

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Вебсередовище управління ІТ-проектами з інтеграцією зовнішніх
сервісів»**

Виконав студент групи ПЗ-22-1
ОЛІЙНИК Валерія Ігорівна

Керівник роботи:
ЧИЖМОТРЯ Олена Геннадіївна

Рецензент:
ТОЛСТОЙ Ігор Анатолійович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ОЛІЙНИК Валерія Ігорівна**

Керівник роботи: **ЧИЖМОТРЕ Олена Геннадіївна**

Тема роботи: «Вебсередовище управління ІТ-проєктами з інтеграцією зовнішніх сервісів»,

затверджена Наказом закладу вищої освіти від «27» лютого 2026 р., №92/с

Вихідні дані для роботи: **засоби розробки вебсередовища для управління ІТ-проєктами з інтеграцією зовнішніх сервісів**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.Г.	01.03.2026р.	01.03.2026р
2	Чижмотря О.Г.	01.03.2026р	01.03.2026р
3	Чижмотря О.Г.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку системи управління ІТ-проєктами та пояснювальної записки. Випускна кваліфікаційна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 30 найменувань (3 стор.), 6 додатків (26 стор.), 21 таблиць та 45 ілюстрацій по тексту. Повний обсяг роботи становить 100 сторінок, з них 72 основного тексту.

Метою роботи є створення вебсередовища управління ІТ-проєктами з використанням сучасних технологій та інтеграцією зовнішніх сервісів для автоматизації процесів планування, контролю та взаємодії між учасниками проєкту.

Архітектура розроблюваної системи базується на клієнт-серверному підході, що забезпечує масштабованість, гнучкість та зручність розширення функціоналу. Для забезпечення взаємодії із зовнішніми сервісами використовуються API, що дозволяє інтегрувати систему з різними сервісами.

Розроблений додаток має зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам створювати проєкти, керувати задачами, контролювати їх виконання та отримувати сповіщення про зміни в реальному часі.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, УПРАВЛІННЯ ПРОЄКТАМИ, ІТ-ПРОЄКТ, ІНТЕГРАЦІЯ, API, ВЕБСЕРЕДОВИЩЕ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА.

					ІПЗ.КР.Б – 121 – 26 – ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Олійник В.І.			Вебсередовище управління ІТ-проєктами з інтеграцією зовнішніх сервісів	Літ.	Арк.
Керівник		Чижмотря О.Г.					3
						Аркушів	100
Рецензент		Толстой І.А.				Житомирська політехніка, група ІПЗ-22-1	
Зав. каф.		Вакалюк Т.А.					

ABSTRACT

The bachelor's qualification work consists of a web application for IT project management and an explanatory note. The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of 30 references (3 pages), six appendices (26 pages), 21 tables, and 45 figures throughout the text. The total length of the thesis is 100 pages, including 72 pages of main text.

The aim of the work is to develop a web-based environment for IT project management using modern technologies and integration with external services to automate planning, monitoring, and collaboration processes among project participants.

The architecture of the developed system is based on a client-server approach, which ensures scalability, flexibility, and ease of functionality extension. Interaction with external services is implemented through APIs, allowing integration with various services.

The developed application features a user-friendly and intuitive interface that enables users to create projects, manage tasks, track their progress, and receive real-time notifications about changes.

KEYWORDS: WEB APPLICATION, PROJECT MANAGEMENT, IT PROJECT, INTEGRATION, API, WEB ENVIRONMENT, CLIENT-SERVER ARCHITECTURE.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ВЕБСЕРЕДОВИЩА УПРАВЛІННЯ ІТ-ПРОЄКТАМИ.....	10
1.1. Постановка задачі.....	10
1.2. Аналіз аналогів програмного продукту	12
1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	16
Висновки до першого розділу	18
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСЕРЕДОВИЩА УПРАВЛІННЯ ІТ- ПРОЄКТАМИ.....	20
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	20
2.2. Розробка бази даних системи	27
2.3. Проєктування алгоритмів функціонування програмної системи	32
2.4. Реалізація функціоналу вебсередовища для керування ІТ- проєктами.....	35
Висновки до другого розділу	40
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСЕРЕДОВИЩЕМ ДЛЯ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ.....	42
3.1. Порядок встановлення та налаштування параметрів вебсередовища.....	42
3.2. Структура інтерфейсу вебсередовища для управління ІТ- проєктами.....	44
3.3. Тестування роботи програмного комплексу.....	63
Висновки до третього розділу	67

ВИСНОВКИ.....	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	70
ДОДАТКИ	73

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс (Application Programming Interface), що використовується для взаємодії між системою та зовнішніми сервісами.

БД – база даних, що використовується для зберігання інформації про проекти, задачі та користувачів.

ПЗ – програмне забезпечення.

CRUD – базові операції управління даними: створення (Create), читання (Read), оновлення (Update), видалення (Delete).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						7

ВСТУП

Актуальність теми зумовлена стрімким розвитком інформаційних технологій та зростанням складності ІТ-проектів, що вимагає застосування ефективних інструментів управління та координації командної роботи. У сучасних умовах цифрової трансформації підприємств значна частина бізнес-процесів пов'язана з розробкою та супроводом програмного забезпечення, що потребує впровадження спеціалізованих інформаційних систем для управління проектною діяльністю.

Особливого значення набуває використання веборієнтованих систем, які забезпечують доступ до функціоналу незалежно від місця перебування користувача та дозволяють організовувати роботу розподілених команд. Водночас важливою тенденцією є інтеграція таких систем із зовнішніми сервісами, що забезпечує автоматизацію інформаційних потоків, зменшення кількості ручних операцій та підвищення загальної ефективності управління проектами.

Інтеграція із зовнішніми сервісами на основі прикладних програмних інтерфейсів (API) дозволяє поєднувати різні програмні продукти в єдине інформаційне середовище, забезпечуючи синхронізацію даних, оперативний обмін інформацією та підтримку прийняття управлінських рішень. Це особливо актуально для ІТ-сфери, де використовуються численні інструменти для розробки, тестування, комунікації та планування.

Метою випускної кваліфікаційної роботи бакалавра є розробка вебсередовища управління ІТ-проектами з інтеграцією зовнішніх сервісів, яке забезпечує автоматизацію процесів планування, контролю та координації виконання задач.

Для досягнення поставленої мети необхідно провести аналіз предметної області та існуючих систем управління ІТ-проектами. Також потрібно дослідити сучасні підходи до інтеграції програмних систем та організації обміну даними. Наступним завданням є формування функціональних і нефункціональних вимог до програмного продукту. Окрему увагу слід

приділити обґрунтуванню вибору архітектурних рішень та інструментальних засобів розробки. Після цього необхідно розробити архітектуру вебсередовища управління проєктами та реалізувати основний функціонал системи. Завершальним етапом є проведення тестування та оцінка ефективності розробленого програмного продукту.

Об'єктом дослідження є процеси управління ІТ-проєктами в умовах використання сучасних інформаційних технологій.

Предметом дослідження є методи та засоби розробки веборієнтованих систем управління проєктами, а також механізми інтеграції із зовнішніми сервісами.

У процесі виконання випускної кваліфікаційної роботи використовуються методи системного аналізу, порівняльного аналізу, об'єктно-орієнтованого проєктування, вебпрограмування, моделювання та тестування програмного забезпечення. Інформаційну базу дослідження становлять наукові та навчально-методичні джерела, офіційна документація використаних технологій, а також матеріали щодо зовнішніх сервісів і прикладних програмних інтерфейсів.

Практичне значення одержаних результатів полягає в тому, що розроблене вебсередовище може використовуватися для організації роботи ІТ-команд, планування проєктів, контролю виконання задач, координації зустрічей і роботи з ресурсами та зовнішніми сервісами. Отримані результати також можуть використовуватися в навчальних цілях як приклад розробки сучасної веборієнтованої системи управління проєктами.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ВЕБСЕРЕДОВИЩА УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

1.1. Постановка задачі

У сучасних умовах розвитку інформаційних технологій управління ІТ-проєктами є складним процесом, який охоплює планування задач, контроль їх виконання, координацію роботи команди та підтримку ефективної комунікації між усіма учасниками. Додаткові труднощі виникають через використання великої кількості окремих сервісів для керування задачами, ведення документації, планування зустрічей, контролю змін і організації командної взаємодії. За таких умов зростає потреба в єдиному інформаційному середовищі, яке забезпечує централізований доступ до даних, спрощує обмін інформацією та підтримує узгодженість робочих процесів.

Вебсередовище управління ІТ-проєктами є програмною системою, що дозволяє організовувати процес розробки, керувати задачами, контролювати їх виконання та забезпечувати взаємодію між учасниками команди через вебінтерфейс. Використання веборієнтованого підходу є доцільним, оскільки воно надає користувачам можливість працювати із системою незалежно від місця перебування, що є особливо важливим для розподілених команд. Крім цього, така організація роботи підвищує прозорість виконання завдань, полегшує доступ до актуальної інформації про стан проєкту та зменшує кількість ручних операцій під час координації спільної діяльності.

Суттєвою ознакою сучасних систем управління проєктами є підтримка інтеграції із зовнішніми сервісами, зокрема Jira, Confluence та Google Calendar. Така взаємодія дає змогу автоматизувати частину робочих процесів, пов'язаних із плануванням, комунікацією та супроводом розробки програмного забезпечення. Завдяки використанню прикладних програмних інтерфейсів забезпечується синхронізація даних між різними платформами, що знижує ризик дублювання інформації, підвищує оперативність її оновлення та покращує загальну керованість проєктом.

Важливою вимогою до такого вебсередовища є не лише наявність окремих функціональних можливостей, а й забезпечення їх узгодженої роботи в межах єдиного програмного рішення. Користувач повинен мати змогу швидко отримувати доступ до актуального стану проєкту, бачити взаємозв'язок між задачами, зустрічами, ресурсами та змінами в командній роботі, не переходячи між великою кількістю сторонніх платформ. Саме тому розроблювана система має бути орієнтована на централізацію даних, спрощення повсякденних робочих дій і зменшення часу, що витрачається на пошук інформації, дублювання операцій та координацію між учасниками проєкту.

Розроблюване вебсередовище повинно забезпечувати користувачам умови для повноцінної роботи з проєктами, задачами та пов'язаною інформацією в межах одного застосунку. Воно має підтримувати створення й оновлення задач, контроль строків виконання, координацію взаємодії між учасниками команди, коментування, сповіщення, інтеграцію із зовнішніми сервісами, а також адміністрування користувачів, ролей і прав доступу. Така побудова дає змогу поєднати в одному середовищі як інструменти щоденної командної роботи, так і засоби організації, контролю та супроводу проєктної діяльності.

У процесі розробки такого програмного продукту важливими є побудова зручного інтерфейсу, проєктування клієнтської та серверної архітектури, реалізація механізмів авторизації та розмежування доступу, створення бази даних і впровадження основного функціоналу для роботи з проєктами, задачами та інтеграціями. Не менш важливим є забезпечення коректної взаємодії між окремими модулями системи та проведення тестування, яке дозволяє оцінити працездатність реалізованих рішень і якість користувацької взаємодії.

У результаті має бути створено вебсередовище, яке забезпечує ефективну організацію управління ІТ-проєктами, підтримує взаємодію між учасниками команди, спрощує контроль виконання задач і надає можливість

інтеграції із зовнішніми сервісами. Це створює підґрунтя для підвищення зручності роботи користувачів, покращення координації дій у команді та подальшого розвитку функціональних можливостей системи.

1.2. Аналіз аналогів програмного продукту

Розробка вебсередовища для управління проєктами потребує глибокого аналізу тому варто озирнутися на вже існуючі рішення. Аналіз популярних інструментів допомагає зрозуміти, які функції дійсно потрібні користувачам, а які залишаються другорядними або незатребуваними.

Сьогодні існує багато інструментів для управління проєктами, тому давайте розглянемо популярні з них.

Одним з найвідоміших сервісів є Trello [\[1\]](#) (рис. 1.1)

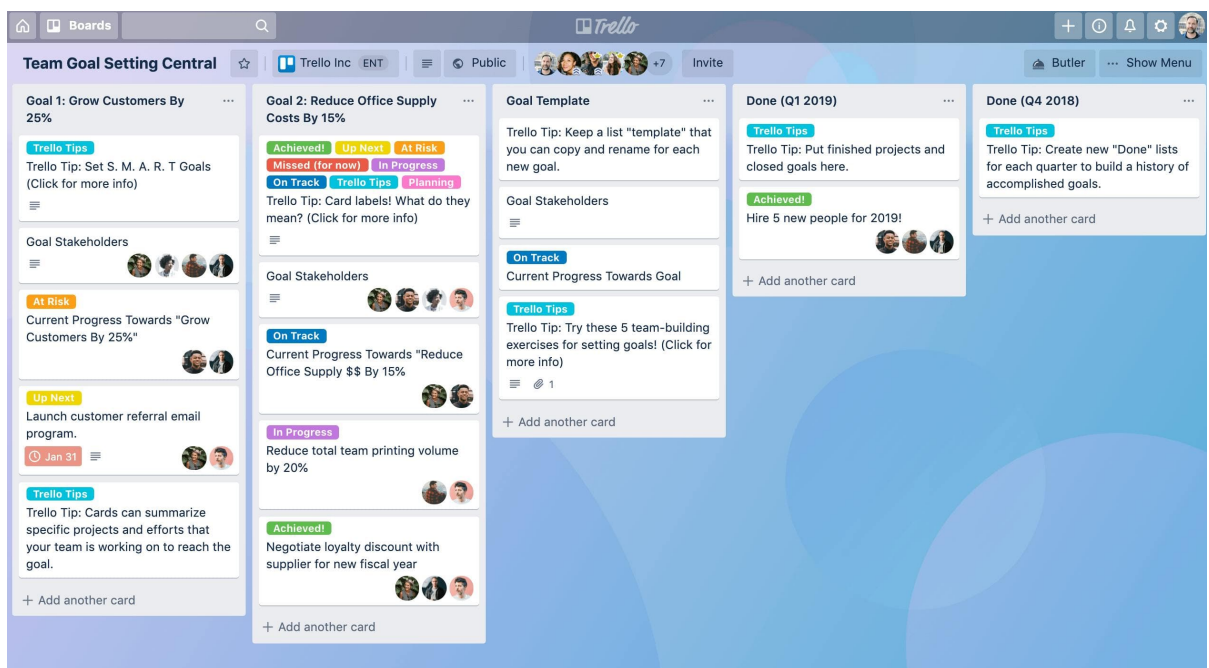


Рис. 1.1. Платформа для управління проєктами Trello

Trello побудований на принципі дошок Kanban, у межах яких задачі представлені у вигляді карток.

До його основних переваг належать простий і зрозумілий інтерфейс, швидке створення задач і проєктів, наявність інтеграцій із зовнішніми сервісами, а також доступність на різних платформах.

Разом із цим система має і певні недоліки. Зокрема, Trello має обмежені можливості для роботи зі складними проєктами, не забезпечує глибоку аналітику, а значна частина розширеного функціоналу доступна лише в платній версії.

Наступний і більш професійний сервіс для управління проєктами є Jira (рис. 1.2).

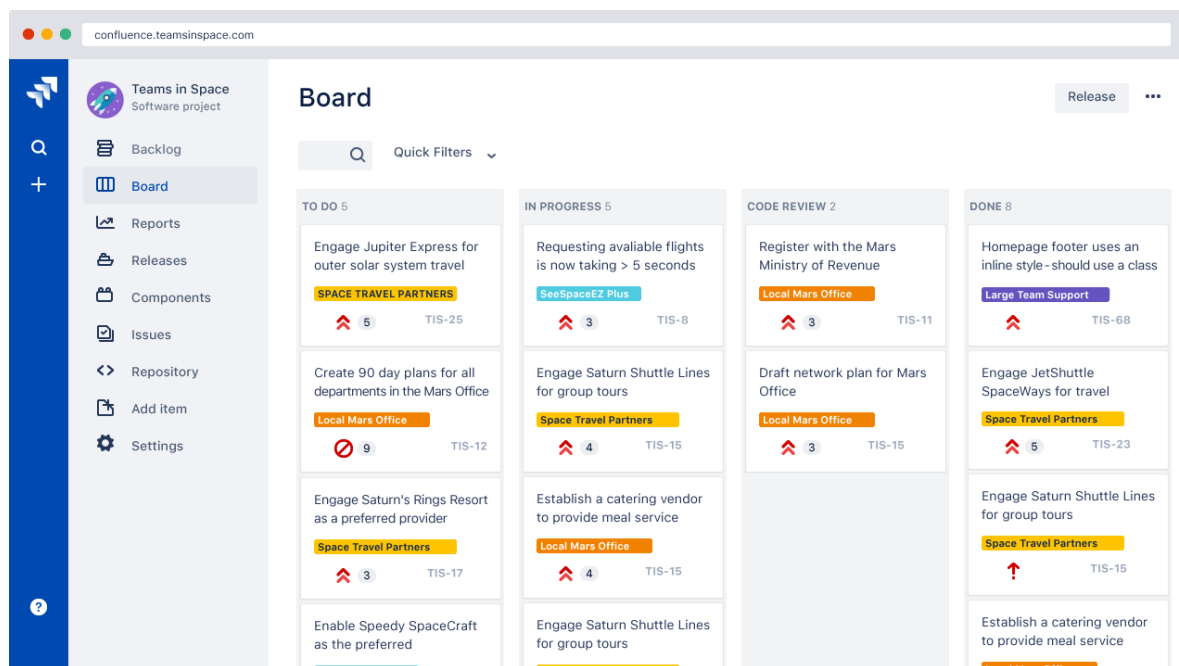


Рис. 1.2. Платформа для управління проєктами Jira [2]

Jira активно використовується в ІТ-компаніях і командах, особливо в тих випадках, коли робота організовується за Agile-методологією.

До її основних переваг належать гнучке налаштування процесів, підтримка різних фреймворків, зокрема Scrum і Kanban, розвинена система звітності та велика кількість інтеграцій із зовнішніми сервісами.

Водночас ця система має і певні недоліки. Для нових користувачів Jira може бути складною в освоєнні, а її інтерфейс нерідко сприймається як перевантажений через велику кількість карток, задач і службових елементів.

Крім того, при роботі на невеликих екранах частина інформації сприймається менш зручно, а повноцінне налаштування проєкту потребує додаткового часу.

Серед аналогів є український проєкт Worksection (рис. 1.3).



Рис. 1.3. Платформа для управління проектами Worksection [3]

Worksection орієнтований на командну роботу та контроль виконання задач.

Його перевагами є зручний і простий інтерфейс, наявність звітів та аналітики, можливість планування задач, а також придатність для невеликих команд із відносно невеликим навантаженням.

Разом із цим сервіс має низку обмежень. Зокрема, у ньому відсутні інтеграції з багатьма зовнішніми сервісами, існують обмеження щодо використання пам'яті для зображень і файлів, а також спостерігається менша гнучкість у налаштуванні порівняно з більш масштабними системами.

Додатково можна відзначити закритість самої системи та окремі проблеми з продуктивністю.

Ще одним з популярних сервісів варто виділити ClickUp [4] (рис. 1.4).

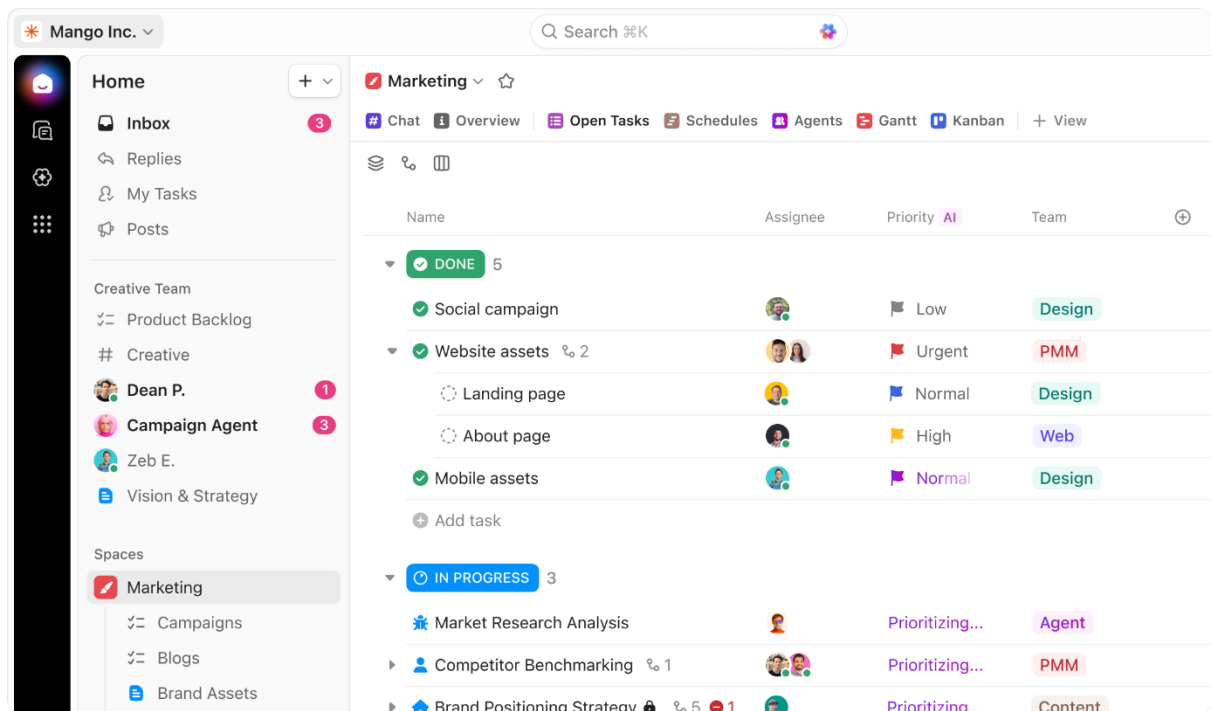


Рис. 1.4. Платформа для управління проектами ClickUp

ClickUp намагається поєднати в межах одного продукту велику кількість різних функцій, однак не завжди сприймається як основний вибір для великих команд і компаній.

До його переваг належать широкий функціонал, підтримка різних форматів відображення задач, гнучкість налаштування проєктів та наявність інтеграцій із зовнішніми сервісами.

Однак ця система також має свої недоліки. Вона є складною для швидкого освоєння, а інтерфейс інколи виглядає надто перевантаженим. Крім того, в окремих випадках під час роботи можуть виникати проблеми з продуктивністю.

Виходячи з перелічених недоліків та переваг, складемо порівняльну таблицю з вищезгаданих аналогів (табл. 1.1).

Таблиця 1.1.

Порівняльна таблиця аналогів системи управління проєктів

Характеристика	Trello	Jira	Worksection	ClickUp
Простота використання	+	-	+	+/-
Гнучкість	-	+	-	+

Продовження таблиці 1.1.

Інтеграції	+	+	-	+
Аналітика	-	+	+	+
Кросплатформеність	+	+	+	+

Проведений аналіз показує, що сучасні системи управління проєктами мають досить широкий функціонал і підтримують інтеграцію з зовнішніми сервісами. Проте жодна з них не є універсальною, одна система є занадто складна, а інші проста, але проблемна з продуктивністю.

Саме тому доцільно розробити власне середовище для керування проєктами, яке поєднуватиме простоту використання з необхідним функціоналом та можливістю інтеграції з зовнішніми сервісами.

1.3. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Розроблюване вебсередовище для управління ІТ-проєктами буде побудоване на клієнт-серверній архітектурі. Такий підхід є одним із найпоширеніших у сучасній веброзробці, оскільки дозволяє чітко розділити обробку даних і користувацький інтерфейс. У результаті система стає більш гнучкою і її легше масштабувати, підтримувати та розвивати в майбутньому.

Серверна частина додатку реалізовуватиметься з використанням платформи Node.js [8]. Ця платформа дозволяє працювати з JavaScript як на сервері, так і на клієнті, що спрощує процес розробки та робить його більш узгодженим. Крім того, Node.js добре підходить для систем, які повинні обробляти велику кількість одночасних запитів, що особливо важливо для багатокористувацьких вебдодатків. Також ця платформа зручна для створення REST API, необхідного для взаємодії із зовнішніми сервісами.

Клієнтська частина буде створена за допомогою бібліотеки React, яка широко використовується для розробки сучасних інтерактивних інтерфейсів. Завдяки компонентному підходу інтерфейс розбивається на окремі незалежні частини, які можна повторно використовувати. Це не лише спрощує розробку,

а й полегшує подальшу підтримку та масштабування додатку. Додатковою перевагою є використання віртуального DOM, що забезпечує швидке оновлення інтерфейсу та покращує продуктивність.

