

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проєктами

Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ



к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "магістр"

на тему: Дослідження методів оцінки тривалості проєктів у програмному забезпеченні

Виконав: студент групи 6141 м



Булавкін М.М.
(підпис)

Керівник роботи:

доцент кафедри ІУСТ, к.т.н.

(посада, науковий ступінь вчене звання)



Ворона М.В.
(підпис)

м. Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами

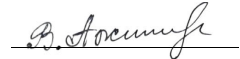
Кафедра інформаційних управляючих систем та технологій

Спеціальність 122 "Комп'ютерні науки"

Освітня програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми



к.т.н, доц. Ажищев В.Ф.

"___" _____ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття ступеня вищої освіти "магістр"

Студенту Булавкін Микола Миколайович

(прізвище, ім'я, по батькові)

1. Тема роботи: Дослідження методів оцінки тривалості проєктів у програмному забезпеченні

Керівник роботи PhD Ворона М.В.

Затверджені наказом ректора № 1355-уч. від "14" листопада 2025 року.

2. Термін подання роботи: 15 грудня 2025 р.

3. Вихідні дані до проєкту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем, матеріали наукового стажування

4. Перелік питань, що належать до розробки (найменування розділів): Завдання на кваліфікаційну роботу; Анотація (українською та англійською мовами); Зміст; Перелік умовних позначень, символів, одиниць та термінів (при необхідності); Вступ; - Дослідження Методів; Проєкт програмного забезпечення; Результати розробки програмного забезпечення; Техніко-Економічне Обґрунтування Розробки; Розділ з охорони праці; Розділ з охорони навколишнього середовища; Висновки; Список використаних джерел; Додатки (технічне завдання, інструкція користувача, програма та методика випробувань програмного забезпечення, текст програми).

5. Перелік презентаційних матеріалів: Титульний слайд, Мета та завдання Кваліфікаційної роботи, Аналіз існуючих рішень, Аналіз вимог, Архітектура програмного забезпечення, Діаграма варіантів використання, Діаграма послідовностей, Діаграма розміщення програмного забезпечення, Діаграма класів, Діаграма потоку даних, Робочий проєкт, Результати розробки, Апробація дослідження, Висновки, Перспективи розвитку проєкту, Заклучний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	<u>Ворона М.В.</u>	05.09.2024	22.12.2024
2	<u>Ворона М.В.</u>	23.12.2024	15.06.2025
3	<u>Ворона М.В.</u>	15.06.2025	25.09.2025
4	<u>Ворона М.В.</u>	25.09.2025	14.11.2025
5	<u>Ворона М.В.</u>	25.09.2025	14.11.2025
6	<u>Ворона М.В.</u>	25.09.2025	14.11.2025

7. Дата видачі завдання 05 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів роботи	Примітка
1	Дослідження та аналіз предметної галузі	22.12.2024	
2	Розробка концепції	15.03.2025	
3	Розділ з розробки проєктних рішень по системі (та її частинах)	15.06.2025	
4	Реалізація проєкту та розробка робочої документації	25.09.2025	
5	Обчислення економічної ефективності	14.11.2025	
6	Вирішення питань охорони праці та навколишнього середовища	14.11.2025	
7	Оформлення пояснювальної записки	14.11.2025	
8	Подання магістерської роботи на перевірку на плагіат	16.11.2025	
9	Подання магістерської роботи рецензенту	20.11.2025	
10	Подання магістерської роботи на державний захист	15.12.2025	

Студент


 (підпис)
Булавкін М.М.

(ПІБ)

Керівник роботи


 (підпис)
Ворона М.В.

(ПІБ)

АНОТАЦІЯ

Дана кваліфікаційна робота присвячена дослідженню та вдосконаленню методів оцінки тривалості проєктів у розробці програмного забезпечення. У сучасних умовах точне прогнозування часу реалізації проєктів є критичним фактором успіху, оскільки дозволяє оптимізувати ресурси, уникнути перевищення бюджету та забезпечити своєчасну здачу продукту. Однак унікальність кожного проєкту та велика кількість впливових факторів ускладнюють отримання точних оцінок, що часто призводить до затримок.

Метою роботи є розробка концепції системи для підвищення точності прогнозування тривалості проєктів шляхом інтеграції методів штучного інтелекту та машинного навчання. Запропонована концепція передбачає автоматичний збір структурованих даних з систем управління проєктами (Jira, Monday.com), репозиторіїв коду (GitHub, GitLab) та інших джерел, створення бази даних історичних проєктів та їх аналіз із застосуванням передових алгоритмів ML, таких як Random Forest, Gradient Boosting та нейронні мережі.

Результатом дослідження є універсальна концепція інструменту, здатного аналізувати параметри нових задач (тип, складність, мова програмування, досвід команди) та надавати реалістичний прогноз їх тривалості. Впровадження подібної системи дозволить значно покращити якість планування, підвищити ефективність управління проєктами та ймовірність їх успішного завершення в галузі розробки ПЗ.

Кваліфікаційна робота викладена на 80 сторінках машинописного тексту та містить: 20 рисунків, 4 додатків, список використаних джерел з 15 найменувань.

Ключові слова: прогнозування тривалості; машинне навчання; управління проєктами; естімація проєктів; штучний інтелект.

ABSTRACT

This master's thesis is dedicated to the research and improvement of project duration estimation methods in software development. In modern conditions, accurate prediction of project timelines is a critical success factor, as it allows for the optimization of resources, prevents budget overruns, and ensures timely product delivery. However, the uniqueness of each project and the large number of influencing factors complicate obtaining accurate estimates, which often leads to delays.

The aim of the work is to develop a system concept for improving the accuracy of project duration forecasting through the integration of artificial intelligence and machine learning methods. The proposed concept involves the automated collection of structured data from project management systems (Jira, Monday.com), code repositories (GitHub, GitLab), and other sources, the creation of a historical project database, and its analysis using advanced ML algorithms such as Random Forest, Gradient Boosting, and neural networks.

The result of the research is a universal concept for a tool capable of analyzing parameters of new tasks (type, complexity, programming language, team experience) and providing a realistic forecast of their duration. The implementation of such a system will significantly improve planning quality, enhance project management efficiency, and increase the likelihood of successful project completion in the field of software development.

The qualification work is laid out on 80 pages of typewritten text and contains: 20 figures, 4 appendices, a list of used sources from 15 titles.

Key words: project duration prediction; machine learning; project management; project estimation; artificial intelligence.

ЗМІСТ

	Стор.
ВСТУП	8
РОЗДІЛ 1. ДОСЛІДЖЕННЯ МЕТОДІВ ОЦІНКИ ТРИВАЛОСТІ ПРОЄКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ	9
1.1 Загальний огляд предметної області	9
1.2 Класифікація підходів до прогнозування	10
1.3 Основні фактори, що впливають на тривалість проєктів	12
1.4 Огляд існуючих інструментів для прогнозування	14
1.5 Виявлення прогалин у предметній області	16
1.6 Концепція системи	17
1.7 Постановка задачі	19
Висновки до розділу 1	19
РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПРОГНОЗУВАННЯ ТРИВАЛОСТІ ІТ-ПРОЄКТІВ	20
2.1. Функціональне проєктування	20
2.2. Архітектурне проєктування системи	23
2.3 Проєктування даних	26
2.4 Проєктування інтерфейсів та інтеграції	31
2.5 Нефункціональні вимоги	42
Висновки до розділу 2	47
РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ОЦІНКИ ТРИВАЛОСТІ ПРОЄКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ	49
3.1 Середовище та інструменти розробки	49
3.2 Реалізація компонентів системи	50
3.3 Впровадження заходів безпеки	55
3.4 Тестування та забезпечення якості	56
3.5 Розгортання системи	57
3.6 Забезпечення експлуатаційної готовності	59
Висновки до розділу 3	60

РОЗДІЛ 4. ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ	61
Висновки до розділу 4	66
РОЗДІЛ 5. ОХОРОНА ПРАЦІ	67
5.1 Загальні положення про охорону праці	67
5.2 Заходи щодо забезпечення безпеки на робочому місці розробника програмного забезпечення	68
Висновки до розділу 5	71
РОЗДІЛ 6. ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА	72
Висновки до розділу 6	76
ВИСНОВКИ (Загальні)	77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	79
ДОДАТКИ	81
Додаток А Технічне завдання на розробку сервісу	81
Додаток Б Інструкція користувача	85
Додаток В Програма та методика випробування програмного забезпечення	90
Додаток Г Фрагмент програмного коду	92

ВСТУП

У сучасному світі інформаційних технологій розробка програмного забезпечення (ПЗ) є однією з найважливіших сфер діяльності, яка активно розвивається. Проекти в галузі програмного забезпечення мають високий рівень складності, а їх успішне завершення залежить від багатьох факторів, серед яких ключовим є точне прогнозування тривалості виконання. Ефективне планування часу проєкту дає змогу оптимізувати використання ресурсів, уникнути перевищення бюджету та забезпечити своєчасну доставку продукту.

Методи оцінки тривалості проєктів у розробці програмного забезпечення є важливим інструментом для управління проєктами, адже вони дозволяють передбачити можливі ризики, планувати етапи розробки, а також визначати реалістичні терміни виконання. Проте, у зв'язку з унікальністю кожного проєкту, часто виникають труднощі у визначенні точних строків, що спричиняє затримки та перевищення витрат. Саме тому вибір ефективного методу оцінки тривалості є критично важливим для успішного завершення проєктів у сфері програмного забезпечення.

