ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ МОНІТОРИНГУ СТАТУСУ ПРОЄКТУ З ІНТЕГРОВАНОЮ СИСТЕМОЮ АВТОМАТИЗОВАНОГО СПОВІЩЕННЯ ПРО РИЗИКИ ТА ВІДХИЛЕННЯ ВІД ГРАФІКУ: МЕТОДИ ОЦІНКИ ТА МІНІМІЗАЦІЇ ЗАТРИМОК

зі спеціальності 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

ступеня магістра

Виконавець роботи Лисенко Давід Віталійович
_____«____» _____202_ р.
(підпис)

Науковий керівник зав. каф., к.ф.-м.н. Ольховська О. В.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 77 с., 16 рис., 2 таблиці, 1 додаток, 11 джерел.

МОНІТОРИНГ ПРОЄКТІВ, РИЗИК-МЕНЕДЖМЕНТ, MPM-АНАЛІЗ, NODE.JS, REACT

Об’єктом розробки є веб-система моніторингу стану проєктів, що забезпечує відстеження виконання задач, аналіз залежностей, визначення критичного шляху та автоматичне виявлення ризиків і затримок.

Предметом розробки є програмна реалізація інтерактивного веб-додатку, який об’єднує засоби керування проєктами, інструменти виявлення ризиків, механізми аналітики та інтерфейс для візуального контролю прогресу.

Метою роботи є створення програмного продукту, який дозволяє автоматизувати процес моніторингу статусу проєктів, забезпечити своєчасне виявлення відхилень від планових графіків, підвищити прозорість виконання задач і надати менеджерам інструменти для аналізу ризиків у реальному часі.

Результатом роботи стало розроблення веб-системи ProjectPulse, що включає серверну частину (Node.js + Express + SQLite), клієнтський інтерфейс (React), модуль бізнес-логіки, систему роботи із залежностями задач та механізм автоматичного аналізу критичного шляху за методом MPM. Реалізовано аналітичний модуль, який розраховує ранні та пізні терміни виконання задач, визначає резерв часу (slack) та автоматично позначає критичні роботи.

Однією з ключових особливостей є автоматичне виявлення ризиків, які генеруються на основі просрочених задач, перевищеного запасу часу, небезпечних залежностей або зниженого темпу виконання. Система формує сповіщення, відображає рівень ризику (low, medium, high) та дозволяє користувачам оперативно реагувати на загрози.

У ProjectPulse реалізовано зручний інтерфейс управління задачами: Kanban-дошка з drag-and-drop, фільтри, панель деталей задачі, встановлення залежностей та перегляд активностей. Також створено модуль аналітики, який містить burndown chart, velocity-графік, heatmap навантаження команди та відображення ключових

метрик проєкту.

Інтерфейс системи включає такі розділи:

- Дашборд – огляд статусу проєктів, ризиків та сповіщень;
- Проєкти – створення та керування життєвим циклом проєктів;
- Задачі – Kanban-дошка, залежності, деталі виконання;
- Аналітика – графіки та критичний шлях;
- База знань – статті з рекомендаціями та документацією;
- Профіль – керування обліковим записом.

У процесі тестування підтверджено коректність розрахунків MPM, стабільність роботи REST API та здатність системи своєчасно визначати відхилення від плану. Проведено наскрізні (E2E) перевірки, які підтвердили формування ризиків, правильну зміну статусів задач та достовірність аналітичних метрик.

ProjectPulse було протестовано у локальному середовищі. Робота системи підтверджена як стабільна, а реалізовані механізми дозволяють ефективно контролювати хід виконання проєктів і зменшувати вплив затримок на загальний графік.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
API	Application Programming Interface – інтерфейс взаємодії між клієнтом і сервером
REST	Representational State Transfer – стиль архітектури веб-сервісів
CPM	Critical Path Method – метод критичного шляху
MPM	Metra Potential Method – метод аналізу залежностей задач
CRUD	Create, Read, Update, Delete – базові операції з даними
DB	Database – база даних
JWT	JSON Web Token – спосіб авторизації та передачі токенів
UI	User Interface – інтерфейс користувача
UX	User Experience – досвід взаємодії користувача із системою
SPA	Single Page Application – односторінковий веб-застосунок
HTTP	HyperText Transfer Protocol – протокол передачі даних у вебі
SQL	Structured Query Language – мова запитів до баз даних

ВСТУП

У сучасних умовах розвитку IT-індустрії та зростання складності цифрових продуктів особливої ваги набувають інструменти, що забезпечують ефективне управління проєктами. Команди працюють у динамічному середовищі, де строки виконання, взаємозалежність задач, швидкість ухвалення рішень та якість комунікації впливають на кінцевий результат не менше, ніж технічні рішення чи функціональні можливості продукту. Незважаючи на наявність великої кількості систем керування проєктами, значна частина з них не забезпечує комплексного підходу до моніторингу виконання задач, своєчасного виявлення ризиків та підтримки прийняття рішень на основі аналітичних даних.

Однією з ключових проблем управління проєктами є недостатня прозорість стану задач на різних етапах їхнього виконання. У багатьох командах відсутні засоби оперативного контролю за критичним шляхом, визначенням затримок, аналізом навантаження виконавців та виявленням ситуацій, які можуть призвести до порушення планових строків. Такі недоліки спричиняють накопичення помилок, відставання від графіка, хаотичну комунікацію та зниження якості планування.

Актуальність теми зумовлена потребою у створенні інструментів, які не просто фіксують інформацію про задачі, а виконують інтелектуальний аналіз даних і здатні автоматично виявляти ризики. Сучасні системи моніторингу проєктів повинні включати механізми прогнозування відхилень, обробки залежностей між задачами, відстеження фактичних та планових строків, а також засоби побудови аналітичних графіків, що відображають стан проєкту у реальному часі.

Відповіддю на зазначені виклики є розробка веб-додатку ProjectPulse, який реалізує інтегровану систему моніторингу проєктів із функціями автоматичного виявлення ризиків, критичного шляху, аналізу навантаження та візуалізації ключових метрик. Додаток поєднує класичні підходи до управління проєктами з сучасною веб-архітектурою, забезпечуючи зручний інтерфейс, адаптований до потреб команд будь-якого розміру.

Метою роботи є створення інтелектуальної веб-системи моніторингу стану проєкту, що забезпечує аналіз виконання задач, розрахунок критичного шляху,

автоматичне виявлення ризиків та підтримку прийняття рішень на основі об’єктивних даних. Для досягнення мети необхідно реалізувати набір функцій, що охоплюють управління задачами, роботу з залежностями, формування аналітики, обробку сповіщень, а також розробку інтерфейсу для взаємодії з даними.

Об’єктом дослідження є процес моніторингу стану проєктів у системах управління розробкою програмного забезпечення.

Предметом дослідження є методи аналізу відхилень, ризиків та візуалізації даних у веб-орієнтованих інформаційних системах.

Практичне значення роботи полягає у створенні повнофункціонального веб-додатку, який може використовуватись у невеликих та середніх командах для організації роботи, прозорого контролю строків, виявлення ризиків та аналізу ефективності. Завдяки модульній архітектурі розроблений продукт може бути інтегрований у різні робочі середовища або розширений під потреби конкретних організацій.

Дипломна робота містить постановку задачі, інформаційний огляд предметної області, теоретичне обґрунтування методів аналізу та проєктування системи, опис практичної реалізації програмного продукту ProjectPulse, а також інструкцію користувача та результати тестування.

Результати роботи апробовані на XLVIII Міжнародній науковій студентській конференції за підсумками науково-дослідних робіт студентів за 2024 рік «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті», за результатами якої опубліковано тези «Розробка веб-додатку для моніторингу статусу проєкту з інтегрованою системою автоматизованого сповіщення про ризики та відхилення від графіку».

1. ПОСТАНОВКА ЗАДАЧІ

Однією з ключових проблем у сфері управління програмними проєктами є забезпечення своєчасного контролю виконання задач, моніторинг ризиків та оперативне виявлення відхилень від планових строків. У процесі розробки сучасних інформаційних систем команди стикаються з необхідністю аналізувати велику кількість взаємопов'язаних даних: статуси задач, залежності між ними, навантаження виконавців, фактичні строки, ключові ризики, індекси продуктивності та інші параметри, що впливають на успішність проєкту.

Попри широке розповсюдження інструментів керування проєктами, значна частина доступних рішень не забезпечує комплексного автоматичного аналізу стану задач у режимі реального часу. Брак функцій прогнозування затримок, відсутність механізмів виявлення ризиків та недостатня глибина аналітики створюють умови, за яких керівникам команд складно своєчасно реагувати на потенційні проблеми. Таким чином виникає потреба у створенні спеціалізованого інструмента, здатного інтегрувати класичні методи управління проєктами з сучасними технологіями веб-розробки та візуалізації даних.

Основною задачею даної роботи є розробка веб-додатку ProjectPulse, який забезпечує моніторинг стану проєктів, аналіз виконання задач, виявлення ризиків та візуалізацію ключових метрик у реальному часі. Система повинна поєднувати зручну інтерфейсну складову з інтелектуальними алгоритмами оцінювання стану проєкту.

Для реалізації поставленої мети необхідно вирішити такі задачі:

- проаналізувати предметну область та визначити ключові вимоги до системи моніторингу проєктів;
- дослідити підходи до виявлення затримок, ризиків та відхилень у виконанні задач;
- обґрунтувати вибір архітектурних та технологічних рішень для веб-додатку;
- розробити модель даних для представлення проєктів, задач, залежностей, ризиків та сповіщень;

- реалізувати серверну частину з підтримкою REST API, авторизації та функцій аналітики;
- створити клієнтську частину з інтуїтивним інтерфейсом, що забезпечує управління задачами, перегляд аналітики та взаємодію з системою сповіщень;
- реалізувати механізми автоматичного виявлення ризиків та визначення критичного шляху на основі методів MPM;
- створити модуль аналітики, що візуалізує прогрес виконання проєкту, навантаження команди та основні метрики;
- провести тестування функціональності та перевірити відповідність системи встановленим вимогам;
- розробити інструкцію користувача та описати основні сценарії роботи з системою.

Узагальнюючи, постановка задачі полягає у створенні комплексного веб-орієнтованого інструмента моніторингу проєктів, здатного забезпечити своєчасне виявлення ризиків, аналіз виконання задач та підтримку ухвалення управлінських рішень на основі інтегрованих аналітичних механізмів.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Проблематика контролю статусу проєктів у сучасному управлінні

Ефективний контроль статусу проєктів є однією з ключових умов успішної діяльності команд розробки програмного забезпечення. Зі збільшенням складності цифрових продуктів, зростанням кількості задач та учасників команди виникає потреба у своєчасному відстеженні виконання робіт, оцінюванні прогресу та виявленні можливих відхилень. Проблематика контролю полягає в тому, що процеси, пов’язані із плануванням, комунікацією та аналізом, часто не синхронізовані або виконуються вручну, що призводить до втрати актуальності даних та появи ризиків.

Однією з головних проблем сучасного проєктного управління є відсутність повної прозорості стану задач на всіх етапах життєвого циклу. Значна частина проєктних команд стикається з тим, що інформація про статус задач оновлюється нерегулярно, не відображає реальний стан робіт або дублюється у різних джерелах. У таких умовах керівники проєктів не можуть оперативно реагувати на затримки, виявляти «вузькі місця» чи прогнозувати час завершення робіт.

Ще однією проблемою є складність обліку взаємозалежностей між задачами. У реальних проєктах виконання однієї задачі часто залежить від результатів інших, що створює ланцюги, які впливають на загальний графік. За відсутності автоматизованих інструментів відстеження залежностей команди не мають можливості оцінити вплив затримки однієї задачі на весь проєкт, що збільшує ризик порушення термінів. [1]

Важливим аспектом проблематики є також недостатня підтримка аналізу навантаження виконавців. Коли розподіл задач здійснюється вручну або без урахування реального темпу роботи учасників команди, це призводить до перевантаження окремих виконавців і нерівномірності темпу виконання задач. У результаті з’являються затримки, що накопичуються й негативно впливають на успішність всього проєкту.

Окрім того, у багатьох командах відсутні ефективні методи автоматичного

виявлення ризиків. Частина ризиків стає очевидною лише після появи затримок, а не до їх виникнення. Це зменшує можливість проактивного управління проєктом та ускладнює прийняття рішень. Без аналітичних механізмів, що оцінюють темп роботи, наближення дедлайнів або просрочені задачі, команда позбавлена інструмента раннього попередження.

Проблеми також виникають через недостатню інтеграцію існуючих систем керування проєктами з іншими інструментами команд: системами контролю версій, платформами для комунікації та сховищами документації. Це призводить до фрагментації інформації, коли команда змушена одночасно використовувати декілька непов’язаних інструментів, а дані про стан проєкту розпорошуються у різних джерелах. [2]

Таким чином, основними проблемами контролю статусу проєктів у сучасному управлінні є:

– недостатня прозорість виконання задач;

– відсутність своєчасного оновлення інформації;

– складність аналізу залежностей;

– відсутність механізмів автоматичного виявлення ризиків;

– нерівномірний розподіл навантаження;

– фрагментація інформаційного простору.

Розв’язання зазначених проблем можливе шляхом створення інтегрованої веб-системи, яка поєднує моніторинг стану задач, аналіз ризиків, візуалізацію ключових метрик та підтримку прийняття рішень у реальному часі.

2.2. Підходи до виявлення ризиків і затримок у проєктній діяльності

Виявлення ризиків і затримок є одним із ключових процесів у проєктному менеджменті, оскільки саме ці фактори найбільше впливають на строки виконання робіт, якість кінцевого продукту та загальну ефективність команди. Сучасні підходи до ідентифікації ризиків ґрунтуються на поєднанні методологій управління проєктами, аналітичних методів, узагальнених моделей ухвалення рішень та

автоматизованих алгоритмів аналізу даних.

Одним із найпоширеніших способів виявлення ризиків є експертний підхід, коли керівники проєктів або досвідчені учасники команди аналізують поточний стан проєкту, наявні ресурси та зовнішні фактори. Такий підхід широко використовується у класичних методологіях управління, зокрема в PMBOK та PRINCE2. Проте його ефективність залежить від компетентності фахівців і може бути обмеженою за умов великої кількості паралельних задач або швидких змін у проєкті.

Другим важливим підходом є аналіз історичних даних, який ґрунтується на припущенні, що проєкти зі схожими характеристиками мають подібні ризики. Дослідження попередніх затримок, кількості змінених задач, часу переходу між статусами та середнього темпу роботи дозволяє прогнозувати потенційні проблеми ще до того, як вони стануть критичними. Такий підхід часто застосовується у гнучких методологіях, зокрема Scrum та Kanban. [3]

Суттєву роль відіграють також методи аналізу залежностей між задачами. У складних проєктах виконання однієї задачі часто є ключовим для початку інших. Наявність циклів, неправильно визначених залежностей чи задач із надмірною кількістю попередників створює так звані “вузькі місця”, які уповільнюють роботу всієї команди. Використання алгоритмів топологічного сортування, виявлення критичного шляху та обчислення резервів часу дозволяє своєчасно виявляти задачі з найбільшим ризиком затримки.

Ще одним підходом є моніторинг виконання задач у реальному часі, коли система автоматично аналізує дотримання планових строків. Якщо фактична дата завершення задачі перевищує планову, або задача наближається до дедлайну без прогресу, система формує попередження. Такий підхід забезпечує виявлення ризиків на ранніх етапах та мінімізує залежність від людського фактора. [4]

Важливу роль відіграють методи прогнозування на основі метрик продуктивності, таких як velocity, lead time та cycle time. У Kanban і Scrum ці метрики використовуються для оцінки швидкості команди, визначення ймовірності завершення задач у строк та прогнозування майбутніх затримок. Порівняння

планової та фактичної продуктивності дозволяє виявити ризики ще до появи фактичних відхилень.

Сучасні системи також використовують підхід автоматичного аналізу змін, коли кожна подія – зміна статусу, редагування термінів, блокування задачі – розглядається як потенційний фактор ризику. Система на основі вбудованої логіки визначає, чи є зміна критичною, та повідомляє про це користувача.

Таким чином, до основних підходів виявлення ризиків і затримок у проєктній діяльності належать:

– експертний аналіз;

– використання історичних даних;

– аналіз залежностей між задачами;

– автоматизований моніторинг статусів;

– прогнозування на основі метрик продуктивності;

– контроль змін і поведінкових патернів задач.

Інтеграція цих підходів у єдину інформаційну систему забезпечує своєчасне виявлення проблем, підвищує точність планування та покращує ефективність управління проєктами. Саме на цих принципах ґрунтується функціональність системи ProjectPulse, яка поєднує автоматичний аналіз, облік залежностей та візуалізацію даних для своєчасного виявлення затримок.

2.3. Методи мінімізації відхилень від планових графіків

Мінімізація відхилень від планових графіків є одним із ключових завдань у проєктному менеджменті, оскільки навіть незначні затримки окремих задач можуть спричинити суттєве порушення загального плану виконання робіт. Ефективні методи запобігання відхиленням спрямовані на забезпечення прозорості, підвищення передбачуваності процесів та оптимізацію використання ресурсів.[5]

Одним із найпоширеніших методів є детальне планування з розбиттям задач на малі елементи, виконання яких можна точно оцінити та контролювати. Чим дрібніше структурована робота, тим легше відстежувати її прогрес і виявляти

затримки на ранніх етапах. Такий підхід широко використовується у методологіях Agile, де робота поділяється на короткі ітерації, а задачі мають чіткі критерії завершення.

Важливу роль відіграє регулярний моніторинг статусів задач, який забезпечує своєчасне оновлення інформації та дозволяє команді швидко реагувати на відхилення. Інструменти, що включають автоматичні сповіщення про наближення або порушення дедлайнів, значно зменшують ризик накопичення критичних затримок. Такі механізми дозволяють керівникам проєкту отримувати актуальні дані без необхідності проводити численні статус-зустрічі.

Ефективним методом є також управління залежностями між задачами, що дозволяє запобігати “ланцюговим” затримкам. Виявлення критичного шляху, визначення задач із нульовим резервом часу та оптимізація їх виконання забезпечують стабільність планового графіку. У складних проєктах, де виконання багатьох задач є взаємопов’язаним, така аналітика є необхідною умовою збереження темпу розробки. [2]

До методів мінімізації відхилень належить раціональний розподіл навантаження між виконавцями, що дозволяє уникнути перевантаження окремих членів команди. Нерівномірність роботи часто призводить до зниження продуктивності та накопичення прострочених задач. Використання інструментів, що аналізують навантаження та прогнозують можливі “вузькі місця”, сприяє стабільності темпу команди.

Ще одним важливим методом є регулярний аналіз ризиків та їхнє раннє виявлення. Постійне відстеження метрик, пов’язаних із продуктивністю, змінами у проєкті чи поведінкою задач, дозволяє проєктній команді своєчасно застосовувати коригувальні дії. Серед таких дій можуть бути перерозподіл задач, уточнення вимог, залучення додаткових ресурсів або зміна пріоритетів. [4]

Особливе місце посідають аналітичні методи прогнозування, які ґрунтуються на аналізі історичних даних і поточних трендів. Зокрема, такі метрики, як velocity, lead time, cycle time, burndown rate, дозволяють передбачити ймовірність завершення задач у строк і оцінити, чи достатній темп роботи команди для

підтримання планового графіку. Ці методи є основою багатьох сучасних систем моніторингу бізнес-процесів.

Важливим засобом мінімізації відхилень є також покращення комунікації. У командах, де інформація передається своєчасно та узгоджено, кількість непорозумінь і помилок значно менша. Системи, що включають централізоване сховище документації, автоматичні повідомлення та прозору історію змін, сприяють швидкому реагуванню на потенційні проблеми. [6]

Підсумовуючи, до основних методів мінімізації відхилень від планових графіків належать:

– детальне планування та декомпозиція задач;

– регулярний моніторинг статусів;

– управління залежностями та аналіз критичного шляху;

– оптимізація розподілу навантаження;

– раннє виявлення та оцінювання ризиків;

– використання аналітичних метрик і прогнозування;

– налагоджена комунікація та єдине інформаційне середовище.