Для зберігання даних обрано реляційну систему управління базами даних MySQL [30]. Вона дозволяє надійно працювати з інформацією про користувачів, проекти, задачі та їх взаємозв'язки. Популярність MySQL у веброзробці забезпечує її стабільність і підтримку великою спільнотою. Для зручності адміністрування використовується phpMyAdmin, який надає зрозумілий графічний інтерфейс для роботи з базою даних.

Локальна розробка та тестування здійснюватимуться в середовищі MAMP, що дозволяє швидко розгорнути всі необхідні компоненти на локальному комп'ютері. Це дає можливість перевіряти роботу додатку на різних етапах без потреби в додаткових налаштуваннях.

Окрему увагу приділено механізму автентифікації користувачів. Для цього буде використано технологію JSON Web Token (JWT), яка забезпечує безпечну передачу даних між клієнтом і сервером. Після успішної авторизації користувач отримує токен, який надалі використовується для доступу до захищених ресурсів. Такий підхід дозволяє відмовитися від зберігання сесій на сервері та забезпечує більш гнучку й масштабовану систему авторизації.

Взаємодія між клієнтською та серверною частинами відбувається через HTTP-запити у форматі JSON, що є простим і ефективним способом обміну даними, а також полегшує інтеграцію із зовнішніми сервісами.

У цілому архітектура системи включає кілька ключових компонентів: клієнтську частину (frontend), серверну частину (backend), базу даних і зовнішні сервіси.

Таким чином, обрані технології та підходи дозволяють створити сучасний, гнучкий і масштабований вебдодаток, який відповідає основним вимогам до систем управління ІТ-проєктами.

Для забезпечення коректної роботи вебдодатку користувацький пристрій повинен відповідати базовим технічним вимогам. Мінімальна конфігурація

передбачає наявність процесора з двома ядрами, не менше 4 ГБ оперативної пам'яті, щонайменше 1 ГБ вільного місця на диску та доступу до мережі Інтернет. Таких характеристик достатньо для запуску клієнтської частини системи, роботи з основними модулями та виконання стандартних користувацьких дій у вебсередовищі.

Для комфортнішої та стабільнішої роботи доцільно використовувати пристрій, який має процесор із чотирма ядрами або більше, не менше 8 ГБ оперативної пам'яті, SSD-накопичувач і стабільне підключення до мережі Інтернет.

Висновки до першого розділу

У першому розділі було проведено аналіз предметної області управління ІТ-проектами, в результаті якого визначено основні вимоги до вебсередовища, призначеного для організації роботи команди, управління задачами та контролю виконання проєктів.

У процесі дослідження аналогічних програмних продуктів було розглянуто та проаналізовано сучасні системи управління проєктами, такі як Trello, Jira, ClickUp та Worksection. Було визначено їх основні функціональні можливості, а також переваги і недоліки, що дозволило сформулювати вимоги до розроблюваної системи.

На основі проведеного аналізу було встановлено, що існуючі рішення мають широкий функціонал, проте не завжди поєднують простоту використання, гнучкість налаштування та достатній рівень інтеграції із зовнішніми сервісами.

Враховуючи отримані результати, було обґрунтовано доцільність розробки власного вебсередовища управління ІТ-проектами з інтеграцією зовнішніх сервісів, яке поєднуватиме необхідний функціонал та зручність використання.

Крім того, на основі сформованих вимог було здійснено вибір інструментальних засобів для реалізації вебдодатку, що забезпечують

ефективність розробки, масштабованість та можливість подальшого розширення системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСЕРЕДОВИЩА УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Розроблюване вебсередовище для управління ІТ-проєктами дозволяє користувачам зручно організовувати роботу, керувати задачами та ефективно взаємодіяти в межах команди. Система реалізована у форматі вебдодатку, тому доступ до неї здійснюється через звичайний браузер без потреби встановлення додаткового програмного забезпечення.

Функціональні можливості системи залежать від ролі користувача. Передбачено три основні ролі: адміністратор, менеджер проєкту (Project Manager) та учасник команди.

Адміністратор відповідає за загальне керування системою, він контролює користувачів, налаштовує доступи та забезпечує коректну роботу платформи.

Менеджер проєкту організовує робочий процес, також, створює проєкти, формує задачі та розподіляє їх між учасниками.

Учасник команди, своєю чергою, працює із задачами, виконує їх і взаємодіє з іншими користувачами.

Для наочного відображення взаємодії користувачів із системою було побудовано діаграму варіантів використання (рис. 2.1).

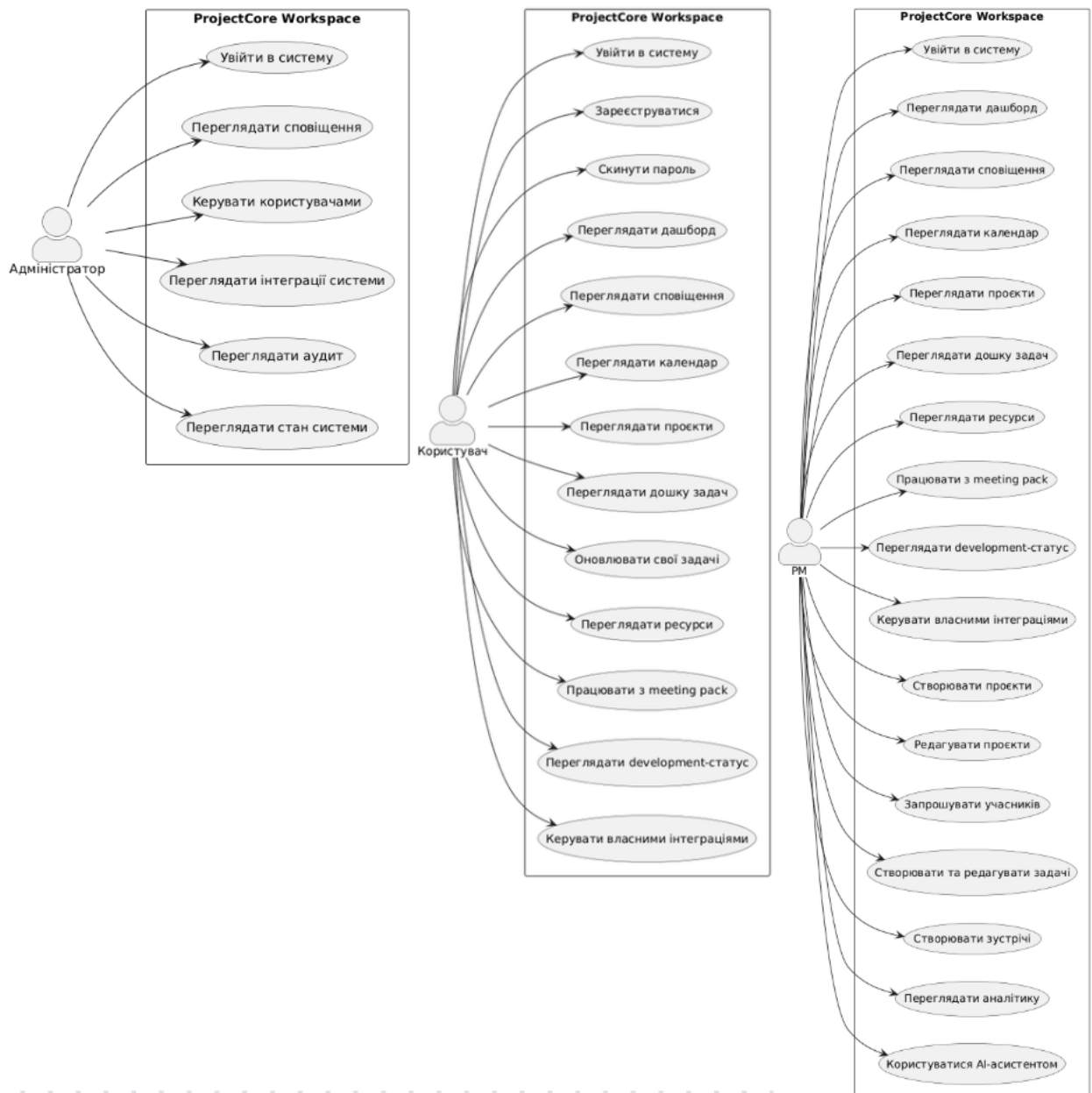


Рис. 2.1. Діаграма варіантів використання

Опис основних варіантів використання наведено в таблицях 2.1 - 2.10.

Таблиця 2.1.

Опис прецеденту «Вхід/Реєстрація»

Варіант використання	Вхід/Реєстрація
Основні актори	Адміністратор, менеджер проєкту, учасник команди
Короткий опис	Користувачі можуть зареєструватися та авторизуватися в системі. Для авторизації

Продовження таблиці 2.1.

	використовується JWT. Також, підтримується вхід через Google.
--	---

Таблиця 2.2.

Опис прецеденту «Перегляд дашборду»

Варіант використання	Перегляд дашборду
Основні актори	Менеджер проєкту, учасник команди
Короткий опис	Користувачі можуть переглядати головний робочий екран із короткою інформацією про задачі, зустрічі, активність та поточний стан роботи.

Таблиця 2.3.

Опис прецеденту «Перегляд проєктів»

Варіант використання	Перегляд проєктів
Основні актори	Менеджер проєкту, учасник команди
Короткий опис	Користувачі можуть переглядати список доступних проєктів, їх основні параметри та переходити до пов'язаних модулів.

Таблиця 2.4.

Опис прецеденту «Управління проєктами»

Варіант використання	Управління проєктами
Основні актори	Менеджер проєкту
Короткий опис	Менеджер проєкту створює нові проєкти, редагує їх параметри, додає опис, керує учасниками та запрошеннями.

Таблиця 2.5.

Опис прецеденту «Управління задачами»

Варіант використання	Управління задачами
Основні актори	Менеджер проєкту, учасник команди
Короткий опис	Менеджер проєкту створює та редагує задачі, призначає виконавців і контролює статуси. Учасники команди переглядають свої задачі, оновлюють їх статус і виконують роботу на дошці.

Таблиця 2.6.

Опис прецеденту «Керування зустрічами та календарем»

Варіант використання	Керування зустрічами та календарем
Основні актори	Менеджер проєкту, учасник команди
Короткий опис	Користувачі можуть переглядати календар подій. Менеджер проєкту може створювати зустрічі, а учасники можуть переглядати свої зустрічі та нагадування про події та зустрічі.

Таблиця 2.7.

Опис прецеденту «Робота з ресурсами та документацією»

Варіант використання	Робота з ресурсами та документацією.
Основні актори	Менеджер проєкту, учасник команди
Короткий опис	Користувачі можуть переглядати й додавати ресурси проєкту, відкривати документи з різних ресурсів в системі, а також, працювати з ними у межах системи.

Таблиця 2.8.

Опис прецеденту «Аналітика та ІІІ-підтримка»

Варіант використання	Аналітика та ІІІ-підтримка
Основні актори	Менеджер проєкту
Короткий опис	Менеджер проєкту може переглядати аналітичні показники проєкту, отримувати короткі підсумки, виявляти ризики та використовувати ІІІ-асистента для формування звітів та допоміжних дій.

Таблиця 2.9.

Опис прецеденту «Перегляд статусу розробки»

Варіант використання	Перегляд статусу розробки
Основні актори	Менеджер проєкту, учасник проєкту
Короткий опис	Користувачі можуть переглядати інформацію про прив'язані репозиторії та останні дії в ньому.

Таблиця 2.10.

Опис прецеденту «Адміністрування системи»

Варіант використання	Адміністрування системи
Основні актори	Адміністратор
Короткий опис	Адміністратор керує користувачами, ролями, статусами аккаунтів, переглядає загальний стан системи та інтеграцій.

Основною метою розробки вебсередовища є створення зручного інструменту для управління ІТ-проєктами, який дозволяє організувати роботу команди, забезпечити контроль виконання задач та підвищити ефективність взаємодії між учасниками.

Система повинна спрощувати процес планування, розподілу задач і моніторингу їх виконання, а також забезпечувати користувачам доступ до актуальної інформації в режимі реального часу.

Під час визначення вимог до програмного продукту було враховано потреби трьох основних ролей, а саме адміністратора, менеджера проєкту та учасника команди.

Адміністратор повинен мати можливість керувати користувачами та їхніми ролями, контролювати роботу системи та переглядати всі проєкти.

Менеджер проєкту повинен мати доступ до створення проєктів, формування задач, призначення виконавців, контролю виконання задач і перегляду аналітики.

Учасник команди повинен мати можливість переглядати власні задачі, виконувати їх, змінювати статус, залишати коментарі та переглядати аналітичну інформацію в межах доступного функціоналу.

До функціональних вимог системи належить забезпечення реєстрації та авторизації користувачів. Також у системі повинна бути реалізована можливість створення проєктів і повноцінного управління задачами.

Окремо має бути передбачена можливість комунікації між користувачами, зокрема через коментарі та пов'язані дії в межах задач і проєктів. Система повинна підтримувати аналітику проєктів, а всі основні дані мають зберігатися в базі даних.

До нефункціональних вимог системи належать вимоги до зручності використання, надійності та продуктивності. Інтерфейс повинен бути простим, зрозумілим і зручним для користувача, а час відгуку системи не повинен перевищувати трьох секунд за стандартних умов роботи. Також система повинна забезпечувати достатній рівень надійності та бути орієнтованою на безперервну роботу в режимі 24/7. З погляду продуктивності система повинна підтримувати одночасну роботу кількох користувачів.

До мінімальних системних вимог належать наявність не менше 2 ГБ оперативної пам'яті, процесора з тактовою частотою від 1.4 GHz, а також

стабільного доступу до мережі Інтернет. Серверна частина повинна підтримувати середовище Node.js, використання системи керування базами даних MySQL та реалізацію REST API для взаємодії між клієнтською і серверною частинами системи.

На основі аналізу функціональних вимог було виділено основні сутності системи та складено діаграму класів (рис. 2.2).

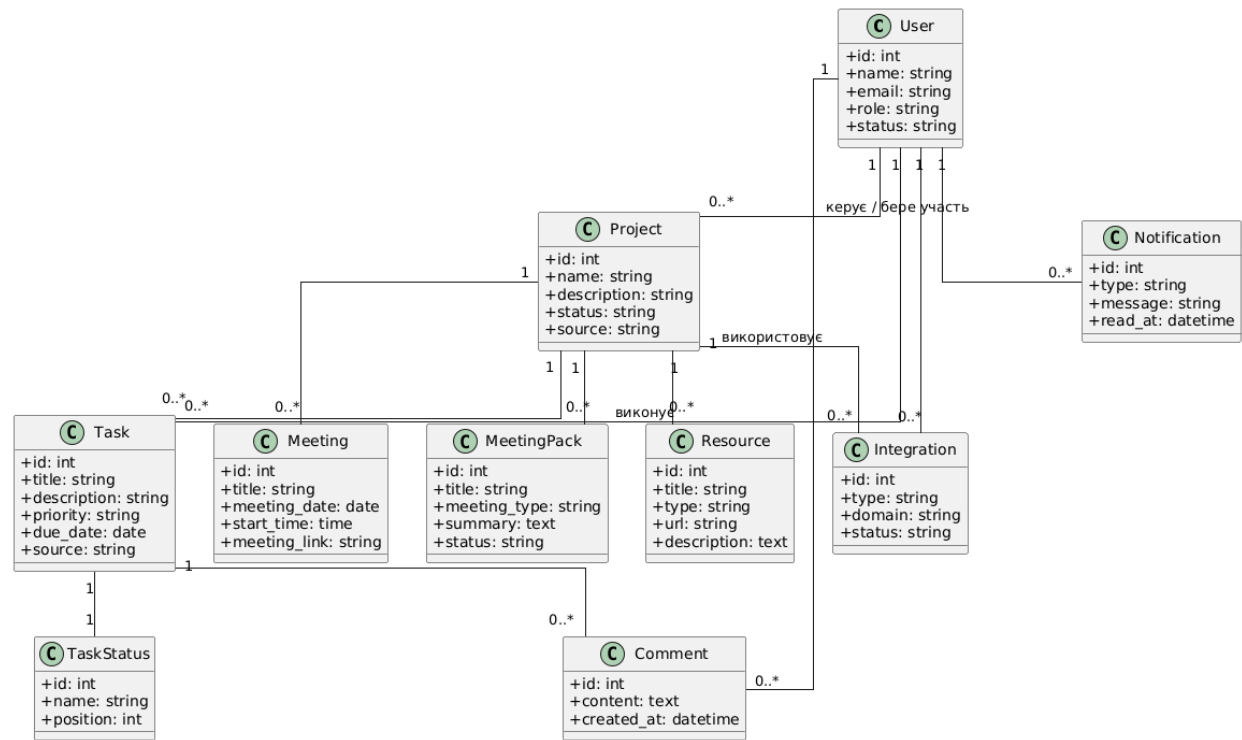


Рис. 2.2. Діаграма класів

Клас User описує користувача системи. Він зберігає основну інформацію: ідентифікатор, ім'я, електронну пошту, пароль, роль у системі та поточний статус акаунта.

Клас Project представляє проєкт, з яким працює команда. У ньому містяться дані про назву проєкту, опис, поточний статус, джерело даних.

Клас Integration описує підключення зовнішніх сервісів. Він використовується для збереження інформації про тип інтеграції, домен сервісу та поточний стан.

Клас Task відповідає за задачі в межах проєкту. Він містить назву задачі, її опис, пріоритет, термін виконання, джерело походження.

Клас TaskStatus використовується для представлення статусу задачі. У ньому зберігаються назва статусу та його позиція у списку або дошці.

Клас Comment призначений для зберігання коментарів до задач. Він забезпечує комунікацію між учасниками.

Клас Meeting описує зустрічі, які пов'язані з проектом. У ньому містяться назва зустрічі, дата, час початку та посилання на зустріч.

Клас MeetingPack використовується для зберігання коротких підсумків, нотаток і матеріалів до робочих зустрічей. Він дозволяє формувати структурований зріз по проекту перед зустріччю або після неї.

Клас Resource описує ресурси проекту. До них належать посилання, документи, таблиці, сторінки джерел інформації, пов'язані з проектом.

Клас Notification використовується для представлення системних оповіщень. Він містить тип повідомлення, його текст і інформацію про те, чи було воно прочитане.

Зв'язки між класами відображають логіку взаємодії основних сутностей у системі. Користувач може бути призначений на багато задач, а один проект, своєю чергою, може містити велику кількість задач. Кожна задача пов'язана з одним конкретним статусом, але при цьому може мати багато коментарів, які створюються різними користувачами. Окремий проект також може містити багато ресурсів, зустрічей і meeting pack, що відображає повноту інформації, яка накопичується в межах проектної діяльності. Користувач може мати кілька інтеграцій із зовнішніми сервісами, а сам проект може використовувати кілька інтеграцій одночасно залежно від потреб команди. Крім того, кожен користувач може отримувати багато системних сповіщень, пов'язаних із задачами, зустрічами, змінами в проекті та іншими подіями в системі.

2.2. Розробка бази даних системи

База даних є однією з ключових частин розробленої системи, оскільки саме вона забезпечує збереження користувачів, проектів, задач та зустрічей, ресурсів, інтеграцій та іншої службової інформації. Для реалізації системи

було використано MySQL, а для зручного створення таблиць, перегляду записів, виконання SQL-запитів і адміністрування структури бази даних використано PHPMyAdmin.

У базі даних системи використовуються 22 основні таблиці: users, projects, project_members, invitations, tasks, task_statuses, task_comments, task_attachments, task_links, task_labels, task_label_links, meetings, meeting_participants, meeting_reminder_logs, meeting_packs, project_resources, integrations, project_integrations, notification_reads, activity_logs, admin_audit_logs, password_reset_tokens.

Нижче наведено опис основних таблиць бази даних (табл. 2.11–2.15)

Таблиця 2.11.

Опис таблиці «users»

Назва поля	Тип	Опис
id	INT	Ідентифікатор користувача
name	VARCHAR	Ім'я користувача
email	VARCHAR	Електронна пошта
password_hash	VARCHAR	Хешований пароль
role	VARCHAR	Роль користувача в системі
status	VARCHAR	Статус акаунта
avatar_url	VARCHAR	Посилання на аватар
google_id	VARCHAR	Ідентифікатор Google-акаунта
created_at	TIMESTAMP	Дата і час створення запису

Таблиця 2.12.

Опис таблиці «projects»

Назва поля	Тип	Опис
id	INT	Ідентифікатор проєкту
name	VARCHAR	Назва проєкту
description	TEXT	Опис проєкту

Продовження таблиці 2.12

Назва поля	Тип	Опис
status	VARCHAR	Джерело проєкту
owner_id	INT	Зовнішній ідентифікатор проєкту
source	VARCHAR	Джерело проєкту
external_id	VARCHAR	Зовнішній ідентифікатор проєкту
created_at	TIMESTAMP	Дата і час створення проєкту

Таблиця 2.13.

Опис таблиці «tasks»

Назва поля	Тип	Опис
id	INT	Ідентифікатор задачі
project_id	INT	Ідентифікатор проєкту
title	VARCHAR	Назва задачі
description	TEXT	Опис задачі
status_id	INT	Ідентифікатор статусу задачі
assignee_id	INT	Ідентифікатор виконавця
priority	VARCHAR	Пріоритет задачі
due_date	DATE	Термін виконання
source	VARCHAR	Джерело задачі
external_id	VARCHAR	Зовнішній ідентифікатор задачі
created_at	TIMESTAMP	Дата і час створення задачі

Таблиця 2.14.

Опис таблиці «meetings»

Назва поля	Тип	Опис
id	INT	Ідентифікатор зустрічі
project_id	INT	Ідентифікатор проєкту
title	VARCHAR	Назва зустрічі

Продовження таблиці 2.14.

Назва поля	Тип	Опис
meeting_type	VARCHAR	Тип зустрічі
description	TEXT	Опис зустрічі
meeting_date	DATE	Дата проведення
start_time	VARCHAR	Час початку
end_time	VARCHAR	Час завершення
meeting_link	VARCHAR	Посилання на зустріч
created_by	INT	Ідентифікатор користувача, який створив зустріч

Таблиця 2.15.

Опис таблиці «integrations»

Назва поля	Тип	Опис
id	INT	Ідентифікатор інтеграції
user_id	INT	Ідентифікатор користувача
type	VARCHAR	Тип інтеграції
email	VARCHAR	Email для підключення, якщо потрібен
domain	VARCHAR	Домен зовнішнього сервісу
access_token	TEXT	Токен доступу
refresh_token	TEXT	Refresh token, якщо підтримується
status	VARCHAR	Статус інтеграції
created_at	TIMESTAMP	Дата створення
updated_at	TIMESTAMP	Дата оновлення

Інші таблиці, які не були детально розглянуті (project_members, invitations, task_statuses, task_comments, task_attachments, task_links, task_labels, task_label_links, meeting_participants, meeting_reminder_logs, meeting_packs, project_resources, project_integrations, notification_reads,

activity_logs, admin_audit_logs, password_reset_tokens), виконують допоміжну або службову функцію та забезпечують повноцінну роботу основних модулів системи. Частина з них пов'язана із задачами, частина - із зустрічами, ресурсами, інтеграціями або діями користувачів у системі.

Більшість таблиць пов'язані з таблицею users, оскільки значна частина сутностей у системі створюється, редагується або використовується конкретним користувачем. Саме тому в багатьох таблицях використовуються поля на зразок user_id, created_by, assignee_id або інші ідентифікатори користувача.

Також, значна кількість таблиць пов'язана з таблицею projects, оскільки проєкт є центральною сутністю системи. Саме через поле project_id визначається, до якого проєкту належить задача, зустріч, meeting pack, ресурс, активність або інший елемент. Такий підхід забезпечує цілісність збереження інформації та спрощує подальшу обробку даних на рівні окремого проєкту. Це дозволяє логічно групувати дані в межах конкретного проєкту та зручно працювати з ними в інтерфейсі системи.

Таблиці, пов'язані із задачами, мають зв'язок із таблицею tasks, що дозволяє зберігати коментарі, вкладення, мітки, зв'язки між задачами та іншу додаткову інформацію саме для конкретної задачі.

Також, таблиці, пов'язані із зустрічами, містять зв'язок із таблицею meetings, що дає можливість зберігати учасників зустрічі, нагадування та додаткові відомості в межах конкретної події.

Окрему роль виконують таблиці інтеграцій і журналів. Таблиці integrations та project_integrations забезпечують зв'язок системи із зовнішніми сервісами, а таблиці activity_logs, notification_reads та admin_audit_logs використовуються для фіксації активності, роботи сповіщень і контролю адміністративних дій.

Таким чином, структура бази даних побудована так, щоб основні сутності системи були пов'язані між собою, а допоміжні таблиці забезпечували зручну, цілісну та розширювану організацію даних.