У цій роботі буде розглянуто основні методи оцінки тривалості проєктів у програмному забезпеченні, включаючи аналітичні, статистичні, експертні та сучасні підходи, такі як використання машинного навчання та інших інноваційних технологій. Також буде проведено аналіз їх переваг, недоліків та можливостей застосування залежно від типу проєкту [1].

Метою цієї курсової роботи є дослідження методів оцінки тривалості проєктів у програмному забезпеченні, виявлення найбільш ефективних підходів та розробка рекомендацій для їх використання в управлінні проєктами. Це дозволить покращити якість планування та підвищити ймовірність успішного виконання проєктів у галузі розробки ПЗ.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ МЕТОДІВ ОЦІНКИ ТРИВАЛОСТІ ПРОЄКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

1.1 Загальний огляд предметної області

1.1.1 Визначення ролі прогнозування у проєктах з розробки ПЗ

Прогнозування тривалості проєктів у сфері розробки програмного забезпечення є одним із ключових аспектів управління проєктами. Точне визначення строків дозволяє ефективно планувати ресурси, розподіляти задачі серед учасників команди та контролювати виконання проєкту відповідно до встановлених термінів. Успішне прогнозування є важливим не лише для задоволення вимог замовника, але й для забезпечення конкурентоспроможності організації в умовах постійних змін ринку ІТ [2].

Об’єкт дослідження — процеси планування та управління проєктами в розробці програмного забезпечення.

Предмет дослідження — методи, моделі та підходи до оцінки тривалості проєктів у програмному забезпеченні.

Мета роботи — системний аналіз існуючих методів оцінки тривалості проєктів розробки ПЗ, порівняльна оцінка їх точності в залежності від характеристик проєкту, а також розробка рекомендацій щодо їх вибору та впровадження для підвищення якості планування виконання проєктів.

1.1.2 Складності пов’язані із визначенням тривалості проєктів

Унікальність кожного проєкту. Кожен проєкт у розробці програмного забезпечення має власну специфіку, яка залежить від багатьох факторів: складність системи, функціональність, інноваційність, які потрібно реалізувати.

Невизначеність у вимогах. Під час розробки програмного забезпечення часто виникають ситуації, коли початкові вимоги змінюються в процесі роботи. Це може бути пов’язано з оновленими потребами замовника, зміною ринкових умов або технічними обмеженнями. Така динамічність значно ускладнює точне прогнозування.

Залежність від людського фактора. Люди, які працюють над проєктом, є головним ресурсом у розробці програмного забезпечення. Продуктивність команди, її досвід, мотивація та здатність вирішувати складні задачі мають значний вплив на тривалість проєкту. Крім того, непередбачувані фактори, як-от плинність кадрів або виснаження, можуть вплинути на строки виконання завдань.

Обмеженість ресурсів. Часто проєкти розробки програмного забезпечення виконуються в умовах обмеженого бюджету, обмеженого часу або дефіциту висококваліфікованих спеціалістів.

1.1.3. Роль точного прогнозування у забезпеченні ефективного управління проєктами

Точне прогнозування тривалості проєктів відіграє ключову роль у їх успішному виконанні. Воно дозволяє:

Оптимізувати використання ресурсів. Завдяки точному прогнозуванню можна уникнути перевантаження команди або простоїв.

Контролювати витрати. Реалістичні строки допомагають зменшити ризик перевищення бюджету.

Забезпечити задоволення замовника. Вчасна доставка продукту зміцнює довіру клієнтів і репутацію компанії.

Підвищити якість управління ризиками. Завдяки прогнозуванню менеджери можуть краще планувати способи реагування на потенційні загрози.

Прогнозування тривалості є основою для створення детального плану проєкту, розподілу задач, контролю прогресу та прийняття стратегічних рішень. Без точного прогнозування зростає ймовірність провалу проєкту через перевищення строків, зростання витрат або незадоволення замовника кінцевим результатом.

1.2 Класифікація підходів до прогнозування

Розподіл методів прогнозування на основі їхньої природи

Методи прогнозування тривалості проєктів у розробці програмного забезпечення поділяються на декілька категорій залежно від їхньої природи та способу отримання результатів [3]. До них належать аналітичні, експертні, статистичні підходи та сучасні технології прогнозування.

1.2.1 Аналітичні методи

Ці методи базуються на формульних моделях і математичних розрахунках, які враховують основні характеристики проєкту.

Модель COCOMO (Constructive Cost Model): Одна з найпопулярніших аналітичних моделей для оцінки тривалості та витрат у розробці ПЗ.

Вона враховує розмір програмного забезпечення (у рядках коду або функціональних точках) та інші параметри, наприклад, складність проєкту, кваліфікацію команди, середовище розробки.

Версія COCOMO II дозволяє враховувати сучасні реалії, такі як використання гнучких методологій та повторне використання коду.

Переваги: точність для проєктів, де відомі кількісні характеристики (розмір, складність).

Недоліки: чутливість до початкових даних, потреба в детальному аналізі до початку проєкту.

1.2.2. Експертні оцінки

Цей підхід базується на думці фахівців, які мають досвід у розробці схожих проєктів.

Метод Delphi. Проводиться анонімне опитування групи експертів. Їхні оцінки аналізуються, після чого проводяться додаткові раунди для досягнення консенсусу.

Planning Poker. Використовується в Agile-проєктах. Учасники команди обговорюють задачі та оцінюють їхню складність і тривалість шляхом вибору карток із числами.

Переваги: швидкість і простота, врахування досвіду команди.

Недоліки: суб'єктивність, залежність від кваліфікації експертів, не працює в маленькій команді

1.2.3. Статистичні підходи

Статистичні методи базуються на аналізі історичних даних та їхньому моделюванні.

Регресійний аналіз. Використовується для виявлення залежностей між параметрами проєкту (наприклад, обсягом роботи) та тривалістю.

Методи апроксимації. Пошук математичних функцій, які найкраще описують залежності у даних.

Переваги: точність за наявності великого обсягу даних.

Недоліки: залежність від якості та обсягу історичних даних.

1.2.4. Сучасні технології прогнозування

Застосування інноваційних підходів, таких як машинне навчання та штучний інтелект, для прогнозування тривалості проєктів.

Машинне навчання. Використання алгоритмів (наприклад, Random Forest, Gradient Boosting) для аналізу великих обсягів даних і виявлення прихованих закономірностей.

Нейронні мережі. Підходять для моделювання складних взаємозалежностей між параметрами проєкту [4].

Переваги: здатність обробляти складні взаємозв'язки, висока точність за наявності якісних даних.

Недоліки: складність реалізації, потреба у великих обсягах даних.

1.3 Основні фактори, що впливають на тривалість проєктів

1.3.1. Характеристики проєкту

Ця група факторів пов'язана з властивостями самого проєкту, які визначають його масштаб, складність та тип програмного продукту.

Масштаб проєкту. Великі проєкти зазвичай потребують більше часу через більшу кількість функцій, складнішу архітектуру та потребу в координації між учасниками.

Наприклад, розробка мобільного додатка для локального використання займе значно менше часу, ніж створення багатофункціональної корпоративної платформи.

Складність проєкту. Складні проєкти можуть включати інноваційні рішення, інтеграцію з різними системами або роботу з нестандартними технологіями. Це вимагає більше часу для дослідження, тестування та відлагодження.

Тип продукту. Різні типи програмного забезпечення (веб-додатки, мобільні додатки, системи реального часу тощо) мають свої особливості розробки. Наприклад, розробка програм для систем, які працюють у режимі реального часу, вимагає ретельного тестування продуктивності.

1.3.2. Команда розробки

Продуктивність команди розробників є одним із найважливіших факторів, що впливають на тривалість проєкту.

Кваліфікація команди. Досвідчена команда з високим рівнем експертизи може виконати проєкт швидше та якісніше. Новачки, навпаки, можуть потребувати більше часу через потребу в навчанні або адаптації до нових технологій.

Чисельність команди. Кількість учасників команди впливає на швидкість виконання завдань. Проте надмірне збільшення команди може призвести до складнощів у комунікації та координації.

Продуктивність. Продуктивність окремих членів команди залежить від їхнього досвіду, мотивації, умов праці та ефективності управління проектом.

1.3.3. Інструменти та технології

Використання сучасних інструментів та технологій може значно вплинути на швидкість виконання завдань.

Фреймворки. Використання готових фреймворків (наприклад, Django для Python або Spring для Java) дозволяє прискорити процес розробки завдяки вже наявному набору функцій.

Мови програмування. Деякі мови програмування забезпечують швидшу розробку за рахунок своєї простоти та великої кількості готових бібліотек (наприклад, Python), тоді як інші потребують більше часу через свою складність (наприклад, C++).

Системи контролю версій та CI/CD. Інструменти для автоматизації (Git, Jenkins тощо) зменшують час на ручне тестування та інтеграцію.

1.3.4. Зовнішні фактори

Фактори, що не залежать безпосередньо від команди розробників, також відіграють значну роль [5].

