django, Spring Boot, Express.js та Ruby on Rails, продемонстрував ключові переваги Laravel. Його синтаксис, орієнтований на зручність розробників, активна підтримка спільноти й регулярні оновлення роблять його ідеальним інструментом для створення масштабованих та зручних у підтримці вебдодатків. Крім того, наявність «стартер-кітів» значно скорочує час розробки, дозволяючи зосередитися на реалізації унікальної бізнес-логіки.

У виборі технологій для реалізації клієнтської частини, було оцінено переваги й недоліки таких інструментів, як Angular, Vue.js, Svelte та React. React, завдяки своїй компонентно-орієнтованій архітектурі, віртуальному DOM і надзвичайно розвиненій екосистемі, виявився найкращим вибором для реалізації інтерактивного інтерфейсу користувача. Його популярність серед розробників і доступність додаткових бібліотек, таких як «shadcn», забезпечують високу швидкість розробки й адаптацію інтерфейсу під потреби проєкту.

Щодо систем керування базами даних, порівняння MySQL, PostgreSQL та MongoDB показало, що MySQL є оптимальним вибором для систем з переважанням операцій читання, як у випадку з системами управління проєктами. Його висока продуктивність запитів SELECT і стабільність у роботі з середніми обсягами даних гарантують швидкий доступ до інформації, що є критично важливим для забезпечення позитивного користувацького досвіду.

Інтеграція між серверною та клієнтською частинами проєкту була значно спрощена завдяки використанню Inertia.js. Цей інструмент дозволяє уникнути складної конфігурації API та забезпечує плавність роботи додатку, поєднуючи

переваги багатосторінкових та односторінкових застосунків. Крім того, його інтеграція з Laravel і React через стартер-кіти зменшує час на налаштування базової інфраструктури.

Таким чином, проведений аналіз і вибір технологій дозволив створити надійну основу для розробки системи, яка інтегрує командну й особисту продуктивність у межах одного програмного продукту. Використання обраного стеку забезпечує баланс між продуктивністю, зручністю розробки й можливістю майбутнього масштабування, що відповідає сучасним вимогам до створення вебдодатків.

Сформульовано вимоги до вебсистеми та спроєктовано реляційну схему бази даних, що складається з 22 таблиць, із них більшість – це частина схеми предметної області, та декілька системних таблиць, необхідних для роботи фреймворку. Це дозволило чітко визначити функціонал для поєднання особистого та командного планування, а також забезпечити надійне зберігання даних і стабільну роботу системних компонентів.

Розроблено архітектуру програмного забезпечення, що поєднує серверну логіку та клієнтський інтерфейс. Схему системи розділено на дві частини, що спростило структуру коду та полегшило його підтримку. Для реалізації командного функціонала використано підхід мультитенантності, де ізольованим середовищем виступає модель робочого простору «Workspace».

Реалізовано підсистему розмежування прав доступу у межах робочих просторів. Впровадження рольової моделі забезпечує сувору конфіденційність даних та дозволяє гнучко налаштовувати дозволи користувачів, унеможливлюючи доступ сторонніх осіб до інформації окремих проєктів.

Було проведено дослідження системи щодо відповідності до вимог, які були поставленні під час планування архітектури та очікуваного результату. Зокрема, проведено тестування продуктивності вебсистеми при різних сценаріях навантаження бази даних.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Кошелюк В. А., Тимчук В. В. Методи організації порядку сортування у впорядкованих списках у програмному забезпеченні. Науковий журнал «Комп'ютерно-інтегровані технології: освіта, наука, виробництво». Луцьк: Луцький НТУ, 2025. Вип. №59. С. 221-227.

2. Кошелюк В. А., Тимчук В. В., Корень В. В. Архітектурні моделі мультитенантності в сучасних saas-системах: порівняльний аналіз та методологія вибору. Науковий журнал «Комп'ютерно-інтегровані технології: освіта, наука, виробництво». Луцьк: Луцький НТУ, 2025. Вип. № 61. С. 98-103.

3. Agile Project Management: A Game-Changer for Modern Projects Elmhurst University Elmhurst University Blog. URL: https://surl.li/ouojmi (дата звернення: 09.09.2025).

4. Інформаційне перевантаження Adaptic World. URL: https://surl.li/xdkuln (дата звернення: 09.09.2025).

5. Getting Things Done: A Simple Step-By-Step Guide todoist. URL: https://surl.li/mjkuvs (дата звернення: 11.09.2025).

6. Метод GTD (Getting Things Done). URL: https://surl.lu/veotlm (дата звернення: 14.09.2025).