2.3. Проектування алгоритмів функціонування програмної системи

Основною метою етапу проектування алгоритмів функціонування програмної системи є опис логіки роботи застосунку, визначення послідовності дій користувачів і реакцій системи, а також встановлення взаємодії між окремими модулями програмного комплексу. На цьому етапі важливо зрозуміти, у якій послідовності виконуються основні операції, які перевірки здійснюються під час обробки запитів і як різні ролі користувачів працюють із системою. Такий підхід дає змогу ще до завершення реалізації сформулювати цілісне уявлення про роботу програмного продукту та забезпечити узгодженість між його складовими.

Розроблене вебсередовище охоплює основні процеси, пов'язані з командною роботою над ІТ-проектами. До них належать авторизація користувачів, розмежування доступу, керування проектами, робота із задачами, календарем, ресурсами, інтеграціями, аналітикою та сповіщеннями. Для відображення загальної логіки роботи системи було побудовано діаграму активності, яка наведена в [Додатку А](#). Вона показує основні сценарії поведінки користувача після входу в систему та демонструє, як змінюється доступний функціонал залежно від ролі. Загальний алгоритм роботи починається з відкриття вебзастосунку та перевірки наявності активної сесії. Якщо користувач ще не увійшов до системи, йому пропонується перейти до сторінки авторизації або реєстрації. Після введення облікових даних система перевіряє правильність введеної інформації та права доступу. Якщо авторизація проходить успішно, користувач отримує доступ до внутрішніх модулів. Якщо виникає помилка або доступ відсутній, перехід до захищених сторінок блокується.

У межах проектування алгоритмів важливо також враховувати не лише окремі дії користувача, а й переходи між модулями системи. Після входу користувач не просто отримує доступ до певної сторінки, а починає рухатися по внутрішньому середовищу залежно від своїх поточних завдань. Наприклад, після відкриття дашборду він може перейти до списку проектів, далі відкрити

конкретний проєкт, переглянути задачі, перейти до календаря або відкрити ресурси. Саме тому алгоритм функціонування системи слід розглядати як послідовність взаємопов'язаних сценаріїв, у межах яких результат однієї дії стає основою для наступної. Такий підхід дозволяє зробити роботу користувача більш послідовною та забезпечити логічну навігацію між функціональними блоками застосунку.

Після входу система визначає роль користувача, оскільки від неї залежить подальший сценарій роботи. Для адміністратора передбачено перехід до адміністративної частини застосунку. Там він може переглядати список користувачів, змінювати їхні ролі, блокувати або розблоковувати акаунти, працювати з інтеграціями, аудитом і службовою інформацією. Якщо в системі авторизовано менеджера проєкту, він потрапляє на робочий дашборд. Звідти менеджер може переглядати та створювати проєкти, керувати задачами, працювати з дошкою, календарем, ресурсами, аналітикою, development-даними та інтеграціями. Саме для цієї ролі алгоритм є найбільш насиченим, оскільки менеджер координує командну роботу та взаємодіє з більшістю модулів системи. Якщо ж у системі авторизовано учасника команди, він потрапляє до власного робочого простору. Там користувач може переглядати проєкти, у яких бере участь, працювати із задачами, змінювати їх статус, переглядати календар, зустрічі, ресурси та сповіщення. Водночас система обмежує доступ до дій, які не відповідають ролі користувача.

Одним із центральних сценаріїв роботи системи є взаємодія користувача з проєктами та задачами. Після відкриття потрібного проєкту система надає доступ до пов'язаних із ним сутностей, зокрема задач, учасників, зустрічей, ресурсів і інтеграцій. У межах цього процесу важливо, щоб кожна дія виконувалася в правильній послідовності. Спочатку користувач обирає проєкт, після чого відкриває потрібний модуль, переглядає наявну інформацію і лише тоді виконує створення, редагування або оновлення даних. Така логіка дозволяє уникнути хаотичної взаємодії з системою та забезпечує зрозумілий порядок роботи з основними об'єктами. Крім цього, вона спрощує контроль

доступу, оскільки більшість операцій виконується в контексті конкретного проєкту.

Важливе місце в алгоритмах функціонування займають і внутрішні реакції системи. До них належать перевірка прав доступу, валідація введених даних, оновлення інформації в базі даних, формування сповіщень та обробка результатів інтеграційної взаємодії із зовнішніми сервісами. Під час роботи з проєктами та задачами система не лише зберігає зміни, а й підтримує узгоджену взаємодію між пов'язаними модулями. Це дозволяє зберігати актуальний стан середовища. Окремо слід зазначити, що частина сценаріїв пов'язана з календарем, зустрічами та додатковими сервісами, зокрема ІІІ-асистентом, який використовується для формування коротких підсумків і допоміжної інформації на основі даних про проєкт.

Окрему роль у загальному алгоритмі функціонування відіграють інтеграції із зовнішніми сервісами, оскільки вони розширюють межі локальної системи та дозволяють обмінюватися даними з іншими платформами. У такому випадку алгоритм охоплює не лише дії користувача та внутрішню обробку запиту, а й перевірку наявності активної інтеграції, формування відповідного запиту до зовнішнього сервісу та обробку отриманої відповіді. Після завершення таких дій система може оновлювати локальні дані, відображати результат синхронізації та формувати сповіщення для користувачів. Це особливо важливо у випадках, коли йдеться про календарні події, development-дані або зміни, які стосуються кількох учасників команди. У такий спосіб алгоритм функціонування охоплює як внутрішню логіку застосунку, так і його взаємодію із зовнішнім інформаційним середовищем.

Отже, побудована діаграма активності дозволяє відобразити загальний алгоритм функціонування системи та послідовність дій для різних ролей користувачів. Вона допомагає краще зрозуміти логіку роботи застосунку, структуру основних сценаріїв використання та взаємозв'язок між окремими модулями. Проєктування алгоритмів функціонування є важливим етапом розробки, оскільки саме воно забезпечує узгодження між вимогами до

системи, особливостями реалізації функціоналу та практичними сценаріями її використання.

2.4. Реалізація функціоналу вебсередовища для керування ІТ-проєктами

Основна увага приділяється структурі програмного рішення, реалізації ключових модулів системи, організації взаємодії між клієнтською та серверною частинами, а також опису найбільш важливих фрагментів коду.

Розроблений програмний комплекс побудовано за клієнт-серверною архітектурою. Клієнтська частина реалізована у вигляді вебзастосунку, що відповідає за інтерфейс користувача, навігацію між сторінками, відображення даних, обробку дій користувача та взаємодію з API. Серверна частина реалізує бізнес-логіку системи, роботу з базою даних, автентифікацію, перевірку прав доступу, інтеграції із зовнішніми сервісами та формування відповідей для клієнта.

Розглянемо загальну структуру програмного рішення.

Клієнтська частина побудована з використанням React [6] та Vite [29], що дозволяє створити динамічний інтерфейс з розподілом на окремі компоненти та сторінки. Для маршрутизації між модулями використовується React Router, а для базового оформлення інтерфейсу та елементів керування застосовано Bootstrap [10] разом із власними стилями.

У застосунку реалізовано окремі сторінки для основних функціональних модулів, таких як: стартова сторінка, сторінки входу та реєстрації, дашборд, сторінка проєктів, дошка задач, мітінг пак, розділ розробки, ресурси, календар, сповіщення, налаштування, аналітика, ші-асистент та адмін модулю.

Для організації єдиного стилю та однакової структури сторінок використано спільний каркас інтерфейсу, який включає sidebar, header та основну робочу область. Це дозволяє підтримувати однакову логіку навігації в усіх внутрішніх модулях системи. Окрема увага була приділена механізму маршрутизації та перевірки доступу. При переході на сторінку система

перевіряє, чи авторизований користувач, а також чи має він право доступу до конкретного модуля. Таким чином, користувач із роллю user не може отримати доступ до функцій адміністратора, а admin не використовує РМ-модулі як основний робочий інтерфейс.

Серверна частина побудована на Node.js із використанням Express. Такий підхід дозволяє реалізувати REST API, організувати обробку запитів від клієнта, взаємодію з базою даних MySQL, механізм авторизації на основі JWT, а також інтеграцію з зовнішніми системами, зокрема Jira, Trello, Google Calendar, Google Docs, Google Sheets, Confluence, Notion, GitLab, GitHub.

Організація серверного коду побудована таким чином, щоб розділити маршрути, контролери, middleware та сервіси. Такий підхід дозволяє уникати надмірного навантаження на один файл і робить код більш зрозумілим.

Однією з ключових частин серверної логіки є механізм авторизації. Після успішного входу користувач отримує JWT-токен, який надалі використовується для доступу до захищених маршрутів. На сервері перевіряється валідність токена, роль користувача та його право доступу до конкретного проєкту або модуля. Важливою частиною серверної реалізації також є робота з ролями.

Система підтримує три основні ролі: admin, pm та user.

Для кожної ролі передбачено власний набір доступних можливостей, що реалізується як на клієнтському, так і на серверному рівнях.

Модуль проєктів забезпечує створення, редагування та перегляд проєктів. Менеджер проєкту може задавати назву, опис, статус та інші параметри, а також запрошувати учасників до спільної роботи.

Проєкт є головною сутністю системи, оскільки саме до нього прив'язуються задачі, зустрічі, ресурси, інтеграції та дані розробки.

Модуль задач реалізує створення, редагування, відображення та зміну станів задач. Для візуального представлення використовується дошка зі статусами.

Кожна задача може містити: назву, опис, пріоритет, дедлайн, виконавця, коментарі та вкладення.

У межах реалізації, також, було передбачено можливість роботи із зовнішніми джерелами задач, зокрема Jira і Trello. Завдяки цьому частина задач може синхронізуватися із зовнішніми сервісами, а локальна дошка зберігає звичний для користувача інтерфейс роботи

Особливу увагу під час реалізації було приділено оновленню статусів задач. Оскільки зовнішні системи можуть мати власну логіку роботи, для інтеграції з Jira було реалізовано м'яке зіставлення локальних статусів зі статусами та переходами зовнішньої системи. Це дозволяє зменшити кількість помилок під час синхронізації та зробити поведінку системи більш стабільною.

Модуль календаря реалізує створення зустрічей, відображення запланованих подій та синхронізацію з Google Calendar. Кожна зустріч пов'язана з проєктом, має дату, час, посилання та список учасників.

Окремо реалізовано логіку нагадувань, яка дає змогу вчасно показувати сповіщення про майбутні зустрічі. Для користувача це виглядає як звичайна робота з календарем, але на рівні серверної логіки система виконує перевірку дати, часу, списку учасників та зберігає відповідну службову інформацію.

Модуль Мітинг-пак використовується для підготовки короткого звіту по проєкту. У межах цього модуля зберігаються блокери, ризики, звіт та інші дані. Цей модуль тісно пов'язаний як і з задачами, так з ІІІ-асистентом, оскільки частина інформації може формуватись автоматично на основі актуального стану проєкту.

Модуль ресурсів реалізує роботу з ресурсами проєкту: звичайними посиланнями, документами, таблицями, Confluence та Notion сторінками та іншими матеріалами.

Особливістю реалізації цього модуля є те, що частина ресурсів відкривається не просто як зовнішнє посилання, а у вигляді окремого workspace-режиму всередині системи. Наприклад, для Confluence, Notion і

Google Docs реалізовано окремі робочі сторінки, що дозволяють переглядати, а в окремих випадках і редагувати вміст без виходу з системи.

Модуль розробки призначений для відображення інформації про стан розробки. У ньому користувач може переглядати прив'язані репозиторії, відкриті merge request або pull request, останні коміти та статус пайплайнів або робочий процес. Під час реалізації цього модуля було використано розділення логіки для GitLab і GitHub. Це дозволило не змішувати правила роботи різних сервісів і зробити обробку даних розробки більш структурованою.

Модуль III-асистента реалізований окремо, він працює з контекстом проєкту. Він може аналізувати задачі, ризики, активність, зустрічі, навантаження команди та іншу інформацію, після чого формувати короткі підсумки або рекомендації. Під час реалізації модуля було передбачено не лише генерацію текстових відповідей, а й виконання окремих дій, наприклад, створення мітінг-пак, формування чернетки задачі, збереження результату в пов'язані документи.

Центр сповіщень призначений для відображення подій, пов'язаних із задачами, коментарями, зустрічами, дедлайнами та іншими важливими змінами в системі.

Адміністративний модуль реалізує керування користувачами, ролями, статусами акаунтів, інтеграціями системи, аудитом та станом системи. Адміністратор може переглядати список користувачів, змінювати їм ролі, блокувати або розблоковувати акаунти, а також переглядати службову інформацію щодо інтеграцій і загального стану системи.

Для кожного типу інтеграції реалізовано окремий маршрут або сервісний шар, який відповідає за перевірку доступу, обмін даними, формування запитів до зовнішнього API та нормалізацію отриманих результатів. У межах системи підтримуються такі сервіси: Jira, Trello, Google Calendar, Google Docs, Google Sheets, Confluence, Notion, GitLab, GitHub.

Під час реалізації важливо було врахувати можливі помилки інтеграцій, наприклад, невалідний токен, недостатні права доступу, відключений API або

відсутність доступу до ресурсу. Для цього на сервері реалізовано окремий механізм нормалізації помилок, а на клієнтській частині дружнє відображення таких ситуацій в інтерфейсі.

Також, було створено діаграму компонентів системи вебсередовища (рис. 2.3) і визначено компоненти програмного комплексу.

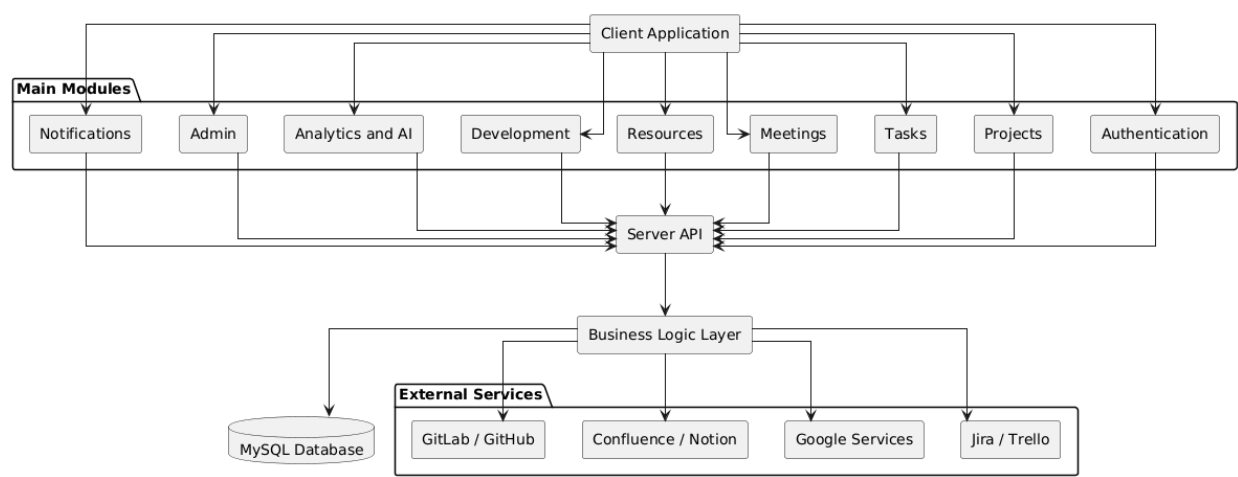


Рис. 2.3. Діаграма компонентів

На діаграмі визначені такі елементи:

Client Application - клієнтський застосунок для роботи користувача із системою.

Authentication Module - модуль авторизації, реєстрації та відновлення пароля.

Project Management Module - модуль створення, редагування та перегляду проєктів.

Task Management Module - модуль роботи із задачами та дошкою.

Meeting and Calendar Module - модуль зустрічей, календаря та нагадувань.

Resources Module - модуль роботи з ресурсами, документами та зовнішніми джерелами.

Development Module - модуль перегляду статусу розробки та репозиторіїв.

Analytics and AI Module - модуль аналітики та ШІ-асистента.

Admin Module - модуль адміністративного керування системою.

Notification Module - модуль системних сповіщень.

Server API - серверний компонент для обробки запитів клієнта.

Business Logic Layer - шар бізнес-логіки та взаємодії із зовнішніми сервісами.

MySQL - компонент збереження даних системи.

Jira/Trello - зовнішні сервіси для синхронізації задач і проєктів.

Google Services - зовнішні сервіси для календаря, документів і таблиць.

Confluence/Notion - зовнішні сервіси для роботи з документацією.

GitLab/GitHub - зовнішні сервіси для відображення активності розробки.

Висновки до другого розділу

У другому розділі було виконано проєктування вебсередовища для управління проєктами та визначено основні принципи його функціонування.

Було проаналізовано варіанти використання системи для різних ролей користувачів: адміністратора, менеджера проєкту та учасника команди. На основі цього сформовано функціональні та нефункціональні вимоги до програмного комплексу.

Також, було спроектовано об'єктно-орієнтовану структуру системи та побудовано діаграму класів, яка відображає основні сутності вебсередовища, їх властивості та взаємозв'язки.

Окрему увагу приділено проєктуванню бази даних, що забезпечує збереження користувачів, проєктів, задач, зустрічей, інтеграцій та іншої службової інформації.

У межах розділу було описано алгоритми функціонування системи та послідовність дій користувачів залежно від їх ролі.

Крім цього, наведено особливості реалізації основних модулів вебсередовища, зокрема модулів проєктів, задач, календаря, ресурсів, інтеграцій, аналітики та ІІІ-асистента.

У результаті виконаного проєктування було сформовано структуру програмного комплексу, яка забезпечує зручне управління проєктами,

підтримує командну взаємодію та дозволяє інтегрувати систему із зовнішніми сервісами.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСЕРЕДОВИЩЕМ ДЛЯ УПРАВЛІННЯ ІТ-ПРОЄКТАМИ

3.1. Порядок встановлення та налаштування параметрів вебсередовища

Для опису фізичної архітектури системи наведено за допомогою діаграми розгортання (рис. 3.1.)

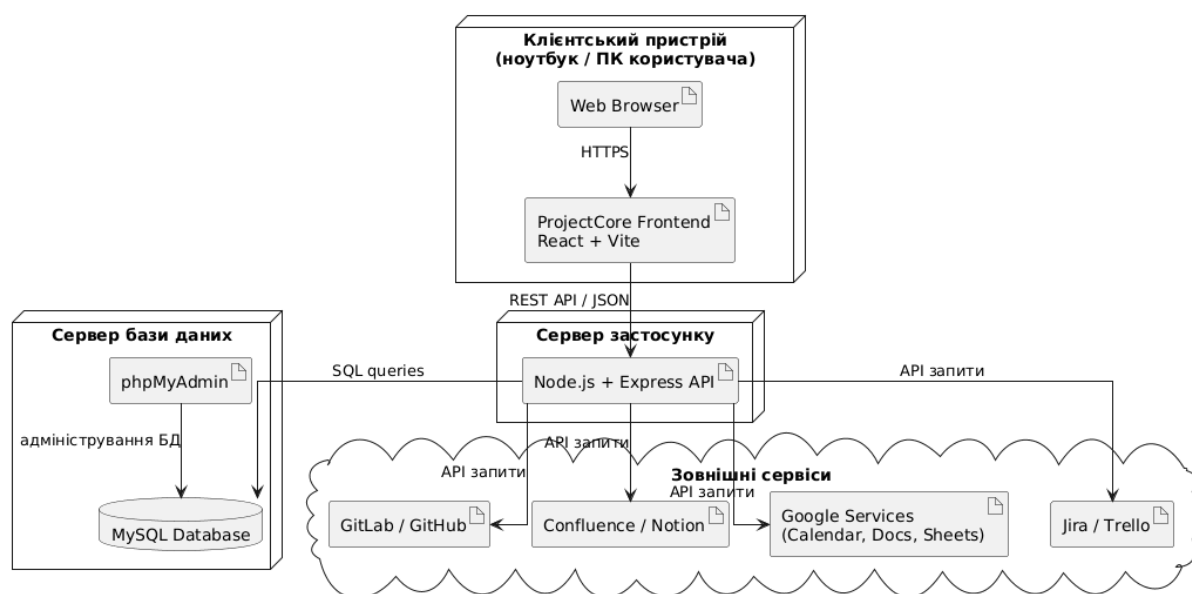


Рис. 3.1. Діаграма розгортання

Діаграма розгортання відображає на яких вузлах розміщуються основні частини системи. Таким чином між ними передаються дані та як організовано взаємодію між програмними компонентами. Завдяки такій діаграмі можна побачити не лише логічну, а й фізичну структуру програмного продукту.

Ця діаграма містить такі основні елементи:

Основні елементи клієнтської частини наведені в таблиці 3.1.

Таблиця 3.1

Опис основних елементів клієнтської частини

Веббраузер	Використовується користувачем для доступу до системи
Клієнтський вебзастосунок	Інтерфейс-частина системи, реалізована на React + Vite, через яку користувач взаємодіє

Продовження таблиці 3.1.

	з проектами, задачами, календарем, ресурсами та іншими модулями
--	---

Основні елементи серверного застосунку наведені в таблиці 3.2.

Таблиця 3.2

Опис основних елементів серверного застосунку

Node.js + Express API	Серверна частина системи, яка приймає запити від клієнта, виконує бізнес-логіку, перевіряє авторизацію, ролі користувачів та обробляє звернення до БД
Сервер застосунку	Сервер застосунку також відповідає за роботу з інтеграціям, сповіщеннями, ШІ-модулем, модулем розробки і адміністративним функціоналом

Основні елементи серверу БД наведені в таблиці 3.3.

Таблиця 3.3

Опис основних елементів серверу БД

MySQL Database	Компонент, у якому зберігаються основні дані системи: користувачі, проекти, задачі, зустрічі, ресурси, інтеграції, журнали активності та інші службові записи
phpMyAdmin	Інструмент адміністрування ДБ, який використовувався для створення таблиць, перевірки структури БД, перегляду записів і виконання SQL-запитів

Основні елементи зовнішніх сервісів наведені в таблиці 3.4.

Опис основних елементів зовнішніх сервісів

Jira/Trello	Використовуються для синхронізації задач і проєктів
Google Services	Забезпечують інтеграцію з Google Calendar, Google Docs і Google Sheets
Confluence/Notion	Використовуються для роботи з документацією та базою знань
GitLab/GitHub	Використовуються для отримання даних розробки, зокрема інформації про репозиторії, коміти, merge request і пайплайни

Користувач працює із системою через браузер, у якому відкривається клієнтський вебзастосунок. Клієнтська частина надсилає запити до серверного API, яке обробляє їх відповідно до логіки системи. Під час виконання запитів серверна частина звертається до БД для отримання або збереження інформації. У разі потреби сервер також надсилає запити до зовнішніх сервісів для синхронізації задач, календаря, документів або інформацію про розробку.

Таким чином, діаграма розгортання дає можливість наочно представити фізичну структуру програмного комплексу, показати основні вузли системи та пояснити, яким чином організовано взаємодію між клієнтською частиною, сервером, БД і зовнішніми сервісами.

3.2. Структура інтерфейсу вебсередовища для управління ІТ-проєктами

Взаємодія користувачів із вебсередовищем здійснюється через вебзастосунок. Інтерфейс розроблений з урахуванням ролі користувача та його основних потреб під час роботи із системою. У програмному комплексі передбачено три основні ролі: учасник команди, менеджер проєкту та адміністратор. Для всіх користувачів використовується спільна базова

структура інтерфейсу, до якої входять бокова панель навігації, основна робоча область, таблиці, картки, форми, модальні вікна та системні повідомлення. Водночас доступний функціонал і набір сторінок змінюються залежно від ролі користувача.

Після відкриття системи користувач потрапляє на стартову сторінку. На ній розміщено коротку інформацію про систему, її основні можливості та елементи переходу до авторизації або початку роботи із застосунком. Головна сторінка формує загальне уявлення про систему та дозволяє користувачу швидко перейти до внутрішніх модулів після входу. На рисунку 3.2. наведено стартову сторінку застосунку.