Вимоги замовника. Зміни у вимогах замовника можуть призвести до збільшення тривалості проєкту. Наприклад, якщо замовник додає нову функціональність на пізніх етапах розробки, це може значно затримати завершення.

Проектна документація. Нечіткі або неповні технічні вимоги ускладнюють планування та виконання завдань. Постійна потреба в уточненнях може призвести до затримок.

Зовнішні умови. Пандемії, економічні кризи, проблеми з постачанням обладнання чи інші глобальні фактори можуть впливати на строки реалізації проєктів.

Висновок

Тривалість проєктів у розробці програмного забезпечення залежить від багатьох факторів, які можуть бути як внутрішніми (характеристики проєкту, команда), так і зовнішніми (вимоги замовника, зовнішні умови). Успішне прогнозування строків можливе лише за умови всебічного аналізу всіх ключових факторів. Для цього важливо використовувати сучасні інструменти та технології, ефективно управляти командою і підтримувати тісний зв'язок із замовником.

1.4 Огляд існуючих інструментів для прогнозування

1.4.1. Інструменти для управління проєктами та прогнозування

Популярні платформи для управління проєктами, які також забезпечують функції прогнозування тривалості задач [6].

Jira. Система управління проєктами, використовується в IT-компаніях.

Можливості.

Планування задач у гнучких методологіях (Scrum, Kanban).

Оцінка тривалості задач за допомогою Planning Poker або інших методів.

Моніторинг виконання задач і аналіз затримок.

Переваги: зручність у використанні, інтеграція з іншими інструментами (Confluence, Bitbucket).

Недоліки: обмежені функції для складних моделей прогнозування.

Microsoft Project. Професійний інструмент для управління проектами з розширеними можливостями прогнозування.

Можливості

Розрахунок залежностей між задачами.

Оцінка часу виконання проєктів із врахуванням обмежень.

Переваги: підтримка великих проєктів, детальний аналіз критичного шляху.

Недоліки: складність освоєння для новачків, висока вартість.

Monday.com. Інтуїтивно зрозуміла платформа для планування проєктів.

Можливості: візуалізація строків за допомогою діаграм Ганта, прогнозування строків виконання на основі динамічного оновлення статусу задач. Переваги: простота у використанні, підтримка інтеграцій із багатьма інструментами. Недоліки: обмежена функціональність для складних проєктів.

1.4.2. Інструменти на основі штучного інтелекту

Сучасні технології, що використовують великі дані та машинне навчання для прогнозування строків проєктів.

Прогнозування на основі машинного навчання. Алгоритми, такі як Random Forest або Gradient Boosting, аналізують історичні дані проєктів і знаходять закономірності. Переваги: автоматичний аналіз великих обсягів даних, точність у разі якісних даних. Недоліки: потреба в значній кількості даних для тренування.

Нейронні мережі. Використовуються для складних взаємозв'язків між факторами, що впливають на строки. Наприклад, визначення тривалості задач на основі даних про складність, команду, технології. Переваги: здатність враховувати нелінійні залежності. Недоліки: складність реалізації, потреба у високих обчислювальних потужностях.

Інструменти штучного інтелекту для прогнозування. Наприклад, Google AutoML, IBM Watson Project Insights. Автоматизують аналіз даних і пропонують прогнози на основі історичних даних. Переваги: економія часу, висока точність. Недоліки: висока вартість, залежність від хмарних сервісів.

Огляд інструментів та методів показує, що вибір залежить від типу проєкту, доступності даних і потреб користувача. Для невеликих проєктів ефективними є інструменти, такі як Jira або Monday.com, тоді як великі корпоративні проєкти потребують використання потужних платформ на кшталт Microsoft Project чи спеціалізованих моделей, як-от PERT і CPM. У сучасних умовах все

більшої популярності набувають інструменти на основі штучного інтелекту, які дозволяють автоматизувати аналіз даних і підвищити точність прогнозування.

1.5. Виявлення прогалин у предметній області

1.5.1. Обмеження існуючих підходів до прогнозування

Сучасні методи (аналітичні, експертні, статистичні та ML) мають суттєві недоліки, що знижують їхню універсальність [7].

- **Жорсткість аналітичних методів.** Інструменти на кшталт COCOMO вимагають детальних специфікацій на ранніх етапах. Це знижує точність у проєктах зі змінними вимогами, де повна інформація спочатку відсутня.
- **Суб'єктивність експертних оцінок.** Точність методів типу Planning Poker залежить від кваліфікації команди. Брак досвіду або неповні дані призводять до значних похибок.
- **Залежність від історичних даних.** Статистичні підходи (регресійний аналіз) потребують великих масивів релевантних даних, які часто відсутні для нових або інноваційних розробок.
- **Ресурсомісність новітніх технологій.** Методи машинного навчання вимагають значних обчислювальних потужностей та специфічної експертизи, що ускладнює їх впровадження в невеликих компаніях.

1.5.2. Проблеми адаптації загальних моделей

Унікальні характеристики програмних проєктів ускладнюють використання універсальних моделей:

- **Масштабування.** Загальні моделі орієнтовані на середньостатистичні показники. Вони потребують складної калібрування для коректної роботи з масштабними корпоративними системами або малими стартапами.
- **Фактор інновацій.** Стандартні підходи часто ігнорують непередбачувані витрати часу на дослідження (R&D), притаманні інноваційним продуктам.
- **Конфлікт методологій.** Моделі, розроблені для Waterfall, неефективні для проєктів, що використовують Agile-підходи через відмінності в життєвому циклі розробки.

1.5.3. Нестача інструментарію для гнучких методологій (Agile, Scrum)

- Специфіка гнучкої розробки створює бар'єри для точного прогнозування:
- **Динамічність вимог.** Ітеративний підхід та часті зміни у вимогах (change requests) роблять моделі з фіксованими параметрами неактуальними.
- **Рівень деталізації.** У Scrum оцінюються окремі задачі (user stories), тоді як більшість інструментів орієнтовані на прогноз проєкту в цілому, ігноруючи мікрорівень.
- **Інструментальний вакуум.** Класичні системи управління (наприклад, MS Project) заточені під каскадні процеси й не забезпечують належної інтеграції з динамікою Agile.

Висновок

Аналіз предметної області виявив критичні прогалини: існуючі підходи залежні від історичних даних, складні у впровадженні та погано адаптуються до унікальних умов проєктів. Особливо гострою є проблема відсутності інструментів, що враховують природу гнучких методологій. Це обґрунтовує необхідність розробки нових адаптивних моделей, здатних ефективно працювати в умовах невизначеності та динамічних змін сучасного IT-ринку.

1.6 Концепція системи

Концепція системи, яка спрямована на підвищення точності прогнозування тривалості проєктів у розробці програмного забезпечення, базується на інтеграції сучасних технологій, таких як штучний інтелект (AI) і машинне навчання (ML). Основна ідея полягає в автоматичному зборі даних про виконання завдань, аналізі цих даних та використанні їх для точного прогнозування часу реалізації майбутніх задач [8].

Ключові аспекти концепції

1. Автоматичний збір даних

Система збирає дані про виконані завдання у реальних проєктах, враховуючи такі параметри:

Тривалість виконання задачі. Скільки часу зайняла конкретна задача.

Тип задачі. Функціональність, виправлення багів, оптимізація тощо.

Мова програмування. Python, Java, C++, JavaScript тощо.

Рівень складності. Оцінка задачі командою перед її виконанням.

Рівень досвіду виконавців. Наприклад, Junior, Middle, Senior.

Розмір команди. Кількість учасників, залучених до виконання задачі.

Дані можуть збиратися з таких джерел:

Системи управління проєктами (наприклад, Jira, Monday.com).

Репозиторії коду (наприклад, GitHub, GitLab).

Системи відстеження часу (наприклад, Toggl).

2. Створення бази даних історичних проєктів

Зібрані дані зберігаються у структурованій базі даних, яка використовується для подальшого аналізу. База даних включає наступне.

Метадані про проєкти (тип, масштаби, технології).

Відомості про задачі (тривалість, складність, виконані дії).

Профілі команд (рівень досвіду, розмір команди).

3. Аналіз даних за допомогою штучного інтелекту

Система використовує алгоритми машинного навчання для аналізу даних і виявлення залежностей між параметрами задачі та часом виконання.

Алгоритми: Random Forest, Gradient Boosting, нейронні мережі.

4. Прогнозування на основі зібраних даних

Коли користувач вводить опис нової задачі, система: аналізує введені параметри (тип задачі, мову програмування, складність тощо).

Використовує навчений AI для прогнозування тривалості виконання [9].

Пропонує користувачеві реалістичний естимейт часу та ресурсів.

1.7 Постановка задачі

Для досягнення поставленої мети та реалізації запропонованої концепції необхідно виконати такі завдання:

1. Провести вибір та обґрунтування алгоритмів машинного навчання, придатних для аналізу історичних даних розробки ПЗ.
2. Спроекувати базу даних для зберігання параметрів задач (складність, тип, стек технологій) та інформації про команду.
3. Розробити архітектуру інформаційної системи прогнозування з урахуванням вимог до інтеграції та обробки даних.
4. Реалізувати програмний модуль (або веб-додаток), що здійснює збір даних та видачу прогнозів на основі навченої моделі.
5. Провести тестування розробленої системи та оцінити точність прогнозування за допомогою метрик похибки.