7. Матриця Ейзенхауера: Пріоритети в управлінні часом і завданнями. URL: https://surl.lu/ncggfz (дата звернення: 14.09.2025).

8. Що таке тайм-менеджмент? – A4 Company. URL: https://surl.li/mvasfc (дата звернення: 15.09.2025).

9. Розкладаємо пріоритети згідно з матрицею Ейзенхауера My Atomy. URL: https://surl.lu/kcdiob (дата звернення: 15.09.2025).

10. Техніка Помодоро: простий секрет продуктивної роботи Ліцей №3 міста Житомира. URL: https://surl.li/jsowbc (дата звернення: 18.09.2025).

11. The Parking Lot Method: A Productivity Technique for Enhanced Workflow. URL: https://surl.li/zuymuf (дата звернення: 18.09.2025).

12. Companies that use Confluence TheirStack.com. URL: https://surl.li/zvhazn (дата звернення: 19.09.2025).

13. Maximizing Agile Efficiency with Jira: Features for Enhanced Project Management. URL: https://surl.li/prierc (дата звернення: 19.09.2025).

14. Products marketplace Atlassian. URL: https://surl.li/vqxphf (дата звернення: 19.09.2025).

15. Printing – or exporting the task list – English Forum / Ask the Community – Asana Forum. URL: https://surl.lu/imbxok (дата звернення: 20.09.2025).

16. Що таке Trello і як програма допомагає в роботі Проєкт менеджера – Wizeclub Education. URL: https://surli.cc/hutmxa (дата звернення: 20.09.2025).

17. Simple Task Manager Template Notion Marketplace. URL: https://surl.li/gkyuic (дата звернення: 20.09.2025).

18. What is Django? Python Web Framework Explained with Features & Examples Lenovo US. URL: https://surl.li/aujjch (дата звернення: 21.09.2025).

19. Python Developers Survey 2023 Results. URL: https://surl.li/hsitwx (дата звернення: 21.09.2025).

20. Is Django Good for Microservices? URL: https://surl.lt/axhmko (дата звернення: 22.09.2025).

21. Django documentation. URL: https://surli.cc/frnlwy (дата звернення: 23.09.2025).

22. 2024 State of the Java Ecosystem Report New Relic. URL: https://surl.li/fjjysw (дата звернення: 23.09.2025).

23. What Is Spring Boot? Oracle APAC. URL: https://surli.cc/owclgr (дата звернення: 23.09.2025).

24. Microservices and Spring Boot – concerns about memory footprint. (Design forum at Coderanch). URL: https://surl.li/ecbnzr (дата звернення: 23.09.2025).

25. Stack Overflow Developer Survey 2023. URL: https://survey.stackoverflow.co/2023/ (дата звернення: 23.09.2025).

26. 2024 Ruby on Rails Community Survey Results. URL: https://railsdeveloper.com/survey/2024/ (дата звернення: 24.09.2025).

27. ORM, Ruby and ActiveRecord. Right off the bat, what is ORM by Violet Moon Medium. URL: https://surl.li/cyaqak (дата звернення: 24.09.2025).

28. Creating and Customizing Rails Generators & Templates – Ruby on Rails Guides. URL: https://surl.li/cqwcky (дата звернення: 25.09.2025).

29. Stack Overflow Developer Survey 2024. URL: https://survey.stackoverflow.co/2024/ (дата звернення: 26.09.2025).

30. Eloquent: Getting Started – Laravel 12.x – The PHP Framework For Web Artisans. URL: https://laravel.com/docs/12.x/eloquent (дата звернення: 27.09.2025).

31. Starter Kits – Laravel 12.x – The PHP Framework For Web Artisans. URL: https://laravel.com/docs/12.x/starter-kits (дата звернення: 28.09.2025).

32. Що буде краще для вашого інтерфейсу – Wezom. URL: https://surl.li/nfpebk (дата звернення: 28.09.2025).

33. Vue 3 Composition API – The Wrong Solution To The Right Problem – DEV Community. URL: https://surl.li/yliyvo (дата звернення: 28.09.2025).

34. Why You Should Start Learning Svelte. URL: https://surl.li/zhoiep (дата звернення: 28.09.2025).

35. Understanding How React Works Under the Hood LinkedIn. URL: https://surl.li/lantdn (дата звернення: 29.09.2025).

36. Що означає PostgreSQL: докладне пояснення – Побутові методики. URL: https://surl.li/vlwfrb (дата звернення: 29.09.2025).

37. PostgreSQL чи MySQL. Як обрати оптимальну базу даних для вашого проєкту. URL: https://surl.li/esjnwh (дата звернення: 30.09.2025).