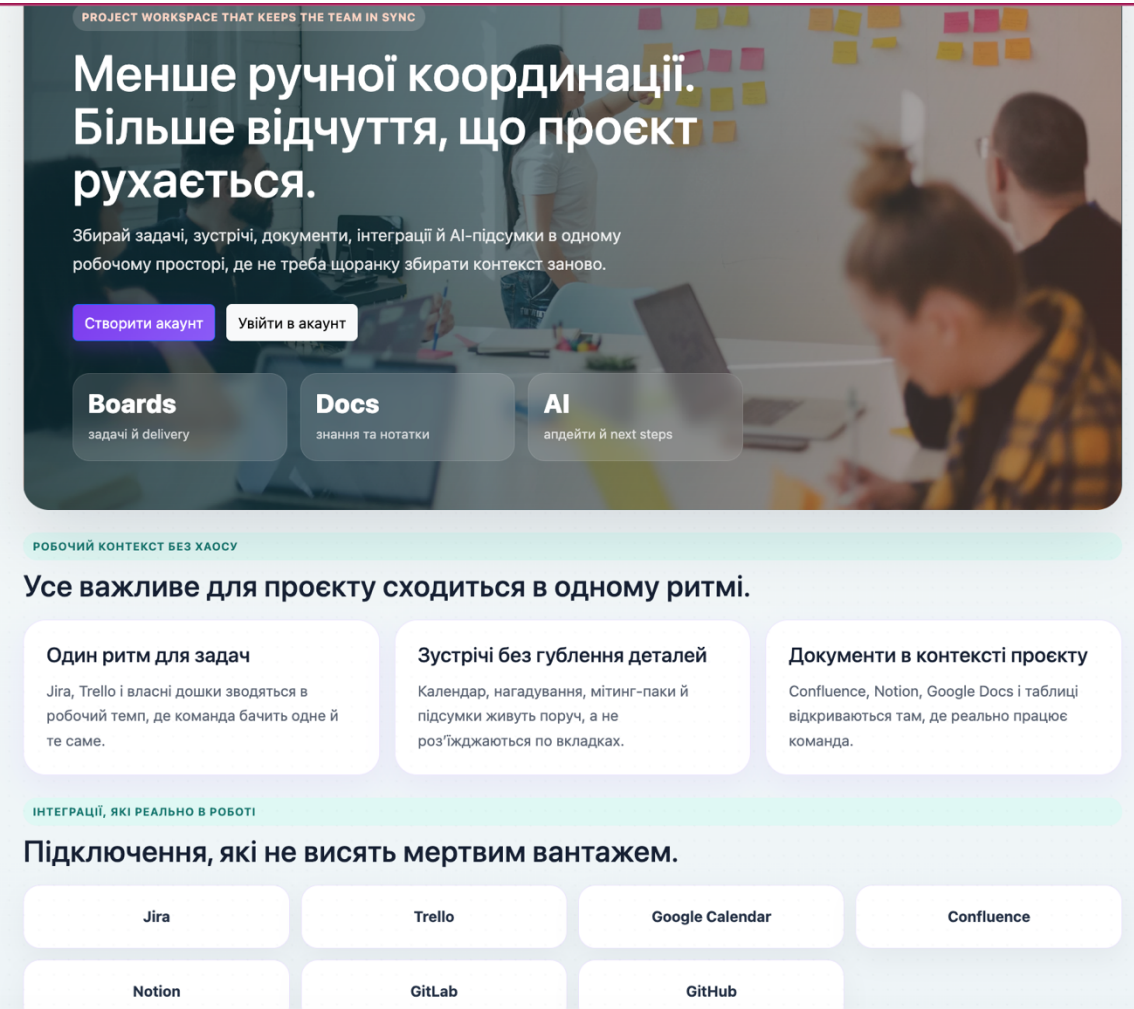


Рис. 3.2. Стартова сторінка

Для початку роботи з вебсередовищем користувач повинен пройти авторизацію. У системі реалізовано сторінки входу або реєстрації, а також сторінки відновлення паролю. Форми побудовані у зрозумілому вигляді і

містять в собі поля введення, кнопки підтвердження та повідомлення про виконання дій. На рисунках 3.3-3.5 наведено сторінку входу, реєстрації та відновлення паролю.

ProjectCore

WORKSPACE FOR STEADY PROJECT FLOW

Заходь у простір, де команда бачить одну й ту саму картину.

Менше переключень між вкладками, більше відчуття, що проект під контролем і нічого не губиться між задачами, зустрічами та документами.

Задачі та дошки

Мітинги й календар

AI та аналітика

- Одна стрічка роботи для задач, зустрічей і документів
- Jira, Trello, Google Calendar, Confluence, Notion і dev-flow поруч
- AI-апдейти й аналітика без зайвого ручного шуму

З поверненням

Увійди, щоб продовжити роботу з проектами.



Вход через аккаунт Google

або

Email

name@company.com

Пароль

Введи пароль

[Забули пароль?](#)

Увійти

Ще немає акаунта? [Створити акаунт](#)

Рис. 3.3. Модуль входу

ProjectCore

START CALM, KEEP MOMENTUM

Створи акаунт і зведи роботу команди в одне місце.

Від першої задачі до підсумку зустрічі. Без хаосу з окремими таблицями, чатами та випадковими лінками по всьому ноуту.

Швидкий старт

Живі інтеграції

Один workspace

- Створюєш акаунт і одразу потрапляєш у свій workspace
- Якщо тебе вже запрошували в проект, доступ підтягнеться автоматично
- Далі можна підключити Jira, Trello, Google і документи під свій процес

Створити акаунт

Можна зареєструватися через Google або заповнити форму нижче.



Регистрация с помощью Google

або

Ім'я

Як до тебе звертатись

Email

name@company.com

Пароль

Не менше 6 символів

Створити акаунт

Вже є акаунт? [Увійти](#)

Рис. 3.4. Модуль реєстрації

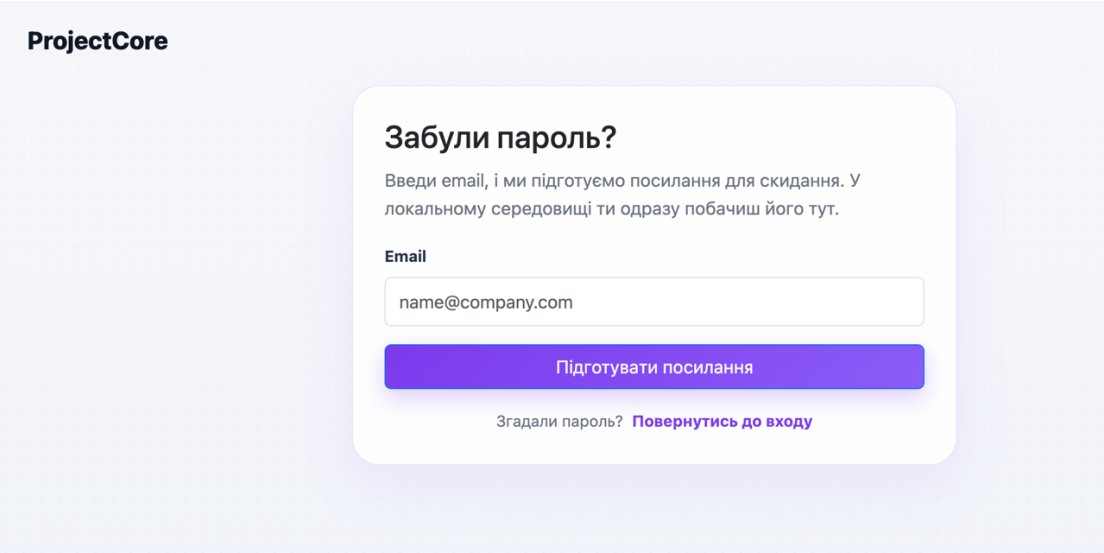


Рис. 3.5. Модуль скидання паролю

Після успішної авторизації система визначає роль користувача і відкриває відповідний набір модулів. Учасник команди потрапляє до власного робочого простору. Інтерфейс побудований таким чином, щоб користувач бачив лише ті модулі, які потрібні для повсякденної роботи. Основною сторінкою є дашборд, на якому відображається коротка інформація про поточні задачі, події, пріоритетні елементи та іншу корисну активність. Дашборд дозволяє активно зорієнтуватись в робочому стані без переходу між великою кількістю сторінок. На рис. 3.6. зображено Дашборд учасника команди

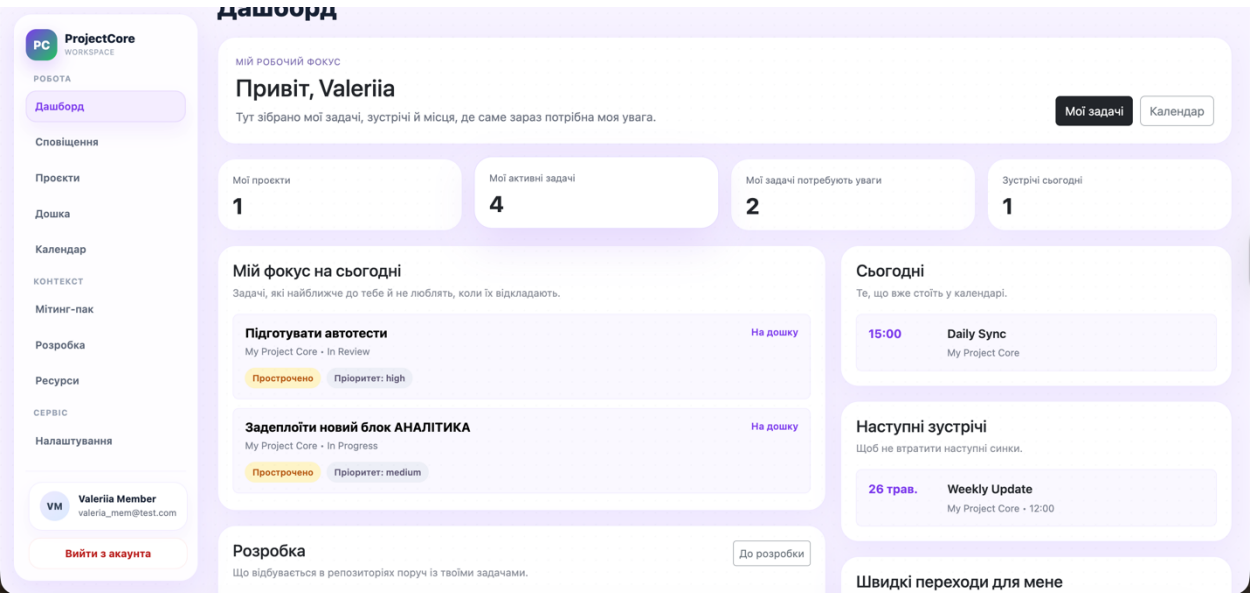


Рис. 3.6. Модуль Дашборд учасника команди

Однією з основних сторінок для користувача є сторінка проєктів. На ній відображаються лише ті проєкти, до яких користувач має доступ і відношення в житті. На цій сторінці він може переглядати загальну інформацію про проєкт та переходити до пов'язаних модулів. Для детальнішого перегляду проєкту можна використовувати модельне вікно, яке відкривається при кліку на проєкт. На рисунках 3.7-3.8 зображена сторінка Проєкти та модальне вікно проєкту.

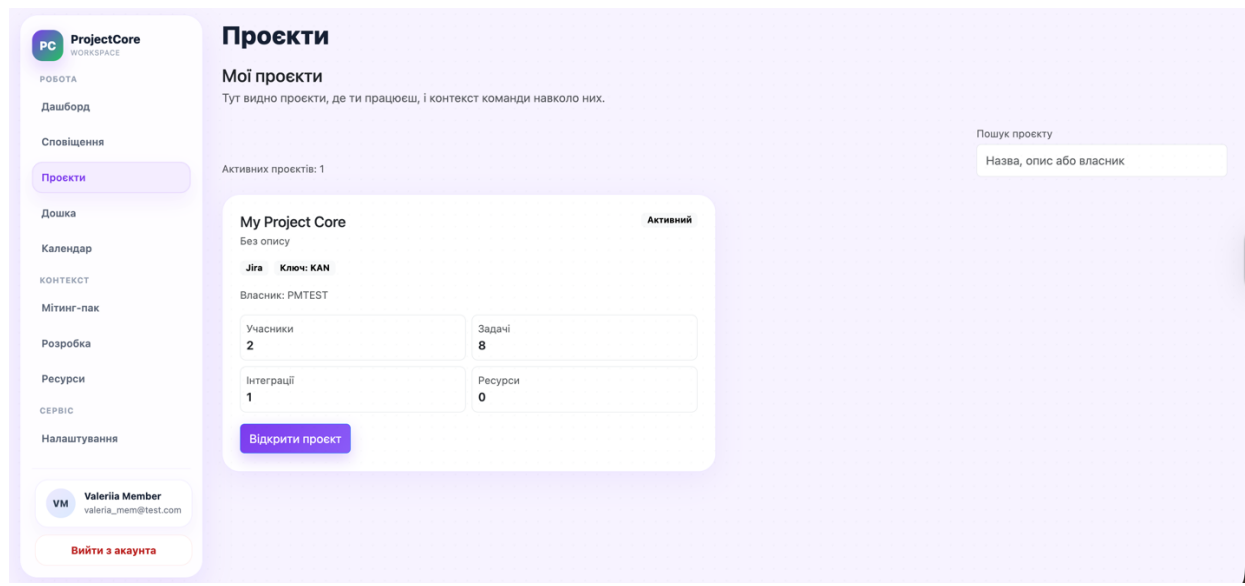


Рис. 3.7. Модуль Проєкти учасника команди

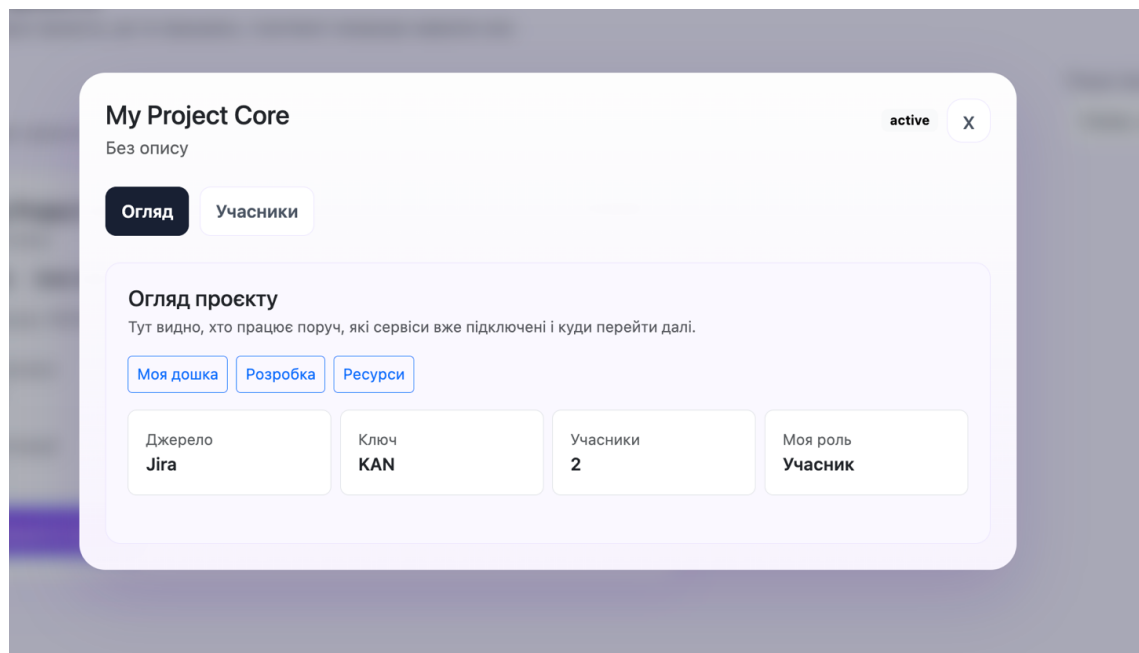


Рис. 3.8. Модальне вікно проєкту учасника команди

Користувач може відкривати окремі задачі, залишати коментарі, додавати вкладення, змінювати статуси власних задач. На рисунках 3.9-3.10 показані дошка задач і модальне вікно окремої задачі.



					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						49

інформацію про них та орієнтуватись в запланованій активності. Для цього в системі передбачено календар та модальні вікна перегляду деталей зустрічі. На рисунках 3.11-3.12 показано календар та модальне вікно окремої зустрічі.

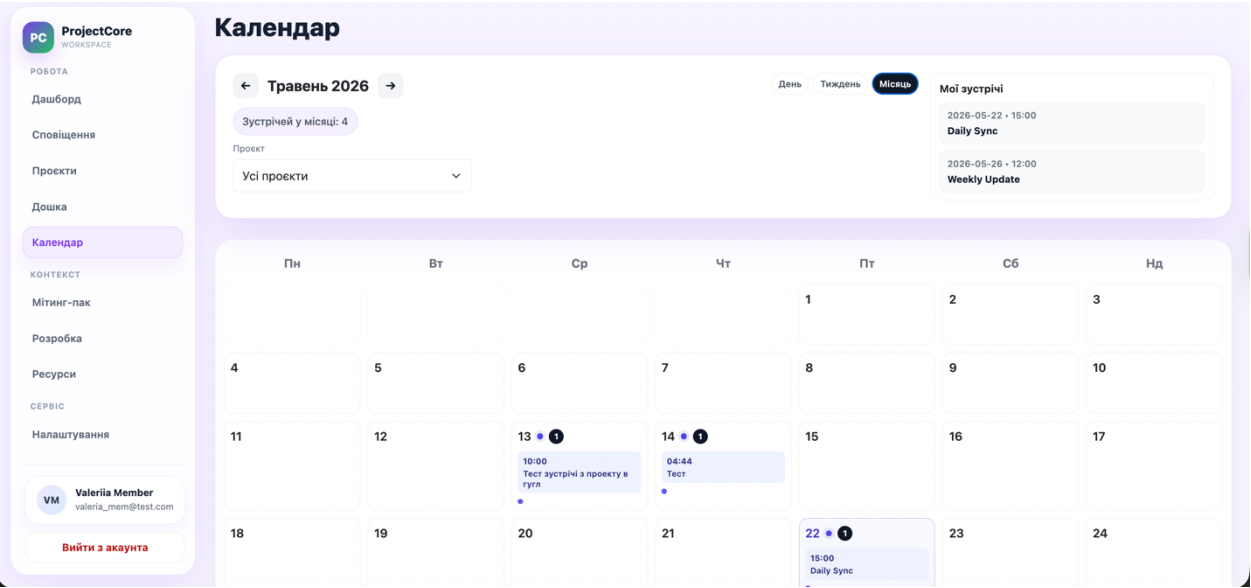


Рис. 3.11. Модуль Календар користувача

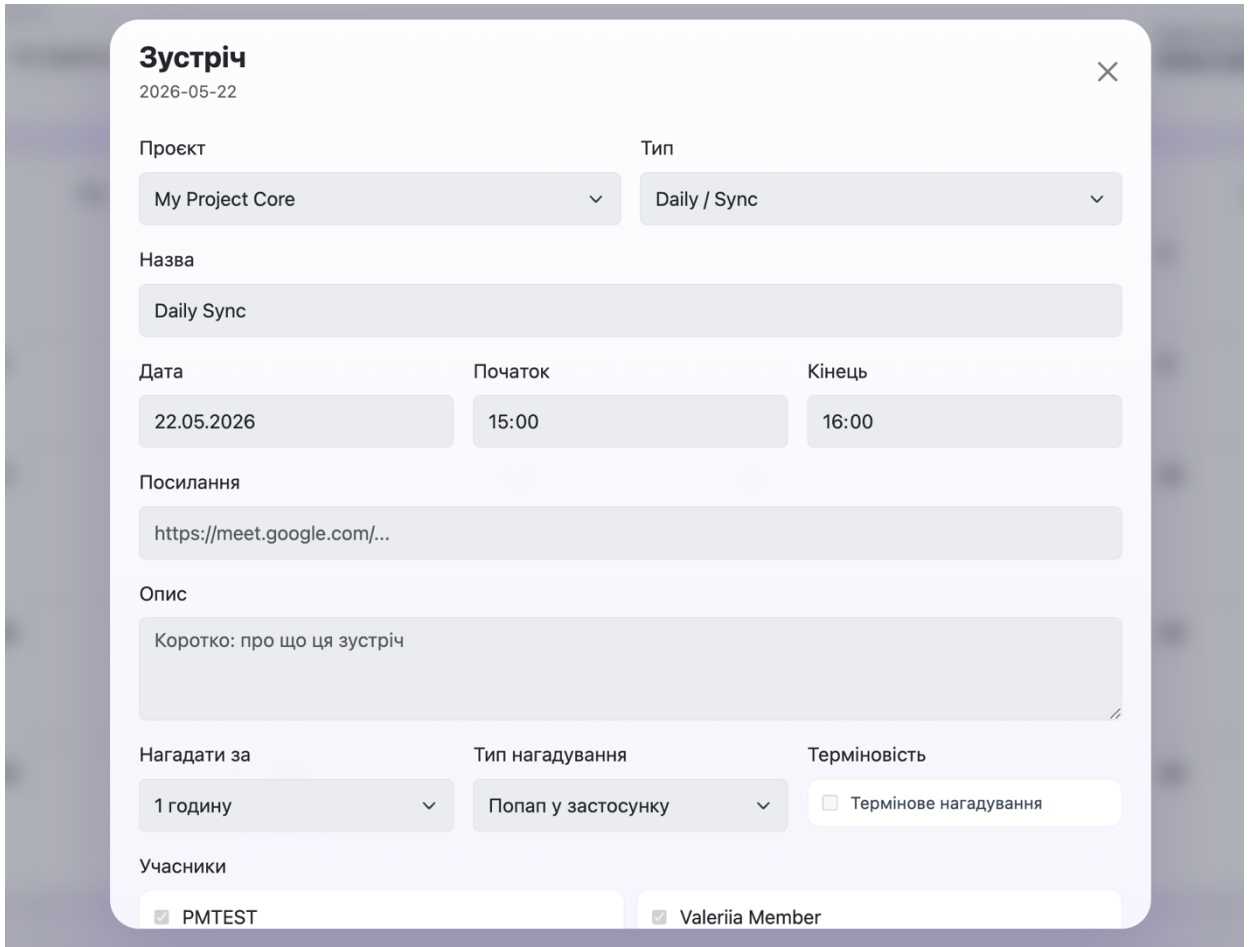


Рис. 3.12. Модальне вікно окремої зустрічі

PCProjectCoreWORKSPACE

РОБОТА

Дашборд

Сповіщення

Проекти

Дошка

Календар

КОНТЕКСТ

Мітинг-пак

Розробка

Ресурси

СЕРВІС

Налаштування

VMValerii Membervalerii_mem@test.com

Вийти з акаунта

Мій daily report

Що вже закрила

Завершено задач: Продовжити задачу: підготувати тест, перевірити створення задач.

Що мене блокує

Явних блокерів зараз не видно.

Мій фокус сьогодні

У роботі зараз: Продовжити задачу: ТЕСТ ЗМІН, Заделлойти новий блок АНАЛІТИКА.

Мої задачі

Moі done1

Moі blocked0

Moі в роботі3

Moі overdue2

На що звернути увагу

2 задачі прострочені, дедлайни вже під ризиком.

Valerii Member має 4 активних задач(и), варто розвантажити.

Стан роботи

Completed25%

Overdue25%

Blocked0%

Velocity2 задач завершено

Останні зміни в проєкті

Показано 6 з 20

Valerii Member оновив(ла) Підготувати автотести2026-05-22 10:22

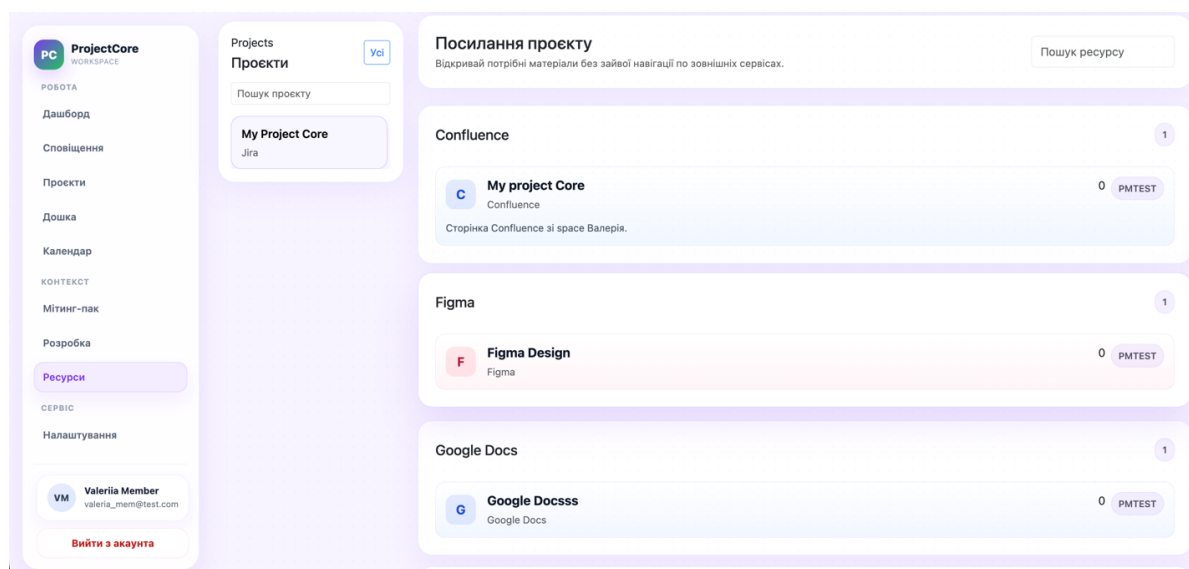
Valerii Member оновив(ла) Протестувати блок Дошка2026-05-22 10:21

PMTEST перемістив(ла) Продовжити задачу: ТЕСТ ЗМІН2026-05-20 12:33

Valerii Member додав(ла) коментар до Зробити дизайн модального вікна задач2026-05-19 18:58

PMTEST створив(ла) задачу Напиши таск для нового блоку аналітика2026-05-16 09:56

Одним із цікавих модулів є сторінка Ресурси. Саме тут кожен користувач може працювати з посиланнями, документами, таблицями і додатковими матеріалами проєкту. Частина ресурсів відкривається у вигляді окремого робочого простору без виходу із системи, що робить взаємодію більш цілісною. На рисунках 3.14-3.15 зображено сторінку ресурсів та сторінку перегляду окремого ресурсу у робочому просторі.