Висновок до розділу 1

Розроблена концепція базується на застосуванні штучного інтелекту для автоматичного аналізу історичних даних про виконання задач. Це дозволяє створити універсальний інструмент, який адаптується до конкретних проєктів і забезпечує точність естимейтів. Система стане незамінним помічником у процесі управління проєктами, підвищуючи ефективність роботи команди та якість планування.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПРОГНОЗУВАННЯ ТРИВАЛОСТІ ІТ-ПРОЄКТІВ

2.1. Функціональне проєктування

2.1.1. Визначення акторів та варіантів використання

Діаграма варіантів використання системи наведена на рисунку 2.1

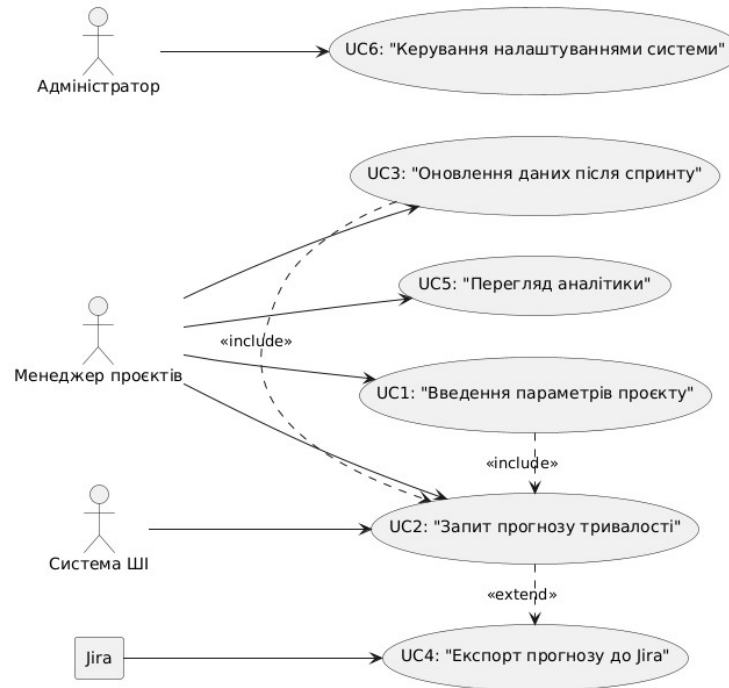


Рисунок 2.1 Use Case Diagram.

Пояснення

- Актори

- **Менеджер проєктів:** Основний користувач, який взаємодіє з системою для отримання прогнозів.

- **Адміністратор:** Налаштовує систему, інтегрує зовнішні сервіси.

- **Система III:** Генетичний алгоритм, який обчислює прогнози.

- **Jira:** Зовнішня система управління проєктами.

- Ключові варіанти використання

1. **UC1:** Введення параметрів проєкту (F, S, E, T, C).

2. **UC2:** Запит прогнозу тривалості (запуск ГА).

3. **UC3:** Оновлення даних після спринту (наприклад, нові задачі, зміна команди).

4. **UC4:** Експорт прогнозованого дедлайну до Jira.

5. **UC5:** Перегляд аналітики (графіки збіжності ГА, порівняння з історичними даними).

6. **UC6:** Керування налаштуваннями (ймовірність мутації, розмір популяції ГА).

2.2.2. Деталізація сценаріїв взаємодії

Sequence Diagram для UC2 (Запит прогнозу тривалості), рисунок 2.2

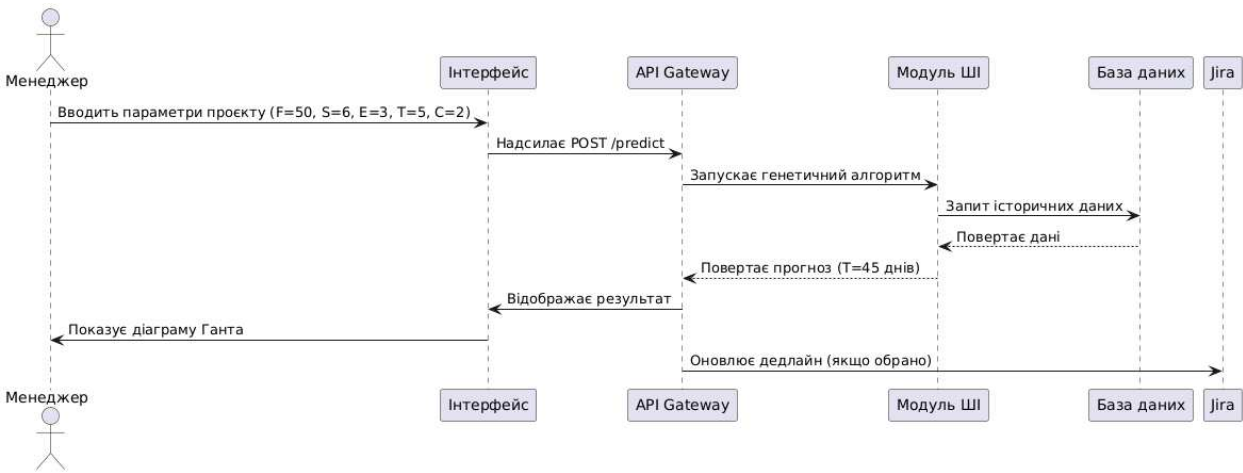


Рисунок 2.2 Діаграма послідовності.

Пояснення

- Менеджер вводить параметри проєкту через інтерфейс.
- Інтерфейс надсилає запит до API Gateway.
- API Gateway запускає модуль ШІ, який використовує ГА для прогнозу.
- Модуль ШІ отримує історичні дані з бази даних для підвищення точності.
- Прогноз повертається через API у вигляді діаграми Ганта.
- Система автоматично оновлює дедлайн у Jira (опційно).

Діаграма активності

Activity Diagram для UC3 (Оновлення даних після спринту). Рисунок 2.3

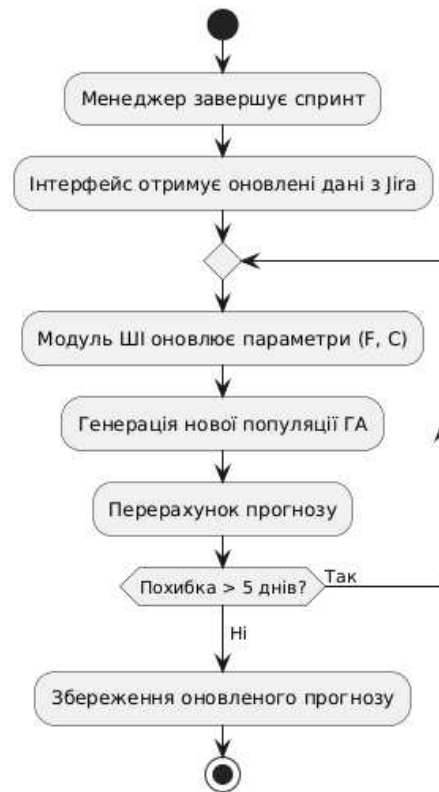


Рисунок 2.3 Діаграма активності.

Пояснення

- Після завершення спринту система отримує оновлені дані (наприклад, нові задачі).
- Модуль ШІ коригує параметри (F) (кількість вимог) та (C) (частота змін).
- ГА перераховує прогноз, доки похибка не стане ≤ 5 днів.
- Остаточний прогноз зберігається в базі даних.

Зв'язок з архітектурою

- **Frontend:** Відповідає за взаємодію з менеджером (UC1, UC5).
- **Backend API:** Обробляє запити (UC2, UC3, UC4).
- **Модуль ШІ:** Реалізує UC2 (прогнозування).
- **База даних:** Зберігає дані для UC2, UC3.
- **Інтеграція з Jira:** Реалізує UC4.

Функціональне проектування визначило:

- Ролі користувачів (менеджер, адміністратор) та їхні взаємодії з системою.
- Ключові сценарії роботи, що підтверджують практичну реалізацію архітектурних рішень.
- Логіку інтеграції з Jira для автоматизації планування.

2.2. Архітектурне проєктування системи

2.2.1. Вибір та обґрунтування архітектурного стилю

Обрано **мікросервісну архітектуру** з наступними обґрунтуваннями:

- **Масштабованість:** Можливість окремо масштабувати модуль ШІ, який потребує більше ресурсів.
- **Гнучкість:** Легка інтеграція зі зовнішніми системами (Jira, Trello) через окремі сервіси.
- **Надійність:** Відмова одного мікросервісу (наприклад, аналітики) не паралізує всю систему.
- **Технологічна незалежність:** Можливість використовувати різні мови/фреймворки для різних компонентів.