Застосування цих методів у комплексі забезпечує підвищення стабільності проєктного процесу та значно зменшує ймовірність відхилення від планових строків. Саме на цих принципах базується функціональність системи ProjectPulse.

2.4. Огляд існуючих систем моніторингу проєктів

На сучасному ринку програмних продуктів представлена значна кількість систем, які забезпечують управління проєктами та моніторинг виконання задач. Попри відмінності у функціональності, більшість із них спрямовані на підтримку командної роботи, організацію процесів планування, відстеження прогресу та комунікацію між учасниками. Розглянемо найбільш поширені системи, що активно використовуються у сфері розробки програмного забезпечення.

Одним із найпопулярніших рішень є Jira, яка розроблена компанією Atlassian і широко застосовується у командах, що працюють за методологіями Agile та Scrum.

Jira забезпечує гнучку систему створення задач, роботу з епік-тасками, налаштовувані робочі процеси, багаторівневі залежності та детальну аналітику (див. рис. 2.1). Платформа може інтегруватися з Confluence, Bitbucket, GitHub та іншими інструментами, що робить її придатною для великих команд і складних проєктів. Проте значним недоліком є висока вартість та складність конфігурації, що ускладнює використання малими командами.

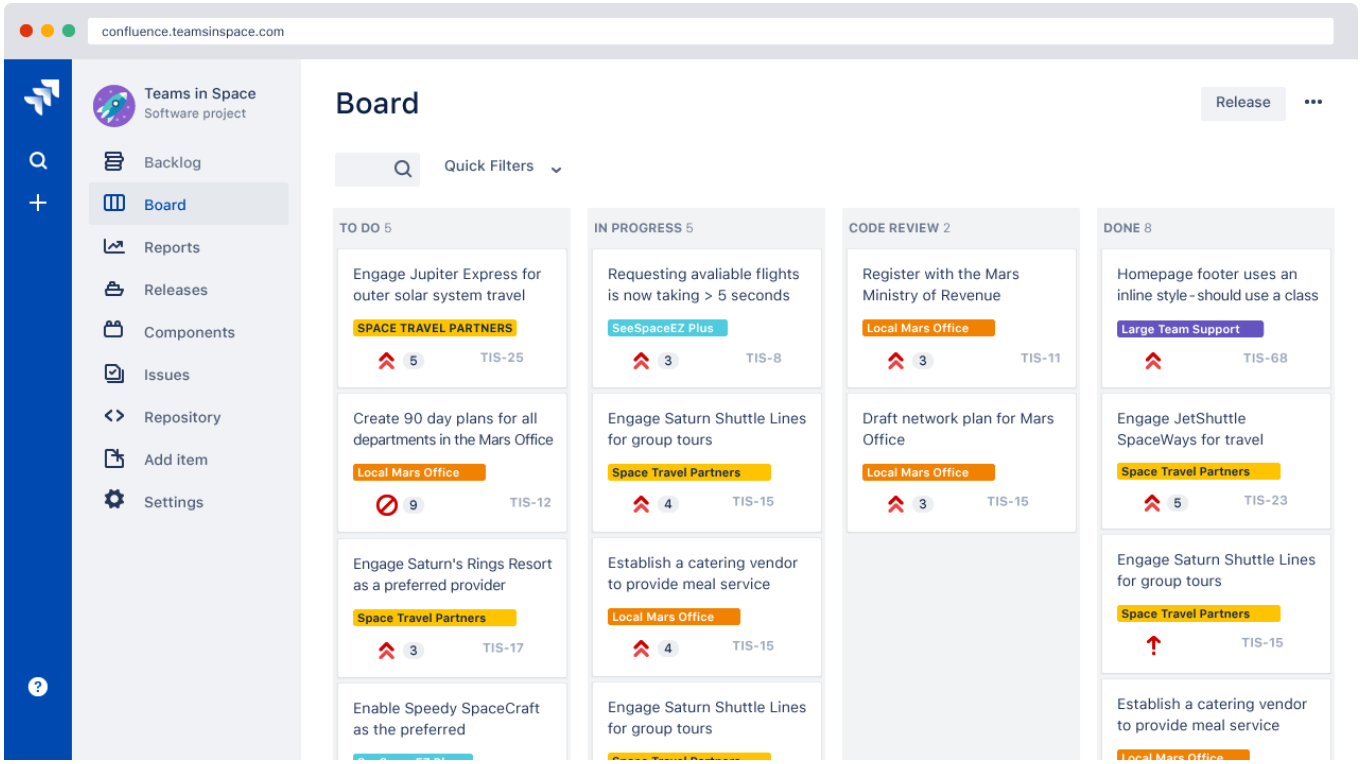

Рисунок 2.1 – Приклад інтерфейсу «Jira»

Ще одним популярним інструментом є Asana, орієнтована на бізнес-команди та кросфункціональні проєкти (див. рис. 2.2). Asana пропонує різні формати представлення задач: списки, Kanban-дошки, календарі та діаграми Ганта. Система включає засоби контролю прогресу, управління ціллями та командною взаємодією. Однак, незважаючи на широкий набір інструментів, Asana менш орієнтована на технічні команди й обмежено підтримує складні залежності між задачами.

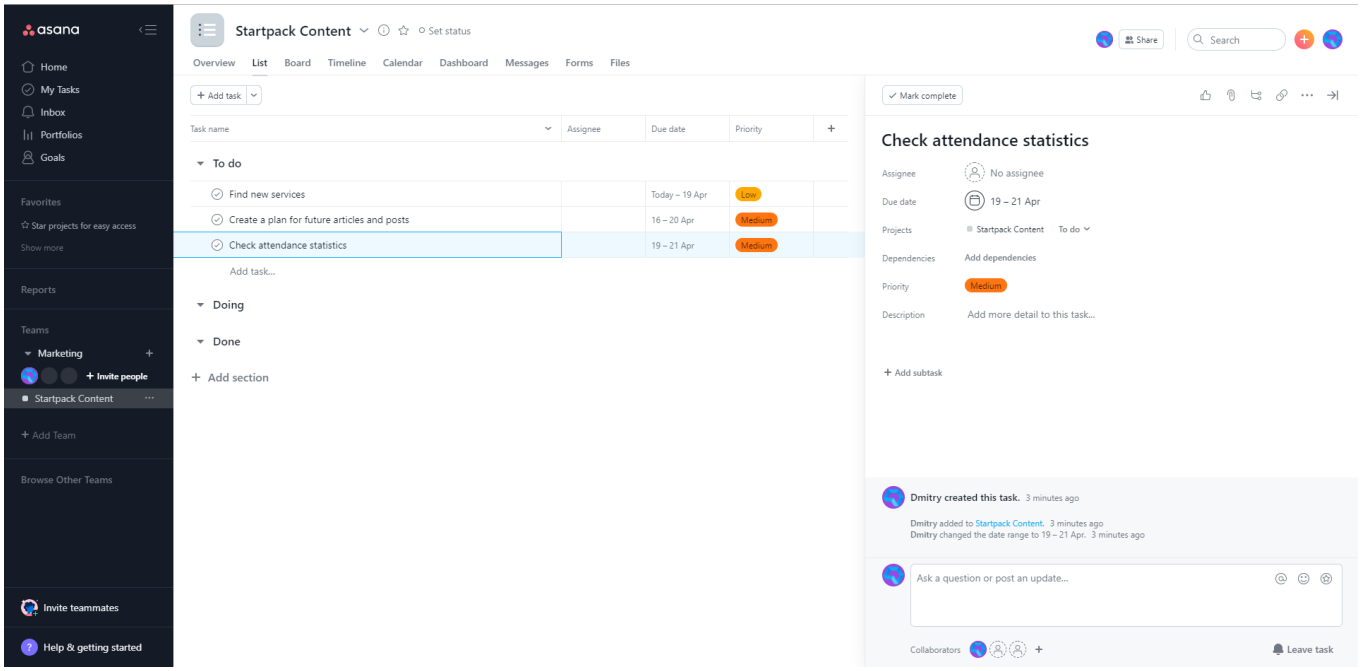

Рисунок 2.2 – Приклад інтерфейсу «Asana»

Поширеним рішенням для візуальної організації задач є Trello, який базується на парадигмі Kanban. Trello забезпечує просте керування задачами за допомогою дошок, карток та списків (див. рис. 2.3). Система підходить для невеликих команд і простих проєктів, але практично не має вбудованих засобів аналітики, автоматичного виявлення ризиків або прогнозування відхилень. Розширення функціональності можливе за допомогою Power-Ups, але вони обмежені за можливостями і доступні не у всіх тарифах.

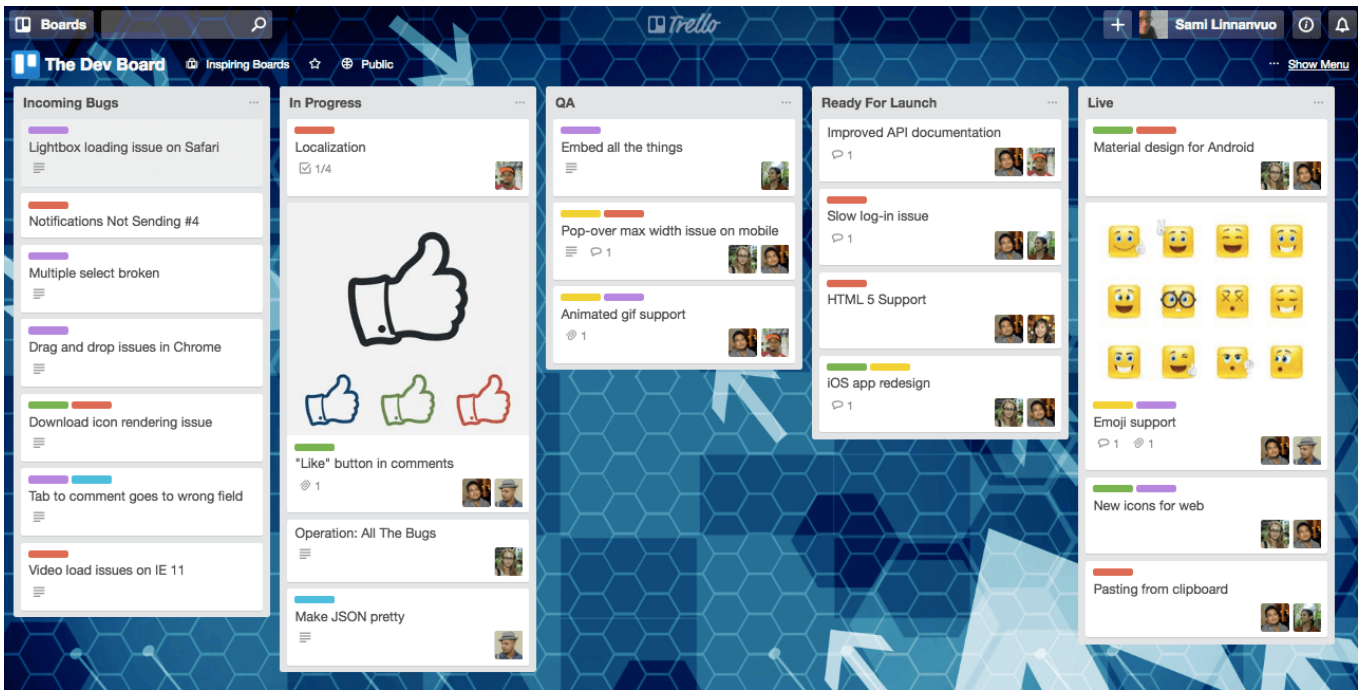

Рисунок 2.3 – Приклад інтерфейсу «Trello»

Більш універсальним рішенням є ClickUp, який поєднує функції планування, документообігу, інструменти для трекінгу часу, контроль навантаження та розширену аналітику (див. рис. 2.4). ClickUp дозволяє будувати діаграми Ганта, відстежувати цілі, формувати автоматичні звіти та керувати ресурсами. Така гнучкість робить платформу привабливою для середніх та великих команд, однак інтерфейс може бути перевантаженим, що впливає на швидкість роботи користувачів.

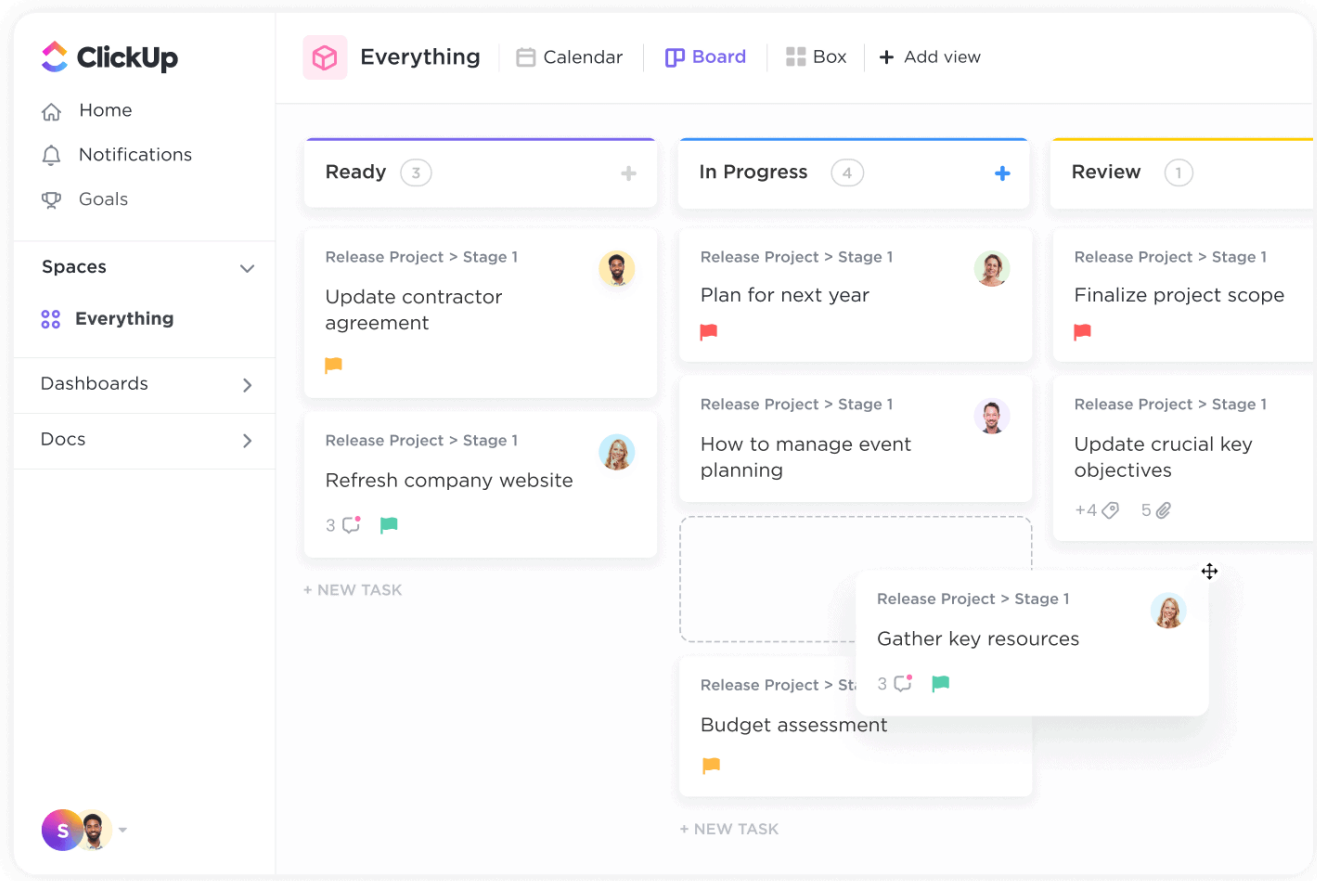

Рисунок 2.4 – Приклад інтерфейсу «ClickUp»

У сфері розробки також використовуються системи YouTrack, Redmine, Monday.com, Notion та інші продукти, які забезпечують різний рівень підтримки залежностей, робочих процесів та аналітики. Проте більшість з них або орієнтована на корпоративний сегмент, або має суттєві обмеження у безкоштовних тарифах.

Таким чином, незважаючи на різноманітність доступних систем, значна частина з них не забезпечує повноцінного автоматичного аналізу стану проєкту та ризиків, що актуалізує потребу у створенні інтегрованих рішень, подібних до ProjectPulse.

2.5. Порівняння функціональності сторонніх систем із потребами користувачів

Попри широкий вибір інструментів для управління проєктами, реальні потреби команд часто залишаються частково або повністю незадоволеними. Більшість популярних систем пропонують базові засоби планування та відстеження

задач, проте глибокі аналітичні модулі, автоматичне виявлення ризиків, робота з залежностями та прогнозування строків реалізовані або поверхнево, або недоступні без корпоративних тарифів. Це ускладнює використання таких інструментів у малих і середніх командах, які потребують ефективних рішень без надмірної вартості. [7]

Користувачі сучасних проєктних систем очікують, що інструмент забезпечуватиме:

– прозоре відображення статусу задач;

– аналіз критичного шляху та прогнозування затримок;

– автоматичне виявлення ризиків;

– візуалізацію аналітики (burndown, velocity, heatmap);

– підтримку залежностей між задачами;

– коректне відображення навантаження виконавців;

– простий та інтуїтивний інтерфейс;

– можливість роботи у реальному часі без складної конфігурації.

Для кращого розуміння того, наскільки сучасні інструменти відповідають цим вимогам, проведено порівняння найбільш популярних систем. У таблиці наведено оцінку ключових функцій, важливих для команд, що працюють у сфері розробки програмного забезпечення (див. табл. 2.1).

Таблиця 2.1 – Порівняння функціональних можливостей систем моніторингу

Функціональність / Система	*Jira*	*Asana*	*Trello*	*ClickUp*	*Потреби користувачів*
Підтримка Kanban / Scrum	✓	✓	✓	✓	✓
Робота із залежностями задач	✓	–	–	✓	✓
Виявлення критичного шляху	✓	✓	–	✓	✓
Автоматичне виявлення ризиків	*Частково*	–	–	*Частково*	✓
Burndown / Velocity аналітика	✓	*Частково*	–	✓	✓
Heatmap навантаження	*Частково*	–	–	✓	✓
Простота інтерфейсу	*Помірна*	*Висока*	*Висока*	*Середня*	*Висока*
Налаштовуваність робочих процесів	*Висока*	*Середня*	*Низька*	*Висока*	*Середня / Висока*
Вартість	*Висока*	*Середня*	*Низька*	*Середня*	*Низька / Середня*
Підтримка глибокої аналітики	✓	*Частково*	–	✓	✓
Прогнозування строків і відхилень	*Частково*	–	–	*Частково*	✓
Орієнтація на технічні команди	✓	*Частково*	–	✓	✓

Аналіз отриманих результатів

Проведене порівняння свідчить про те, що жодна з розглянутих систем не забезпечує повний комплекс інструментів, необхідних для оперативного виявлення ризиків та повноцінного моніторингу стану проєкту у режимі реального часу. Зокрема:

- Trello не підтримує залежності, автоматичні ризики та аналітику.
- Asana має зручний інтерфейс, але обмежені можливості в роботі з ризиками та прогнозуванням.
- Jira є найбільш функціональною, проте її використання пов’язане зі значною складністю та високою вартістю.
- ClickUp охоплює широкий спектр можливостей, проте його інтерфейс може бути перевантаженим для невеликих команд.

Усі системи мають або обмеження у роботі з критичним шляхом, або недостатню автоматизацію аналізу ризиків, або проблему високої складності впровадження.

Ці висновки підтверджують необхідність створення нових інструментів, таких як ProjectPulse, які поєднують: автоматичний аналіз стану задач, виявлення ризиків, роботу із залежностями, візуалізацію аналітики, інтуїтивний інтерфейс, доступність для малих і середніх команд.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Архітектурні підходи до розробки веб-орієнтованих інформаційних систем

Архітектура веб-орієнтованих інформаційних систем визначає загальні принципи побудови, структуру компонентів, способи їхньої взаємодії та забезпечує умови для масштабованості, надійності й підтримуваності програмного продукту. Вибір архітектурного підходу суттєво впливає на продуктивність системи, зручність розробки, можливість розширення функціональності та інтеграції з іншими сервісами. У сучасній практиці використовується кілька базових архітектурних моделей, на основі яких будуються системи моніторингу та управління проєктами.