					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						51

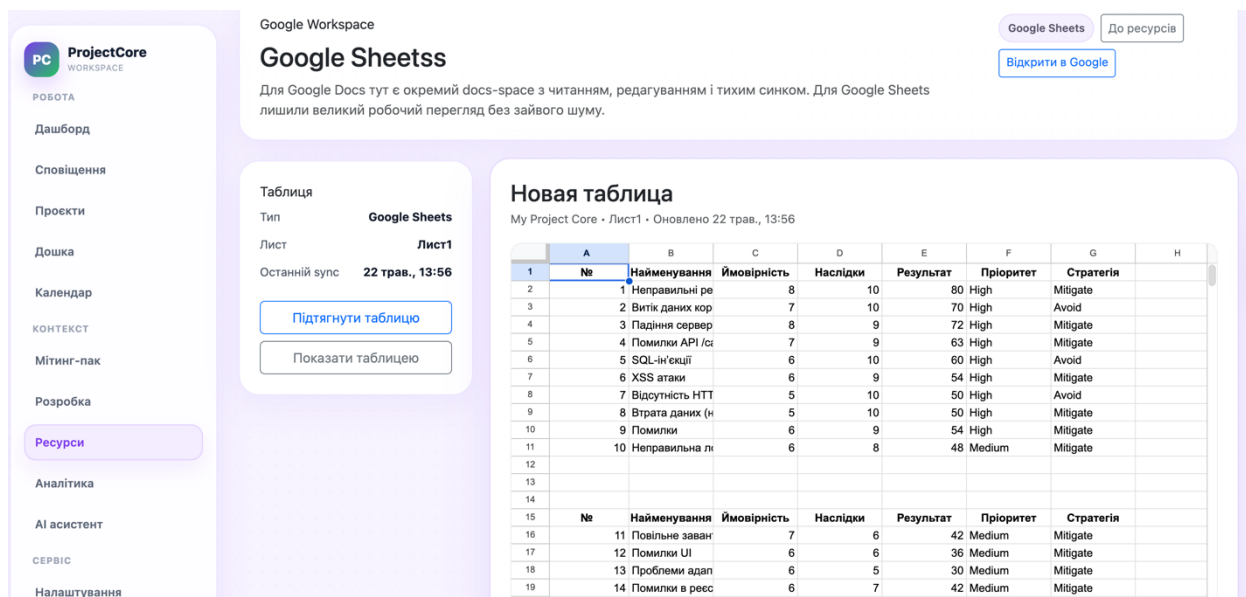


Рис. 3.15. Сторінка перегляду окремого ресурсу

Учасник команди, менеджер проєкту та адміністратор має доступ до центру сповіщень, де відображаються зміни по задачах, коментарях, зустрічах та інших подіях. Це дозволяє швидко реагувати на зміни та які елементи потребують уваги. На рисунку 3.16. зображено центр сповіщень

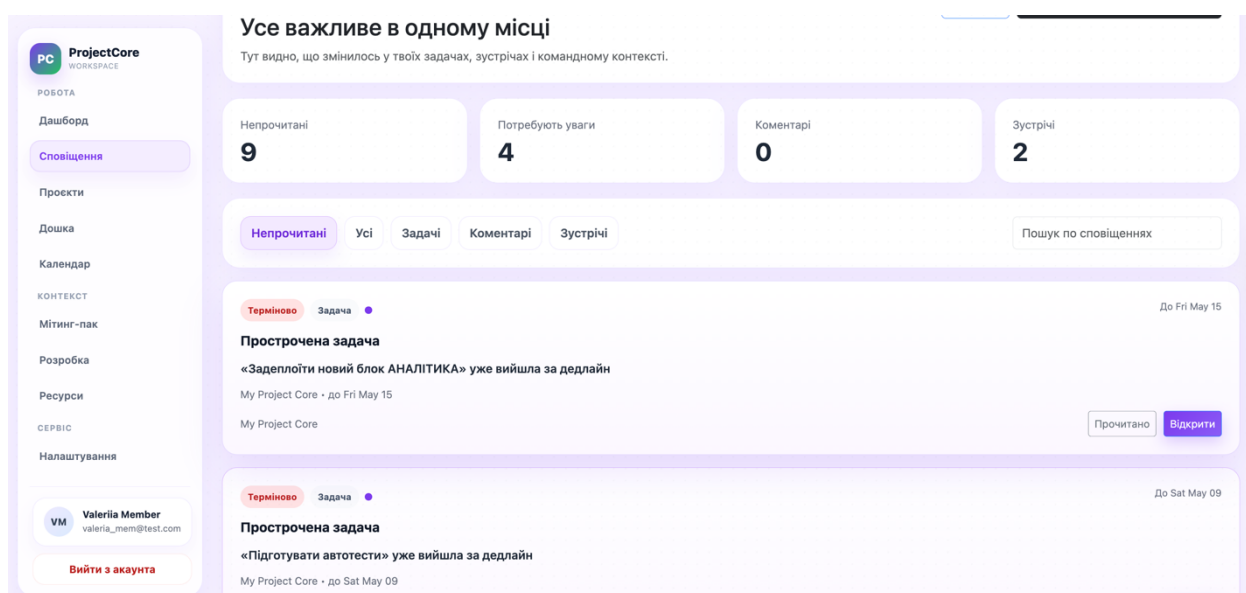


Рис. 3.16. Модуль Сповіщення

На сторінці налаштувань користувач може керувати власними інтеграціями. Тут відображаються підключені сервіси, можливість їх налаштування, а також інформація щодо підключення токенів і зовнішніх систем. На рисунках 3.17-3.18 зображено сторінку налаштувань та модальне вікно підключення інтеграцій

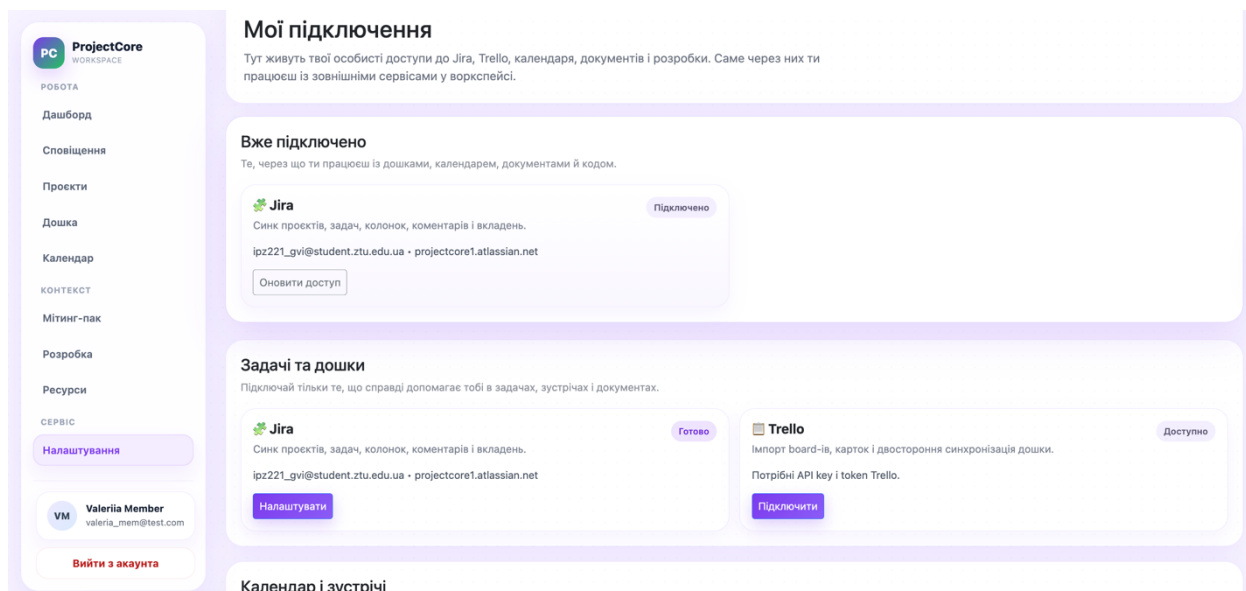


Рис. 3.17. Модуль інтеграцій

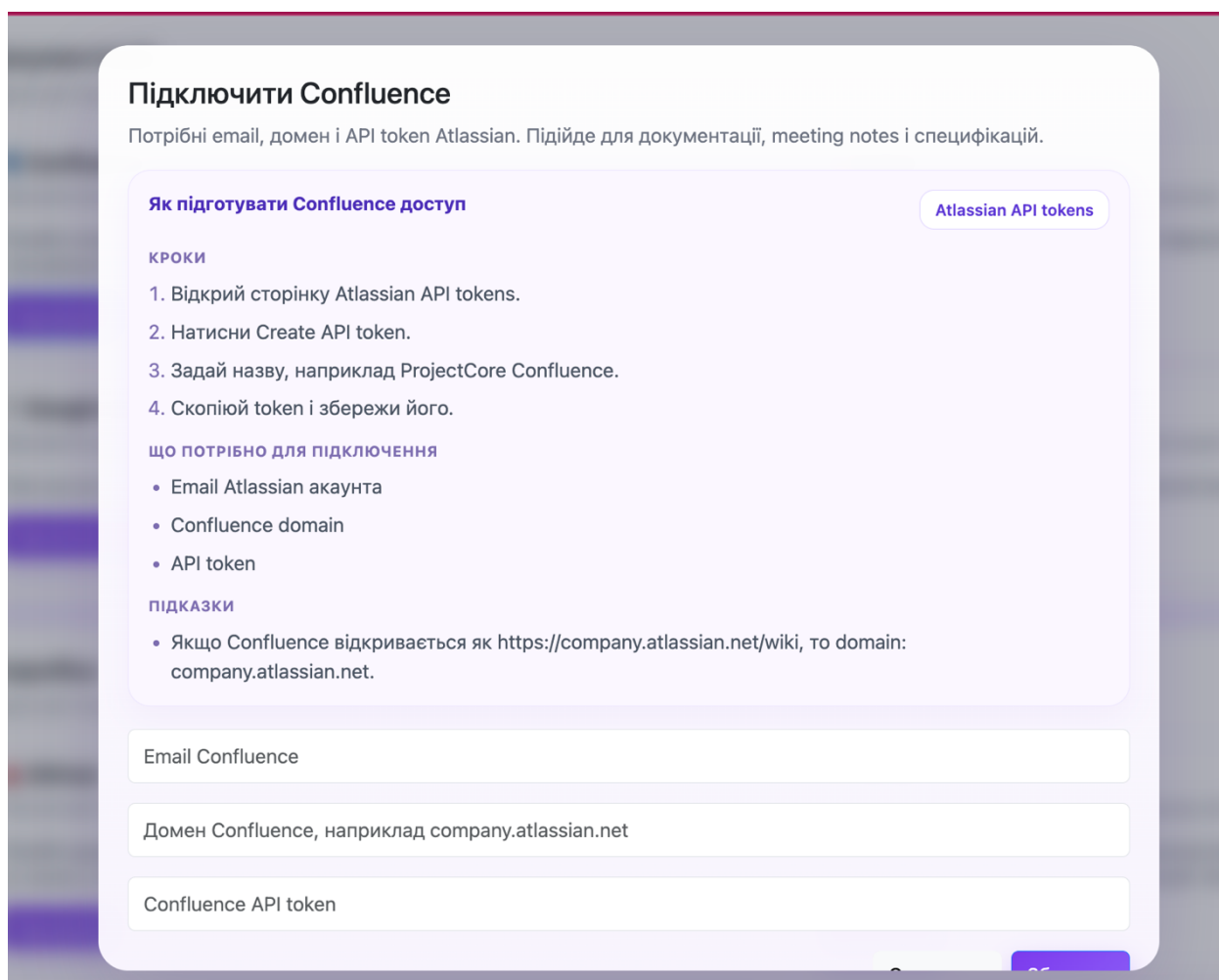


Рис. 3.18. Модальне вікно підключення інтеграцій

Перейдемо до інтерфейсу менеджера проєкту. Інтерфейс побудовано на тій самій базовій структурі сторінок що й інтерфейс учасника команди, але має значно ширший функціонал. Після входу менеджер потрапляє на власний

дашборд, де відображаються ключові показники кожного проєкту, кількість активних задач, прострочені елементи, зустрічі, навантаження та інші дані. На рисунку 3.19. показано дашборд менеджера.

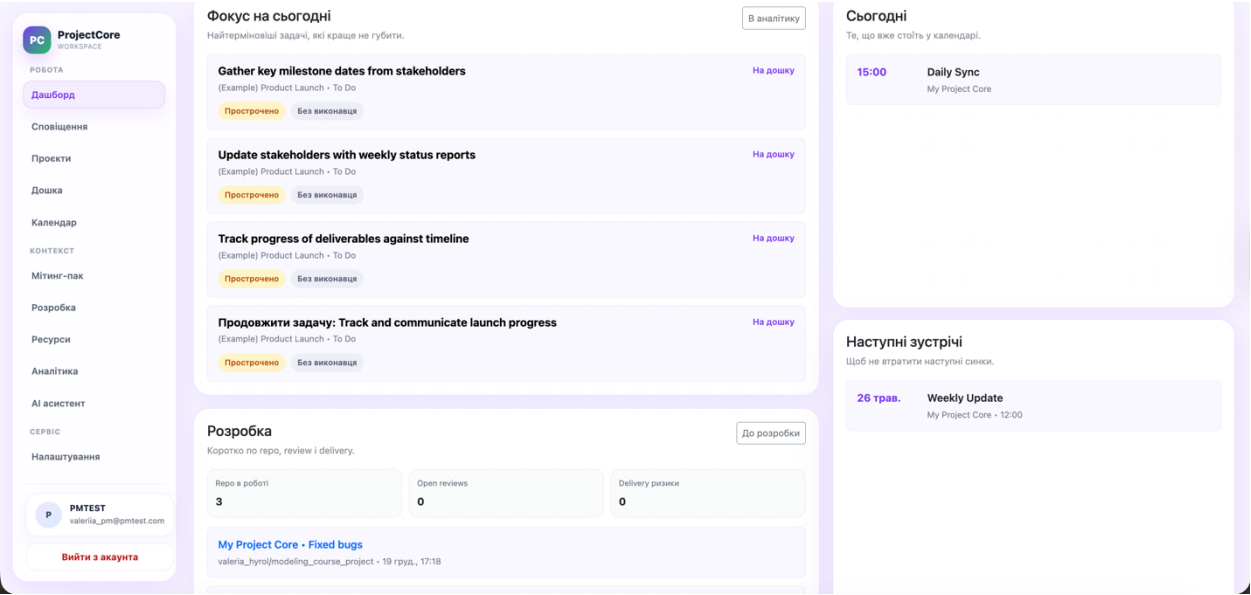


Рис. 3.19. Модуль Дашборд менеджера проєкту

Менеджер проєкту має можливість створювати нові проєкти, редагувати вже наявні, додавати опис, керувати параметрами проєкту та запрошувати учасників до команди. Усі ці дії виконуються через форми та модальні вікна, які забезпечують зручне введення й редагування даних. На рисунках 3.20-3.23 показано створення проєкту, редагування проєкту та додавання учасника.

Створити проєкт

Обери, чи створюємо власний проєкт вручну, чи імпортуємо його з підключеної Jira або Trello.

Вручну

Із Jira

Із Trello

Назва проєкту

Наприклад, Marketing Portal

Опис

Коротко опиши мету або контекст проєкту

Після створення цей проєкт одразу буде синхронізовано з Jira і з'явиться там як окремий Jira-проєкт.

Статус

Дата старту

Кінцева дата

Активний

дд.мм.рррр

дд.мм.рррр

Скасувати

Створити проєкт

Рис. 3.20. Створення проєкту

Створити проєкт

Обери, чи створюємо власний проєкт вручну, чи імпортуємо його з підключеної Jira або Trello.

Вручну

Із Jira

Із Trello

Назва проєкту

Наприклад, Marketing Portal

Опис

Коротко опиши мету або контекст проєкту

Після створення цей проєкт одразу буде синхронізовано з Jira і з'явиться там як окремий Jira-проєкт.

Статус

Дата старту

Кінцева дата

Активний

дд.мм.рррр

дд.мм.рррр

Скасувати

Створити проєкт

Рис. 3.21. Редагування проєкту

My Project Core

Без опису

active

X

Огляд

Команда

Запрошення

Додати людей у проєкт

Запроси нову людину по email або додай уже існуючого користувача.

Запросити по email

Додати з системи

Email учасника

Роль у проєкті

Надіслати запрошення

name@example.com

Учасник

Учасники проєкту

Тут можна керувати ролями й складом команди.

PMTEST

valeriia_pm@pmtest.com

Власник

Valeriia Member

valeria_mem@test.com

Учасник

Учасник

Видалити

Рис. 3.22. Запрошення учасника до проєкту

Окремо для менеджера реалізовано повне керування задачами, можна створювати задачі, редагувати їх, призначати виконавців, задавати дедлайни, працювати зі статусами та контролювати хід виконання на дошці. На рисунках 3.23-3.24 показані модальні вікна створення та редагування задачі.

×

Нова задача

Створи задачу, і вона одразу з'явиться на дошці.

Назва

Наприклад, підготувати РМ-дошку

Опис

Що саме потрібно зробити?

Статус

То Do

Пріоритет

Середній

Виконавець

Без виконавця

Дедлайн

дд.мм.rrrr

Закрити

Створити задачу

×

Напиши таск для нового блоку аналітика

Редагуй поля, коментуй і синхронізуй зміни із зовнішньою дошкою.

Назва

Напиши таск для нового блоку аналітика

Опис

Мета: напиши таск для нового блоку аналітика.

Очікуваний результат: є готовий артефакт або реалізований функціонал без поломки поточного flow.

Залежності: явних blocked залежностей зараз не видно.

Статус

Пріоритет

To Do

Високий

Виконавець

Дедлайн

Без виконавця

дд.мм.рррр

Джерело: Jira

Jira key: KAN-11

Оновлено: 2026-05-16

Теги задачі

Познач задачі темами, типами або зонами відповідальності.

Для цієї задачі ще немає тегів

Календар для менеджера проєкту має розширений характер використання, оскільки він може не лише переглядати зустрічі, а й створювати їх, задавати дату, час, учасників та посилання на подію. Це дозволяє використовувати календар як один із центральних інструментів планування. На рисунках 3.25-3.26 зображено модальне вікно створення зустрічі і відображення зустрічі в календарі.

Нова зустріч

2026-05-26

Проект

My Project Core

Тип

Daily / Sync

Назва

Дата

26.05.2026

Початок

--:--

Кінець

--:--

Посилання

https://meet.google.com/...

Опис

Коротко: про що ця зустріч

Нагадати за

1 годину

Тип нагадування

Попап у застосунку

Терміновість

☐ Термінове нагадування

Учасники

☐ PMTEST

☐ Valeriia Member

Рис. 3.25. Модальне вікно створення зустрічі

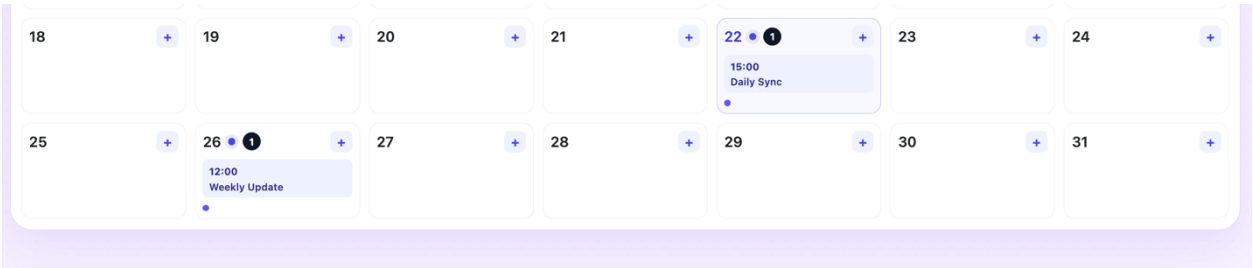


Рис. 3.26. Відображення створених зустрічей

У системі також реалізовано розділ розробки, у якому менеджер може переглядати інформацію про репозиторії, коміти, merge request або pull request, а також статуси збірок. Це дозволяє об'єднати керування проектом і технічний контекст розробки в межах одного середовища. На рисунку 3.27. зображено розділ розробки.

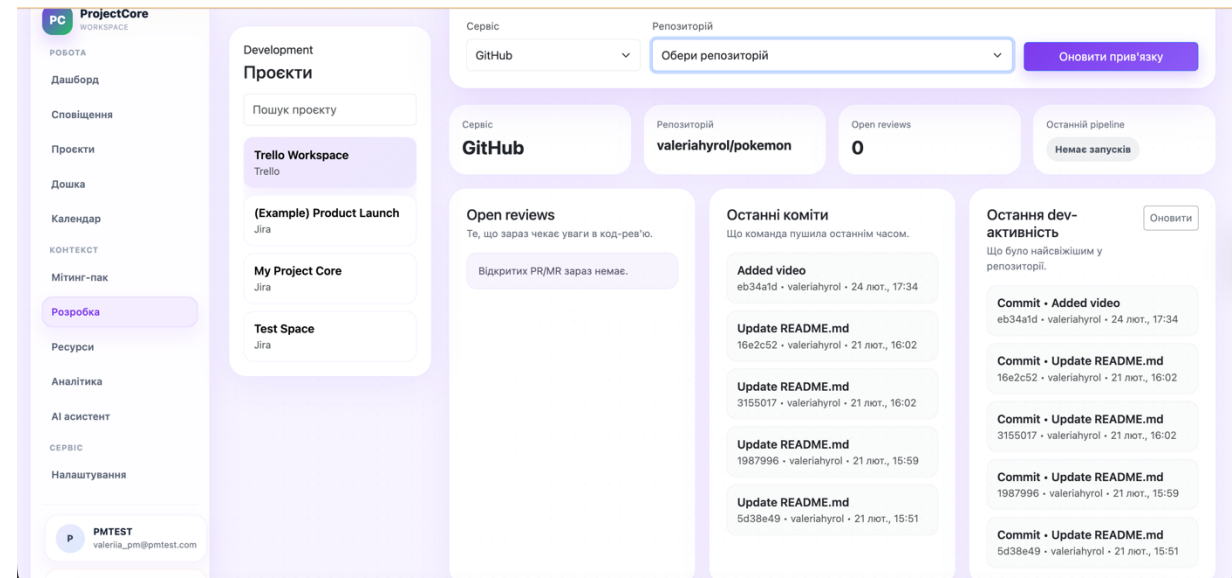


Рис. 3.27. Модуль Розробка

Для аналітичної оцінки стану проекту менеджер може використовувати окрему сторінку аналітики. На ній відображаються показники по задачах, виконанню, простроченнях, ризиках та інших аспектах роботи. Це дозволяє приймати рішення на основі узагальнених даних. На рисунку 3.28. показано сторінку аналітики.