Архітектурна схема (високого рівня) наведена на рисунку 2.4

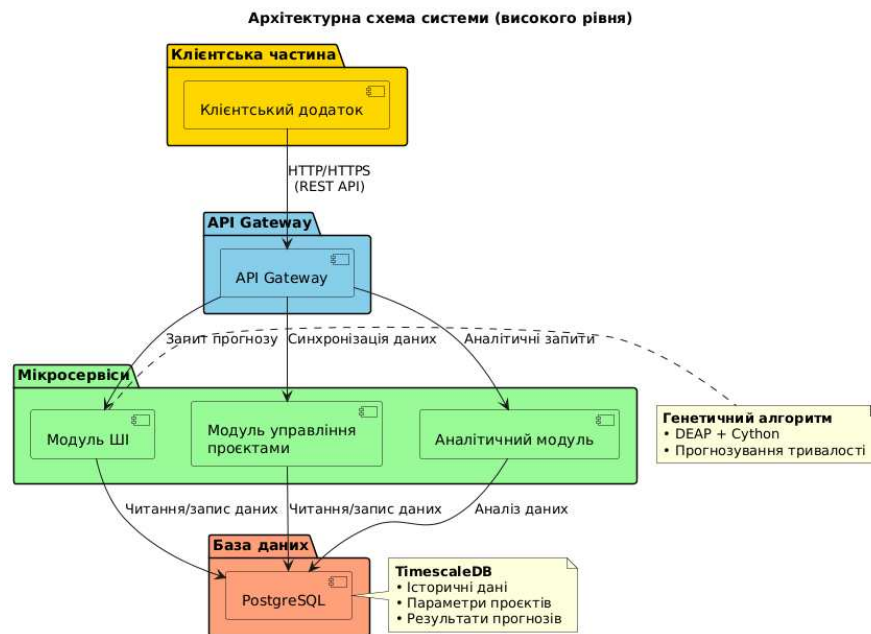


Рисунок 2.4 Архітектурна схема

2.2.2. Компонентна архітектура системи

Діаграма компонентів наведена на рисунку 2.5

Основні модулі

Технології: React.js + D3.js для інтерактивних діаграм.

Технології: FastAPI (Python) з підтримкою асинхронності.

Технології: Python, бібліотеки DEAP, NumPy, інтеграція з Cython для оптимізації.

Технології: PostgreSQL + TimescaleDB для часових рядів.

Технології: Jira REST API, бібліотека jira-python.

Аналітичний модуль

- Відповідальність: Генерація звітів, аналіз ефективності прогнозів.
- Технології: Python (Pandas, Matplotlib).

2.2.3. Вибір технологічного стеку

Технологічний стек представлений у таблиці 2.1

Таблиця 2.1 Технології розробки.

Компонент	Технології	Обґрунтування
Frontend	React.js, D3.js	Гнучкість, велика спільнота, підтримка інтерактивної візуалізації.
Backend API	FastAPI (Python)	Швидкість, асинхронність, інтуїтивна інтеграція з Python-бібліотеками для ШІ.
Модуль ШІ	Python, DEAP, Cython	DEAP — спеціалізована бібліотека для ГА; Cython прискорює обчислення.
База даних	PostgreSQL, TimescaleDB	Підтримка структурних даних і часових рядів, масштабованість.
Інтеграція	Jira REST API, jira-python	Стандарт для інтеграції з Jira, простота використання.
Контейнеризація	Docker, Kubernetes	Забезпечення ізольованості мікросервісів, автоматичне масштабування.

Запропонована архітектура забезпечує гнучкість, масштабованість та ефективну інтеграцію з інструментами управління проєктами. Вибір технологій обґрунтовано вимогами до продуктивності ($MAE \leq 5$ днів) та інтеграції з Agile-середовищем.

2.3. Проектування даних

2.3.1. Концептуальна модель даних (Class Diagram)

Діаграма класів наведена на рисунку 2.6

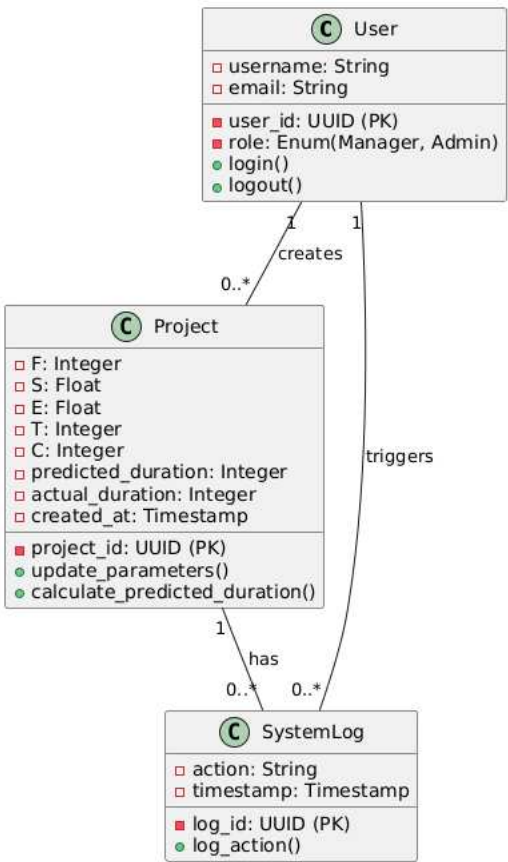


Рисунок 2.6 Class Diagram.

Пояснення:

- **Класи:**
 1. **User:** Містить дані користувачів (менеджер/адмін).
 2. **Project:** Основна сутність для зберігання параметрів проєкту та результатів прогнозу.
 3. **SystemLog:** Записує дії користувачів (наприклад, оновлення параметрів, експорт до Jira).
- **Зв'язки:**
 - Користувач створює проєкти.
 - Користувач ініціює записи в журналі.
 - Проєкт може мати кілька записів у журналі.

2.3.2. Логічна модель бази даних (ER-діаграма)

Логічна модель бази даних наведена на рисунку 2.7

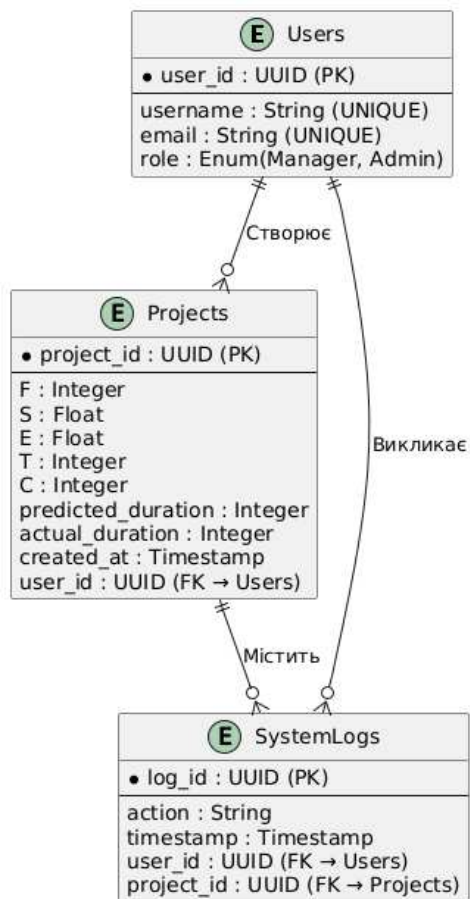


Рисунок 2.7 Entity-relationship model.

Таблиці та поля БД наведено у таблиці 2.2

Таблиця 2.2 Опис таблиць

Таблиця	Поле	Тип даних	Обмеження
Users	user_id	UUID	PRIMARY KEY
	username	VARCHAR(50)	UNIQUE, NOT NULL
	email	VARCHAR(100)	UNIQUE, NOT NULL
	role	ENUM	NOT NULL
Projects	project_id	UUID	PRIMARY KEY
	F	INTEGER	NOT NULL
	S	FLOAT	NOT NULL
	E	FLOAT	NOT NULL
	T	INTEGER	NOT NULL
	C	INTEGER	NOT NULL
	predicted_duration	INTEGER	NOT NULL
	actual_duration	INTEGER	NOT NULL

Таблиця	Поле	Тип даних	Обмеження
	actual_duration	INTEGER	NULL
	created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP
	user_id	UUID	FOREIGN KEY (Users)
SystemLog	log_id	UUID	PRIMARY KEY
	action	VARCHAR(200)	NOT NULL
	timestamp	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP
	user_id	UUID	FOREIGN KEY (Users)
	project_id	UUID	FOREIGN KEY (Projects)

Ключові зв'язки:

- **Users ↔ Projects:** Один користувач (Manager/Admin) може створити багато проєктів (user_id у Projects як зовнішній ключ).
- **Users ↔ SystemLogs:** Один користувач може викликати багато логів (user_id у SystemLogs).
- **Projects ↔ SystemLogs:** Один проєкт може мати багато записів у логах (project_id у SystemLogs).

Нормалізація:

- **1НФ:** Усі атрибути атомарні. Наприклад, role зберігається як ENUM, а не строка з кількома значеннями [10].
- **2НФ:** Усі неключові атрибути повністю залежать від первинного ключа. Наприклад, predicted_duration залежить від project_id, а не від user_id.
- **3НФ:** Немає транзитивних залежностей. Наприклад, email користувача не впливає на прогноз тривалості.

2.3.3. Фізична модель бази даних

Ключові рішення:

- **СУБД:**
 - **PostgreSQL + TimescaleDB** (обґрунтування):
 - Підтримка часових рядів для аналітики прогнозів.
 - Можливість секціонування таблиць за часом (created_at).
 - JSONB для зберігання додаткових параметрів (наприклад, історія мутацій ГА).
- **Стратегії індексування:**
 - **Таблиця Projects:**
 - Складний індекс (F, S) для швидкого пошуку проєктів зі схожими характеристиками.
 - Індекс created_at для фільтрації за датою створення.