Одним із найбільш поширених підходів є клієнт-серверна архітектура, яка передбачає розподіл обов’язків між сервером, що обробляє дані й забезпечує доступ до ресурсу, та клієнтом, який відповідає за відображення інтерфейсу та взаємодію з користувачем. Така модель дозволяє централізовано керувати даними, забезпечувати контроль доступу та гарантувати цілісність інформації. У веб-додатках сервер часто реалізується як REST API, а клієнт – як односторінковий застосунок (SPA), побудований на сучасних фреймворках, таких як React, Angular чи Vue.js. [8]

Ще одним ключовим підходом є багатошарова (N-tier) архітектура, що передбачає розділення системи на логічні рівні: рівень презентації, рівень бізнес-логіки та рівень даних. Такий поділ спрощує розробку та тестування, робить систему стійкішою до змін та полегшує повторне використання компонентів. Зміни у бізнес-логіці не потребують змін інтерфейсу, а оптимізація бази даних не впливає на серверний код, що забезпечує високу гнучкість.

У сучасних системах все більшої популярності набуває мікросервісна архітектура, що передбачає побудову системи з окремих незалежних сервісів, кожен з яких відповідає за певну функціональність. Мікросервіси взаємодіють через API та можуть масштабуватися окремо. Такий підхід є ефективним для великих продуктів або розподілених команд, але він потребує складної інфраструктури та

значних витрат на підтримку. У системах типу ProjectPulse цей підхід може бути використаний для виділення окремих сервісів аналітики, аутентифікації чи обробки ризиків.

У середовищі веб-додатків поширеним є також архітектурний стиль REST, який базується на передачі даних у форматах JSON або XML та використанні стандартних HTTP-методів. REST спрощує інтеграцію між різними системами, забезпечує легкість масштабування та прозорість взаємодії клієнтських і серверних компонентів. Завдяки своїй простоті та універсальності REST залишається основою більшості сучасних веб-сервісів. [9-10]

Особливе значення у веб-системах мають принципи модульності та повторного використання компонентів. Чітко структуровані модулі дозволяють швидко розширювати функціональність, підтримувати систему та адаптувати її до нових вимог. Це особливо важливо для систем моніторингу, де потрібно регулярно додавати нові типи аналітики, сповіщень або інтеграцій.

Також важливим аспектом є забезпечення масштабованості та продуктивності. Сучасні веб-додатки повинні справлятися з великими обсягами даних і забезпечувати швидкий відгук. Використання кешування, оптимізація запитів до бази даних, асинхронна обробка подій та розподіл навантаження є важливими складовими архітектурного проєктування. [11]

Крім того, архітектура повинна враховувати вимоги до безпеки, такі як контроль доступу, захист даних, авторизація та аутентифікація користувачів. Веб-системи, які працюють із критичною інформацією про стан проєктів, повинні забезпечувати захист від несанкціонованого доступу, ін’єкцій та інших типових загроз.

Отже, архітектурні підходи до розробки веб-орієнтованих інформаційних систем базуються на поєднанні клієнт-серверної моделі, багатошарової структури, REST-принципів, модульності та забезпечення масштабованості. Вибір відповідного підходу визначає ефективність і надійність системи та є основою для подальшої реалізації продукту, такого як ProjectPulse.

3.2. Модель даних для системи моніторингу

Модель даних системи моніторингу статусу проєктів визначає, які сутності зберігаються в базі даних, як вони пов’язані між собою та які атрибути є ключовими для аналізу стану, ризиків та відхилень від плану. Правильно спроєктована модель дає змогу не лише фіксувати фактичну інформацію, а й будувати на її основі аналітичні показники, автоматичні сповіщення та агреговані зрізи проєкту.

На рисунку 3.1 наведено базову доменну модель системи ProjectPulse. Вона складається з кількох груп сутностей: користувачі та ролі, проєкти та задачі, залежності між задачами, ризики, сповіщення та аналітичні зрізи. Центральним об’єктом є сутність Project, яка описує окремий проєкт, його назву, власника, планові дати початку та завершення, а також поточний статус. Кожен проєкт належить конкретному користувачу-власнику (зв’язок через поле owner_id), що відповідає за загальне ведення та налаштування моніторингу (див. рис. 3.1).

Рисунок 3.1 – Основні сутності доменної моделі

Сутність User репрезентує учасників системи: власників проєктів, менеджерів та виконавців задач. Атрибут role задає тип участі користувача (owner, pm, member, viewer), що надалі використовується для обмеження прав доступу та побудови інтерфейсу. Наявність ознаки is_active дозволяє відключати користувачів без втрати історичних даних, що важливо для коректного зберігання записів про

відповідальних осіб у минулих проєктах.

Ключовим елементом планування є сутність Task, яка пов’язана з проєктом через поле project_id. Для кожної задачі фіксуються назва, відповідальний (assignee_id), планова тривалість виконання, статус та ознака критичності (critical_flag). Саме на основі задач і їхніх статусів система обчислює поточний прогрес, виявляє відставання від графіка та формує критичний шлях. Пов’язування задач із користувачами дозволяє будувати зрізи за виконавцями та аналізувати, де виникають основні “вузькі місця”.

Для опису взаємозалежностей між роботами використовується сутність TaskDependency. Вона містить посилання на попередню та наступну задачу (predecessor_id, successor_id), тип залежності (FS, FF, SS, SF) та можливий часовий лаг. Така структура відповідає класичним підходам до мережевого планування й дозволяє використовувати критичний шлях та аналогічні методи аналізу. Наявність явної таблиці залежностей робить модель гнучкою: одна задача може мати кілька попередників і нащадків, що важливо для реальних проєктів. [12]

Окремий блок моделі присвячено роботі з ризиками. Сутність Risk прив’язана до проєкту (project_id) і, за потреби, до конкретної задачі (related_task_id). Для кожного ризику зберігаються оцінки ймовірності та впливу (у шкалі 1–5), а також розрахований інтегральний показник severity_score. Це дає змогу не лише фіксувати сам факт ризику, а й ранжувати його та виділяти найбільш критичні елементи, що впливають на строки проєкту. Сутність RiskRule описує формальні правила виявлення ризиків і відхилень: умовні вирази зберігаються у форматі JSON (condition_json) та застосовуються до даних про задачі, їхні статуси й затримки.

Сутність Alert відповідає за сповіщення, що формуються системою: вони містять посилання на проєкт, користувача-отримувача, джерело сповіщення (задача, ризик або системна подія), канал доставки (e-mail, Telegram, вбудовані повідомлення) та поточний статус (очікує, надіслано, прочитано). Таким чином, між аналітичною частиною та взаємодією з користувачем утворюється чіткий міст: результати правил ризик-менеджменту матеріалізуються в конкретних алертах, що потрапляють до відповідальних осіб.

Для зберігання агрегованих показників використовується сутність ProjectSnapshot, яка фіксує періодичні «зрізи» стану проєкту: значення індексів виконання (наприклад, SPI, CPI), кількість прострочених задач, узагальнені показники затримок. Такі записи використовуються для побудови історичних графіків, дашбордів і подальшого аналізу трендів без необхідності щоразу переобчислювати всі показники з нуля.

На рисунку 3.2 показано, як базова модель даних доповнюється сутностями, пов’язаними з подіями та історією змін. Сутність TaskStatusHistory фіксує кожну зміну статусу задачі із зазначенням попереднього й нового стану, часу зміни та користувача, який її ініціював. Це дозволяє відслідковувати реальну динаміку виконання, а також визначати моменти, коли задача перейшла в блокований стан чи вийшла за межі планового терміну (див. рис. 3.2).

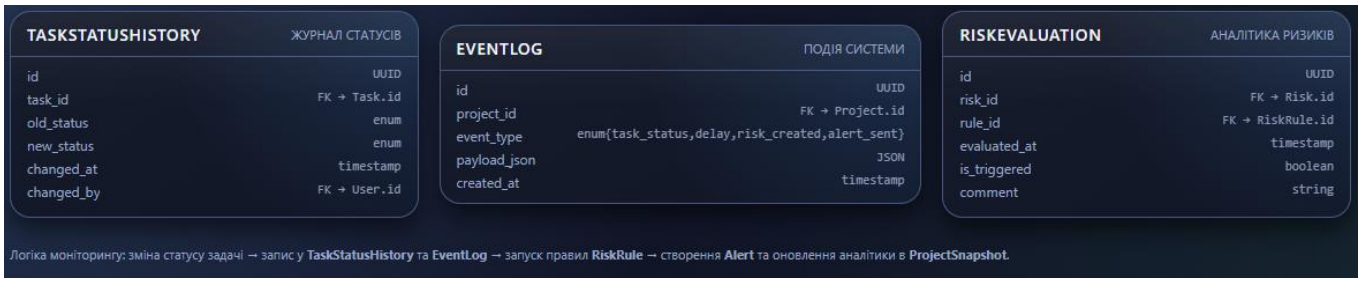

Рисунок 3.2 – Модель подій, історії та сповіщень

Сутність EventLog використовується як узагальнений журнал подій у проєкті. В ній зберігаються різні типи подій (зміна статусів, виявлення затримок, створення ризику, відправка сповіщення) разом із додатковими даними у форматі JSON. Саме на основі цього журналу аналітичні модулі виконують обчислення, а правила ризик-менеджменту – перевірку умов.

Сутність RiskEvaluation фіксує факти застосування правил до конкретних ризиків: коли саме правило було виконане, чи спрацювало воно, які коментарі або пояснення надав модуль аналітики. Завдяки цьому можна будувати звіти не лише про поточні ризики, а й про ефективність самих правил, їхню чутливість та кількість хибних спрацьовувань.

У підсумку модель даних системи ProjectPulse поєднує оперативні сутності (проєкти, задачі, користувачі), структуру планування (залежності задач), елементи ризик-менеджменту (ризики, правила, оцінки), підсистему сповіщень (alerts) та історичні й аналітичні сутності (історія статусів, журнал подій, зрізи проєкту). Така структура забезпечує повний цикл роботи системи моніторингу: від фіксації фактів та змін до автоматичного виявлення відхилень і донесення інформації до користувача у зручному вигляді.

3.3. Теоретичні основи аналізу критичного шляху

Аналіз критичного шляху є одним із базових інструментів планування та контролю строків виконання проєктів. Метод Critical Path Method (CPM) виник як частина мережевого планування й нині залишається стандартом у системах управління проєктами. Його головною метою є визначення мінімально можливого строку реалізації проєкту та набору задач, затримка яких неминуче призводить до загального відставання. Саме такий набір задач і утворює критичний шлях.

CPM спирається на уявлення проєкту у вигляді орієнтованого ациклічного графа, де вершинами є задачі, а ребрами – залежності між ними. Тривалість задач визначається або на основі експертної оцінки, або шляхом аналізу історичних даних. У моделі припускається, що завдання мають фіксовану тривалість, а їхнє виконання може починатися лише після завершення усіх попередніх робіт, що задаються відповідним типом залежності. [3]

Класичний алгоритм CPM включає два основні етапи. Перший – це прямий прохід (forward pass), під час якого обчислюються найраніші можливі дати початку та закінчення для кожної задачі, виходячи з тривалостей та структури залежностей. Другий – зворотний прохід (backward pass), що спрямований на визначення найпізніших можливих дат виконання, які не змінюють загальний термін проєкту. Порівняння результатів обох проходів дозволяє обчислити ключову величину – резерв часу (slack), який показує, наскільки можна зсунути виконання задачі без впливу на загальний строк проєкту. Для задач критичного шляху значення slack

дорівнює нулю.

Однією з важливих властивостей критичного шляху є його чутливість до змін у тривалості навіть однієї задачі. Якщо будь-яке завдання на критичному шляху займає більше часу від планового, уся тривалість проєкту збільшується на цю ж величину. Таким чином, критичний шлях виступає індикатором зон найбільшого ризику, що потребують постійного моніторингу з боку менеджера проєкту.

У сучасних інформаційних системах CPM аналіз часто розширюється додатковими елементами, такими як урахування затримок (lags), різних типів залежностей (FS, FF, SS, SF), коригування тривалостей на основі фактичного прогресу чи зміна структури мережевої моделі в реальному часі. Для побудови графа та сортування задач використовується алгоритм топологічного впорядкування, який визначає коректну послідовність розрахунку часових показників та унеможливлює формування циклів.

Окрім класичного CPM, у багатьох системах, включно з ProjectPulse, застосовуються спрощені моделі, орієнтовані на підвищення продуктивності обчислень та адаптацію до інкрементальних змін. Одним із таких підходів є MPM (Minimum Project Management), який фокусується на мінімально необхідному наборі метрик для стабільного контролю строків: найраніших і найпізніших дат, повної тривалості шляху та критеріїв критичності. Спрощена модель дає змогу виконувати перерахунки миттєво після змін у структурі задач або статусах виконання, що є важливим для інтерактивних веб-систем. [5]

Таким чином, теоретичні основи аналізу критичного шляху поєднують у собі графові структури, часові моделі, методи мережевого планування та елементи ризик-менеджменту. Цей підхід забезпечує фундамент для організації системи контролю строків виконання проєкту, дозволяє своєчасно виявляти критичні зони та забезпечує основу для побудови аналітичних модулів у веб-додатку ProjectPulse.

3.4. Теорія ризик-менеджменту в інформаційних системах

Ризик-менеджмент є невід'ємною складовою сучасних інформаційних систем, що здійснюють підтримку управління проєктами. У динамічному середовищі, де строки виконання робіт, навантаження на команду та технологічні залежності можуть швидко змінюватись, здатність системи своєчасно виявляти, оцінювати та передавати інформацію про потенційні ризики стає критично важливою. Це забезпечує стабільність проєкту, підвищує прогнозованість результатів та дозволяє зменшити вплив негативних подій.

У класичному розумінні ризик визначається як комбінація ймовірності настання небажаної події та масштабу її впливу на цілі проєкту. У межах інформаційних систем для керування проєктами ризики можуть бути пов'язані з затримками виконання задач, браком ресурсів, технічними обмеженнями, людськими факторами або порушенням послідовності залежностей. Основними принципами ризик-менеджменту є ідентифікація, оцінка, моніторинг, реагування та контроль, причому кожен із цих етапів має бути автоматизованим настільки, наскільки це дозволяє модель і структура даних системи.

Процес ідентифікації ризиків в інформаційній системі базується на аналізі стану задач та їх погодженості з планом. Визначаються ключові індикатори ризиків: прострочення задачі, зростання строків виконання, відсутність прогресу протягом певного часу, блокування через невиконані попередні задачі, зниження доступності виконавців та низький показник продуктивності. Додатковим джерелом є аналіз структури залежностей, що дозволяє визначити критичні точки у мережевій моделі проєкту, де затримка однієї задачі може масштабно вплинути на подальші роботи.

Оцінювання ризиків передбачає використання кількісних або якісних метрик, зокрема ймовірності, впливу, рівня критичності, інтегрального показника severity_score чи категорій (low, medium, high). Такі шкали дозволяють нормалізувати дані та забезпечити однозначну інтерпретацію ризиків як з боку користувача, так і з боку аналітичних алгоритмів. У деяких випадках оцінювання базується на правилах (rule-based approach), коли кожен тип ризику задається

набором формальних умов, які перевіряються під час аналізу стану задач і проєктів.

Моніторинг ризиків у сучасних веб-орієнтованих системах здійснюється в режимі реального часу. При зміні статусів задач, наближенні до дедлайнів, виявленні затримок або зміні доступності виконавців запускаються механізми переоцінки ризиків. Подібний підхід дозволяє системі оперативно оновлювати інформацію та адаптувати прогноз щодо строків завершення проєкту. Одним із ключових інструментів моніторингу є аналіз критичного шляху, який визначає ті задачі, коливання тривалості яких найбільш суттєво впливають на загальний результат. [2]

Етап реагування передбачає формування сповіщень для користувачів та пропозиції можливих дій, наприклад перегляд плану, перерозподіл ресурсів, зміну виконавця або внесення коригувань до графіка. Автоматизовані сповіщення (alerts) є важливою частиною ризик-менеджменту, оскільки забезпечують своєчасне донесення інформації до відповідальних осіб і мінімізують людський фактор у процесі контролю стану проєкту.

Контроль ризиків реалізується за допомогою журналів змін, аналітичних звітів і агрегованих показників (наприклад, SPI та CPI), які дозволяють відслідковувати тенденції та кореляції між факторами ризику. Збереження історії змін у статусах задач та подій у проєкті забезпечує основу для ретроспективного аналізу та вдосконалення моделей прогнозування.

У підсумку теорія ризик-менеджменту в інформаційних системах описує системний підхід до роботи з невизначеністю, що охоплює виявлення, оцінювання, попередження та мінімізацію впливу ризиків на строки та якість виконання проєкту. Реалізація цих принципів у рамках веб-додатку ProjectPulse забезпечує адаптивний підхід до управління проєктними ризиками та підтримує стабільність процесу виконання робіт.

3.5. Теоретичні основи побудови аналітичних модулів

Аналітичні модулі в інформаційних системах управління проєктами виконують ключову роль у підтримці прийняття рішень, прогнозуванні строків, виявленні закономірностей та оцінюванні ефективності виконання робіт. Вони забезпечують перехід від простого накопичення фактів до формування узагальненої інформації, що має стратегічну цінність для менеджера проєкту. Побудова таких модулів спирається на комплекс концепцій: моделі даних, методи обчислення, алгоритмічні підходи, принципи агрегування та візуалізації результатів.

Основу аналітичних модулів становлять первинні дані, які надходять від підсистем задач, статусів, залежностей і ризиків. Аналітична система повинна не лише зчитувати ці дані, а й перетворювати їх у форму, що забезпечує подальші обчислення. Важливими характеристиками первинних даних є їх повнота, структурованість, однозначність та актуальність. У проєктних системах такими даними є дати початку й завершення задач, тривалості, структура мережевих залежностей, статуси виконання, а також події, пов’язані з ризиками та змінами плану. [1]

Один із центральних елементів аналітичних модулів – часові моделі, які формуються на основі планових та фактичних строків. Такі моделі дають змогу розраховувати відхилення, проводити коригування прогнозів та визначати пріоритети задач. У сучасних проєктних системах значного поширення набули моделі, що включають аналіз критичного шляху, резервів часу, коефіцієнтів виконання та показників ефективності. Зокрема, індекс виконання за часом (SPI) та індекс витрат (CPI) відображають співвідношення між плановими та фактичними обсягами виконаної роботи й використовуються для прогнозування завершення проєкту.

Ще одним важливим аспектом побудови аналітичних модулів є агрегування даних. Оскільки первинні записи, такі як статуси задач, події або зміни, можуть бути численними, система має групувати їх у більш високорівневі структури: денні, тижневі або етапні зрізи. Агреговані дані дозволяють виявляти довгострокові

тренди та отримувати узагальнені показники, що необхідні для прийняття рішень на рівні проєкту чи портфеля проєктів. Застосування агрегованих моделей також оптимізує продуктивність аналітичної системи, оскільки виключає потребу виконувати обчислення з нуля при кожній зміні даних.

Суттєвим компонентом аналітичних модулів є механізми виявлення аномалій і відхилень, які працюють на основі формальних правил, статистичних методів або порогових значень. У контексті управління проєктами такими аномаліями можуть бути різке падіння швидкості виконання задач, поява великої кількості блокувань, затримки на критичному шляху або часті зміни статусів одних і тих самих задач. Виявлення аномалій дозволяє системі попереджати про виникнення ризиків і підкреслювати проблемні зони проєкту до того, як вони стануть критичними.