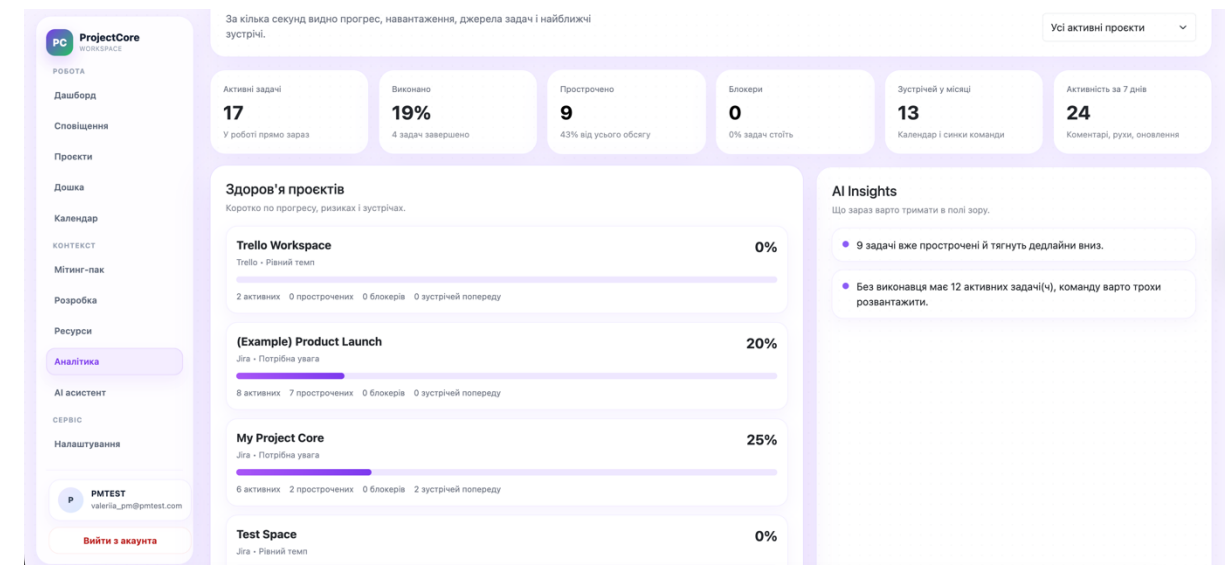


Рис. 3.28. Модуль Аналітика

PCProjectCoreWORKSPACE

РОБОТА

Дашборд

Сповіщення

Проекти

Дошка

Календар

КОНТЕКСТ

Мітинг-пак

Розробка

Ресурси

Аналітика

AI асистент

СЕРВІС

Налаштування

PMTEST
valeria_pm@pmtest.com

Розмова з асистентом

Написати питання своїми словами або стартувати з готового сценарію для my project core.

Підсумуй станДе ризикиХто переважанийДо чого готуватись сьогодніЩо робити далі

Апдейт для стейкхолдерів

До чого готуватись сьогодні

До чого готуватись сьогодні

Сьогодні в календарі 1 зустріч(ч). Паралельно в роботі 6 активних задач, тож перед синками варто пробігтись по найризиковіших пунктах.

Зустрічі: Daily Sync (15:00)

Фокус: Підготувати автотести, Задеплойти новий блок АНАЛІТИКА, Написати таск для нового блоку аналітика, Продовжити задачу: ТЕСТ ЗМІН

ЩО Я ВРАХУВАВ

проект My Project Core6 активних задач2 прострочених0 blocked1 зустрічі сьогодні

Оновити статуси перед зустрічамиПідготувати короткий апдейт по blocked і overdue задачам

Подивитися на учасників, які найбільше завантажені

Показати зустрічіЩо далі в календаріЗберегти в Meeting PackЗберегти в Google Docs

Зберегти в NotionЗберегти в Confluence

Наприклад: де ризики, дай апдейт або напиши нову задачу

Надіслати

Перейдемо до адміністрування системи. Інтерфейс адміністратора орієнтований на системне керування користувачами й платформою загалом. Однією з головних сторінок є список користувачів, де адміністратор може

переглядати акаунти, змінювати ролі, блокувати або розблоковувати доступ. На рисунках 3.31-3.32 показано сторінку керування користувачами та приклад зміни ролі.

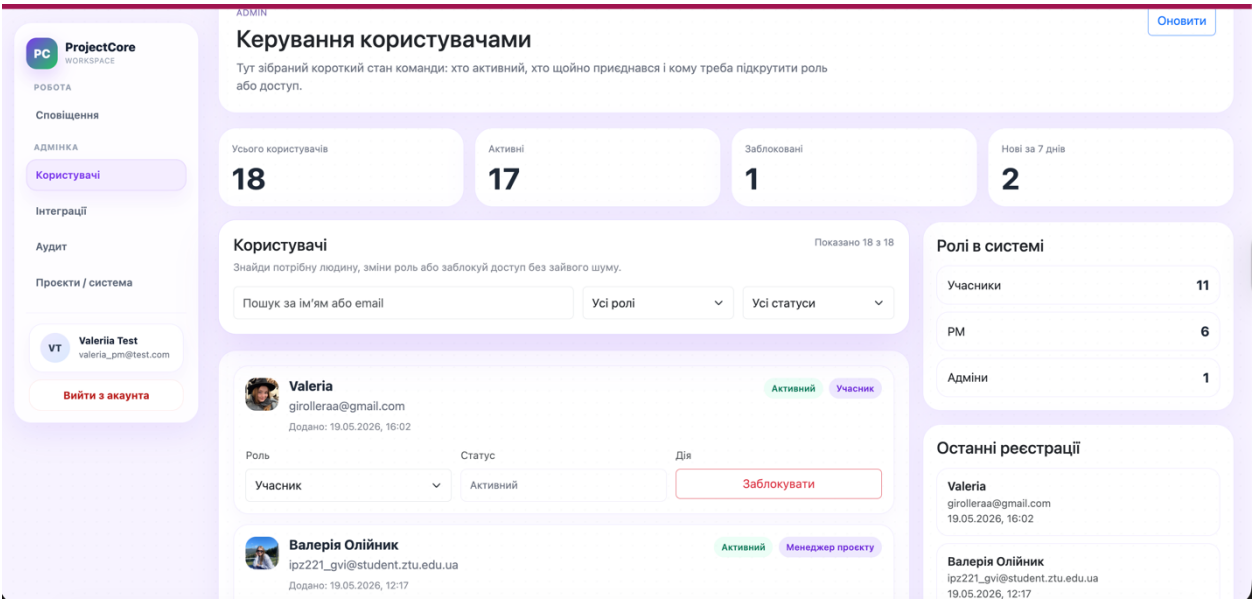


Рис. 3.31. Модуль керування користувачами

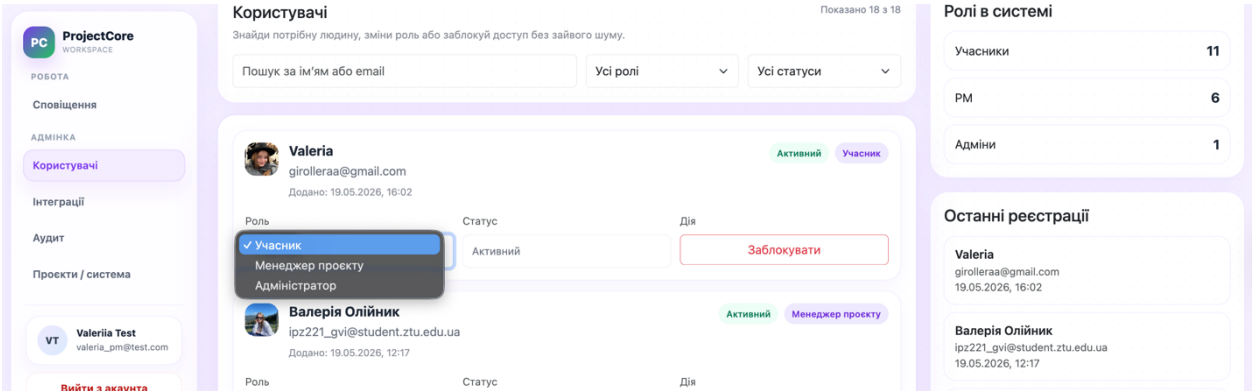


Рис. 3.32. Приклад зміни ролі користувача

Також адміністратор має окремі сторінки для перегляду інтеграцій системи, аудиту адміністративних дій і загального стану системи. Ці розділи дозволяють контролювати службову інформацію, бачити історію змін та оцінювати поточний стан роботи платформи. На рисунках 3.33-3.35 показані інтеграції системи, аудит та сторінку системного огляду.

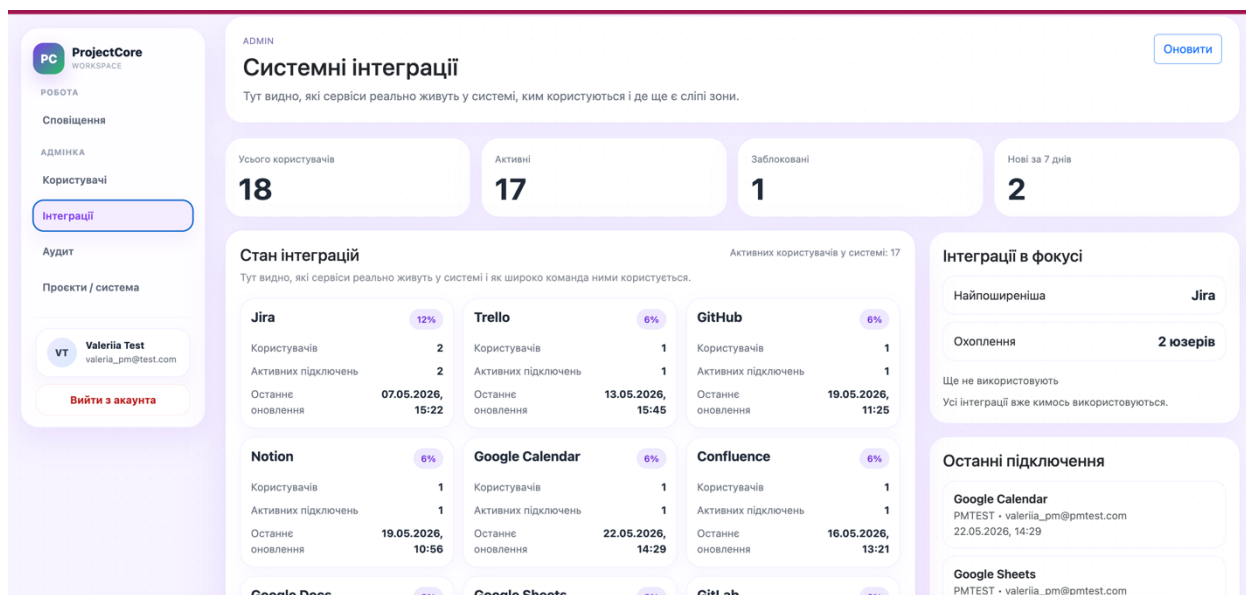


Рис. 3.33. Модуль системні інтеграції

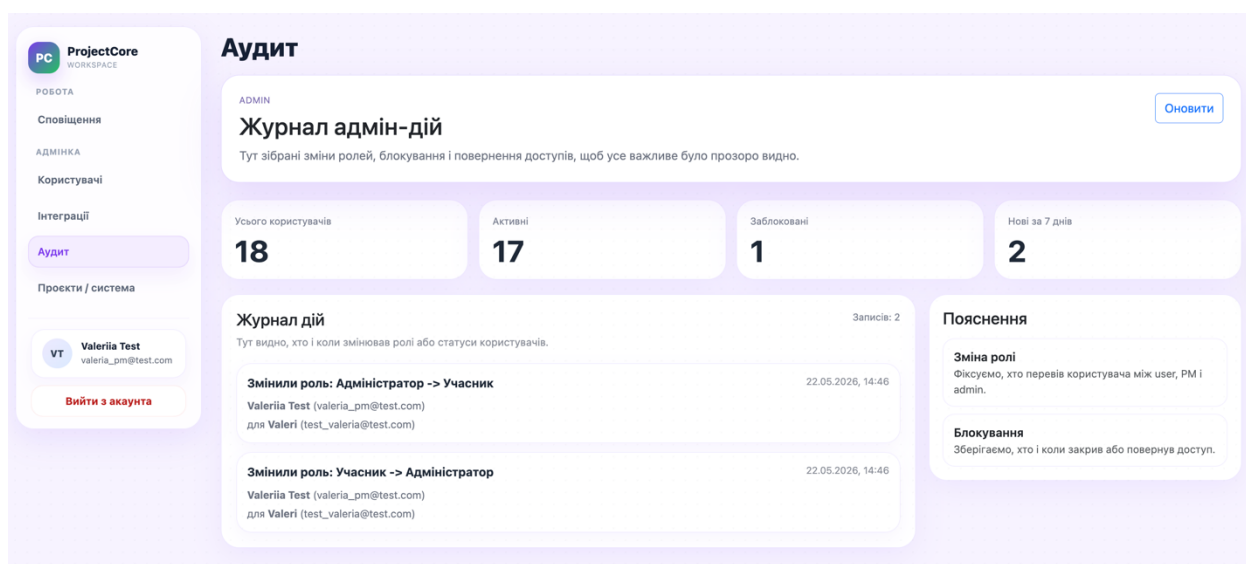


Рис. 3.34. Модуль Аудит

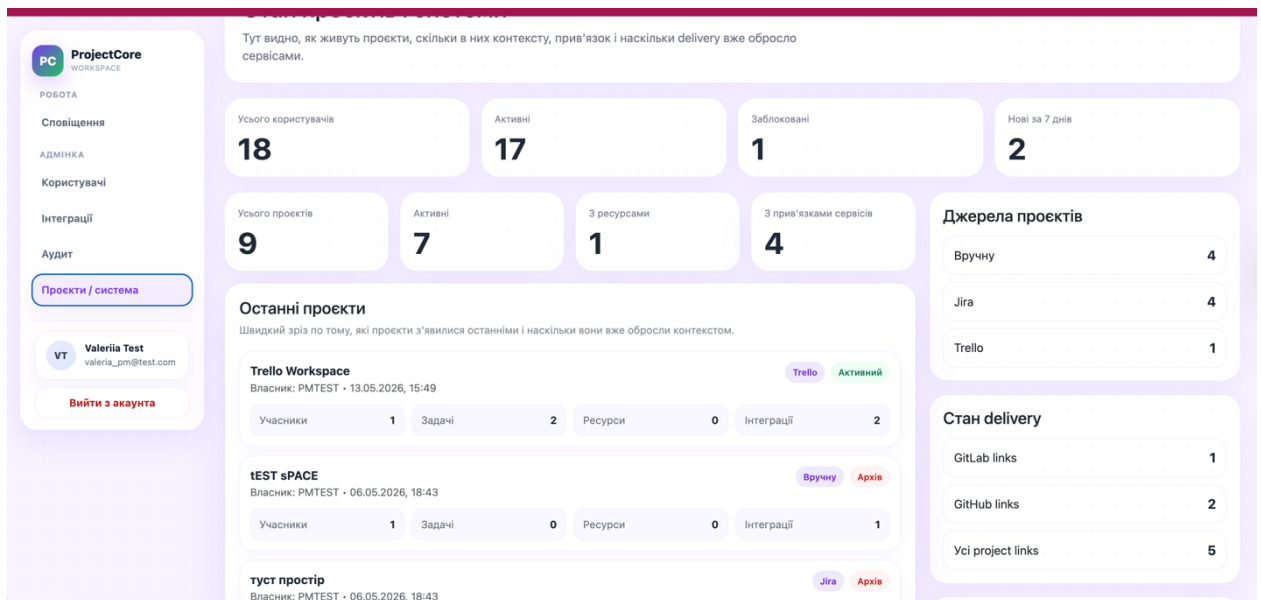


Рис. 3.35. Модуль стану системи

Отже, структура інтерфейсу вебсередовища побудована з урахуванням ролей користувачів і особливостей їх роботи. Стартова сторінка забезпечує вхідну точку для взаємодії із системою, а внутрішній інтерфейс об'єднує роботу з проєктами, задачами, зустрічами, ресурсами, інформацією про розробку, аналітикою, ШІ-функціями та адміністративними можливостями. Такий підхід дозволяє зробити роботу із системою зрозумілою, структурованою та зручною для різних категорій користувачів.

3.3. Тестування роботи програмного комплексу

Для оцінки якості клієнтської частини застосунку було використано Google Lighthouse [18]. Такий підхід дозволяє отримати узагальнені показники продуктивності, доступності, відповідності рекомендованим практикам розробки та базової SEO-оптимізації. Перевірка виконувалась окремо для мобільного та десктопного режимів відображення сторінки. На рисунках 3.36-3.37 зображено результати мобільного та десктоп тестування.

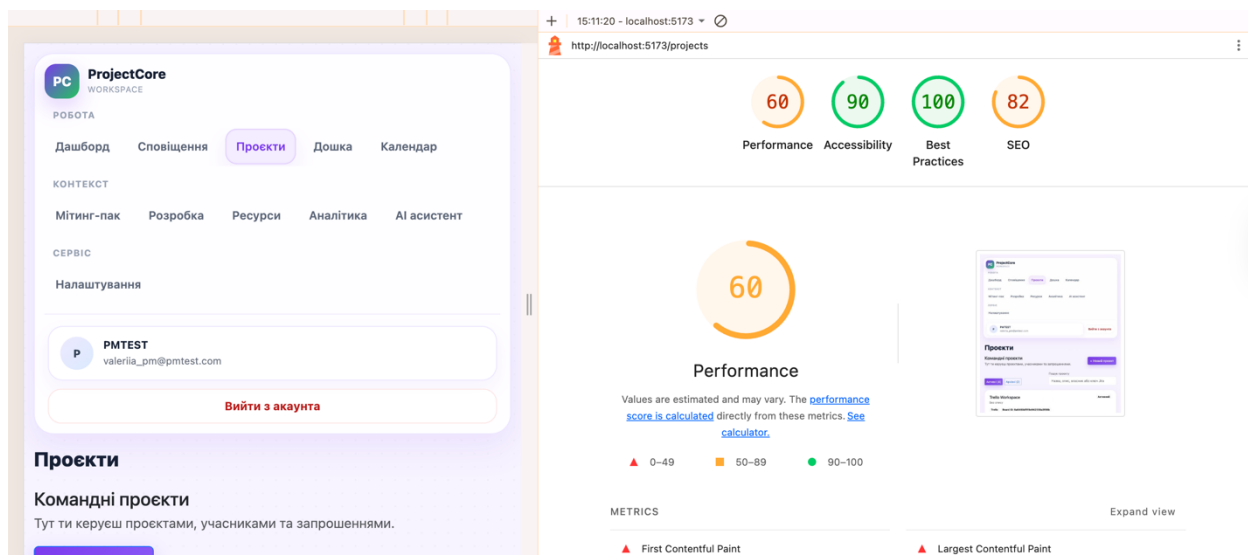


Рис. 3.36. Результат тестування десктоп версії

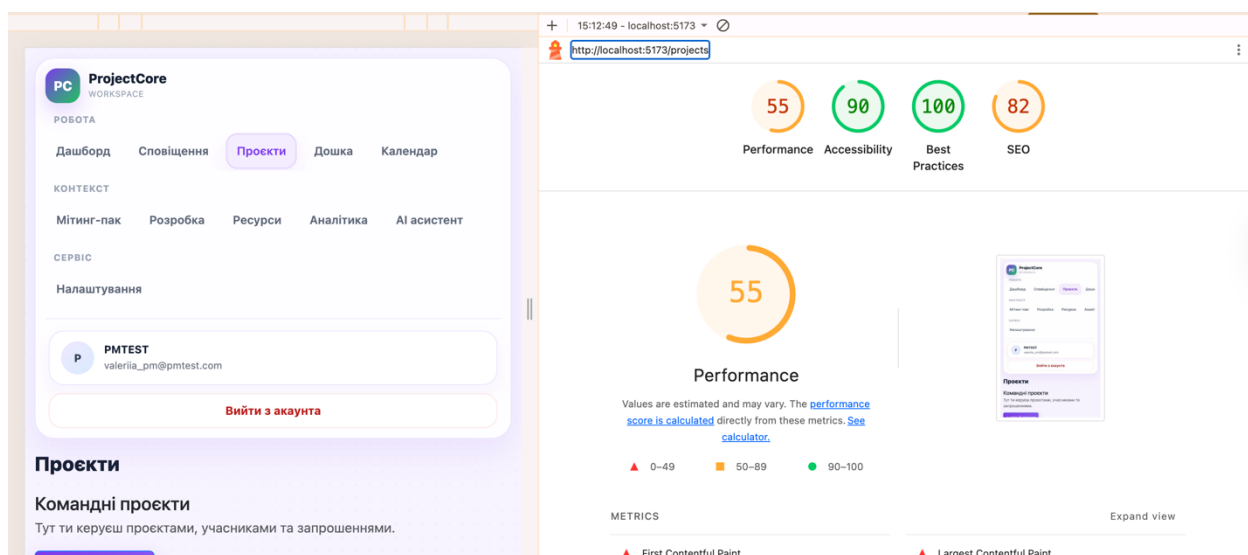


Рис. 3.37. Результат тестування мобільної версії

Отримані результати показують, що застосунок загалом відповідає сучасним вимогам веброзробки, має коректно організований інтерфейс і достатній рівень зручності для користувачів.

Разом із тим результати перевірки показали, що найбільш слабким місцем залишається продуктивність сторінки. Значення в межах 55–60 балів означає, що система працює коректно, однак потребує додаткової оптимізації швидкодії. Насамперед це стосується зменшення обсягу ресурсів, оптимізації рендерингу інтерфейсу та покращення завантаження окремих компонентів сторінки.

Для покриття всього додатку відповідним тестуванням, було складено тест-кейси і проведено детальну перевірку кожного з модулів список тест-кейсів наведено в таблиці 3.5.

Таблиця 3.5

Опис перевірки кожного модуля

Крок	Очікування	Статус
Перехід за адресою /dashboard без авторизації	Система не відкриває сторінку	Виконано
Перехід за адресою /admin/users без авторизації	Система не відкриває сторінку	Виконано
Відправка порожньої форми реєстрації	Система не дозволяє завершити реєстрацію та показує помилку	Виконано
Введення некоректного email при реєстрації	Система повідомляє про неправильний формат	Виконано
Введення зареєстрованого email при реєстрації	Система показує повідомлення про те, що користувач з таким email вже існує	Виконано
Авторизація під роллю керівника проєкту	Система відкриває робочий простір менеджера проєкту	Виконано
Створення нового проєкту з валідними даними	Новий проєкт успішно створюється і з'являється у списку	Виконано
Створення нової задачі	Задача успішно створена і відображається на дошці	Виконано

Продовження таблиці 3.5

Додавання коментаря до задачі	Коментар з'являється в картці задачі	Виконано
Перегляд коментаря учасником команди	Користувач бачить призначену йому задачу	Виконано
Створення нової зустрічі з назвою, датою та учасниками	Зустріч успішно створюється та відображається в календарі	Виконано
Авторизація під роллю адміністратора	Система відкриває адміністративний інтерфейс	Виконано
Зміна ролі користувача	Нова роль успішно зберігається	Виконано
Блокування користувача	Статус акаунта змінюється на заблокований	Виконано
Спроба створити зустріч під роллю користувача в календарі	Система не надає доступу до створення зустрічі	Виконано

Отже, результати ручного тестування та перевірки продуктивності показали, що вебсередовище коректно реалізує основні функціональні можливості та забезпечує стабільну роботу ключових модулів системи. Проведені тест-кейси підтвердили правильну роботу механізмів авторизації, розмежування прав доступу, керування проектами, задачами, зустрічами, ресурсами, сповіщеннями та адміністративним функціоналом. Додатково перевірка за допомогою Lighthouse показала, що застосунок має хороший рівень доступності та загалом відповідає сучасним практикам веброзробки. Водночас результати аналізу вказують на можливість подальшого покращення продуктивності окремих сторінок, особливо тих, які містять велику кількість

інтерфейсних елементів або даних. Загалом результати тестування підтверджують, що вебсередовище є працездатною, зручною у використанні та має достатній рівень якості для подальшого використання й розвитку.

Висновки до третього розділу

У цьому розділі результати ручного тестування та перевірки продуктивності показали, що вебсередовище коректно реалізує основні функціональні можливості та забезпечує стабільну роботу ключових модулів системи. Проведені тест-кейси підтвердили правильну роботу механізмів авторизації, розмежування прав доступу, керування проєктами, задачами, зустрічами, ресурсами, сповіщеннями та адміністративним функціоналом.

Додатково перевірка за допомогою Lighthouse показала, що застосунок має хороший рівень доступності та загалом відповідає сучасним практикам веброзробки. Водночас результати аналізу вказують на можливість подальшого покращення продуктивності окремих сторінок, особливо тих, які містять велику кількість інтерфейсних елементів або даних.

Загалом результати тестування підтверджують, що вебсередовище є працездатною, зручною у використанні та має достатній рівень якості для подальшого використання й розвитку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано вебсередовище для управління IT-проектами ProjectCore Workspace. Розроблена система призначена для організації командної роботи, управління задачами, контролю виконання проєктів та взаємодії із зовнішніми сервісами.

Під час виконання роботи було проведено аналіз предметної області та досліджено сучасні системи управління проєктами, зокрема Trello, Jira, Worksection і ClickUp. Це дозволило визначити основні функціональні вимоги до майбутньої системи, а також виділити підходи та можливості, які доцільно реалізувати у власному програмному рішенні.