- **Таблиця SystemLogs:**
 - Індекс timestamp для аналізу активності за період.
 - Індекс user_id для моніторингу дій користувачів.
- **Партиціонування:**
 - **Горизонтальне секціонування** таблиці Projects за полем created_at:


```
CREATE TABLE projects_2023 PARTITION OF projects
FOR VALUES FROM ('2023-01-01') TO ('2024-01-01');
```
 - **Вертикальне секціонування:** Відокремлення часто використовуваних полів (F, S, T) від рідких (actual_duration).
- **Налаштування зберігання:**
 - **Табличні простори:** Для оптимізації I/O — дані зберігаються на SSD, індекси на HDD.
 - **Реплікація:** Налаштовано асинхронну реплікацію для резервування.
- **Безпека:**
 - **Шифрування:**
 - Диски: AES-256.
 - Дані: Шифрування колонок за допомогою pgcrypto (наприклад, email у Users).
 - **Ролі:**
 - admin_role: Повний доступ до всіх таблиць.
 - manager_role: Читання/запис у Projects, лише читання SystemLogs.
 - **Аудит:** Тригери для запису спроб несанкціонованого доступу.
- **Оптимізація запитів:**
 - **Матеріалізовані подання:** Для звітів з аналітикою (наприклад, "Середня тривалість проєктів за місяць").
 - **Кешування:** Використання Redis для кешування результатів популярних запитів.
 - **Оптимізація JOIN-запитів:** Використання індексів та обмеження кількості рядків за допомогою LIMIT.

Приклад оптимізованого запиту:

```
-- Пошук проєктів з F > 30 та S > 5.0
SELECT * FROM projects
WHERE F > 30 AND S > 5.0
ORDER BY created_at DESC
LIMIT 10;
```

Зв'язок з архітектурою

- **Нормалізація** забезпечує ефективне зберігання даних для модуля ІІІ.

- **Партиціонування** підтримує масштабованість системи (розділ 2.1.1).
- **Індекси** відповідають вимогам до швидкості ($MAE \leq 5$ днів).

Фізична модель реалізована з урахуванням продуктивності, безпеки та майбутнього масштабування. Вона повністю відповідає функціональним вимогам системи.

2.4. Проектування інтерфейсів та інтеграції

2.4.1. Інтерфейси взаємодії з зовнішніми системами

Система інтегрується зі зовнішніми сервісами управління проєктами (Jira, Trello) через REST API. Основні цілі інтеграції:

- **Імпорт задач:** Отримання актуальних задач, їх параметрів (кількість user stories, статуси).
- **Експорт прогнозів:** Оновлення дедлайнів у зовнішніх системах.
- **Синхронізація змін:** Відстеження змін у вимогах після спринтів.

Протоколи та формати даних

- **Протокол:**
 - **REST API** — через простоту, масштабованість та підтримку JSON.
 - **HTTPS** — для шифрування даних.
- **Формат даних:**
 - **JSON** — стандарт для обміну даними між веб-сервісами.
 - Приклад запиту до Jira:

```
{ "fields": { "summary": "Прогнозований дедлайн", "duedate": "2023-12-31" }}
```

- **Аутентифікація:**

- **OAuth 2.0** для Jira (наприклад, токени з обмеженим терміном дії).
- **API-ключі** для Trello.
- Приклад заголовка запиту з токеном:

```
Authorization: Bearer <token>
```

Специфікація точок інтеграції табл. 2.3-2.4

Таблиця 2.3 - для Jira

Ендпоінт	Метод	Опис
/rest/api/3/issue/{issueKey}	GET	Отримання інформації про задачу.
/rest/api/3/issue/{issueKey}	PUT	Оновлення дедлайну (duedate).
/rest/api/3/search	POST	Пошук задач за критеріями (напр., статус "Open").

Приклад оновлення дедлайну:

```
curl -X PUT -H "Authorization: Bearer {token}" \
  -H "Content-Type: application/json" \
  -d '{"fields": {"duedate": "2023-12-31"}}' \
  https://jira.example.com/rest/api/3/issue/PRJ-123
```

Таблиця 2.3 - для Trello

Ендпоінт	Метод	Опис
/1/cards/{cardId}	PUT	Оновлення дати завершення картки.
/1/boards/{boardId}/cards	GET	Отримання списку карток.

Обробка помилок та відмовистійкість

- **Retry-механізми:** Повторні спроби запитів при помилках 5xx (наприклад, 3 спроби з інтервалом 10 сек).
- **Логування:** Збереження деталей помилок у таблиці SystemLogs з полями error_code, timestamp, endpoint.
- **Webhook-и:** Підписка на зміни в Jira/Trello для миттєвої синхронізації.

Jira Webhook URL: https://our-system.com/api/webhooks/jira

Схема взаємодії наведена на рисунку 2.8

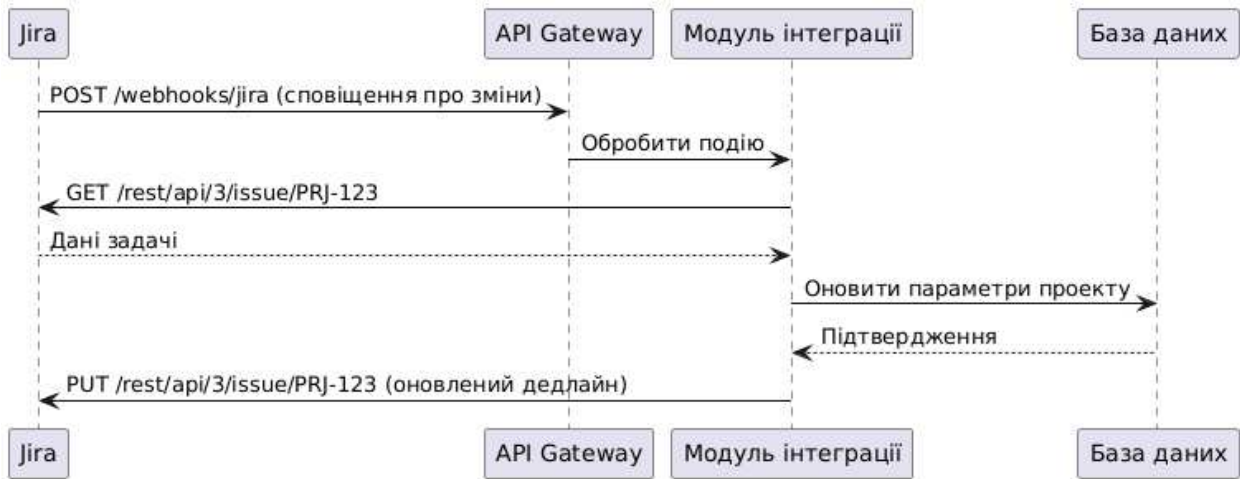


Рисунок 2.8 Sequence Diagram.

Обґрунтування рішень

- **REST API** обрано через універсальність і сумісність з SaaS-платформами.
- **JSON** дозволяє легко інтегруватися з Python-бібліотеками (requests).
- **OAuth 2.0** забезпечує безпеку без зберігання паролів.

Спроектвані інтерфейси забезпечують надійну двосторонню інтеграцію з Jira/Trello. Використання REST API, JSON та сучасних механізмів безпеки робить систему сумісною з інструментами Agile-команд.

2.4.2. Проектування API системи

Система надає RESTful API для взаємодії з фронтендом та зовнішніми сервісами [11]. Основні ендпоінти:

Базовий URL

https://api.project-predictor.com/v1

Список основних та ендпоінтів наведен у таблицях 2.3, 2.4, 2.5, 2.6.

Ресурси та ендпоінти

1. Проекти (/projects) Таблиця 2.4.

Таблиця 2.4 Управління проектами.

Метод	Ендпоінт	Опис
GET	/projects	Список усіх проектів (з фільтрами).
POST	/projects	Створення нового проекту.
GET	/projects/{id}	Деталі проекту за ID.
PUT	/projects/{id}	Оновлення параметрів проекту.
POST	/projects/{id}/predict	Запуск прогнозування тривалості.

Приклади:

- **Запит створення проекту (POST /projects):**

```
{ "F": 50, "S": 6.0, "E": 3.5, "T": 5, "C": 2}
```
- **Відповідь (201 Created):**

```
{ "project_id": "a1b2c3d4", "predicted_duration": 45, "created_at": "2023-10-20T12:00:00Z"}
```
- **Запит прогнозу (POST /projects/{id}/predict):**

```
curl -X POST -H "Authorization: Bearer {token}" \
https://api.project-predictor.com/v1/projects/a1b2c3d4/predict
```
- **Відповідь (200 OK):**

```
{ "project_id": "a1b2c3d4", "predicted_duration": 45, "confidence": 0.85}
```

2. Користувачі (/users) Таблиця 2.5

Таблиця 2.5 Управління користувачами (менеджери, адміністратори).

Метод	Ендпоінт	Опис
POST	/users/login	Авторизація (отримання JWT).
POST	/users	Реєстрація нового користувача.
GET	/users/{id}	Інформація про користувача.

Приклад авторизації (POST /users/login):

```
{ "email": "manager@example.com", "password": "secure123"}
```

Відповідь (200 OK):

```
{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", "role": "Manager" }
```

3. Логги (/logs) Таблица 2.6

Таблица 2.6 Отримання системних логів.