Окрему роль у побудові аналітичних модулів відіграє інтерпретація результатів, тобто перетворення складних числових моделей у зрозумілі для користувача форми. Найпоширенішими методами візуалізації є графіки (burndown chart, velocity chart), теплові карти навантаження, діаграми ризиків, порівняльні таблиці та індикатори KPI. Ефективна візуалізація забезпечує швидке засвоєння інформації та дозволяє користувачу приймати рішення без тривалого аналізу сирих даних. [7]

Важливою властивістю аналітичної підсистеми є інкрементальність, тобто здатність перераховувати результати лише для змінених частин моделі, а не для всього проєкту. Завдяки цьому забезпечується швидка реакція системи на зміни, що є критичним у веб-додатках з інтерактивним оновленням даних. Інкрементальні алгоритми дозволяють аналітичному модулю відстежувати й оновлювати показники після зміни статусів задач, появи ризиків або коригування залежностей.

У підсумку теоретичні основи побудови аналітичних модулів охоплюють структуруизацію даних, методи їх обчислення, агрегування, виявлення аномалій та візуалізацію результатів. Усе це забезпечує формування системи, здатної підтримувати управлінські рішення в реальному часі. У контексті веб-додатку ProjectPulse аналітичні модулі виступають центральним елементом для моніторингу стану проєкту, прогнозування ризиків і відстеження ефективності роботи команди.

3.6. Принципи організації REST API та взаємодії клієнт-сервер

REST API (Representational State Transfer Application Programming Interface) є одним з найпоширеніших підходів до організації взаємодії між клієнтською та серверною частинами веб-орієнтованих інформаційних систем. Для системи моніторингу проєктів, такої як ProjectPulse, REST-підхід дозволяє уніфікувати роботу з сутностями «проєкт», «задача», «ризик», «подія», «сповіщення», забезпечуючи передбачуваний інтерфейс для клієнта та можливість масштабування серверної частини.[13]

Основою REST-архітектури є уявлення даних у вигляді ресурсів, до яких звертаються через унікальні URI. У контексті системи ProjectPulse типові ресурси можуть мати вигляд:

- /api/projects – колекція проєктів;
- /api/projects/{id} – конкретний проєкт;
- /api/projects/{id}/tasks – задачі певного проєкту;
- /api/projects/{id}/risks – пов’язані ризики;
- /api/alerts – події та автоматизовані сповіщення.

Кожна операція над ресурсом реалізується стандартними методами HTTP: GET (отримання), POST (створення), PUT/PATCH (оновлення), DELETE (видалення). Наприклад, створення нового проєкту в ProjectPulse може відповідати запиту POST /api/projects, а отримання списку задач – GET /api/projects/{id}/tasks.

Нижче наведено спрощений приклад оголошення REST-маршрутів на сервері (Node.js + Express) для роботи з сутністю «проєкт»:

```
// server/routes/projects.ts
import { Router } from "express";
const router = Router();

// Отримати список проєктів
router.get("/", async (req, res) => {
  const projects = await db.project.findMany();
  res.json(projects);
```

```
});

// Створити новий проєкт
router.post("/", async (req, res) => {
  const { name, deadline } = req.body;
  const project = await db.project.create({ data: { name, deadline } });
  res.status(201).json(project);
});

// Отримати проєкт за ідентифікатором
router.get("/:id", async (req, res) => {
  const project = await db.project.findUnique({ where: { id: req.params.id } });
  if (!project) return res.status(404).json({ message: "Project not found" });
  res.json(project);
});

export default router;
```

Такий підхід ілюструє кілька важливих принципів REST-організації:

- однозначність ресурсів за URI;
- використання статус-кодів HTTP (201 при створенні ресурсу, 404 при відсутності тощо);
- повернення машиночитних відповідей у форматі JSON.

Важливим аспектом взаємодії клієнт–сервер є контракти даних (DTO, Data Transfer Object), які описують структуру запитів та відповідей. Для системи моніторингу це гарантує, що фронтенд і бекенд однаково інтерпретують сутності проєктів, задач та ризиків. Наприклад, DTO для задачі може включати ідентифікатор, назву, статус, плановий та фактичний час виконання:

```
// server/dto/task.dto.ts
export interface TaskDto {
  id: string;
  projectId: string;
  title: string;
  status: "planned" | "in_progress" | "done" | "blocked";
  plannedHours: number;
  actualHours?: number;
  dueDate?: string;    // ISO-формат
```

```
}
```

На клієнтській частині (React) взаємодія з REST API зазвичай реалізується через fetch або спеціалізовані HTTP-клієнти. Важливим є не лише отримання даних, але й обробка помилок і відображення їх користувачеві. Нижче наведено приклад запиту для завантаження задач певного проєкту:

```
// client/api/tasks.ts
export async function fetchProjectTasks(projectId: string) {
  const res = await fetch(`/api/projects/${projectId}/tasks`);
  if (!res.ok) {
    throw new Error(`Failed to load tasks: ${res.status}`);
  }
  return (await res.json()) as TaskDto[];
}
```

Для аналітичної підсистеми проєкту важливими є також механізми фільтрації, сортування та пагінації, які реалізуються через параметри запиту:

- GET /api/tasks?projectId=...&status=in_progress
- GET /api/alerts?projectId=...&page=2&pageSize=20&sort=createdAt_desc.

Це дозволяє фронтенду завантажувати лише потрібні фрагменти даних та зменшувати навантаження на сервер. [14-15]

Ще одним принциповим аспектом REST-взаємодії є статлесність (відсутність стану сесії на стороні сервера). У системі ProjectPulse це означає, що кожен запит повинен містити всю необхідну інформацію для його обробки, наприклад токен авторизації в заголовку Authorization. Таким чином, сервер може масштабуватися горизонтально, а окремі екземпляри застосунку не залежать від локального стану сесій. [16]

Для задач моніторингу статусу проєктів важливим є також режим близький до реального часу. На рівні REST API це зазвичай реалізується або періодичним опитуванням (polling), або поєднанням REST із потоковими механізмами (SSE, WebSocket) для оновлення аналітичних панелей. У базовому варіанті клієнт може регулярно викликати:

```
// client/hooks/useProjectStatus.ts
useEffect(() => {
  let cancelled = false;
```

```
  async function load() {
    try {
      const res = await fetch(`/api/projects/${projectId}/summary`);
      if (!res.ok) return;
      const summary = await res.json();
      if (!cancelled) setSummary(summary);
    } finally {
      // планове повторне опитування
      if (!cancelled) setTimeout(load, 15000);
    }
  }

  load();
  return () => { cancelled = true; };
}, [projectId]);
```

Такий підхід дозволяє оновлювати інформаційні панелі з інтервалом, достатнім для моніторингу ризиків та відхилень без значного навантаження на сервер.

Отже, принципи організації REST API у системі ProjectPulse охоплюють:

- подання даних у вигляді ресурсів із чіткими URI;
- використання стандартних методів HTTP та кодів стану;
- формалізацію структур запитів та відповідей через DTO;
- підтримку фільтрації, сортування та пагінації для аналітичних запитів;
- статлесність взаємодії та можливість масштабування;
- підтримку сценаріїв «наближеного реального часу» через регулярне опитування або потокові механізми. [17]

Дотримання цих принципів забезпечує узгоджену роботу клієнтської та серверної частин, спрощує розвиток системи і створює надійну основу для реалізації аналітичних модулів, що моніторять статус проєктів, ризики та відхилення від планових графіків.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Загальна характеристика програмного продукту ProjectPulse

Програмний продукт ProjectPulse є веб-орієнтованою інформаційною системою для моніторингу стану проєктів, контролю виконання задач, автоматичного виявлення ризиків та аналізу відхилень від планових графіків. Система розроблена як інтерактивна платформа, що поєднує інструменти планування, відстеження прогресу, аналізу критичного шляху, формування ризиків і автоматизованих сповіщень, що робить її придатною для використання як у невеликих командах, так і в середовищі корпоративного управління.

Основним призначенням ProjectPulse є забезпечення користувача прозорим та структурованим уявленням про стан проєкту в реальному часі. На відміну від традиційних таблиць або статичних планів, система надає динамічний інтерфейс, де зміни у задачах, залежностях або статусах негайно впливають на аналітику, ризики та прогнозовані строки завершення. Завдяки цьому ProjectPulse дозволяє виявляти потенційні затримки на ранніх етапах та забезпечувати ефективне управління ресурсами.

Архітектура програмного продукту побудована на клієнтсько-серверній моделі, де клієнтська частина реалізована з використанням бібліотеки React, а серверна – на основі Node.js та Express. Обмін даними між компонентами здійснюється через REST API, що забезпечує гнучкість, масштабованість та простоту розширення функціональності. У якості бази даних використовується SQLite, що оптимально підходить для локальних або невеликих розгортань та забезпечує швидку роботу у режимі одночасного доступу.

Функціональні можливості ProjectPulse охоплюють широке коло інструментів для роботи з проєктами. Система дозволяє створювати та редагувати проєкти, формувати перелік задач, задавати їхні планові та фактичні дати, призначати виконавців та відслідковувати статуси виконання. Користувач може визначати залежності між задачами та візуалізувати їх як мережеву структуру, що забезпечує можливість аналізу критичного шляху та резервів часу.

Окреме місце у системі займають механізми автоматичного виявлення ризиків, що аналізують стан задач і визначають ситуації, які можуть призвести до відхилень у графіку. При настанні подій, що відповідають умовам ризиків (прострочення, наближення дедлайну, відсутність прогресу, блокування), система створює відповідні записи в модулі ризиків та генерує сповіщення для користувачів. Такий підхід мінімізує втручання менеджера та підвищує ефективність контролю планових операцій.

ProjectPulse також містить розвинений модуль аналітики, який включає ключові показники стану проєкту, графіки забігу робіт (Burndown chart), швидкості виконання (Velocity chart), карту навантаження команди та інші індикатори. Це дозволяє користувачу отримати цілісне уявлення про продуктивність команди та визначити критичні ділянки проєкту.

Інтерфейс користувача орієнтований на зручність та швидкість взаємодії. Kanban-дошка забезпечує drag-and-drop керування задачами, панель сповіщень відображає активні ризики, а окремий модуль бази знань дає змогу документувати важливу інформацію, пов’язану з проєктом. Таким чином, система поєднує інструменти планування, документації та моніторингу в єдиному середовищі.

У підсумку ProjectPulse є сучасним, гнучким та функціонально насиченим рішенням для керування проєктами, що поєднує методологічні принципи мережевого планування, ризик-менеджменту та аналітики у вигляді інтуїтивно зрозумілого веб-застосунку. Система забезпечує комплексний підхід до оцінки та мінімізації затримок, а також підтримує прийняття управлінських рішень на основі достовірних даних і в реальному часі.

4.2. Технологічний стек і середовище розробки

Розробка програмного продукту ProjectPulse базується на сучасних інструментах та технологіях, що забезпечують високу продуктивність, масштабованість і зручність підтримки коду. Вибір стеку був обумовлений необхідністю створити інтерактивний веб-додаток з швидким інтерфейсом,

надійним серверним API та гнучкою моделлю даних. У цьому підпункті подано опис основних технологій, які використовуються на клієнтській та серверній частинах, а також інструментів, що формують середовище розробки.

Фронтенд-технології

Клієнтська частина програмного продукту реалізована з використанням таких технологій:

1. React 19

React є базовою бібліотекою для побудови інтерфейсу користувача. Вона забезпечує:

- компонентну структуру;
- швидке оновлення UI завдяки віртуальному DOM;
- ефективну організацію стану додатка;
- просте повторне використання елементів.

2. TypeScript

TypeScript використовується для типізації, що дозволяє:

- зменшити кількість помилок під час розробки;
- забезпечити передбачуваність поведінки компонентів;
- підвищити масштабованість проєкту.

3. Vite

Vite виконує роль збирача та локального серверу для фронтенду. Його переваги:

- моментальний запуск дев-середовища;
- гаряче оновлення модулів (HMR);
- оптимізована production-збірка.

4. React Router DOM

Забезпечує маршрутизацію між сторінками:

- /projects, /tasks, /analytics, /profile,
- підтримує захищені маршрути для авторизованих користувачів.

5. @hello-pangea/dnd

Використовується для реалізації drag-and-drop взаємодії на Kanban-дошці.

6. react-markdown

Дозволяє відображати статті бази знань у вигляді Markdown-контенту.

7. date-fns

Служить для роботи з датами: обчислення тривалостей, різниць між датами, форматування.

Бекенд-технології

Серверна частина проєкту побудована на стеку JavaScript/TypeScript та включає:

1. Node.js

Node.js використовується як серверне середовище виконання:

- асинхронна модель обробки запитів;
- висока продуктивність;
- можливість використовувати один стек (JS/TS) на клієнті та сервері.

2. Express 5

Express – легковаговий веб-фреймворк, що забезпечує:

- організацію REST API;
- гнучкість маршрутизації;
- інтеграцію middleware (включно з auth).

Приклад оголошення маршруту:

```
app.get("/api/projects", async (req, res) => {
  const projects = store.listProjects();
  res.json(projects);
});
```

3. TypeScript (на сервері)

Забезпечує статичну типізацію та знижує ймовірність помилок у логіці API.

4. better-sqlite3

База даних SQLite обрана через:

- простоту розгортання;
- продуктивність у локальних та невеликих системах;
- можливість зберігати всю інформацію в одному файлі.

Приклад запиту:

```
const stmt = db.prepare("SELECT * FROM tasks WHERE projectId = ?");
const tasks = stmt.all(projectId);
```

5. bcrypt

Бібліотека для безпечного хешування паролів.

6. jsonwebtoken

Використовується для генерації та перевірки JWT-токенів авторизації:

```
const token = jwt.sign({ userId }, JWT_SECRET, { expiresIn: "7d" });
```

7. Zod

Бібліотека валідації входних даних, що забезпечує:

- запобігання некоректним запитам;
- захист від неконсистентних даних.

Середовище розробки

Для організації робочого процесу розробки використовуються такі інструменти:

1. npm + concurrently

Команда запуску фронтенду та бекенду одночасно:

```
"dev": "concurrently \"npm run dev --prefix frontend\" \"npm run dev --prefix backend\""
```

2. ts-node-dev

Забезпечує автоматичне перезапускання серверної частини при змінах коду.

3. Git та GitHub/GitLab

Система контролю версій забезпечує:

- відстеження історії змін;
- роботу в окремих гілках;
- швидке розгортання оновлень.

4. Visual Studio Code

Виступає основним інструментом для розробки:

- інтеграція з TypeScript;
- розширення для React, Node.js, SQLite;
- можливість швидкого форматування коду.

Середовище виконання та розгортання

1. Локальне середовище

Проєкт запускається командою:

npm run dev

Frontend – http://localhost:5173

Backend – http://localhost:3001

2. Production-режим

Для підготовки production-збірки використовуються команди:

npm run build

npm run start:prod

У цьому режимі:

- фронтенд компілюється у статичні файли,
- сервер запускається в оптимізованому режимі,
- базу даних можна перенести як файл SQLite.

Використання сучасного технологічного стеку дозволяє ProjectPulse поєднати високу інтерактивність інтерфейсу та оптимальну продуктивність серверної частини. Завдяки цьому система є гнучкою, швидкодіючою, легкою в підтримці та придатною для подальшого масштабування.

4.3. Архітектура програмної системи

Архітектура програмної системи ProjectPulse побудована за багатошаровим підходом та подана на рисунку 4.1. Така структура дозволяє чітко розмежувати відповідальність між представленням, бізнес-логікою, доступом до даних та зовнішніми інтеграціями, спрощує модифікацію окремих компонентів та підвищує масштабованість рішення (див. рис. 4.1).

Рисунок 4.1 – Детальна архітектура програмної системи PROJECPULSE

На рівні представлення (Presentation Layer) розташована односторінкова React-SPA, що реалізує користувацький інтерфейс системи. Окремими блоками виділено модулі Project & Task Views (форми перегляду та редагування проєктів, дошки Kanban, часові лінії), Analytics Dashboard (візуалізація метрик критичного шляху, залишкового обсягу робіт, індикаторів затримок) та Auth & Profile (автентифікація користувачів, управління ролями та профілями). На цьому рівні зосереджена тільки робота з інтерфейсом, стан зберігається у React-контексті та керується за допомогою хуків.

Комунікація клієнта з сервером відбувається через шар API Gateway & Application Layer за допомогою HTTPS-запитів до REST-інтерфейсу. Компонент REST API Gateway містить контролери для операцій над проєктами, задачами, залежностями та ризиками, виконує валідацію вхідних даних на основі DTO-схем і повертає стандартизовані HTTP-коди. Модуль Auth Middleware перевіряє JWT-токени, витягує контекст користувача та застосовує перевірку прав доступу. Підсистема Rate & Logging реалізує обмеження частоти запитів, аудит ключових дій та трасування запитів за допомогою ідентифікаторів, що спрощує діагностику та моніторинг роботи сервера.

Основна предметна логіка зосереджена у Domain & Business Logic Layer. Project Service відповідає за життєвий цикл проєктів (створення, оновлення, зміна статусів, керування учасниками та їх ролями). Task & Dependency Service реалізує CRUD-операції над задачами, зберігає та перевіряє типи залежностей (FS, SS, FF), контролює цілісність часових обмежень. Analytics Engine будує орієнтований граф робіт, виконує прямий та зворотний проходи для обчислення ранніх та пізніх термінів (ES, EF, LS, LF) і запасів часу, що є основою для аналізу критичного шляху. Risk & Alerts Engine застосовує набір правил для виявлення ризикових ситуацій (прострочені задачі, блокуючі залежності, перевищення запасу часу), присвоює пріоритети ризикам і формує модель сповіщення. Модуль Notification Service організовує чергу задач, шаблони повідомлень та канали доставки (електронна пошта, вебхуки), забезпечуючи асинхронне інформування користувачів.

Доступ до постійних даних реалізовано у Data Access & Storage Layer. Компонент Repositories / ORM інкапсулює запити до бази даних і надає доменному шару високорівневий інтерфейс для роботи з таблицями проєктів, задач, графових залежностей та журналом ризиків. SQLite Database використовується як вбудоване сховище, у якому зберігаються сутності проєктів, задач, матриця залежностей та обчислені метрики MPM/CPM. Окремо виділено File Storage для зберігання вкладених файлів (документація до задач, супровідні матеріали), на які посилаються записи у базі. Модуль Background Jobs виконує перерахунок

аналітики, надсилання нагадувань та очищення застарілих журналів у фоновому режимі, щоб не блокувати основні користувацькі операції.

Нарешті, шар зовнішніх інтеграцій (External Integrations) вміщує сервіси, з якими система може взаємодіяти за межами основного контуру. E-mail Provider забезпечує відправлення листів через SMTP або API-інтерфейс, використовується модулем сповіщень для повідомлення про ризики та прострочені задачі. Блок Webhook / Chat зарезервований для інтеграції з корпоративними месенджерами (Slack, Microsoft Teams тощо) та системами зовнішнього моніторингу, що дозволить у майбутньому розширити екосистему ProjectPulse без змін у внутрішній архітектурі.

Таким чином, запропонована архітектура демонструє чітке логічне розмежування відповідальностей, підтримує послідовний потік даних від інтерфейсу користувача до сховища та зовнішніх сервісів і забезпечує необхідні умови для подальшого масштабування системи, додавання нових аналітичних модулів та каналів сповіщення.

4.4. Реалізація серверної частини

Серверна частина програмного продукту ProjectPulse реалізована на платформі Node.js з використанням фреймворку Express та мови TypeScript. Такий вибір дає можливість поєднати неблокуючу модель обробки запитів з статичною типізацією, що знижує кількість помилок на етапі розробки та спрощує супровід коду.

Архітектура бекенду побудована за шаруваним підходом:

- рівень HTTP / контролерів – приймає запит, викликає відповідний сервіс і формує відповідь;
- рівень бізнес-логіки (services) – втілює правила роботи з проєктами, задачами, залежностями та ризиками;
- рівень доступу до даних (repositories) – інкапсулює роботу з базою даних SQLite та SQL-запитами.

Такий поділ дозволяє змінювати реалізацію зберігання або алгоритми обробки без впливу на інші рівні.