У процесі проєктування було сформовано архітектуру програмного комплексу, визначено структуру клієнтської та серверної частин, а також спроектовано базу даних для збереження інформації про користувачів, проєкти, задачі, зустрічі, ресурси та інтеграції. Для реалізації системи використано сучасні вебтехнології: React, Node.js та Express і MySQL. У межах роботи реалізовано систему авторизації та розмежування прав доступу на основі ролей користувачів.

У системі передбачено окремі можливості для адміністратора, менеджера проєкту та учасника команди. Користувачі можуть створювати проєкти, працювати із задачами, використовувати календар і зустрічі, взаємодіяти через коментарі та сповіщення, а також працювати з документацією та ресурсами проєкту.

Окрему увагу було приділено реалізації інтеграцій із зовнішніми сервісами. У системі підтримується робота з Jira, Trello, Google Calendar, Google Docs, Google Sheets, Confluence, Notion, GitLab та GitHub. Завдяки цьому вдалося об'єднати управління задачами, роботу з документацією, календарями та активністю розробки в межах одного вебсередовища. Також у системі реалізовано модуль аналітики та ШІ-асистент, який дозволяє формувати короткі підсумки проєктів, аналізувати ризики, створювати meeting

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						68

раск та допомагати користувачам у виконанні окремих робочих процесів. Використання такого функціоналу підвищує ефективність роботи команди та спрощує управління проєктами.

У ході виконання роботи було розроблено структуру інтерфейсу вебзастосунку та описано взаємодію користувачів із системою. Інтерфейс побудовано з урахуванням сучасних вимог до зручності використання, що забезпечує швидкий доступ до основних функцій і комфортну роботу з проєктами, задачами, календарем, ресурсами та аналітикою.

Крім цього, було проведено тестування системи та перевірку продуктивності. Результати тестування підтвердили коректну роботу основних функціональних модулів, а також достатній рівень якості реалізованого вебсередовища.

Отже, у результаті виконання кваліфікаційної роботи було створено сучасне вебсередовище для управління ІТ-проєктами, яке поєднує зручний інтерфейс, гнучке управління задачами, підтримку командної взаємодії та інтеграцію із зовнішніми сервісами. Розроблена система може слугувати основою для подальшого розвитку та розширення функціональних можливостей у сфері управління ІТ-проєктами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Trello. Trello [Електронний ресурс]. URL: <https://trello.com/>
2. Atlassian. Jira Agile Project Management Software [Електронний ресурс]. URL: <https://www.atlassian.com/software/jira/agile>
3. Worksection. Worksection [Електронний ресурс]. URL: <https://worksection.com/hello/>
4. Capterra. ClickUp Reviews 2026 [Електронний ресурс]. URL: <https://www.capterra.co.uk/software/158833/clickup>
5. PlantUML. Official Documentation [Електронний ресурс]. URL: <https://plantuml.com/>
6. React Team. React Documentation [Електронний ресурс]. URL: <https://react.dev/>
7. React Router Team. React Router Official Documentation [Електронний ресурс]. URL: <https://reactrouter.com/>
8. Node.js Foundation. Node.js Documentation [Електронний ресурс]. URL: <https://nodejs.org/en/docs/>
9. Express.js. Express – Node.js Web Application Framework [Електронний ресурс]. URL: <https://expressjs.com/>
10. Bootstrap Team. Bootstrap Documentation [Електронний ресурс]. URL: <https://getbootstrap.com/>
11. Google for Developers. Using OAuth 2.0 to Access Google APIs [Електронний ресурс]. URL: <https://developers.google.com/identity/protocols/oauth2>
12. GitHub Docs. GitHub REST API Documentation [Електронний ресурс]. URL: <https://docs.github.com/en/rest?apiVersion=latest>
13. GitLab Docs. REST API. [Електронний ресурс] URL: <https://docs.gitlab.com/api/rest/>
14. Atlassian Developer. The Jira Cloud Platform REST API [Електронний ресурс]. URL: <https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/>

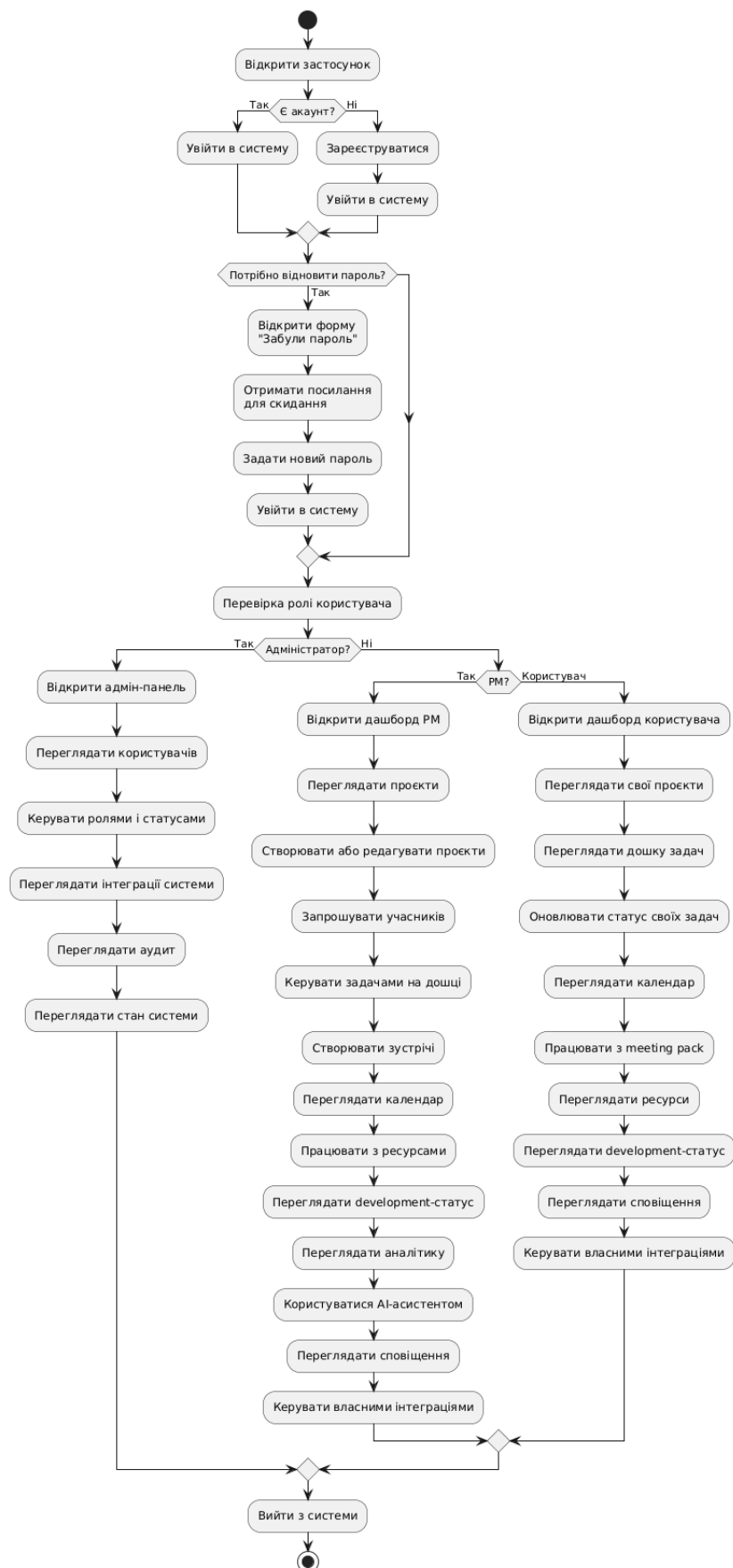
15. Atlassian Support. Get started with Trello's REST API [Электронный ресурс]. URL: <https://support.atlassian.com/trello/docs/getting-started-with-trello-rest-api/>
16. Atlassian Developer. The Confluence Cloud REST API [Электронный ресурс]. URL: <https://developer.atlassian.com/cloud/confluence/rest/v1/intro/>
17. Notion Developers. Getting Started with the Notion API [Электронный ресурс]. URL: <https://developers.notion.com/guides/get-started>
18. Chrome for Developers. Introduction to Lighthouse [Электронный ресурс]. URL: <https://developer.chrome.google.cn/docs/lighthouse/overview/>
19. Google for Developers. Display the Sign in with Google Button. [Электронный ресурс] URL: <https://developers.google.com/identity/gsi/web/guides/display-button>
20. Google for Developers. Google Sheets API Overview [Электронный ресурс]. URL: <https://developers.google.com/sheets/api/guides/concepts>
21. Google for Developers. Google Docs API: Document [Электронный ресурс]. URL: <https://developers.google.com/docs/api/concepts/document>
22. Google for Developers. Events: list – Google Calendar API [Электронный ресурс]. URL: <https://developers.google.com/calendar/api/v3/reference/events/list>
23. GitLab Docs. Get Started with GitLab CI/CD. [Электронный ресурс] URL: <https://docs.gitlab.com/ci/>
24. Notion Developers. Working with Databases. [Электронный ресурс] URL: <https://developers.notion.com/guides/data-apis/working-with-databases>
25. Microsoft Playwright. Playwright Documentation. [Электронный ресурс] URL: <https://playwright.dev/>
26. Google for Developers. How User Authorization Works. [Электронный ресурс] URL: <https://developers.google.com/identity/oauth2/web/guides/how-user-authz-works>

27. GitHub Docs. About the REST API. [Електронний ресурс] URL:
<https://docs.github.com/en/rest/about-the-rest-api/about-the-rest-api>
28. Методичні матеріали до виконання кваліфікаційної роботи [Електронний ресурс]. URL:
https://learn.ztu.edu.ua/pluginfile.php/409083/mod_folder/content/0/%D0%9F%D0%9F05%20%D0%9A%D0%B2%D0%B0%D0%BB%D1%96%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0%20%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0.pdf
29. Vite Team. Getting Started | Vite [Електронний ресурс]. URL:
<https://vite.dev/guide/>
30. Oracle. MySQL Documentation [Електронний ресурс]. URL:
<https://dev.mysql.com/doc/>

ДОДАТКИ

					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						73

Діаграма активності вебсередовища для управління ІТ-проектами



Лістинг реалізації маршрутизації та обмеження доступу на клієнтській частині

```
import { useEffect, useState } from "react";
import { BrowserRouter, Routes, Route, Navigate } from "react-router-dom";
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/dist/js/bootstrap.bundle.min.js';

import LandingPage from "../pages/LandingPage";
import LoginPage from "../pages/LoginPage";
import RegisterPage from "../pages/RegisterPage";
import ForgotPasswordPage from "../pages/ForgotPasswordPage";
import ResetPasswordPage from "../pages/ResetPasswordPage";
import Dashboard from "../pages/Dashboard";
import Projects from "../pages/Projects";
import Boards from "../pages/Boards";
import MeetingPack from "../pages/MeetingPack";
import Development from "../pages/Development";
import Resources from "../pages/Resources";
import ProjectResources from "../pages/ProjectResources";
import ConfluenceWorkspace from "../pages/ConfluenceWorkspace";
import GoogleWorkspaceResource from "../pages/GoogleWorkspaceResource";
import NotionWorkspace from "../pages/NotionWorkspace";
import Analytics from "../pages/Analytics";
import Calendar from "../pages/Calendar";
import AIAssistant from "../pages/AIAssistant.jsx";
import Settings from "../pages/Settings.jsx";
import AdminPage from "../pages/AdminPage.jsx";
import NotificationsPage from "../pages/NotificationsPage.jsx";
import SystemFeedbackHost from "../components/SystemFeedbackHost.jsx";
import { clearStoredAuth, getRoleHome, getStoredToken, getStoredUser } from
"./config/access.js";
import { readJsonResponse } from "../config/http.js";
import { showSystemError, showSystemMessage } from
"./config/systemFeedback.js";

function App() {

  const [token, setToken] = useState(() => getStoredToken());
  const [user, setUser] = useState(() => getStoredUser());
  const [authChecked, setAuthChecked] = useState(() => !getStoredToken());
  const role = user?.role;
  const homePath = token ? getRoleHome(role) : "/login";

  useEffect(() => {
    const syncAuthFromStorage = () => {
      setToken(getStoredToken());
      setUser(getStoredUser());
      setAuthChecked(!getStoredToken());
    };

    window.addEventListener("auth-changed", syncAuthFromStorage);

    return () => {
      window.removeEventListener("auth-changed", syncAuthFromStorage);
    };
  }, []);
```

```

useEffect(() => {
  let cancelled = false;

  if (!token) {
    return undefined;
  }

  fetch("http://localhost:5001/api/auth/me", {
    headers: {
      Authorization: `Bearer ${token}`
    }
  })
    .then(async (response) => {
      const data = await readJsonResponse(response);

      if (!response.ok) {
        const err = new Error(data?.message || "Не вдалося
перевірити сесію");
        err.status = response.status;
        throw err;
      }

      if (cancelled) {
        return;
      }

      localStorage.setItem("user", JSON.stringify(data));
      setUser(data);
      setAuthChecked(true);
    })
    .catch((err) => {
      if (cancelled) {
        return;
      }

      if (err.status === 401 || err.status === 403) {
        clearStoredAuth();
        setToken(null);
        setUser(null);
      } else {
        showSystemMessage({
          type: "info",
          title: "Сервер недоступний",
          message: "Ми не змогли перевірити сесію"
        });
      }

      setAuthChecked(true);
    });

  return () => {
    cancelled = true;
  };
}, [token]);

useEffect(() => {
  const handleUnhandledRejection = (event) => {
    if (!event.reason) {
      return;
    }

    showSystemError(
      event.reason,
      "Несподівана помилка",

```

```

        "Одна з дій завершилась з помилкою. Спробуй ще раз."
    );
};

const handleWindowError = (event) => {
    if (!event.error && !event.message) {
        return;
    }

    showSystemError(
        event.error || event.message,
        "Помилка в інтерфейсі",
        "Сторінка спіткнулась на неочікуваній помилці."
    );
};

window.addEventListener("unhandledrejection",
handleUnhandledRejection);
window.addEventListener("error", handleWindowError);

return () => {
    window.removeEventListener("unhandledrejection",
handleUnhandledRejection);
    window.removeEventListener("error", handleWindowError);
};
}, []);

const guardRoute = (element, allowedRoles) => {
    if (!authChecked) {
        return null;
    }

    if (!token) {
        return <Navigate to="/login" replace />;
    }

    if (!allowedRoles.includes(role)) {
        return <Navigate to={homePath} replace />;
    }

    return element;
};

return(
    <BrowserRouter>

        <SystemFeedbackHost />

        {!authChecked ? null : (

            <Routes>

                <Route path="/" element={<LandingPage/>} />
                <Route path="/login" element={token ? <Navigate to={homePath}
replace /> : <LoginPage/>} />
                <Route path="/register" element={token ? <Navigate
to={homePath} replace /> : <RegisterPage/>} />
                <Route path="/forgot-password" element={token ? <Navigate
to={homePath} replace /> : <ForgotPasswordPage/>} />
                <Route path="/reset-password" element={<ResetPasswordPage/>}
/>

                <Route path="/dashboard" element={guardRoute(<Dashboard />,
["pm", "user"])} />

```

```

        <Route path="/projects" element={guardRoute(<Projects />,
["pm", "user"])} />
        <Route path="/boards" element={guardRoute(<Boards />, ["pm",
"user"])} />
        <Route path="/meeting-pack" element={guardRoute(<MeetingPack
/>, ["pm", "user"])} />
        <Route path="/development" element={guardRoute(<Development
/>, ["pm", "user"])} />
        <Route path="/development/:id"
element={guardRoute(<Development />, ["pm", "user"])} />
        <Route path="/resources" element={guardRoute(<Resources />,
["pm", "user"])} />
        <Route path="/resources/:id"
element={guardRoute(<ProjectResources />, ["pm", "user"])} />
        <Route path="/resources/:id/confluence/:resourceId"
element={guardRoute(<ConfluenceWorkspace />, ["pm", "user"])} />
        <Route path="/resources/:id/google/:resourceId"
element={guardRoute(<GoogleWorkspaceResource />, ["pm", "user"])} />
        <Route path="/resources/:id/notion/:resourceId"
element={guardRoute(<NotionWorkspace />, ["pm", "user"])} />
        <Route path="/calendar" element={guardRoute(<Calendar />,
["pm", "user"])} />
        <Route path="/notifications"
element={guardRoute(<NotificationsPage />, ["pm", "user", "admin"])} />
        <Route path="/settings" element={guardRoute(<Settings />,
["pm", "user"])} />
        <Route path="/analytics" element={guardRoute(<Analytics />,
["pm"])} />
        <Route path="/aiassistant" element={guardRoute(<AIAssistant
/>, ["pm"])} />
        <Route path="/integrations" element={<Navigate to="/settings"
replace />} />
        <Route path="/admin" element={<Navigate to="/admin/users"
replace />} />
        <Route path="/admin/users" element={guardRoute(<AdminPage
section="users" />, ["admin"])} />
        <Route path="/admin/integrations"
element={guardRoute(<AdminPage section="integrations" />, ["admin"])} />
        <Route path="/admin/audit" element={guardRoute(<AdminPage
section="audit" />, ["admin"])} />
        <Route path="/admin/projects" element={guardRoute(<AdminPage
section="projects" />, ["admin"])} />
        <Route path="*" element={<Navigate to={token ? homePath :
"/"} replace />} />
    </Routes>

    )}

</BrowserRouter>

);
}

export default App;

```

Лістинг реалізації middleware авторизації та перевірки прав доступу

```

import jwt from "jsonwebtoken";
import { db } from "../config/db.js";

const JWT_SECRET = process.env.JWT_SECRET || "secret123";

export const requireAuth = async (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith("Bearer ")) {
    return res.status(401).json({
      message: "Потрібна авторизація"
    });
  }

  const token = authHeader.split(" ")[1];

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    const [users] = await db.promise().query(
      "SELECT id, name, email, role, status FROM users WHERE id = ?
LIMIT 1",
      [decoded.id]
    );

    if (!users.length) {
      return res.status(401).json({
        message: "Користувача не знайдено"
      });
    }

    if (users[0].status === "blocked") {
      return res.status(403).json({
        message: "Акаунт заблоковано"
      });
    }

    req.user = users[0];
    next();
  } catch {
    return res.status(401).json({
      message: "Недійсний токен"
    });
  }
};

export const requireRole = (...roles) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({
        message: "Потрібна авторизація"
      });
    }

    if (!roles.includes(req.user.role)) {
      return res.status(403).json({
        message: "Недостатньо прав доступу"
      });
    }

    next();
  };
};

```

```

};

const getIdFromRequest = (req) => {
  return (
    req.params.id ||
    req.params.project_id ||
    req.body?.project_id ||
    req.query?.project_id ||
    null
  );
};

export const requireProjectAccess = async (req, res, next) => {
  if (!req.user) {
    return res.status(401).json({
      message: "Потрібна авторизація"
    });
  }

  if (req.user.role === "admin") {
    return next();
  }

  const projectId = getIdFromRequest(req);

  if (!projectId) {
    return res.status(400).json({
      message: "Не вказано проект"
    });
  }

  try {
    const [projects] = await db.promise().query(
      "SELECT id, owner_id FROM projects WHERE id=?",
      [projectId]
    );

    if (!projects.length) {
      return res.status(404).json({
        message: "Проект не знайдено"
      });
    }

    if (projects[0].owner_id === req.user.id) {
      req.project = projects[0];
      return next();
    }

    const [members] = await db.promise().query(
      "SELECT id, project_role FROM project_members WHERE project_id=? AND user_id=?",
      [projectId, req.user.id]
    );

    if (!members.length) {
      return res.status(403).json({
        message: "У вас немає доступу до цього проекту"
      });
    }

    req.project = projects[0];
    req.projectMember = members[0];
    next();
  } catch (err) {

```

```

        console.error("PROJECT ACCESS ERROR:", err);
        return res.status(500).json({
            message: "Не вдалося перевірити доступ до проекту"
        });
    }
};

export const requireProjectRole = (...roles) => {
    return (req, res, next) => {

        if (!req.user) {
            return res.status(401).json({
                message: "Потрібна авторизація"
            });
        }

        if (req.user.role === "admin") {
            return next();
        }

        if (req.project?.owner_id === req.user.id) {
            return next();
        }

        if (!req.projectMember) {
            return res.status(403).json({
                message: "Немає доступу до проекту"
            });
        }

        if (!roles.includes(req.projectMember.project_role)) {
            return res.status(403).json({
                message: "Недостатньо прав"
            });
        }

        next();
    };
};

```

Лістинг реалізації інтеграції з Jira

```

const translateJiraErrorMessage = (message) => {
  const normalized = String(message || "").trim();

  if (!normalized) {
    return "Помилка Jira API";
  }

  if (normalized.includes("事务不存在或者您没有查看的权限")) {
    return "Jira не дозволяє доступ до цієї задачі або проекту для цього акаунта."
  }

  return normalized;
};

const extractJiraErrorText = (data) => {
  const errorMessages = Array.isArray(data?.errorMessages) ?
data.errorMessages.filter(Boolean) : [];
  const fieldErrors = data?.errors && typeof data.errors === "object"
    ? Object.values(data.errors).filter(Boolean)
    : [];

  return [...errorMessages, ...fieldErrors].join(", ");
};

const isBenignJiraSyncError = (message) => {
  const normalized = String(message || "").toLowerCase();

  return normalized.includes("already a member of the project role")
    || normalized.includes("you cannot edit the configuration of this
project");
};

const normalizeStatusValue = (value) => String(value || "")
  .trim()
  .toLowerCase()
  .replace(/[_-]+/g, " ")
  .replace(/[^p{L}\p{N}\s]+/gu, " ")
  .replace(/\s+/g, " ")
  .trim();

const getStatusGroup = (value) => {
  const normalized = normalizeStatusValue(value);

  if (!normalized) {
    return "";
  }

  const groups = [
    {
      name: "todo",
      keywords: [
        "todo", "to do", "open", "new", "backlog", "queue", "queued",
        "черга", "до виконання", "нова", "новий", "відкрита",
        "відкрито", "беклог"
      ]
    },
    {
      name: "in_progress",

```

```

        keywords: [
            "in progress", "progress", "doing", "development", "dev",
"work in progress",
            "started", "start progress", "start", "active",
            "у роботі", "в роботі", "робота", "в процесі", "виконується",
"активна"
        ]
    },
    {
        name: "review",
        keywords: [
            "review", "code review", "in review", "on review",
            "на рев ю", "на рев'ю", "рев ю", "рев'ю", "перевірка",
"перегляд"
        ]
    },
    {
        name: "qa",
        keywords: [
            "qa", "testing", "test", "verification", "verify", "uat",
            "на qa", "qa перевірка", "тест", "тестування", "перевірка qa"
        ]
    },
    {
        name: "blocked",
        keywords: [
            "blocked", "block", "on hold", "hold",
            "заблок", "блокер", "на паузі", "пауза"
        ]
    },
    {
        name: "done",
        keywords: [
            "done", "complete", "completed", "closed", "resolved",
"finished",
            "ready", "success", "released",
            "заверш", "готово", "закрит", "виконано"
        ]
    }
];

for (const group of groups) {
    if (group.keywords.some((keyword) => normalized.includes(keyword))) {
        return group.name;
    }
}

return "";
};

const getTransitionScore = (targetStatusName, transition) => {
    const targetNormalized = normalizeStatusValue(targetStatusName);
    const targetGroup = getStatusGroup(targetStatusName);
    const transitionName = normalizeStatusValue(transition?.name);
    const transitionToName = normalizeStatusValue(transition?.to?.name);
    const transitionToGroup = getStatusGroup(transition?.to?.name);
    const category = String(transition?.to?.statusCategory?.key ||
    "").toLowerCase();

    if (!targetNormalized) {
        return 0;
    }

    if (transitionToName === targetNormalized) {

```

```

        return 100;
    }

    if (transitionName === targetNormalized) {
        return 95;
    }

    if (transitionToName && (transitionToName.includes(targetNormalized) ||
targetNormalized.includes(transitionToName))) {
        return 88;
    }

    if (transitionName && (transitionName.includes(targetNormalized) ||
targetNormalized.includes(transitionName))) {
        return 82;
    }

    if (targetGroup && transitionToGroup && targetGroup ===
transitionToGroup) {
        let score = 70;

        if (targetGroup === "done" && category === "done") {
            score += 8;
        }

        if (targetGroup === "todo" && category === "new") {
            score += 8;
        }

        if (targetGroup === "in_progress" && category === "indeterminate") {
            score += 8;
        }

        return score;
    }

    if (targetGroup === "done" && category === "done") {
        return 58;
    }

    if (targetGroup === "todo" && category === "new") {
        return 58;
    }

    if ((targetGroup === "in_progress" || targetGroup === "review" ||
targetGroup === "qa") && category === "indeterminate") {
        return 48;
    }

    return 0;
};

export const getJiraClient = (integration) => {
    const email = integration.email;
    const domain = integration.domain;

    if (!email || !domain) {
        throw new Error("Jira integration is missing email or domain");
    }

    const token = Buffer
        .from(`${email}:${integration.access_token}`)
        .toString("base64");