Метод	Ендпоінт	Опис
GET	/logs	Фільтрація логів за датою, користувачем.

Приклад запити (GET /logs?user_id=u1&start_date=2023-10-01)

Відповідь (200 OK):

```
[
  {
    "log_id": "log1",
    "action": "Project created",
    "timestamp": "2023-10-20T12:05:00Z",
    "user_id": "u1"
  }
]
```

4. Інтеграція з Jira (/integrations/jira) Таблица 2.7

Таблица 2.7 Синхронізація з Jira.

Метод	Ендпоінт	Опис
POST	/integrations/jira/sync	Синхронізація задач з Jira.
POST	/integrations/jira/export	Експорт прогнозу до Jira.

Приклад експорту (POST /integrations/jira/export):

```
{ "project_id": "a1b2c3d4", "jira_issue_key": "PRJ-123" }
```

Відповідь (200 OK):

```
{ "status": "success", "jira_due_date": "2023-12-31" }
```

Формати запитів/відповідей

- **Content-Type:** application/json.
- **Стандартні коди відповідей:**
 - 200 OK: Успішний запит.
 - 201 Created: Ресурс створено.
 - 400 Bad Request: Помилка валідації (напр., F < 0).
 - 401 Unauthorized: Не валідний токен.
 - 404 Not Found: Ресурс не знайдено.

Приклад помилки (400 Bad Request):

```
{ "error": "Invalid parameters", "details": [ "F must be greater than 0" ] }
```

Аутентифікація

- **JWT-токен** передається в заголовку:
Authorization: Bearer <token>
- **Ролі:**
 - Manager: Доступ до /projects, /logs (read-only).
 - Admin: Доступ до всіх ендпоінтів.

Документація

API документовано за допомогою **Swagger/OpenAPI 3.0**.

Приклад специфікації:

```
openapi: 3.0.0
info:
  title: Project Predictor API
  version: 1.0.0
paths:
  /projects:
    post:
      summary: Створення проєкту
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Project'
      responses:
        '201':
          description: Проєкт створено
```

API охоплює всі ключові сценарії використання системи, від управління проєктами до інтеграції з Jira. Використання стандартних методів

HTTP та JSON забезпечує зручність для розробників.

2.4.3. Концепція інтерфейсу користувача (UI)

Інтерфейс системи орієнтований на зручність, інтуїтивність та візуалізацію даних. Основні екрани та їхні ключові елементи наведені на рисунках 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15.

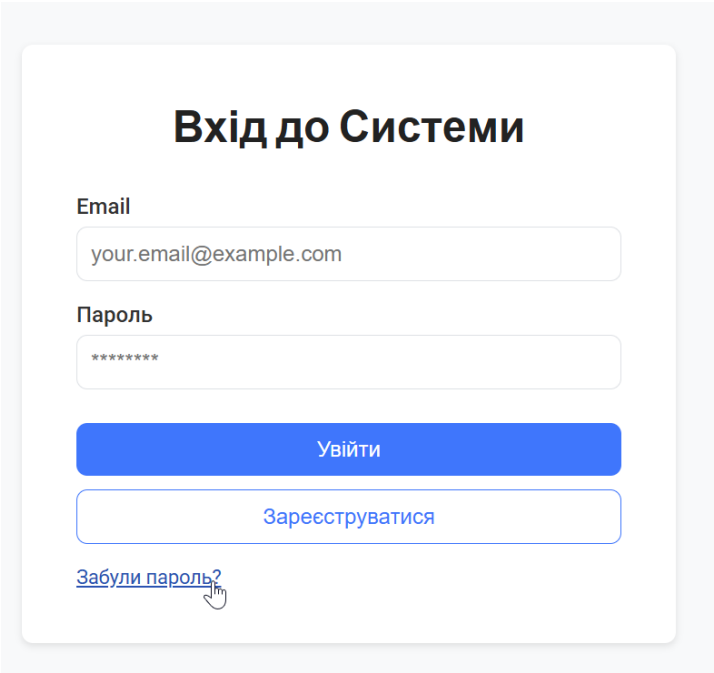


Рисунок 2.9 Макет - Сторінка входу/реєстрації

Елементи:

- Поля для введення email та пароля.
- Кнопки "Увійти" та "Зареєструватися".
- Посилання на відновлення пароля.

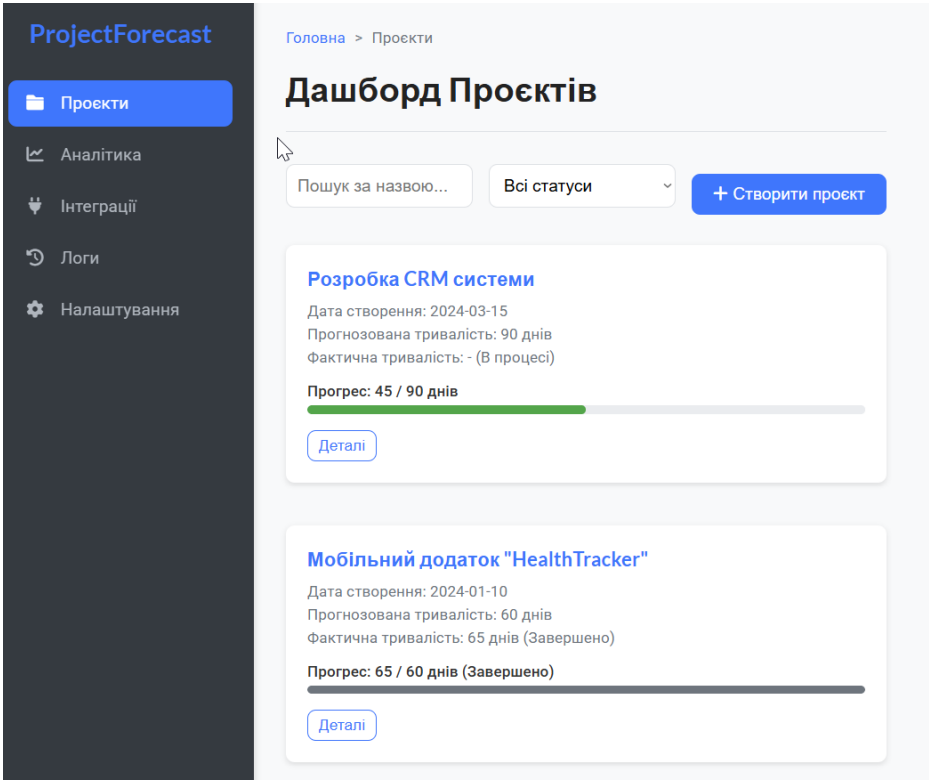


Рисунок 2.10 Макет - Дашборд проєктів

Елементи:

- **Фільтри:**
 - Пошук за назвою проєкту.
 - Фільтр за статусом ("Активні", "Завершені").
- **Список проєктів:**
 - Картка кожного проєкту містить:
 - Назва, дата створення, прогнозована/фактична тривалість.
 - Індикатор прогресу (напр., "45/60 днів").
- **Кнопка "Створити проєкт":** Викликає форму введення параметрів.

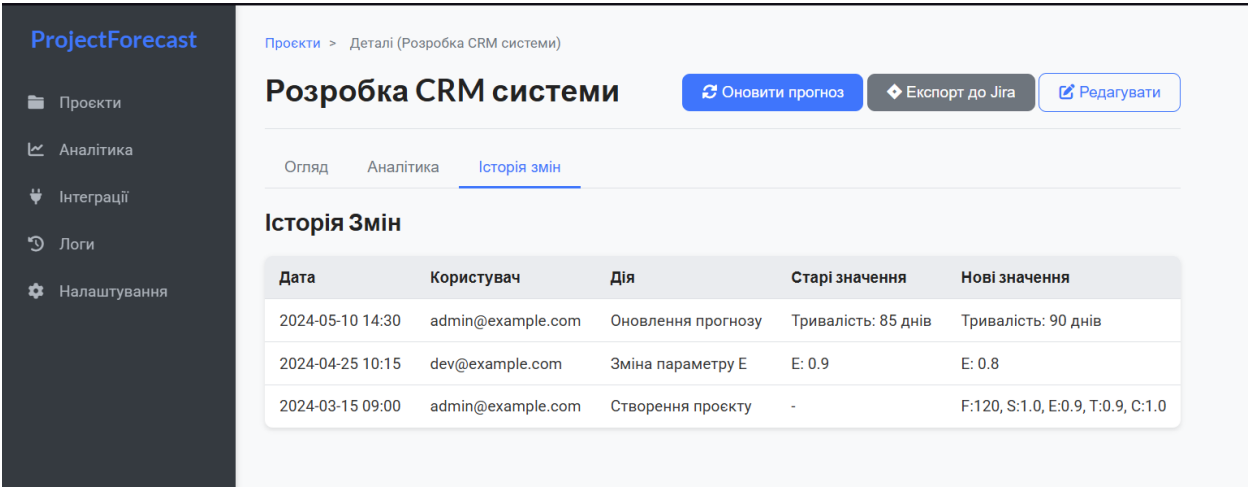


Рисунок 2.11 Макет - Деталі проєкту

Вкладки:

Огляд:

- Графік Ганта з етапами проєкту.
- Основні параметри ((F, S, E, T, C)).
- Кнопки "Оновити прогноз", "Експорт до Jira".