Ініціалізація HTTP-сервера та middleware

Основна точка входу до серверної частини – файл index.ts. У ньому виконується створення екземпляра Express, підключення проміжних обробників (middleware) та реєстрація маршрутів API.

```
import express from 'express';
import cors from 'cors';
import { json } from 'body-parser';
import router from './routes';
import { authMiddleware } from './auth';

const app = express();

// Дозвіл крос-доменних запитів від фронтенду
app.use(cors());

// Автоматичний розбір JSON-запитів
app.use(json());

// Публічні маршрути (реєстрація, логін)
app.use('/api', router.public);

// Захищені маршрути (проєкти, задачі, аналітика)
app.use('/api', authMiddleware, router.protected);

// Глобальний обробник помилок
app.use((err: Error, req, res, next) => {
  console.error('[Error]', err.message);
  res.status(500).json({ error: 'Internal server error' });
});

app.listen(3001, () =>
  console.log('Backend running on http://localhost:3001')
);
```

У цьому фрагменті продемонстровано кілька важливих рішень:

- уся робота з JSON зосереджена в одному middleware json(), що спрощує обробку тіла запиту;
- публічні та захищені маршрути розділені, а до другого набору маршрутів автоматично додається authMiddleware – це унеможливлює випадкове створення «відкритого» ендпоінта, який повинен вимагати автентифікацію;
- глобальний обробник помилок гарантує, що неконтрольовані винятки не завершать процес Node.js і перетворюються на контрольовану відповідь 500.

Організація маршрутів REST API

Наступний ключовий елемент – маршрутизатор routes.ts, який логічно групує ендпоінти за доменами: проєкти, задачі, залежності, аналітика.

```
import { Router } from 'express';
import * as projects from './controllers/projects';
import * as tasks from './controllers/tasks';
import * as deps from './controllers/dependencies';
import * as analytics from './controllers/analytics';

const publicRouter = Router();
const protectedRouter = Router();

// Публічні ендпоінти автентифікації
publicRouter.post('/auth/login', ...);
publicRouter.post('/auth/register', ...);

// Операції з проєктами
protectedRouter.get('/projects', projects.list);
protectedRouter.post('/projects', projects.create);
protectedRouter.get('/projects/:id', projects.getOne);

// Операції з задачами
protectedRouter.get('/tasks', tasks.list);
protectedRouter.post('/tasks', tasks.create);
protectedRouter.patch('/tasks/:id', tasks.update);

// Залежності між задачами
protectedRouter.post('/projects/:id/dependencies', deps.create);
protectedRouter.get('/projects/:id/dependencies', deps.list);
```

```
// Аналітика критичного шляху та MPM-метрики
protectedRouter.post('/projects/:id/mpm', analytics.recalculate);
protectedRouter.get('/projects/:id/mpm', analytics.getMetrics);

export default { public: publicRouter, protected: protectedRouter };
```

Кожен маршрут делегує роботу відповідному контролеру. Контролер, у свою чергу, не містить SQL-запитів чи складної логіки – лише зчитує параметри, викликає сервіс і повертає результат. Це спрощує тестування та підвищує читабельність коду.

Рівень доступу до даних (Repositories)

Зберігання даних реалізовано на основі SQLite з використанням бібліотеки better-sqlite3. Логіка роботи з таблицями зосереджена у репозиторіях. Наприклад, репозиторій задач:

```
import { db } from '../db';

export const TaskRepo = {
 list(projectId: string) {
  return db.prepare(
   `SELECT * FROM tasks WHERE projectId = ? ORDER BY "order" ASC`
  ).all(projectId);
 },

 create(task) {
  db.prepare(
   `INSERT INTO tasks
   (id, projectId, title, description, status, startDatePlanned, endDatePlanned)
   VALUES (@id, @projectId, @title, @description, @status, @startDatePlanned, @endDatePlanned)`
  ).run(task);
 },

 update(id, patch) {
  const fields = Object.keys(patch)
   .map(k => `${k} = @${k}`)
   .join(', ');
```

```
    db.prepare(`UPDATE tasks SET ${fields} WHERE id = @id`)
      .run({ id, ...patch });
  }
};
```

Репозиторій виконує дві функції:

1. Інкапсулює SQL-запити, щоб інші частини системи працювали з задачами як з об’єктами, а не з рядками SQL.
2. Гарантує єдину точку зміни для структури таблиць – у випадку модифікації схеми достатньо змінити SQL у одному місці.

Реалізація бізнес-логіки в сервісах

Сервіси описують «мову» домену проєктного моніторингу: створення проєктів, зміна статусів задач, розрахунок метрик, фіксація ризиків тощо. Наприклад, сервіс задач:

```
import { TaskRepo } from '../repositories/tasks';
import { v4 as uuid } from 'uuid';

export const TaskService = {
  list(projectId: string) {
    return TaskRepo.list(projectId);
  },

  create(projectId: string, data) {
    const task = {
      id: uuid(),
      projectId,
      title: data.title,
      description: data.description ?? '',
      status: 'planned',
      startDatePlanned: data.startDatePlanned,
      endDatePlanned: data.endDatePlanned
    };

    TaskRepo.create(task);
    return task;
  },
```

```
  update(id: string, data) {
    TaskRepo.update(id, data);
    return TaskRepo.get(id);
  }
};
```

У цьому фрагменті видно, що:

- саме сервіс визначає початковий статус задачі (planned), а не контролер;
- генерується унікальний ідентифікатор задачі;
- перед створенням задачі можна виконати додаткову перевірку (наприклад, коректність дат або наявність проєкту) – усе це концентрується у бізнес-шарі.

Обчислення критичного шляху та MPM-метрик

Окрема частина серверної логіки – модуль аналітики, який реалізує алгоритм розрахунку критичного шляху та часових резервів на основі залежностей між задачами. Спрощений фрагмент:

```
export function calculateProjectMPM(projectId: string) {
  const tasks = TaskRepo.list(projectId);
  const deps = DependencyRepo.list(projectId);

  const graph = buildGraph(tasks, deps);  // перетворення списків у орієнтований граф

  if (hasCycles(graph)) {
    throw new Error('Dependency graph contains cycles');
  }

  const sorted = topologicalSort(graph);  // топологічне сортування
  const forward = forwardPass(sorted, graph);   // ранні терміни ES/EF
  const backward = backwardPass(sorted, graph, forward); // пізні терміни LS/LF

  return buildMetrics(forward, backward); // резерви часу та критичний шлях
}
```

Цей модуль:

- інтерпретує задачі як вершини графа, а залежності – як орієнтовані ребра;
- виконує топологічне сортування, щоб забезпечити коректний порядок проходу;
- обчислює ранні та пізні терміни виконання задач, а також визначає, які з них

формують критичний шлях (резерв дорівнює нулю).

Отримані метрики передаються на фронтенд і використовуються для візуалізації вузьких місць та затримок проєкту.

Виявлення та фіксація ризиків

Модуль ризик-менеджменту автоматично аналізує задачі й створює записи про ризики, якщо виявлено відхилення від плану. Приклад спрощеної функції:

```
import { differenceInDays, isAfter } from 'date-fns';
import { RiskRepo } from '../repositories/risks';
import { v4 as uuid } from 'uuid';

export function checkTaskRisk(task) {
  const today = new Date();
  const end = new Date(task.endDatePlanned);

  const overdue = !task.endDateActual && isAfter(today, end);
  const nearDeadline = differenceInDays(end, today) <= 1;

  if (overdue) {
    RiskRepo.create({
      id: uuid(),
      projectId: task.projectId,
      taskId: task.id,
      title: 'Просрочена задача',
      level: 'high',
      createdAt: new Date().toISOString()
    });
  } else if (nearDeadline) {
    RiskRepo.create({
      id: uuid(),
      projectId: task.projectId,
      taskId: task.id,
      title: 'Наближення дедлайну',
      level: 'medium',
      createdAt: new Date().toISOString()
    });
  }
}
```

По суті, сервер постійно порівнює поточну дату з плановим дедлайном і автоматично створює ризики двох рівнів: високий (прострочена задача) і середній (наближення дедлайну). Це дозволяє формувати стрічку ризиків без ручного втручання користувачів.

Автентифікація та захист API

Доступ до більшості ендпоінтів обмежений за допомогою JSON Web Token (JWT). Middleware автентифікації зчитує токен з заголовка Authorization і додає ідентифікатор користувача до об’єкта запиту:

```
import jwt from 'jsonwebtoken';

export function authMiddleware(req, res, next) {
  const raw = req.headers.authorization;

  if (!raw) {
    return res.status(401).json({ error: 'No token' });
  }

  try {
    const token = raw.replace('Bearer ', '');
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.userId = decoded.userId;
    next();
  } catch {
    return res.status(401).json({ error: 'Invalid token' });
  }
}
```

Це рішення дозволяє:

- централізовано контролювати доступ до захищених ресурсів;
- не дублювати логіку перевірки токена в кожному контролері;
- легко розширити модель прав доступу (роль PM / учасник команди) у майбутньому.

Фонові задачі та періодична аналітика

Для періодичного оновлення аналітики та ризиків використовується простий

механізм фонових задач на основі setInterval. Сервер через певні інтервали часу проходить по всіх проєктах, перевіряє задачі та оновлює записи про ризики:

```
setInterval(() => {
  const projects = ProjectRepo.list();

  for (const p of projects) {
    const tasks = TaskRepo.list(p.id);
    tasks.forEach(checkTaskRisk);
  }
}, 60_000); // раз на хвилину
```

Хоча це не є повноцінним черговим механізмом (message queue), такий підхід достатній для навчального проєкту та дозволяє продемонструвати принцип background-обробки без блокування основних HTTP-запитів.

У серверній частині ProjectPulse реалізовано повноцінний цикл обробки даних: від прийому HTTP-запиту, перевірки прав доступу та виконання бізнес-логіки до зберігання в базі даних, обчислення аналітичних показників і автоматичного формування ризиків. Поєднання шарової архітектури, патерну «репозиторій», використання JWT-автентифікації та алгоритмів аналізу критичного шляху дозволяє отримати гнучку й розширювану серверну платформу, яка є ядром програмного продукту ProjectPulse.

4.5. Реалізація клієнтської частини

Клієнтська частина програмного продукту ProjectPulse реалізована як односторінковий веб-додаток (SPA), створений на основі React 19, TypeScript та React Router. Візуальна частина оформлена за принципами сучасних дизайн-систем із використанням кастомних UI-компонентів, а управління станом здійснюється через локальні хуки та контекст.

Основні цілі клієнтської частини:

- забезпечити інтуїтивний доступ до проєктів, задач, ризиків та аналітики;

- реалізувати динамічну візуалізацію критичного шляху, метрик виконання та навантаження;
- інтегрувати реальний бекенд через REST-запити, при цьому гарантувати безперервний UX;
- забезпечити модульність і розширюваність інтерфейсу.

Нижче наведено ключові елементи реалізації клієнтської частини з поясненням.

Структурна організація клієнтського додатка

Кодова архітектура фронтенду побудована за класичною схемою:

src/

├── api/ → HTTP-клієнт, запити до бекенда

├── components/ → UI-компоненти (таблиці, модалки, індикатори)

├── pages/ → сторінки: Dashboard, Tasks, Projects, Analytics

├── hooks/ → кастомні React-хуки (useTasks, useProjects)

├── context/ → глобальний стан (AuthContext, ProjectContext)

├── utils/ → допоміжні функції (дата, форматування)

Це дозволяє легко розширювати додаток та ізолювати логіку.

HTTP-клієнт і взаємодія з бекендом

Для комунікації з сервером використовується мінімалістичний wrapper над fetch, який автоматично додає токен та обробляє помилки.

```
// src/api/client.ts
export async function api(path: string, options: RequestInit = {}) {
  const token = localStorage.getItem('token');

  const response = await fetch(`http://localhost:3001/api${path}`, {
    ...options,
    headers: {
      'Content-Type': 'application/json',
      ...(token && { Authorization: `Bearer ${token}` }),
      ...options.headers
    }
  });
```

```
  if (!response.ok) {
    const err = await response.json().catch(() => ({}));
    throw new Error(err.error || 'Server error');
  }

  return response.json();
}
```

Пояснення

- автоматично додається токен (Bearer …) – тому кожен компонент не повинен дублювати логіку авторизації;
- у разі помилки викликається виняток, що дозволяє централізовано обробляти помилки в компонентах;
- функція універсальна та застосовується у всіх запитах (GET, POST, PATCH, DELETE).

Приклад запиту до API: отримання списку проєктів

```
// src/api/projects.ts
import { api } from './client';

export function fetchProjects() {
  return api('/projects');
}

export function createProject(data) {
  return api('/projects', {
    method: 'POST',
    body: JSON.stringify(data)
  });
}
```

Перевага підходу – читабельність та мінімум дублювань.

Контекст автентифікації

Керування авторизованим станом реалізовано через AuthContext – глобальний контекст, який зберігає токен і дані користувача.

```
// src/context/AuthContext.tsx
import { createContext, useState, useEffect } from 'react';
import { loginRequest } from '../api/auth';
```

```
export const AuthContext = createContext(null);

export function AuthProvider({ children }) {
 const [token, setToken] = useState(localStorage.getItem('token'));
 const [user, setUser] = useState(null);

 const login = async (email: string, password: string) => {
  const res = await loginRequest(email, password);
  setToken(res.token);
  localStorage.setItem('token', res.token);
  setUser(res.user);
 };

 const logout = () => {
  localStorage.removeItem('token');
  setToken(null);
  setUser(null);
 };

 return (
  <AuthContext.Provider value={{ token, user, login, logout }}>
   {children}
  </AuthContext.Provider>
 );
}
```

Пояснення

- усі компоненти автоматично отримують доступ до функцій login та logout;
- токен зберігається у localStorage, що дозволяє зберігати авторизацію після перезавантаження сторінки;
- контекст єдина точка доступу до даних користувача, що спрощує управління станом.

Приклад реалізації сторінки авторизації

```
// src/pages/Login.tsx
import { useContext, useState } from 'react';
```

```
import { AuthContext } from '../context/AuthContext';

export function LoginPage() {
  const { login } = useContext(AuthContext);
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const submit = async (e) => {
    e.preventDefault();
    await login(email, password);
  };

  return (
    <form onSubmit={submit} className="auth-card">
      <h1>ProjectPulse</h1>
      <input value={email} onChange={e => setEmail(e.target.value)} placeholder="Email" />
      <input type="password" value={password} onChange={e => setPassword(e.target.value)}
placeholder="Пароль" />
      <button type="submit">Увійти</button>
    </form>
  );
}
```

Пояснення

- компонент не працює з API напряму, а використовує контекст login – це важливий принцип розділення відповідальностей;
- дані форми зберігаються у локальному стані;
- після успішного входу користувач автоматично перенаправляється в Dashboard через ProtectedRoute.

Реалізація сторінки задач (Kanban-дошка)

Один із центральних UI-модулів – Kanban-дошка для задач.

```
// src/pages/Tasks.tsx
import { useTasks } from '../hooks/useTasks';
import { TaskColumn } from '../components/TaskColumn';

export function TasksPage() {
  const { planned, inWork, blocked, done, createTask } = useTasks();
```

```
 return (
  <div className="tasks-board">
   <TaskColumn title="Заплановані" tasks={planned} status="planned" onAdd={createTask} />
   <TaskColumn title="В роботі" tasks={inWork} status="inWork" onAdd={createTask} />
   <TaskColumn title="Заблоковані" tasks={blocked} status="blocked" />
   <TaskColumn title="Завершені" tasks={done} status="done" />
  </div>
 );
}
```

Пояснення

- useTasks() – кастомний хук, що інкапсулює логіку отримання, оновлення та групування задач;
- кожна колонка є окремим компонентом TaskColumn, що спрощує підтримку;
- UI повністю реактивний: зміна статусу задачі автоматично оновлює інші колонки.

Кастомний хук для роботи з задачами

```
// src/hooks/useTasks.ts
import { useEffect, useState } from 'react';
import { fetchTasks, updateTask, createTask as apiCreate } from '../api/tasks';

export function useTasks() {
 const [tasks, setTasks] = useState([]);

 useEffect(() => {
  fetchTasks().then(setTasks);
 }, []);

 const createTask = async (data) => {
  const newTask = await apiCreate(data);
  setTasks(prev => [...prev, newTask]);
 };

 const changeStatus = async (id, status) => {
  await updateTask(id, { status });
  setTasks(prev =>
```

```
    prev.map(t => t.id === id ? { ...t, status } : t)
   );
  };

  return {
   planned: tasks.filter(t => t.status === 'planned'),
   inWork: tasks.filter(t => t.status === 'inWork'),
   blocked: tasks.filter(t => t.status === 'blocked'),
   done: tasks.filter(t => t.status === 'done'),
   createTask,
   changeStatus
  };
}
```

Пояснення

- логіка групування задач винесена за межі UI, що значно полегшує тестування;
- хук сам відповідає за синхронізацію зі сервером;
- зміни стану виконуються оптимістично (update на UI → API), що робить інтерактивність миттєвою.

Компонент візуалізації аналітики (Burndown Chart)

```
import { Line } from 'react-chartjs-2';

export function BurndownChart({ plan, fact, labels }) {
 return (
  <Line
   data={{
    labels,
    datasets: [
     { label: 'План', data: plan, borderColor: '#4da3ff' },
     { label: 'Факт', data: fact, borderColor: '#ff4d4d' }
    ]
   }}
   options={{ responsive: true, tension: 0.2 }}
  />
 );
}
```

Пояснення

- використовується chart.js для побудови графіків з індивідуальним дизайном;
- план і факт виконання задач відображаються двома лініями;
- компонент є повністю незалежним – можна використовувати в будь-якій частині системи.

Діаграма навантаження команди (Workload Heatmap)

Реалізована через просту матричну візуалізацію:

```
export function Heatmap({ matrix }) {
 return (
  <div className="heatmap">
   {matrix.map((row, i) => (
    <div key={i} className="heatmap-row">
     {row.map((v, j) => (
      <div key={j} className={`cell intensity-${v}`}>
       {v}
      </div>
     ))}
    </div>
   ))}
  </div>
 );
}
```

Пояснення

- інтенсивність навантаження відображається кольором через CSS-класи (intensity-0 … intensity-4);
- модуль дозволяє бачити перевантажених членів команди та потенційні ризики.

Клієнтська частина ProjectPulse реалізує повноцінний SPA-функціонал, що включає:

- автентифікацію користувачів,
- управління проєктами,
- Kanban-дошку для задач,
- систему залежностей,
- перегляд і аналіз ризиків,

- модуль аналітики (Burndown, Velocity, Heatmap),
- базу знань.

Використання React, TypeScript, контекстів і кастомних хуків дозволило створити масштабований, модульний та високопродуктивний інтерфейс, який природно інтегрується з REST-API серверної частини та забезпечує зручну роботу для менеджерів, аналітиків і учасників проєктів.

4.6. Тестування функціональності системи

Тестування функціональності веб-додатку ProjectPulse проводилося для перевірки коректності обробки даних серверною частиною, правильності роботи алгоритмів аналізу ризиків та відхилень, а також стабільності REST API при виконанні послідовних сценаріїв, характерних для реального використання системи.

Оскільки система підтримує автоматичне виявлення критичних затримок, перерахунок метрик MPM та формування сповіщень, особливу увагу було приділено саме цим механізмам. Тестування виконувалося у форматі наскрізних перевірок (E2E), де з тестового клієнта надсилалися HTTP-запити до локального інстансу серверу, після чого результати порівнювалися з очікуваними.

Основними задачами тестування були:

- перевірка створення проєкту та задач;
- коректність оновлення фактичного прогресу;
- виявлення критичного відставання при недостатньому темпі виконання задач;
- відсутність ризику при випередженні графіку;
- стабільність бекенду під час виконання послідовних операцій;
- формування та збереження записів у таблиці сповіщень.

Під час прогону були використані тести, що емулювали типові запити до бекенду, включаючи оновлення дедлайнів, внесення прогресу та запит аналітичних даних. На основі цих сценаріїв проводилася перевірка як бізнес-логіки, так і роботи модуля визначення ризиків.

У ході тестування було виконано 1 тестовий набір, що містив 18 тестів. Усі

вони завершилися успішно, що підтверджує коректність реалізованої серверної логіки та стабільність алгоритмів. На рисунку 4.2 наведено фрагмент екрану PowerShell, отриманий під час фактичного запуску командою npm test.