```

```

const baseUrl = `https://${domain}`;

const request = async (url, options = {}) => {
  const hasCustomContentType = Object.keys(options.headers || {}).some(
    (key) => key.toLowerCase() === "content-type"
  );
  const isFormDataBody = typeof FormData !== "undefined" &&
options.body instanceof FormData;

  const res = await fetch(baseUrl + url, {
    ...options,
    headers: {
      Authorization: `Basic ${token}`,
      Accept: "application/json",
      ...(isFormDataBody && !hasCustomContentType ? { "Content-
Type": "application/json" } : {}),
      ...(options.headers || {})
    }
  });

  const raw = await res.text();
  let data = null;

  try {
    data = raw ? JSON.parse(raw) : null;
  } catch {
    data = raw || null;
  }

  if (!res.ok) {
    const extractedMessage = extractJiraErrorText(data) ||
data?.message || "Jira API error";

    if (!isBenignJiraSyncError(extractedMessage)) {
      console.log("JIRA ERROR RAW:", data);
    }

    throw new Error(
      translateJiraErrorMessage(
        extractedMessage ||
        "Jira API error"
      )
    );
  }

  return data;
};

const toAdfDocument = (text = "") => ({
  type: "doc",
  version: 1,
  content: [
    {
      type: "paragraph",
      content: text
        ? [{ type: "text", text }]
        : []
    }
  ]
});

const fromAdf = (node) => {
  if (!node) {
    return "";
  }

```

```

    }

    if (typeof node === "string") {
        return node;
    }

    if (Array.isArray(node)) {
        return node.map(fromAdf).join("");
    }

    if (node.type === "text") {
        return node.text || "";
    }

    if (node.type === "hardBreak") {
        return "\n";
    }

    const content = Array.isArray(node.content) ?
node.content.map(fromAdf).join("") : "";

    if (node.type === "paragraph") {
        return content ? `${content}\n` : "";
    }

    if (node.type === "bulletList" || node.type === "orderedList") {
        return `${content}\n`;
    }

    if (node.type === "listItem") {
        return `- ${content}`.trimEnd() + "\n";
    }

    return content;
};

const normalizePlainText = (value) => fromAdf(value).replace(/\n{3,}/g,
"\n\n").trim();

const resolvePriorityName = (priority) => {
    if (!priority) {
        return null;
    }

    const map = {
        low: "Low",
        medium: "Medium",
        high: "High"
    };

    return map[String(priority).toLowerCase()] || priority;
};

return {
    getCurrentUser: async () => request("/rest/api/3/myself"),

    createUser: async (emailAddress, products = ["jira-software"]) =>
request("/rest/api/3/user", {
    method: "POST",
    body: JSON.stringify({
        emailAddress,
        products
    })
}),
},

```

```

    getProjects: async () => {
        const data = await request("/rest/api/3/project/search");
        return data.values || [];
    },

    getProjectRoles: async (projectIdOrKey) =>
request(`/rest/api/3/project/${projectIdOrKey}/role`),

    getBoardsForProject: async (projectIdOrKey) => {
        const data = await
request(`/rest/agile/1.0/board?projectKeyOrId=${encodeURIComponent(projectIdO
rKey)}`);
        return data.values || [];
    },

    getBoardConfiguration: async (boardId) =>
request(`/rest/agile/1.0/board/${boardId}/configuration`),

    createProject: async ({ name, description = "", key }) => {
        const me = await request("/rest/api/3/myself");
        const leadAccountId = me?.accountId;

        if (!leadAccountId) {
            throw new Error("Не вдалося визначити Jira-користувача для
створення проекту");
        }

        return request("/rest/api/3/project", {
            method: "POST",
            body: JSON.stringify({
                assigneeType: "PROJECT_LEAD",
                description,
                key,
                leadAccountId,
                name,
                projectTemplateKey: "com.atlassian.jira-core-project-
templates:jira-core-simplified-project-management",
                projectTypeKey: "business"
            })
        });
    },

    archiveProject: async (projectIdOrKey) =>
request(`/rest/api/3/project/${projectIdOrKey}/archive`, {
        method: "POST"
    }),

    restoreProject: async (projectIdOrKey) =>
request(`/rest/api/3/project/${projectIdOrKey}/restore`, {
        method: "POST"
    }),

    getProjectStatuses: async (projectIdOrKey) => {
        const data = await
request(`/rest/api/3/project/${projectIdOrKey}/statuses`);
        const seen = new Map();

        for (const issueType of data || []) {
            for (const status of issueType.statuses || []) {
                if (!seen.has(status.name)) {
                    seen.set(status.name, status);
                }
            }
        }
    }
}

```

```

    }

    return Array.from(seen.values());
  },

  getBoardStatuses: async (projectIdOrKey) => {
    const [boards, projectStatuses] = await Promise.all([
      request(`/rest/agile/1.0/board?projectKeyOrId=${encodeURIComponent(projectIdOrKey)}`),
      request(`/rest/api/3/project/${projectIdOrKey}/statuses`)
    ]);
    const board = (boards.values || []).find(b => b.id === projectIdOrKey);

    if (!board) {
      return [];
    }

    const statusesById = new Map();

    for (const issueType of projectStatuses || []) {
      for (const status of issueType.statuses || []) {
        statusesById.set(String(status.id), status);
      }
    }

    const configuration = await
    request(`/rest/agile/1.0/board/${board.id}/configuration`);
    const columns = configuration.columnConfig?.columns || [];
    const orderedStatuses = [];
    const seen = new Set();

    for (const column of columns) {
      for (const columnStatus of column.statuses || []) {
        const resolvedStatus =
          statusesById.get(String(columnStatus.id)) ||
          columnStatus;
        const name = String(resolvedStatus.name || "").trim();
        const normalizedName = name.toLowerCase();

        if (!name || seen.has(normalizedName)) {
          continue;
        }

        seen.add(normalizedName);
        orderedStatuses.push({
          ...resolvedStatus,
          columnName: column.name
        });
      }
    }

    return orderedStatuses;
  },

  getIssues: async (projectExternalId = null) => {
    const jql = projectExternalId
      ? `project = "${projectExternalId}" ORDER BY created DESC`
      : "created >= -30d ORDER BY created DESC";

    const data = await request("/rest/api/3/search/jql", {
      method: "POST",
      body: JSON.stringify({
        jql,

```

```

        maxResults: 50,
        fields: [
            "summary",
            "description",
            "status",
            "project",
            "assignee",
            "priority",
            "duedate",
            "comment",
            "labels"
        ]
    });

    return data.issues || [];
},

    getIssue: async (issueIdOrKey) =>
    request(`/rest/api/3/issue/${issueIdOrKey}?fields=summary,description,status,project,assignee,priority,duedate,comment,labels`),

    getIssueWithAttachments: async (issueIdOrKey) =>
    request(`/rest/api/3/issue/${issueIdOrKey}?fields=summary,description,status,project,assignee,priority,duedate,comment,labels,attachment`),

    createIssue: async ({ projectKey, title, description = "", priority = null, dueDate = null, labels = [] }) => {
        const payload = {
            fields: {
                project: { key: projectKey },
                summary: title,
                issuetype: { name: "Task" },
                description: toAdfDocument(description)
            }
        };

        const resolvedPriority = resolvePriorityName(priority);

        if (resolvedPriority) {
            payload.fields.priority = { name: resolvedPriority };
        }

        if (dueDate) {
            payload.fields.duedate = dueDate;
        }

        if (Array.isArray(labels) && labels.length) {
            payload.fields.labels = labels;
        }

        return request("/rest/api/3/issue", {
            method: "POST",
            body: JSON.stringify(payload)
        });
    },

    updateIssue: async (issueIdOrKey, { title, description, priority, dueDate, labels }) => {
        const fields = {};

        if (typeof title === "string") {
            fields.summary = title;
        }
    }
}

```

```

        if (typeof description === "string") {
            fields.description = toAdfDocument(description);
        }

        if (priority !== undefined) {
            const resolvedPriority = resolvePriorityName(priority);
            fields.priority = resolvedPriority ? { name: resolvedPriority
} : null;
        }

        if (dueDate !== undefined) {
            fields.duedate = dueDate || null;
        }

        if (labels !== undefined) {
            fields.labels = Array.isArray(labels) ? labels : [];
        }

        return request(`/rest/api/3/issue/${issueIdOrKey}`, {
            method: "PUT",
            body: JSON.stringify({ fields })
        });
    },

    deleteIssue: async (issueIdOrKey) =>
request(`/rest/api/3/issue/${issueIdOrKey}`, {
    method: "DELETE"
}),

    addAttachment: async (issueIdOrKey, { fileName, mimeType, buffer })
=> {
        const form = new FormData();
        const blob = new Blob([buffer], {
            type: mimeType || "application/octet-stream"
        });

        form.append("file", blob, fileName);

        return request(`/rest/api/3/issue/${issueIdOrKey}/attachments`, {
            method: "POST",
            headers: {
                "X-Atlassian-Token": "no-check"
            },
            body: form
        });
    },

    deleteAttachment: async (attachmentId) =>
request(`/rest/api/3/attachment/${attachmentId}`, {
    method: "DELETE"
}),

    getIssueComments: async (issueIdOrKey) => {
        const data = await
request(`/rest/api/3/issue/${issueIdOrKey}/comment`);
        return (data.comments || []).map((comment) => ({
            id: comment.id,
            source: "jira",
            author_name: comment.author?.displayName || "Jira",
            author_email: comment.author?.emailAddress || "",
            content: normalizePlainText(comment.body),
            created_at: comment.created,
            updated_at: comment.updated

```

```

    }));
  },

  getIssueRemoteLinks: async (issueIdOrKey) => {
    const data = await
request(`/rest/api/3/issue/${issueIdOrKey}/remotelink`);
    return Array.isArray(data) ? data : [];
  },

  createIssueRemoteLink: async (issueIdOrKey, { title, url, summary =
"", globalId = null }) => {
    const payload = {
      object: {
        url,
        title
      }
    };

    if (summary) {
      payload.object.summary = summary;
    }

    if (globalId) {
      payload.globalId = globalId;
    }

    return request(`/rest/api/3/issue/${issueIdOrKey}/remotelink`, {
      method: "POST",
      body: JSON.stringify(payload)
    });
  },

  deleteIssueRemoteLink: async (issueIdOrKey, linkId) =>
request(`/rest/api/3/issue/${issueIdOrKey}/remotelink/${linkId}`, {
    method: "DELETE"
  }),

  addComment: async (issueIdOrKey, content) =>
request(`/rest/api/3/issue/${issueIdOrKey}/comment`, {
    method: "POST",
    body: JSON.stringify({
      body: toAdfDocument(content)
    })
  }),

  getIssueTransitions: async (issueIdOrKey) => {
    const data = await
request(`/rest/api/3/issue/${issueIdOrKey}/transitions`);
    return data.transitions || [];
  },

  transitionIssueToStatus: async (issueIdOrKey, targetStatusName) => {
    const issue = await
request(`/rest/api/3/issue/${issueIdOrKey}?fields=status`);
    const currentStatusName = issue?.fields?.status?.name || "";

    if (
      normalizeStatusValue(currentStatusName) ===
normalizeStatusValue(targetStatusName)
      || (getStatusGroup(currentStatusName) &&
getStatusGroup(currentStatusName) === getStatusGroup(targetStatusName))
    ) {
      return {
        skipped: true,

```

```

        already_in_status: true,
        current_status_name: currentStatusName
    };
}

const transitions = await
request(`/rest/api/3/issue/${issueIdOrKey}/transitions`);
const availableTransitions = transitions.transitions || [];
const scoredTransitions = availableTransitions
    .map((transition) => ({
        transition,
        score: getTransitionScore(targetStatusName, transition)
    }))
    .filter((item) => item.score > 0)
    .sort((left, right) => right.score - left.score);
const target = scoredTransitions[0]?.transition || null;

if (!target) {
    throw new Error(`У Jira немає доступного переходу в статус
"${targetStatusName}"`);
}

return request(`/rest/api/3/issue/${issueIdOrKey}/transitions`, {
    method: "POST",
    body: JSON.stringify({
        transition: {
            id: target.id
        }
    })
});
},

findUserByEmail: async (userEmail) => {
    if (!userEmail) {
        return null;
    }

    const data = await
request(`/rest/api/3/user/search?query=${encodeURIComponent(userEmail)}`);
    const normalizedEmail = String(userEmail).trim().toLowerCase();
    const candidates = (data || []).filter((user) => user &&
user.accountId && user.active !== false);
    const exactEmailMatch = candidates.find(
        (user) => String(user.emailAddress || "").toLowerCase() ===
normalizedEmail
    );

    if (exactEmailMatch) {
        return exactEmailMatch;
    }

    const realUsers = candidates.filter((user) =>
String(user.accountType || "").toLowerCase() !== "app");

    if (realUsers.length === 1) {
        return realUsers[0];
    }

    return null;
},

addUserToProjectRole: async (projectIdOrKey, roleId, accountId) =>
request(`/rest/api/3/project/${projectIdOrKey}/role/${roleId}`, {
    method: "POST",

```

```

        body: JSON.stringify({
            user: [accountId]
        })
    }},

    setIssueAssignee: async (issueIdOrKey, userReference) => {
        let accountId = null;

        if (typeof userReference === "object" &&
            userReference?.accountId) {
            accountId = userReference.accountId;
        } else if (userReference) {
            const match = await
            request(`/rest/api/3/user/search?query=${encodeURIComponent(userReference)}`)
            ;

            const exactEmail = (match || []).find(
                (item) => String(item.emailAddress || "").toLowerCase()
                === String(userReference).toLowerCase()
            );
            const realUsers = (match || []).filter(
                (item) => item && item.accountId &&
                String(item.accountType || "").toLowerCase() !== "app"
            );

            accountId = exactEmail?.accountId || (realUsers.length === 1
            ? realUsers[0].accountId : null);
        }

        return request(`/rest/api/3/issue/${issueIdOrKey}/assignee`, {
            method: "PUT",
            body: JSON.stringify({
                accountId
            })
        });
    },

    extractPlainText: normalizePlainText
};
};

```

Лістинг реалізації інтеграції з Google Calendar

```

import { db } from "../config/db.js";

const GOOGLE_CLIENT_ID = process.env.GOOGLE_CLIENT_ID || "";
const GOOGLE_CLIENT_SECRET = process.env.GOOGLE_CLIENT_SECRET || "";
const GOOGLE_REDIRECT_URI = process.env.GOOGLE_REDIRECT_URI ||
"http://localhost:5001/api/google-calendar/callback";
const GOOGLE_CALENDAR_SCOPE = "https://www.googleapis.com/auth/calendar";

const buildUrlEncodedBody = (params) => {
  const body = new URLSearchParams();

  Object.entries(params).forEach(([key, value]) => {
    if (value !== undefined && value !== null && value !== "") {
      body.set(key, value);
    }
  });

  return body.toString();
};

const readGoogleJson = async (response) => {
  const raw = await response.text();
  let data = null;

  try {
    data = raw ? JSON.parse(raw) : null;
  } catch {
    data = null;
  }

  if (!response.ok) {
    console.error("GOOGLE CALENDAR RAW ERROR:", data || raw ||
response.statusText);
    const message = data?.error_description || data?.error?.message ||
data?.error || raw || "Google Calendar API error";
    throw new Error(message);
  }

  return data;
};

const saveGoogleTokens = async (integration, tokenData) => {
  if (!integration?.id || !tokenData?.access_token) {
    return;
  }

  await db.promise().query(
    `
      UPDATE integrations
      SET access_token = ?,
          refresh_token = ?,
          updated_at = CURRENT_TIMESTAMP
      WHERE id = ?
    `,
    [
      tokenData.access_token,
      tokenData.refresh_token || integration.refresh_token || null,
      integration.id
    ]
  );
};

```

```

integration.access_token = tokenData.access_token;

if (tokenData.refresh_token) {
    integration.refresh_token = tokenData.refresh_token;
}
};

export const isGoogleCalendarConfigured = () => {
    return Boolean(GOOGLE_CLIENT_ID && GOOGLE_CLIENT_SECRET &&
GOOGLE_REDIRECT_URI);
};

export const buildGoogleCalendarAuthUrl = (state) => {
    if (!isGoogleCalendarConfigured()) {
        throw new Error("Google Calendar OAuth ще не налаштований на
сервері");
    }

    const params = new URLSearchParams({
        client_id: GOOGLE_CLIENT_ID,
        redirect_uri: GOOGLE_REDIRECT_URI,
        response_type: "code",
        access_type: "offline",
        prompt: "consent",
        include_granted_scopes: "true",
        scope: GOOGLE_CALENDAR_SCOPE,
        state

    });

    return
`https://accounts.google.com/o/oauth2/v2/auth?${params.toString()}`;
};

export const exchangeGoogleCalendarCode = async (code) => {
    const response = await fetch("https://oauth2.googleapis.com/token", {
        method: "POST",
        headers: {
            "Content-Type": "application/x-www-form-urlencoded"
        },
        body: buildUrlEncodedBody({
            code,
            client_id: GOOGLE_CLIENT_ID,
            client_secret: GOOGLE_CLIENT_SECRET,
            redirect_uri: GOOGLE_REDIRECT_URI,
            grant_type: "authorization_code"
        })
    });

    return readGoogleJson(response);
};

export const refreshGoogleCalendarToken = async (refreshToken) => {
    const response = await fetch("https://oauth2.googleapis.com/token", {
        method: "POST",
        headers: {
            "Content-Type": "application/x-www-form-urlencoded"
        },
        body: buildUrlEncodedBody({
            client_id: GOOGLE_CLIENT_ID,
            client_secret: GOOGLE_CLIENT_SECRET,
            refresh_token: refreshToken,
            grant_type: "refresh_token"
        })
    });

```

```

    });

    return readGoogleJson(response);
};

const googleApiRequest = async (integration, url, options = {}, allowRefresh
= true) => {
    const response = await fetch(url, {
        ...options,
        headers: {
            Authorization: `Bearer ${integration.access_token}`,
            "Content-Type": "application/json",
            ...(options.headers || {})
        }
    });

    if (response.status === 401 && allowRefresh && integration.refresh_token)
    {
        const tokenData = await
refreshGoogleCalendarToken(integration.refresh_token);
        await saveGoogleTokens(integration, tokenData);
        return googleApiRequest(integration, url, options, false);
    }

    return readGoogleJson(response);
};

export const getGoogleCalendarList = async (integration) => {
    return googleApiRequest(integration,
"https://www.googleapis.com/calendar/v3/users/me/calendarList");
};

export const listGoogleCalendarEvents = async (integration, { dateFrom,
dateTo, calendarId = "primary" }) => {
    const params = new URLSearchParams({
        singleEvents: "true",
        orderBy: "startTime",
        timeMin: new Date(`${dateFrom}T00:00:00Z`).toISOString(),
        timeMax: new Date(`${dateTo}T23:59:59Z`).toISOString()
    });

    return googleApiRequest(
        integration,

`https://www.googleapis.com/calendar/v3/calendars/${encodeURIComponent(calend
arId)}/events?${params.toString()}`
    );
};

export const createGoogleCalendarEvent = async (integration, event,
calendarId = "primary") => {
    return googleApiRequest(
        integration,

`https://www.googleapis.com/calendar/v3/calendars/${encodeURIComponent(calend
arId)}/events?sendUpdates=all`,
        {
            method: "POST",
            body: JSON.stringify(event)
        }
    );
};

export const updateGoogleCalendarEvent = async (integration, eventId, event,

```

```

calendarId = "primary") => {
  return googleApiRequest(
    integration,

    `https://www.googleapis.com/calendar/v3/calendars/${encodeURIComponent(calendarId)}/events/${encodeURIComponent(eventId)}?sendUpdates=all`,
    {
      method: "PUT",
      body: JSON.stringify(event)
    }
  );
};

export const deleteGoogleCalendarEvent = async (integration, eventId,
calendarId = "primary") => {
  const response = await fetch(

    `https://www.googleapis.com/calendar/v3/calendars/${encodeURIComponent(calendarId)}/events/${encodeURIComponent(eventId)}?sendUpdates=all`,
    {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${integration.access_token}`
      }
    }
  );

  if (!response.ok) {
    const raw = await response.text();
    let data = null;

    try {
      data = raw ? JSON.parse(raw) : null;
    } catch {
      data = null;
    }

    throw new Error(data?.error?.message || "Не вдалося видалити подію з Google Calendar");
  }

  return true;
};

```

Лістинг реалізації патернів Strategy та Factory Method

1. DevelopmentProviderFactory.js

```
import { GitlabDevelopmentStrategy } from "../GitlabDevelopmentStrategy.js";
import { GithubDevelopmentStrategy } from "../GithubDevelopmentStrategy.js";

export class DevelopmentProviderFactory {
  static create(linkedIntegration) {
    if (!linkedIntegration?.type) {
      throw new Error("Не вдалося визначити тип dev-інтеграції");
    }

    if (linkedIntegration.type === "gitlab") {
      return new GitlabDevelopmentStrategy();
    }

    if (linkedIntegration.type === "github") {
      return new GithubDevelopmentStrategy();
    }

    throw new Error(`Непідтримуваний dev-провайдер:
    ${linkedIntegration.type}`);
  }
}
```

2. BaseDevelopmentStrategy.js

```
export class BaseDevelopmentStrategy {
  parseMeta(rawMeta) {
    try {
      return rawMeta ? JSON.parse(rawMeta) : {};
    } catch {
      return {};
    }
  }

  getLinkedRepoName(linkedRepo = {}) {
    return linkedRepo.path_with_namespace
      || linkedRepo.full_name
      || linkedRepo.name
      || "";
  }
}
```

3. GithubDevelopmentStrategy.js

```
import { getGithubRepoStatus } from "../githubService.js";
import { BaseDevelopmentStrategy } from "../BaseDevelopmentStrategy.js";

export class GithubDevelopmentStrategy extends BaseDevelopmentStrategy {
  async buildStatus(linkedIntegration) {
    const meta = this.parseMeta(linkedIntegration.meta);
    const status = await getGithubRepoStatus({
      domain: linkedIntegration.domain,
      accessToken: linkedIntegration.access_token,
      repoFullName: linkedIntegration.external_id
    });

    return {
      linked: true,
      provider: "github",
      linked_repo: {

```

```

        id: linkedIntegration.external_id,
        ...meta,
        ...status.repo
    },
    open_reviews_count: status.open_pull_requests_count,
    open_reviews: status.open_pull_requests.map((item) => ({
        id: item.id,
        number: item.number,
        title: item.title,
        web_url: item.html_url,
        branch_label: `${item.head_ref} → ${item.base_ref}`,
        author_name: item.author_name,
        updated_at: item.updated_at || item.created_at
    })),
    latest_delivery: status.latest_workflow_run
    ? {
        kind: "workflow_run",
        status: status.latest_workflow_run.status,
        ref: status.latest_workflow_run.ref,
        web_url: status.latest_workflow_run.web_url,
        updated_at: status.latest_workflow_run.updated_at
    }
    : null,
    recent_commits: status.recent_commits,
    recent_activity: status.recent_activity
};
}
}

```

4. GitlabDevelopmentStrategy.js

```

import { getGitlabRepoStatus } from "../gitlabService.js";
import { BaseDevelopmentStrategy } from "../BaseDevelopmentStrategy.js";

export class GitlabDevelopmentStrategy extends BaseDevelopmentStrategy {
    async buildStatus(linkedIntegration) {
        const meta = this.parseMeta(linkedIntegration.meta);
        const status = await getGitlabRepoStatus({
            domain: linkedIntegration.domain,
            accessToken: linkedIntegration.access_token,
            repoId: linkedIntegration.external_id
        });

        return {
            linked: true,
            provider: "gitlab",
            linked_repo: {
                id: linkedIntegration.external_id,
                ...meta,
                ...status.repo
            },
            open_reviews_count: status.open_merge_requests_count,
            open_reviews: status.open_merge_requests.map((item) => ({
                id: item.id,
                number: item.iid,
                title: item.title,
                web_url: item.web_url,
                branch_label: `${item.source_branch} →
                ${item.target_branch}`,
                author_name: item.author_name,
                updated_at: item.updated_at || item.created_at
            })),
            latest_delivery: status.latest_pipeline
            ? {
                kind: "pipeline",

```

```
        status: status.latest_pipeline.status,  
        ref: status.latest_pipeline.ref,  
        web_url: status.latest_pipeline.web_url,  
        updated_at: status.latest_pipeline.updated_at  
    }  
    : null,  
    recent_commits: status.recent_commits,  
    recent_activity: status.recent_activity  
};  
}  
}
```

					ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						100