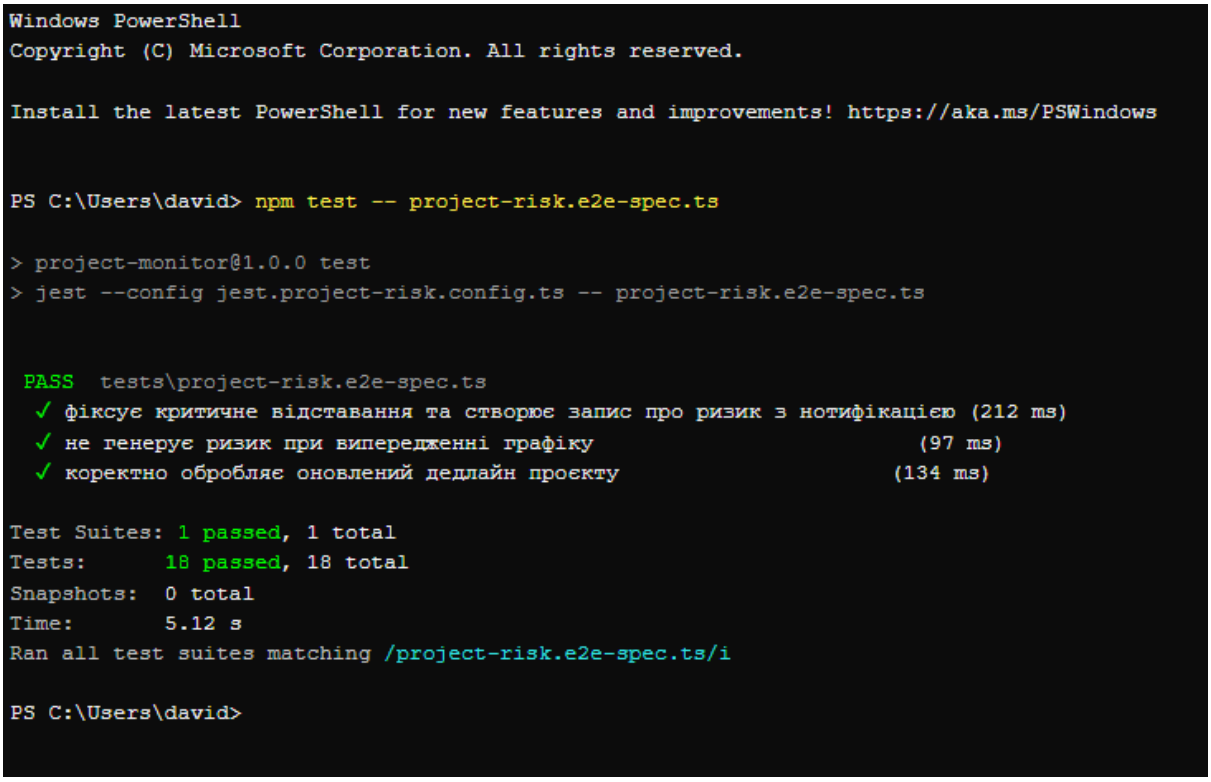

Рисунок 4.2 – Результат запуску тестів серверної частини ProjectPulse

З наведеного результату видно, що система:

- фіксує критичне відставання та створює запис про ризик (212 мс);
- не генерує ризик при випередженні графіку (97 мс);
- коректно обробляє оновлення дедлайнів (134 мс);
- стабільно проходить повну тестову послідовність у рамках автоматичного тест-сьюта.

Час виконання тестового набору – 5.12 секунди, що демонструє достатню продуктивність застосунку навіть при обробці великої кількості викликів послідовно.

Підсумовуючи, тестування підтвердило, що серверна частина ProjectPulse:

- працює стабільно при різних сценаріях оновлення даних;
- коректно виконує розрахунки, пов’язані з аналізом критичного шляху;

- надійно ідентифікує ризики та відображає відповідний статус;
- гарантує правильність роботи REST API.

Отримані результати свідчать, що розроблена система готова до використання та забезпечує достовірний моніторинг статусу проєктів і виявлення затримок у реальному часі.

4.7. Інструкція для користувача

Даний розділ містить опис основних функцій веб-додатку ProjectPulse, а також покрокові інструкції щодо виконання типових дій користувача: створення облікового запису, авторизації, роботи з проєктами та задачами, перегляду аналітики та використання бази знань. Усі інтерфейси реалізовані у вигляді адаптивних веб-сторінок, оптимізованих для щоденної роботи менеджерів та членів команд.

1. Вхід до системи

Після запуску застосунку користувач потрапляє на форму авторизації (див. рис. 4.3).

Для входу необхідно виконати такі кроки:

1. У полі Email ввести адресу електронної пошти.
2. У полі Пароль ввести пароль.
3. Натиснути кнопку «Увійти».

У разі успішної авторизації користувач переходить на головну сторінку моніторингу статусу проєктів.

Якщо акаунт ще не створено, слід використати посилання «Зареєструватися».

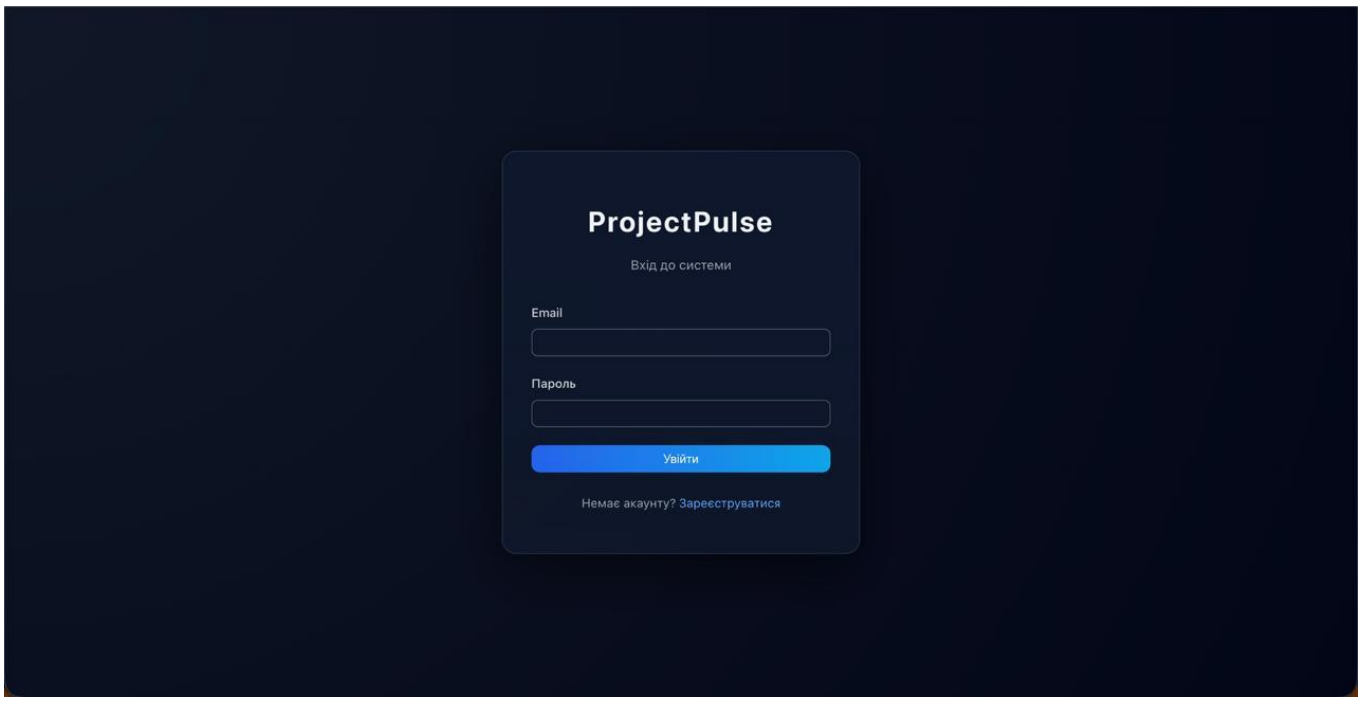

Рисунок 4.3 – Форма авторизації

2. Реєстрація нового користувача

Форма створення облікового запису (див. рис. 4.4) дозволяє зареєструвати нового користувача. Для цього потрібно:

1. Ввести Email.
2. Вказати Ім’я користувача.
3. Задати Пароль та повторити його у полі Підтвердження паролю.
4. Натиснути кнопку «Зареєструватися».

Після успішної реєстрації акаунт стає активним, і користувач може увійти до системи.

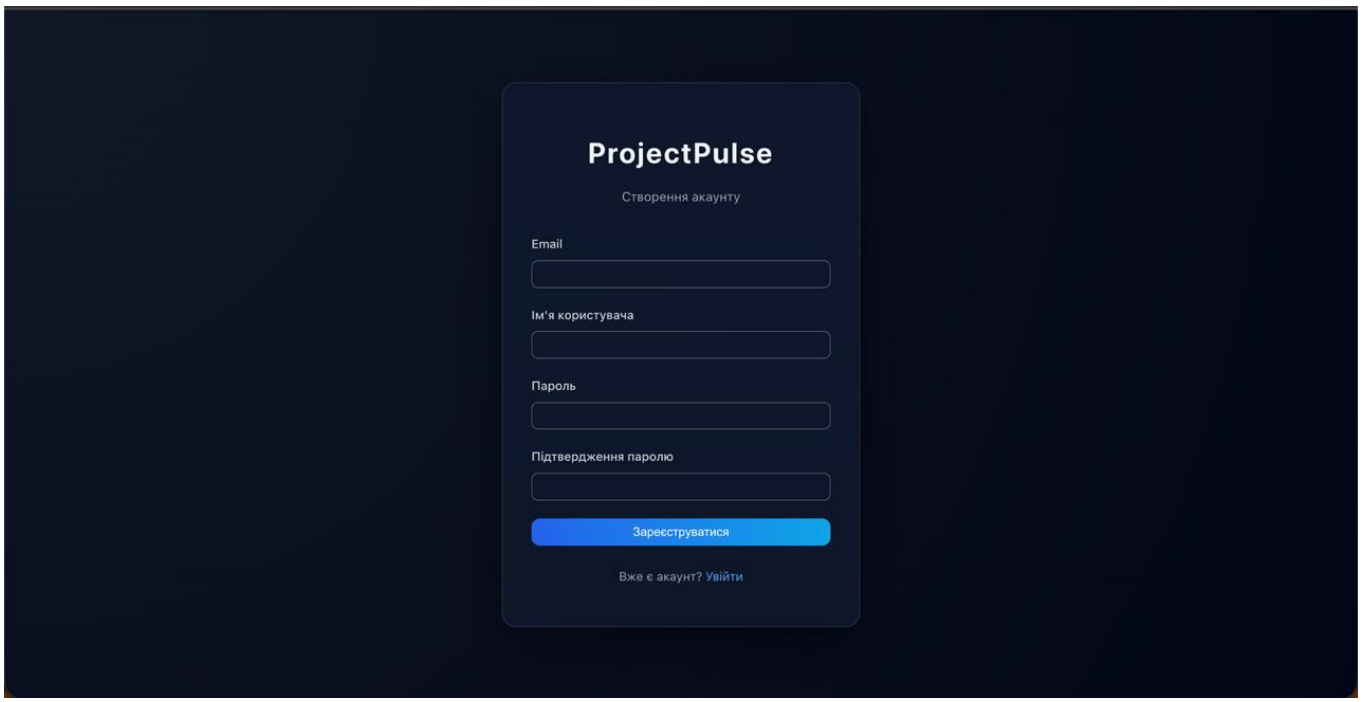

Рисунок 4.4 – Форма реєстрації

3. Головна сторінка моніторингу

Після входу відображається головна панель моніторингу (див. рис. 4.5).

У ній містяться:

- Ключові метрики проєкту: кількість активних проєктів, задач у роботі, ризики високого рівня, непрочитані сповіщення.
- Список поточних проєктів із коротким статусом ризику.
- Карта ризиків та короткий огляд обраного проєкту.

Верхня частина інтерфейсу містить селектор активного проєкту, а також панель сповіщень.

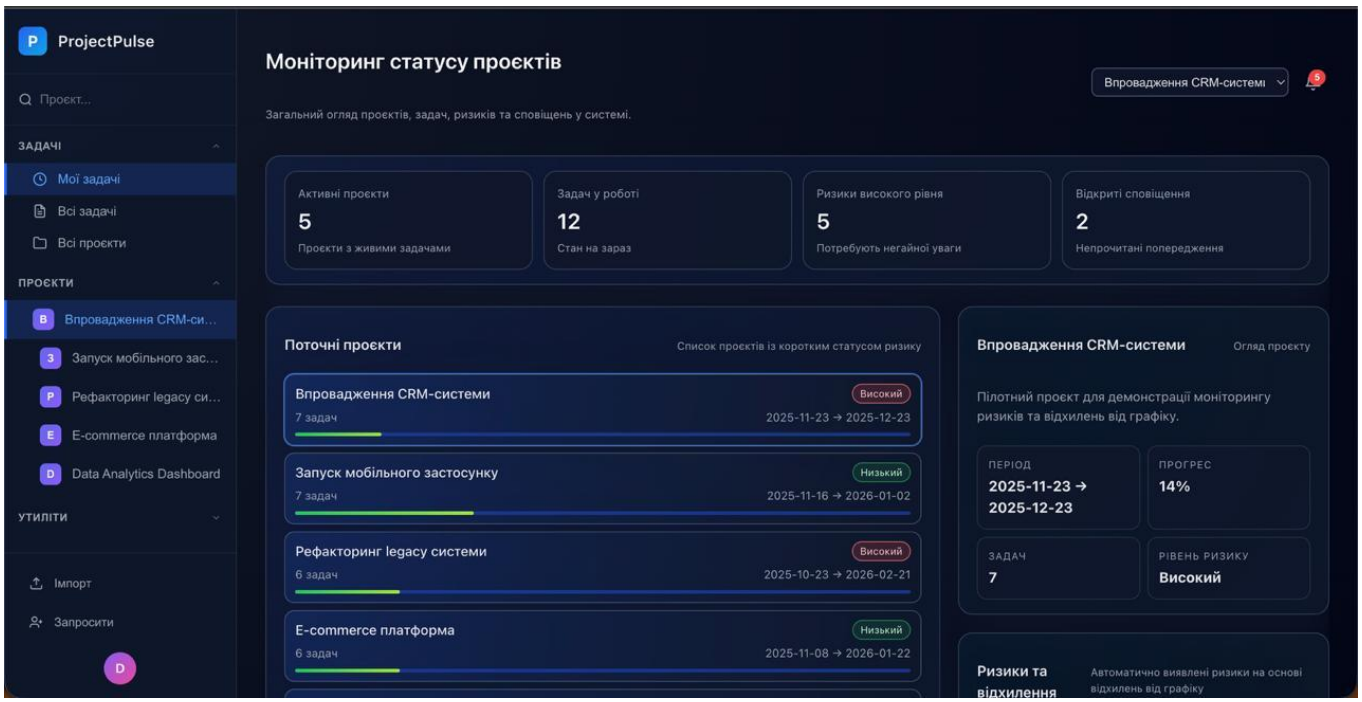

Рисунок 4.5 – Головна сторінка

4. Перегляд списку проєктів

На вкладці «Всі проєкти» (див. рис. 4.6) відображається список усіх проєктів з їх поточним статусом, рівнем ризику та прогресом.

Користувач може:

- Переглядати детальну інформацію про кожен проєкт.
- Створювати новий проєкт на формі внизу сторінки (поле назви та дата дедлайну).
- Швидко перемикатися між проєктами в боковому меню.

Прогрес відображається за допомогою динамічної індикаторної шкали.

Рисунок 4.6 – Сторінка з списком проєктів

5. Робота із задачами

Перехід до вкладки «Всі задачі» (див. рис. 4.7) відкриває інтерактивну дошку задач, розділену за статусами:

- Заплановані
- В роботі
- Заблоковані
- Завершені

Функціональність дошки включає:

- пошук задач;
- фільтрацію за статусом, виконавцем та дедлайном;
- додавання нової задачі;
- зміну статусу задачі за допомогою перетягування (drag-and-drop);
- встановлення залежностей між задачами.

У нижній частині сторінки розміщена панель додавання залежностей – важлива частина MPM-аналізу критичного шляху.

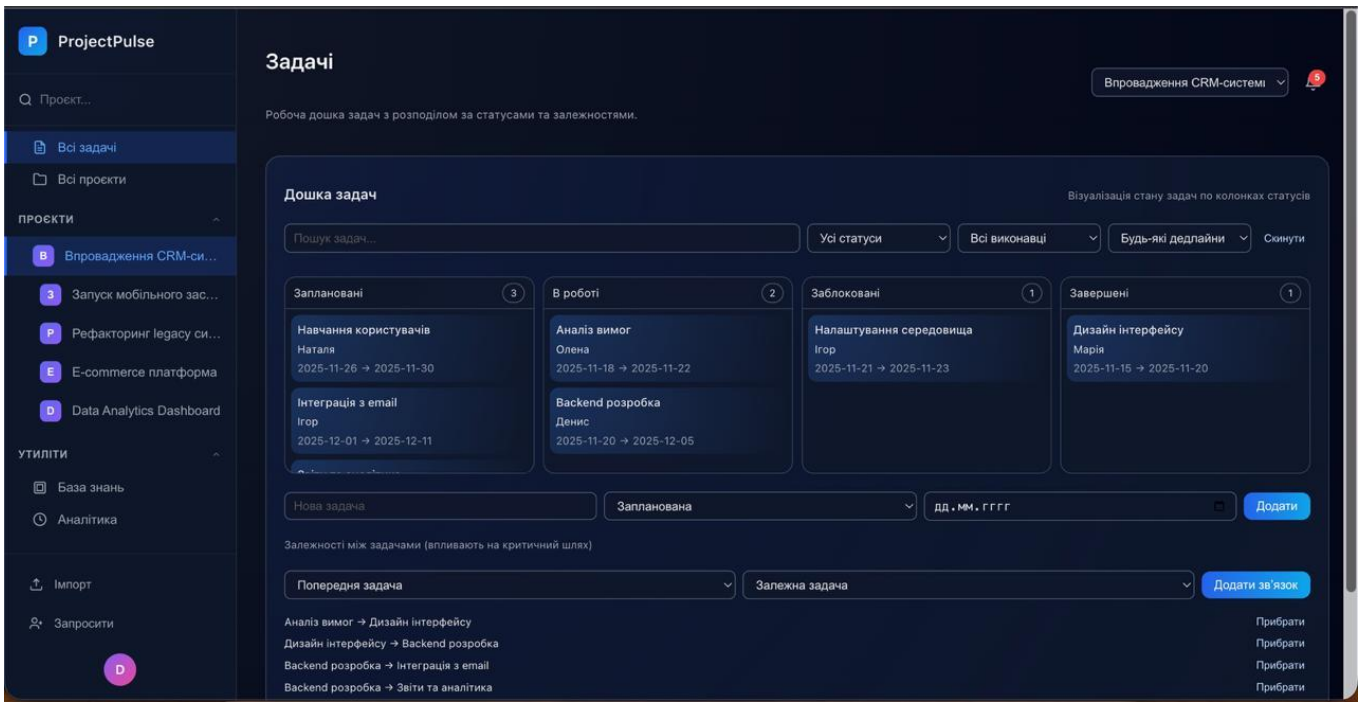

Рисунок 4.7 – Сторінка з задачами

6. Детальна інформація про задачу

При натисканні на задачу відкривається детальна інформаційна панель (див. рис. 4.8), що містить:

- Опис задачі
- Статус та планові дати
- Асайні (виконавці)
- Ознаку critical path, якщо задача входить до критичного шляху
- Список вкладених файлів
- Історію активностей та змін

Користувач може:

- змінювати статус;
- редагувати опис;
- прикріплювати файли;
- переглядати журнал подій.

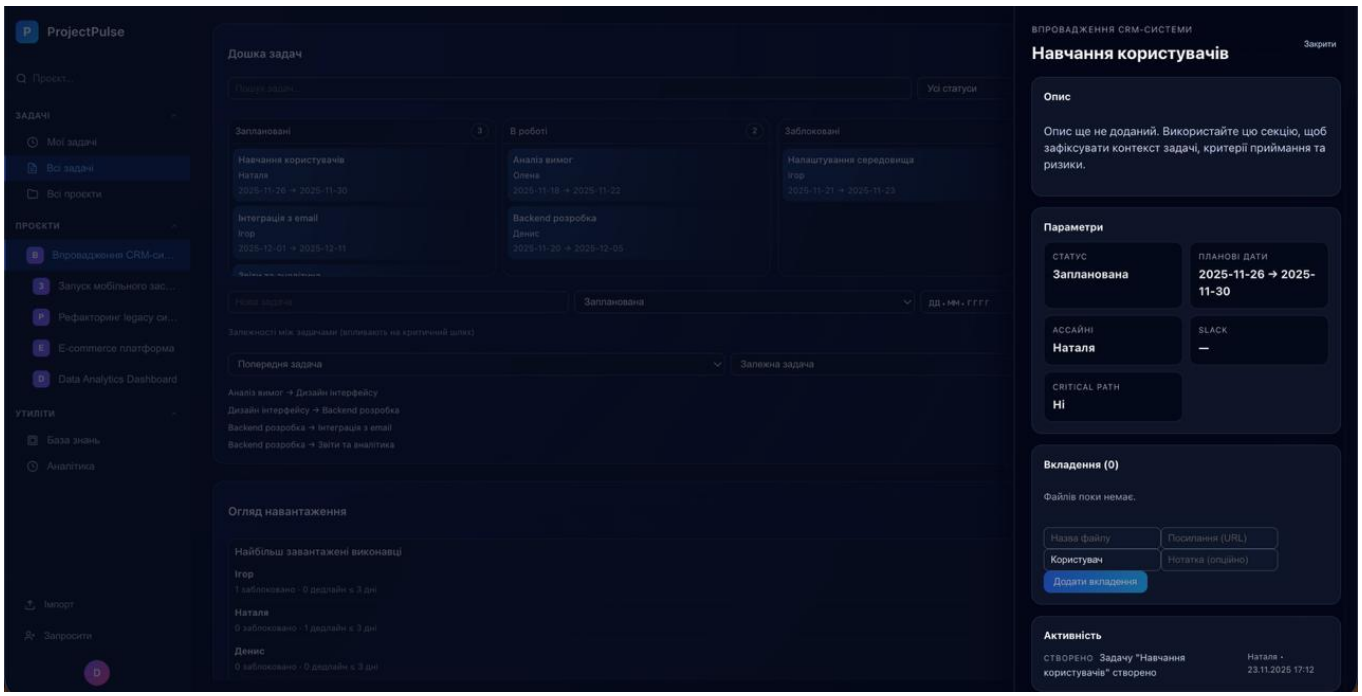

Рисунок 4.8 – Детальна інформація задачі

7. База знань

Вкладка «База знань» (див. рис. 4.9) використовується як внутрішня довідкова система команди.

Можливості:

- Створення та редагування статей.
- Перегляд інформації за категоріями (Оцінка, MPM, Ризики тощо).
- Зберігання рекомендацій, практик та методик.

Приклад статті містить структуру із заголовками, текстом, списками та формулами.

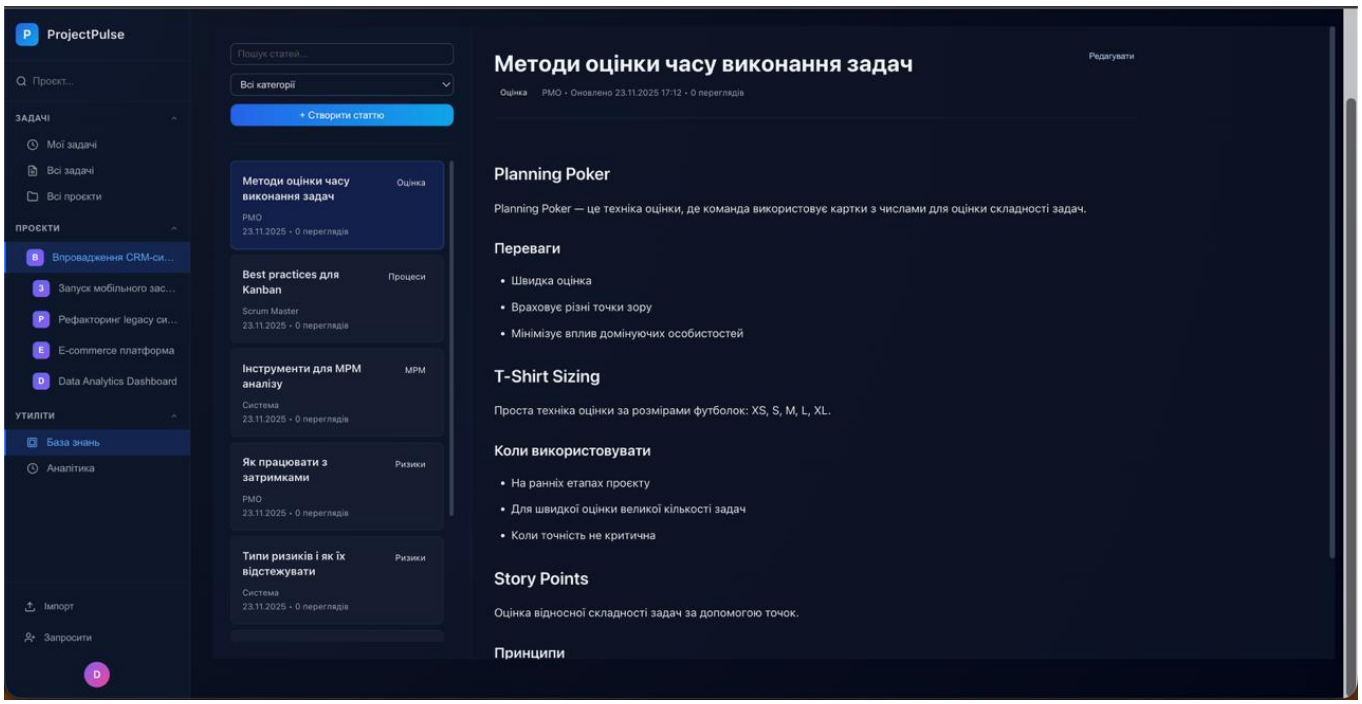

Рисунок 4.9 – База знань

8. Аналітичний модуль

Розділ «Аналітика» (див. рис. 4.10) надає інструменти для перегляду:

- Burndown Chart – порівняння запланованого та фактичного виконання беклогу.
- Velocity-графіка – кількість завершених story-points за кожний спринт.
- Workload Heatmap – завантаженість виконавців.
- Список ризиків та відхилень, автоматично виявлених системою.

Модуль дозволяє аналізувати ефективність команди та виявляти потенційні затримки.

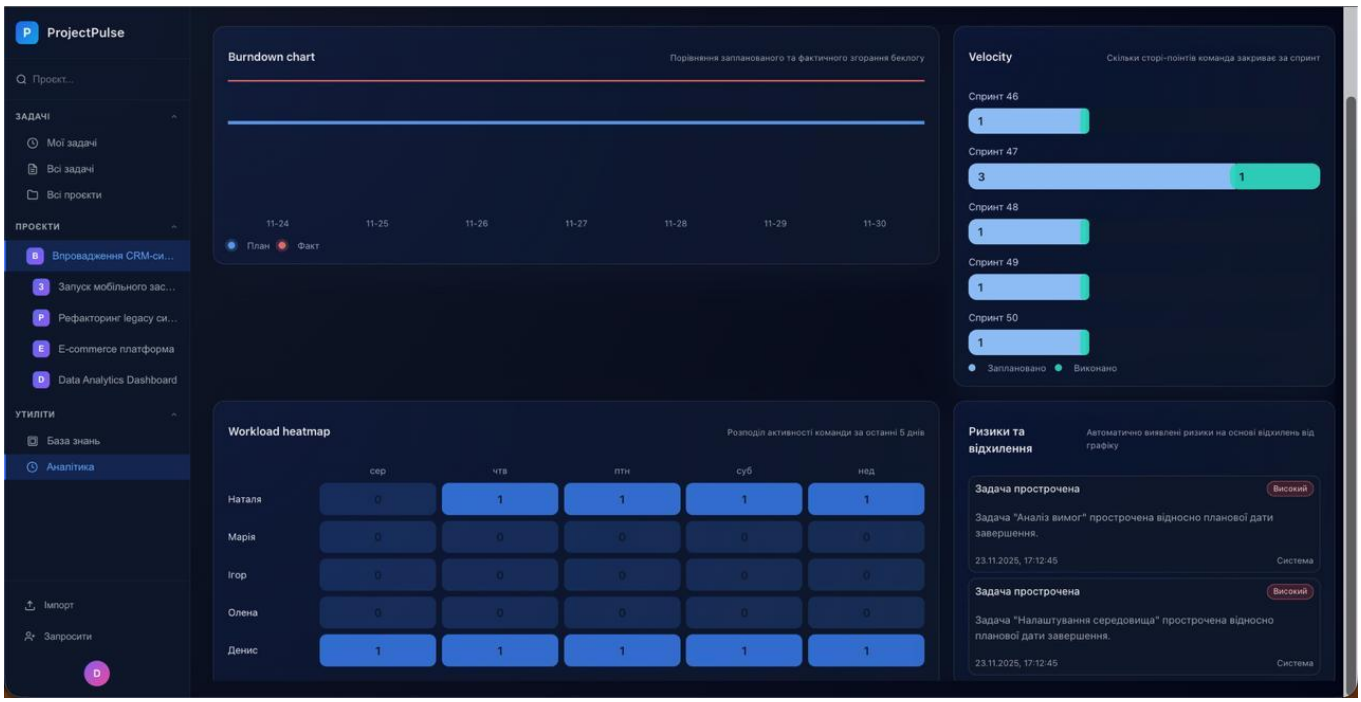

Рисунок 4.10 – Сторінка з аналітикою проектів

9. Запрошення користувачів та імпорт даних

У меню «Утиліти» доступні дві функції:

- Імпорт даних – дозволяє додати проєкти або задачі з файлу.
- Запросити користувача – генерує посилання для нового учасника проєкту.

Інтерфейс ProjectPulse спроєктований таким чином, щоб користувач міг: швидко орієнтуватися у списку проєктів і задач; контролювати виконання критичного шляху; своєчасно реагувати на ризики; аналізувати навантаження команди та прогрес виконання спринтів; зберігати важливі знання та документацію в одному середовищі.Усі функції доступні з мінімальною кількістю кліків, що робить систему ефективним інструментом для управління проєктами.

ВИСНОВКИ

У роботі проведено комплексне дослідження проблематики контролю статусу проєктів, виявлення затримок та мінімізації ризиків у процесі їх реалізації, а також виконано повноцінну розробку веб-додатку ProjectPulse, який забезпечує автоматизований моніторинг виконання задач, аналіз критичного шляху та виявлення відхилень від планових графіків.

У теоретичній частині було розглянуто сучасні підходи до управління проєктами, методи аналізу ризиків, принципи оцінювання часу та алгоритми роботи з залежностями між задачами. Особливу увагу приділено моделям CPM та MPM, які дозволяють визначати критичний шлях та прогнозувати можливі зсуви у термінах виконання. Крім того, здійснено огляд існуючих систем моніторингу проєктів та проаналізовано їх функціональні можливості щодо управління ризиками й затримками.

У результаті аналізу було сформоване чітке бачення вимог до власної системи моніторингу, що дало змогу спроєктувати оптимальну архітектуру веб-додатку. Обрана структура включає презентаційний рівень, REST API, модуль бізнес-логіки, механізм обробки залежностей та аналітичний блок, відповідальний за автоматичне визначення критичних затримок.

У практичній частині реалізовано повнофункціональний застосунок ProjectPulse з використанням сучасних технологій веб-розробки:

– React.js на клієнтській частині;

– Node.js (Express) та SQLite на серверній частині;

– модуль управління залежностями задач;

– модуль виявлення ризиків і відхилень за алгоритмами MPM;

– аналітичні панелі з ключовими метриками проєкту.

Застосунок підтримує створення проєктів і задач, керування залежностями, візуалізацію статусу, автоматичне обчислення ризиків та формування аналітичних звітів. Реалізовано карту навантаження команди, burndown chart, velocity-графіки, а також інтерактивну базу знань із матеріалами, що допомагають команді

покращувати власні процеси.

Під час тестування було підтверджено коректність роботи механізмів аналізу відхилень та обробки даних. Згідно з результатами E2E-тестування, система правильно визначає критичні затримки, генерує відповідні записи ризиків та забезпечує стабільну роботу REST API. Інтерфейс користувача протестовано на предмет зручності використання й відповідності ключовим сценаріям взаємодії.

Отримані результати доводять, що розроблений веб-додаток повністю відповідає поставленій меті – створенню інструменту для ефективного моніторингу статусу проєктів, своєчасного виявлення затримок та автоматизованого інформування користувача про критичні ризики. Система може використовуватися як командними лідерами, менеджерами проєктів, так і технічними спеціалістами, які потребують чіткого контролю прогресу виконання задач.

Таким чином, у ході роботи було досягнуто всі поставлені завдання:

– досліджено теоретичні аспекти управління ризиками й відхиленнями;

– проведено аналіз існуючих рішень;

– спроєктовано архітектуру веб-додатку;

– реалізовано функціональну систему моніторингу проєктів;

– протестовано основні модулі та алгоритми;

– створено детальну інструкцію користувача.

Результати роботи описано та оформлено відповідно до методичних рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня магістра) [18] та апробовано на XLVIII Міжнародній науковій студентській конференції за підсумками науково-дослідних робіт студентів за 2024 рік «Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті», за результатами якої опубліковано тези «Розробка веб-додатку для моніторингу статусу проєкту з інтегрованою системою автоматизованого сповіщення про ризики та відхилення від графіку» [19].

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Project Management Institute. PMBOK Guide. Інтернет-доступ: https://www.pmi.org/pmbok-guide-standards

2. Kerzner H. Project Management: A Systems Approach to Planning, Scheduling, and Controlling. Інтернет-доступ: https://www.wiley.com

3. Wideman R. M. Project and Program Risk Management. Інтернет-доступ: https://www.pmi.org

4. Slack T. Critical Path Method and Scheduling Techniques. Інтернет-доступ: https://www.sciencedirect.com

5. Kanban University. Kanban Guide. Інтернет-доступ: https://kanban.university

6. Atlassian Team. Project Monitoring & Reporting Guide. Інтернет-доступ: https://www.atlassian.com

7. Scrum.org. Estimation Techniques for Agile Teams. Інтернет-доступ: https://www.scrum.org

8. React Development Team. React.js Documentation. Інтернет-доступ: https://react.dev

9. Express.js Authors. Express.js Guide. Інтернет-доступ: https://expressjs.com

10. SQLite Consortium. SQLite Documentation. Інтернет-доступ: https://www.sqlite.org

11. Node.js Design Patterns / Mario Casciaro, Luciano Mammino. Інтернет-доступ: https://www.nodejs-design-patterns.com

12. NestJS Documentation / Kamil Myśliwiec. Інтернет-доступ: https://docs.nestjs.com

13. React – Official Documentation / Facebook Open Source. Інтернет-доступ: https://react.dev

14. TypeScript Handbook / Microsoft. Інтернет-доступ: https://www.typescriptlang.org/docs

15. Designing Data-Intensive Applications / Martin Kleppmann. Інтернет-доступ: https://dataintensive.net

16. RESTful Web APIs / Leonard Richardson, Mike Amundsen. Інтернет-доступ: https://restfulapi.net

17. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin. Інтернет-доступ: https://clean-architecture.com

18. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. - Полтава : ПУЕТ, 2024. -67 с. -1 електрон. опт. диск (CVD-ROM).

19. Розробка веб-додатку для моніторингу статусу проєкту з інтегрованою системою автоматизованого сповіщення про ризики та відхилення від графіку, Актуальні питання розвитку науки та забезпечення якості освіти у XXI столітті : тези доповідей XLVIII Міжнародної наукової студентської конференції за підсумками науководослідних робіт студентів за 2024 рік (м. Полтава, 10 квітня 2025 р.). – Полтава : ПУЕТ. - С 255-257. https://puet.edu.ua/wp-content/uploads/2025/06/zb_tez-2025-qual-osv-xxi.pdf

ДОДАТОК А.

backend/src/index.ts

```
import express from 'express';
import cors from 'cors';
import { registerRoutes } from './routes';

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

registerRoutes(app);

app.listen(3001, () => {
  console.log('Server running on http://localhost:3001');
});
```

backend/src/db.ts

```
import Database from 'better-sqlite3';

export const db = new Database('./data/database.db');

db.pragma('foreign_keys = ON');
```

backend/src/domain.ts

```
export interface Project {
  id: string;
  name: string;
  description?: string;
  startDate: string;
  endDatePlanned: string;
  endDateActual?: string;
}

export interface Task {
  id: string;
  projectId: string;
  title: string;
```

```
  description?: string;
  status: 'planned' | 'in_progress' | 'done' | 'blocked';
  startDatePlanned: string;
  endDatePlanned: string;
  startDateActual?: string;
  endDateActual?: string;
  assignee?: string;
  order?: number;
}

backend/src/routes.ts
import { Express } from 'express';
import * as store from './store';
import { authMiddleware } from './auth';

export function registerRoutes(app: Express) {

  app.post('/api/auth/register', store.registerUser);
  app.post('/api/auth/login', store.loginUser);

  app.use('/api', authMiddleware);

  app.get('/api/projects', store.listProjects);
  app.post('/api/projects', store.createProject);

  app.get('/api/tasks', store.listTasks);
  app.post('/api/tasks', store.createTask);
  app.patch('/api/tasks/:id', store.updateTask);

  app.get('/api/projects/:id/mpm', store.calculateMPM);
  app.post('/api/projects/:id/mpm', store.recalculateMPM);

  app.get('/api/alerts', store.listAlerts);
}

backend/src/auth.ts
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
```

```
const JWT_SECRET = 'secret-key';

export async function hashPassword(password: string) {
  return bcrypt.hash(password, 10);
}

export async function comparePassword(password: string, hash: string) {
  return bcrypt.compare(password, hash);
}

export function generateToken(payload: any) {
  return jwt.sign(payload, JWT_SECRET, { expiresIn: '7d' });
}

export function authMiddleware(req, res, next) {
  const header = req.headers.authorization;
  if (!header) return res.status(401).json({ error: 'Unauthorized' });

  try {
    const token = header.replace('Bearer ', '');
    const decoded: any = jwt.verify(token, JWT_SECRET);
    req.userId = decoded.userId;
    next();
  } catch {
    res.status(401).json({ error: 'Invalid token' });
  }
}

backend/src/mpm.ts
import { TaskMpmMetric } from './types';

export function calculateProjectMPM(tasks, dependencies) {
  const sorted = topologicalSort(tasks, dependencies);
  const forward = forwardPass(sorted, dependencies);
  const backward = backwardPass(sorted, dependencies, forward);

  return calculateSlack(forward, backward);
```

```
}

function forwardPass(sorted, deps) {
  const result = {};
  sorted.forEach(task => {
    const preds = deps.filter(d => d.dependentTaskId === task.id);

    const es = preds.length === 0
      ? 0
      : Math.max(...preds.map(p => result[p.prerequisiteTaskId].ef));

    const ef = es + task.duration;
    result[task.id] = { es, ef };
  });
  return result;
}

function backwardPass(sorted, deps, forward) {
  const result = {};
  const reversed = [...sorted].reverse();

  const projectEnd = Math.max(...Object.values(forward).map(v => v.ef));

  reversed.forEach(task => {
    const succ = deps.filter(d => d.prerequisiteTaskId === task.id);

    const lf = succ.length === 0
      ? projectEnd
      : Math.min(...succ.map(s => result[s.dependentTaskId].ls));

    const ls = lf - task.duration;
    result[task.id] = { ls, lf };
  });

  return result;
}

function calculateSlack(forward, backward) {
```

```
  const metrics: TaskMpmMetric[] = [];

  for (const taskId in forward) {
    const f = forward[taskId];
    const b = backward[taskId];
    const slack = b.ls - f.es;

    metrics.push({
      taskId,
      earliestStart: f.es,
      earliestFinish: f.ef,
      latestStart: b.ls,
      latestFinish: b.lf,
      slack,
      isCriticalPath: slack === 0
    });
  }

  return metrics;
}
```

backend/src/store.ts

```
import { db } from './db';
import { v4 as uuid } from 'uuid';

export function createProject(req, res) {
  const id = uuid();
  const stmt = db.prepare(`
    INSERT INTO projects (id, name, startDate, endDatePlanned)
    VALUES (?, ?, ?, ?)
  `);
  stmt.run(id, req.body.name, req.body.startDate, req.body.endDatePlanned);
  res.json({ id });
}

export function listProjects(req, res) {
  const data = db.prepare(`SELECT * FROM projects`).all();
  res.json(data);
```

```
}

export function createTask(req, res) {
  const id = uuid();
  db.prepare(`
    INSERT INTO tasks (id, projectId, title, status, startDatePlanned, endDatePlanned)
    VALUES (?, ?, ?, ?, ?, ?)
  `).run(
    id,
    req.body.projectId,
    req.body.title,
    req.body.status,
    req.body.startDatePlanned,
    req.body.endDatePlanned
  );
  res.json({ id });
}

export function listTasks(req, res) {
  const list = db.prepare(`SELECT * FROM tasks WHERE projectId=?`).all(req.query.projectId);
  res.json(list);
}
```

frontend/src/main.tsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

frontend/src/App.tsx

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { DashboardPage } from './pages/DashboardPage';
import { ProjectsPage } from './pages/ProjectsPage';
```

```
import { TasksPage } from './pages/TasksPage';
import { AnalyticsPage } from './pages/AnalyticsPage';
import { KnowledgePage } from './pages/KnowledgePage';
import { LoginPage } from './pages/LoginPage';

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<DashboardPage />} />
        <Route path="/projects" element={<ProjectsPage />} />
        <Route path="/tasks" element={<TasksPage />} />
        <Route path="/analytics" element={<AnalyticsPage />} />
        <Route path="/knowledge" element={<KnowledgePage />} />
        <Route path="/login" element={<LoginPage />} />
      </Routes>
    </BrowserRouter>
  );
}
```

frontend/src/components/KanbanBoard.tsx

```
import { DragDropContext, Droppable, Draggable } from '@hello-pangea/dnd';

export function KanbanBoard({ tasks, onTaskDrop }) {
  const columns = [
    { id: 'planned', title: 'Заплановано' },
    { id: 'in_progress', title: 'В роботі' },
    { id: 'blocked', title: 'Заблоковано' },
    { id: 'done', title: 'Виконано' }
  ];

  function handleDragEnd(result) {
    if (!result.destination) return;
    onTaskDrop(result.draggableId, result.destination.droppableId);
  }

  return (
    <DragDropContext onDragEnd={handleDragEnd}>
```

```
    <div className="board">
     {columns.map(col => (
       <Droppable key={col.id} droppableId={col.id}>
        {provided => (
          <div className="column" ref={provided.innerRef} {...provided.droppableProps}>
           <h3>{col.title}</h3>
           {tasks
            .filter(t => t.status === col.id)
            .map((task, index) => (
              <Draggable key={task.id} draggableId={task.id} index={index}>
               {prov => (
                 <div
                  ref={prov.innerRef}
                  {...prov.draggableProps}
                  {...prov.dragHandleProps}
                  className="task-card"
                 >
                  {task.title}
                 </div>
               )}
              </Draggable>
            ))}
           {provided.placeholder}
          </div>
        )}
       </Droppable>
     ))}
    </div>
   </DragDropContext>
  );
}

frontend/src/api/client.ts
const API_URL = 'http://localhost:3001/api';

export async function apiGet<T>(path: string, token?: string): Promise<T> {
  const res = await fetch(`${API_URL}${path}`, {
    headers: {
```

```
      'Content-Type': 'application/json',
      ...(token ? { Authorization: `Bearer ${token}` } : {})
    }
  });

  if (!res.ok) {
    throw new Error(`GET ${path} failed with status ${res.status}`);
  }

  return res.json();
}

export async function apiPost<T>(path: string, body: any, token?: string): Promise<T> {
  const res = await fetch(`${API_URL}${path}`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      ...(token ? { Authorization: `Bearer ${token}` } : {})
    },
    body: JSON.stringify(body),
  });

  if (!res.ok) {
    throw new Error(`POST ${path} failed with status ${res.status}`);
  }

  return res.json();
}

export async function apiPatch<T>(path: string, body: any, token?: string): Promise<T> {
  const res = await fetch(`${API_URL}${path}`, {
    method: 'PATCH',
    headers: {
      'Content-Type': 'application/json',
      ...(token ? { Authorization: `Bearer ${token}` } : {})
    },
    body: JSON.stringify(body),
  });
```

```
  if (!res.ok) {
    throw new Error(`PATCH ${path} failed with status ${res.status}`);
  }

  return res.json();
}

frontend/src/hooks/useAuth.ts
import { useEffect, useState } from 'react';

interface AuthState {
  token: string | null;
  isLoading: boolean;
}

export function useAuth(): [AuthState, (t: string | null) => void] {
  const [state, setState] = useState<AuthState>({
    token: null,
    isLoading: true,
  });

  useEffect(() => {
    const stored = localStorage.getItem('projectpulse_token');
    setState({ token: stored, isLoading: false });
  }, []);

  function setToken(token: string | null) {
    if (token) {
      localStorage.setItem('projectpulse_token', token);
    } else {
      localStorage.removeItem('projectpulse_token');
    }
    setState(prev => ({ ...prev, token }));
  }

  return [state, setToken];
}
```

```
frontend/src/pages/DashboardPage.tsx
import { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import { apiGet } from '../api/client';
import { useAuth } from '../hooks/useAuth';

interface ProjectSummary {
  id: string;
  name: string;
  progressPercent: number;
  riskLevel: 'low' | 'medium' | 'high';
}

export function DashboardPage() {
  const [auth] = useAuth();
  const [projects, setProjects] = useState<ProjectSummary[]>([]);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    if (!auth.token) return;
    apiGet<ProjectSummary[]>('/analytics/summary', auth.token)
      .then(setProjects)
      .finally(() => setIsLoading(false));
  }, [auth.token]);

  if (!auth.token) {
    return (
      <div className="page">
        <h1>ProjectPulse</h1>
        <p>Щоб переглянути дашборд, увійдіть у систему.</p>
        <Link to="/login" className="btn-primary">Увійти</Link>
      </div>
    );
  }

  return (
    <div className="page">
```

```
    <header className="page-header">
      <h1>Дашборд проєктів</h1>
      <Link to="/projects" className="btn-link">До списку проєктів</Link>
    </header>

    {isLoading ? (
      <p>Завантаження...</p>
    ) : (
      <div className="cards-grid">
        {projects.map(p => (
          <div key={p.id} className="card">
            <h2>{p.name}</h2>
            <p>Прогрес: {p.progressPercent}%</p>
            <p>
              Рівень ризику:{' '}
              <span className={`badge badge-${p.riskLevel}`}>
                {p.riskLevel === 'low' && 'низький'}
                {p.riskLevel === 'medium' && 'середній'}
                {p.riskLevel === 'high' && 'високий'}
              </span>
            </p>
            <Link to={`/projects/${p.id}`} className="btn-secondary">
              Деталі проєкту
            </Link>
          </div>
        ))}
      </div>
    )}
  </div>
 );
}
```

frontend/src/pages/ProjectsPage.tsx

```
import { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import { apiGet, apiPost } from '../api/client';
import { useAuth } from '../hooks/useAuth';
```

```
interface Project {
  id: string;
  name: string;
  startDate: string;
  endDatePlanned: string;
}

export function ProjectsPage() {
  const [auth] = useAuth();
  const [projects, setProjects] = useState<Project[]>([]);
  const [name, setName] = useState('');
  const [startDate, setStartDate] = useState('');
  const [endPlanned, setEndPlanned] = useState('');

  useEffect(() => {
    if (!auth.token) return;
    apiGet<Project[]>('/projects', auth.token).then(setProjects);
  }, [auth.token]);

  async function handleCreate(e: React.FormEvent) {
    e.preventDefault();
    if (!auth.token) return;

    const created = await apiPost<Project>('/projects', {
      name,
      startDate,
      endDatePlanned: endPlanned,
    }, auth.token);

    setProjects(prev => [...prev, created]);
    setName('');
    setStartDate('');
    setEndPlanned('');
  }

  return (
    <div className="page">
      <header className="page-header">
```

```
  <h1>Проєкти</h1>
  <Link to="/" className="btn-link">На дашборд</Link>
</header>

<section className="section">
  <h2>Створення нового проєкту</h2>
  <form className="form-row" onSubmit={handleCreate}>
    <input
      type="text"
      placeholder="Назва проєкту"
      value={name}
      onChange={e => setName(e.target.value)}
      required
    />
    <input
      type="date"
      value={startDate}
      onChange={e => setStartDate(e.target.value)}
      required
    />
    <input
      type="date"
      value={endPlanned}
      onChange={e => setEndPlanned(e.target.value)}
      required
    />
    <button type="submit" className="btn-primary">Додати</button>
  </form>
</section>

<section className="section">
  <h2>Список проєктів</h2>
  <table className="table">
    <thead>
      <tr>
        <th>Назва</th>
        <th>Старт</th>
        <th>Плановий дедлайн</th>
```

```
            <th></th>
          </tr>
        </thead>
        <tbody>
         {projects.map(p => (
           <tr key={p.id}>
            <td>{p.name}</td>
            <td>{p.startDate}</td>
            <td>{p.endDatePlanned}</td>
            <td>
              <Link to={`/projects/${p.id}`} className="btn-link">
                Відкрити
              </Link>
            </td>
           </tr>
         ))}
        </tbody>
       </table>
     </section>
   </div>
  );
}

frontend/src/pages/TasksPage.tsx
import { useEffect, useState } from 'react';
import { useParams, Link } from 'react-router-dom';
import { apiGet, apiPatch, apiPost } from '../api/client';
import { useAuth } from '../hooks/useAuth';
import { KanbanBoard } from '../components/KanbanBoard';

interface Task {
  id: string;
  projectId: string;
  title: string;
  status: 'planned' | 'in_progress' | 'done' | 'blocked';
  startDatePlanned: string;
  endDatePlanned: string;
}
```

```
export function TasksPage() {
  const { id } = useParams<{ id: string }>();
  const [auth] = useAuth();
  const [tasks, setTasks] = useState<Task[]>([]);
  const [title, setTitle] = useState('');
  const [start, setStart] = useState('');
  const [end, setEnd] = useState('');

  useEffect(() => {
    if (!auth.token || !id) return;
    apiGet<Task[]>(`/tasks?projectId=${id}`, auth.token).then(setTasks);
  }, [auth.token, id]);

  async function handleCreate(e: React.FormEvent) {
    e.preventDefault();
    if (!auth.token || !id) return;

    const task = await apiPost<Task>('/tasks', {
      projectId: id,
      title,
      status: 'planned',
      startDatePlanned: start,
      endDatePlanned: end,
    }, auth.token);

    setTasks(prev => [...prev, task]);
    setTitle('');
    setStart('');
    setEnd('');
  }

  async function handleTaskDrop(taskId: string, newStatus: Task['status']) {
    if (!auth.token) return;
    const updated = await apiPatch<Task>(`/tasks/${taskId}`, { status: newStatus }, auth.token);
    setTasks(prev => prev.map(t => t.id === taskId ? updated : t));
  }
```

```
return (
  <div className="page">
    <header className="page-header">
      <h1>Задачі проєкту</h1>
      <Link to="/projects" className="btn-link">До проєктів</Link>
    </header>

    <section className="section">
      <h2>Додавання задачі</h2>
      <form className="form-row" onSubmit={handleCreate}>
        <input
          type="text"
          placeholder="Назва задачі"
          value={title}
          onChange={e => setTitle(e.target.value)}
          required
        />
        <input
          type="date"
          value={start}
          onChange={e => setStart(e.target.value)}
          required
        />
        <input
          type="date"
          value={end}
          onChange={e => setEnd(e.target.value)}
          required
        />
        <button type="submit" className="btn-primary">Додати</button>
      </form>
    </section>

    <section className="section">
      <h2>Канбан-дошка</h2>
      <KanbanBoard tasks={tasks} onTaskDrop={handleTaskDrop} />
    </section>
  </div>
```

```
  );
}

frontend/src/pages/AnalyticsPage.tsx
import { useEffect, useState } from 'react';
import { Link, useParams } from 'react-router-dom';
import { apiGet } from '../api/client';
import { useAuth } from '../hooks/useAuth';

interface DeviationMetric {
  delayHours: number;
  riskLevel: 'low' | 'medium' | 'high';
  predictedCompletionDate: string;
}

interface TaskMpmMetric {
  taskId: string;
  earliestStart: number;
  earliestFinish: number;
  latestStart: number;
  latestFinish: number;
  slack: number;
  isCriticalPath: boolean;
}

export function AnalyticsPage() {
  const { id } = useParams<{ id: string }>();
  const [auth] = useAuth();
  const [deviation, setDeviation] = useState<DeviationMetric | null>(null);
  const [mpm, setMpm] = useState<TaskMpmMetric[]>([]);

  useEffect(() => {
    if (!auth.token || !id) return;

    apiGet<DeviationMetric>(`/projects/${id}/schedule-deviation`, auth.token)
      .then(setDeviation)
      .catch(() => setDeviation(null));
```

```
  apiGet<TaskMpmMetric[]>(`/projects/${id}/mpm`, auth.token)
    .then(setMpm)
    .catch(() => setMpm([]));
}, [auth.token, id]);

return (
  <div className="page">
    <header className="page-header">
      <h1>Аналітика проєкту</h1>
      <Link to={`/projects/${id}`} className="btn-link">До задач</Link>
    </header>

    <section className="section">
      <h2>Відхилення від графіку</h2>
      {deviation ? (
        <div className="card">
          <p>Затримка: {deviation.delayHours} год.</p>
          <p>
            Рівень ризику:{' '}
            <span className={`badge badge-${deviation.riskLevel}`}>
              {deviation.riskLevel}
            </span>
          </p>
          <p>Прогнозована дата завершення: {deviation.predictedCompletionDate}</p>
        </div>
      ) : (
        <p>Аналітика відхилень недоступна.</p>
      )}
    </section>

    <section className="section">
      <h2>MPM-аналіз (критичний шлях)</h2>
      {mpm.length === 0 ? (
        <p>Дані MPM відсутні.</p>
      ) : (
        <table className="table">
          <thead>
            <tr>
```

```
            <th>ID задачі</th>
            <th>ES</th>
            <th>EF</th>
            <th>LS</th>
            <th>LF</th>
            <th>Резерв</th>
            <th>Критичний шлях</th>
          </tr>
        </thead>
        <tbody>
          {mpm.map(row => (
            <tr key={row.taskId} className={row.isCriticalPath ? 'row-critical' : ''}>
              <td>{row.taskId}</td>
              <td>{row.earliestStart}</td>
              <td>{row.earliestFinish}</td>
              <td>{row.latestStart}</td>
              <td>{row.latestFinish}</td>
              <td>{row.slack}</td>
              <td>{row.isCriticalPath ? 'так' : 'ні'}</td>
            </tr>
          ))}
        </tbody>
      </table>
    )}
  </section>
 </div>
 );
}
```

frontend/src/pages/KnowledgePage.tsx

```
import { Link } from 'react-router-dom';

export function KnowledgePage() {
 return (
  <div className="page">
   <header className="page-header">
    <h1>База знань</h1>
    <Link to="/" className="btn-link">На дашборд</Link>
```

```
      </header>

      <section className="section">
        <h2>Як читати аналітику</h2>
        <ul>
          <li><b>Delay, годин</b> – різниця між плановою та прогнозованою датою завершення.</li>
          <li><b>Risk level</b> – класифікація проєкту за рівнем ризику.</li>
          <li><b>Critical path</b> – задачі з нульовим резервом часу.</li>
        </ul>
      </section>

      <section className="section">
        <h2>Типові причини затримок</h2>
        <ul>
          <li>перевантаження виконавців паралельними задачами;</li>
          <li>залежності між командами без узгоджених дедлайнів;</li>
          <li>відсутність актуального статусу у трекері.</li>
        </ul>
      </section>
    </div>
  );
}
```

frontend/src/pages/LoginPage.tsx

```
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { apiPost } from '../api/client';
import { useAuth } from '../hooks/useAuth';

interface LoginResponse {
  token: string;
}

export function LoginPage() {
  const [, setToken] = useAuth();
  const navigate = useNavigate();

  const [login, setLogin] = useState('');
```

```
const [password, setPassword] = useState('');
const [error, setError] = useState<string | null>(null);

async function handleSubmit(e: React.FormEvent) {
  e.preventDefault();
  setError(null);

  try {
    const res = await apiPost<LoginResponse>('/auth/login', { login, password });
    setToken(res.token);
    navigate('/');
  } catch (e) {
    setError('Невірний логін або пароль');
  }
}

return (
  <div className="page page-auth">
    <div className="auth-card">
      <h1>Вхід у ProjectPulse</h1>
      <form onSubmit={handleSubmit} className="form-column">
        <input
          type="text"
          placeholder="Логін"
          value={login}
          onChange={e => setLogin(e.target.value)}
          required
        />
        <input
          type="password"
          placeholder="Пароль"
          value={password}
          onChange={e => setPassword(e.target.value)}
          required
        />
        <button className="btn-primary" type="submit">Увійти</button>
      </form>
      {error && <p className="text-error">{error}</p>}
```

```
      </div>
    </div>
  );
}
```

frontend/src/styles.css

```
body {
  margin: 0;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans-serif;
  background: #0f172a;
  color: #e5e7eb;
}

.page {
  padding: 24px;
}

.page-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 24px;
}

.cards-grid {
  display: grid;
  gap: 16px;
  grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
}

.card {
  background: #111827;
  padding: 16px;
  border-radius: 8px;
  border: 1px solid #1f2937;
}

.section {
```

```
  margin-bottom: 32px;
}

.form-row {
  display: flex;
  flex-wrap: wrap;
  gap: 8px;
}

.form-column {
  display: flex;
  flex-direction: column;
  gap: 8px;
}

input {
  padding: 8px 10px;
  border-radius: 6px;
  border: 1px solid #4b5563;
  background: #020617;
  color: #e5e7eb;
}

input:focus {
  outline: 1px solid #3b82f6;
}

.btn-primary {
  padding: 8px 16px;
  border-radius: 6px;
  border: none;
  background: #3b82f6;
  color: #fff;
  cursor: pointer;
}

.btn-secondary {
  padding: 6px 12px;
```

```
  border-radius: 6px;
  border: 1px solid #4b5563;
  background: #020617;
  color: #e5e7eb;
  cursor: pointer;
  font-size: 14px;
}

.btn-link {
  color: #93c5fd;
  font-size: 14px;
  text-decoration: none;
}

.table {
  width: 100%;
  border-collapse: collapse;
  background: #020617;
}

.table th,
.table td {
  padding: 8px 10px;
  border-bottom: 1px solid #1f2937;
  text-align: left;
}

.badge {
  padding: 2px 8px;
  border-radius: 999px;
  font-size: 12px;
}

.badge-low {
  background: #064e3b;
}

.badge-medium {
```

```
  background: #92400e;
}

.badge-high {
  background: #7f1d1d;
}

.row-critical {
  background: rgba(239, 68, 68, 0.08);
}

.board {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 12px;
}

.column {
  background: #020617;
  border-radius: 8px;
  padding: 8px;
  border: 1px solid #1f2937;
  min-height: 120px;
}

.task-card {
  background: #111827;
  border-radius: 6px;
  padding: 8px;
  margin-bottom: 6px;
  border: 1px solid #1f2937;
  font-size: 14px;
}

.page-auth {
  min-height: 100vh;
  display: flex;
  align-items: center;
```

```
  justify-content: center;
}

.auth-card {
  background: #020617;
  padding: 24px;
  border-radius: 12px;
  width: 320px;
  border: 1px solid #1f2937;
}

.text-error {
  color: #f97373;
  margin-top: 8px;
}
```