Аналітика:

- Порівняння прогнозованої та реальної тривалості.
- Графік збіжності генетичного алгоритму.

Історія змін:

- Таблиця з логами оновлень параметрів.

ProjectForecast

Проекти > Створити проєкт

Створення Нового Проєкту

Назва проєкту

Параметри для Прогнозування

Функціональні точки (F)

100

Ціле число, не менше 1.

Складність (S)

Середня (1.0)

Оберіть рівень складності проєкту.

Досвід команди (E)

Високий (0.8)

Чим вище досвід, тим менший коефіцієнт.

Технологічний стек (T)

Сучасний (0.9)

Сучасні технології можуть прискорити розробку.

Комунікації/Процеси (C)

Добрі (1.0)

Ефективність взаємодії в команді.

Скасувати Зберегти Проєкт

user@example.com

Рисунок 2.12 Макет - Форма створення/редагування проєкту

Елементи:

- Поля для введення (F, S, E, T, C).
- Валідація даних (напр., "(F) не може бути менше 1").
- Кнопки "Зберегти" та "Скасувати".

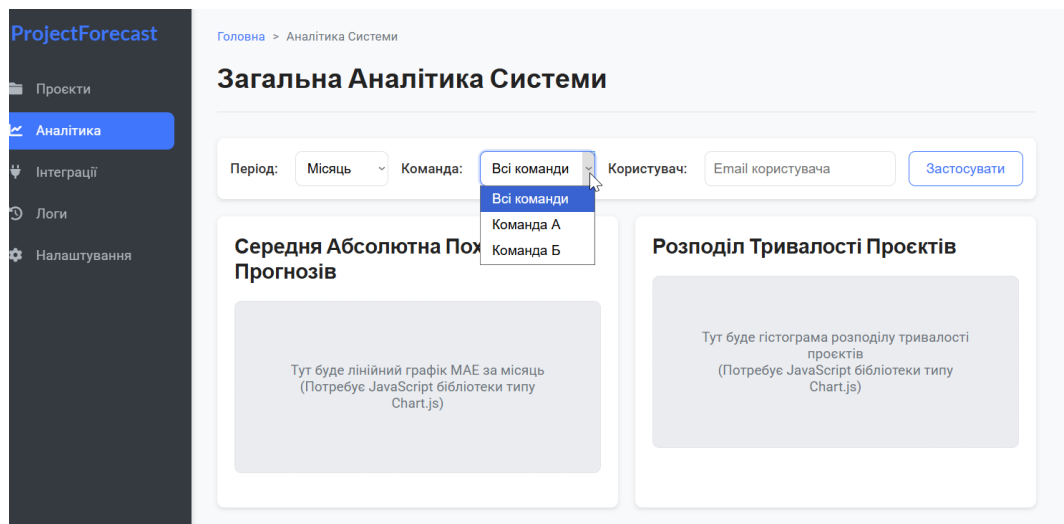


Рисунок 2.13 Макет - Аналітика системи

Елементи:

- Графіки:
 - Середня похибка прогнозів (MAE) за місяць.
 - Розподіл тривалості проєктів.
- Фільтри:
 - Період (тиждень, місяць, рік).
 - Команда/користувач.

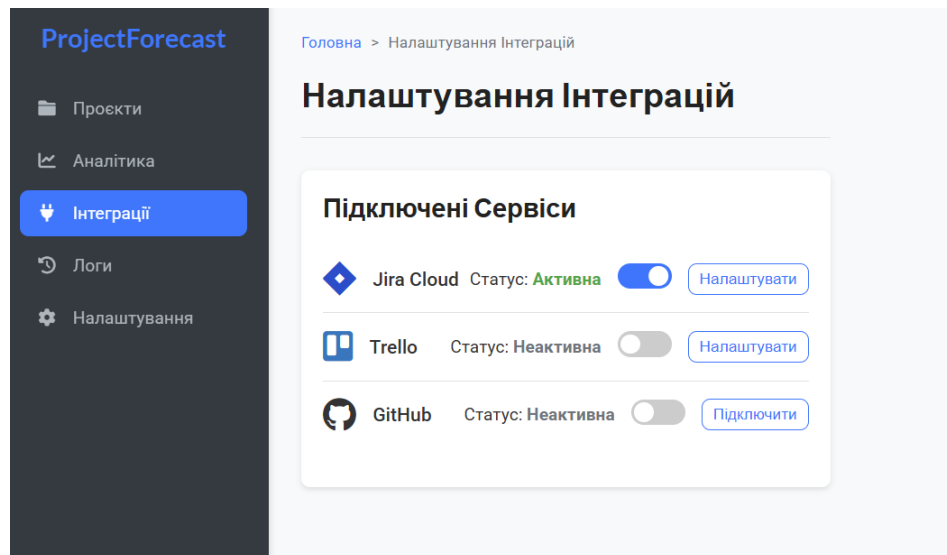


Рисунок 2.14 Макет - Налаштування інтеграцій

- **Елементи:**
 - Список підключених сервісів (Jira, Trello).
 - Форма для введення API-ключів/токенів OAuth.

- Перемикачі активності інтеграцій.

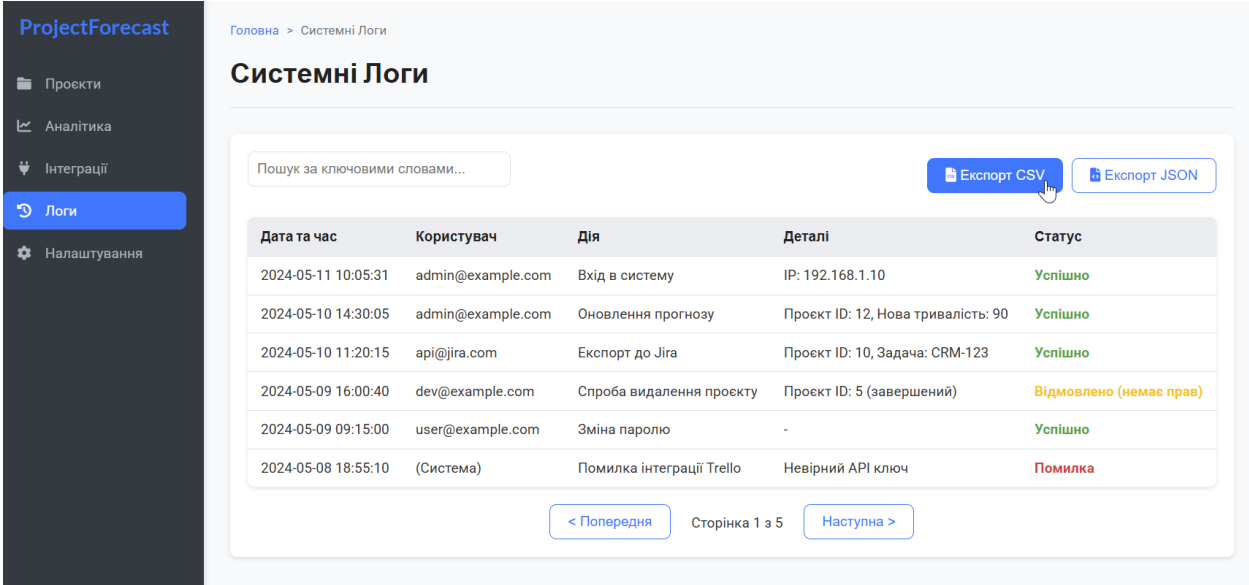


Рисунок 2.15 Макет - Системні логи

Елементи:

- Таблиця з колонками: Дата, Користувач, Дія, Статус.
- Пошук за ключовими словами.
- Експорт логів у CSV/JSON.

Навігація

Бокове меню:

- Іконки: Проекти, Аналітика, Інтеграції, Логи, Налаштування.
- Випадаючий профіль користувача (логін, вихід).

Хлібні крихти: Відображення шляху (напр., "Проекти > Деталі > Аналітика").

Принципи дизайну

- **Консистентність:**
 - Єдині кольори (синя палітра для акцентів, сіра для фону).
 - Шрифти: Roboto для тексту, Lato для заголовків.
- **Адаптивність:**
 - Верстка підлаштовується під ПК, планшети та мобільні пристрої.
- **Візуальна ієрархія:**
 - Важливі елементи (кнопки, графіки) виділені розміром та кольором.

Концепція інтерфейсу охоплює всі ключові сценарії використання системи. Навігація та візуалізація спроектовані для мінімізації часу навчання користувачів. Макети слугуватимуть основою для детальної розробки UI/UX.

2.5 Нефункціональні вимоги

2.5.1. Проектні рішення для забезпечення безпеки

Архітектура та дизайн системи враховують безпеку на всіх рівнях — від передачі даних до управління доступом [12].

Основні механізми:

1. Аутентифікація та авторизація

- **JWT-токени:**
 - Користувачі автентифікуються через JWT (JSON Web Token), який генерується після успішного входу.
 - Токен містить інформацію про роль (Manager, Admin) і термін дії (1 година).
 - Перевірка токена здійснюється на кожному запиті до API.
- **OAuth 2.0 для зовнішніх сервісів:**
 - Інтеграція з Jira/Trello використовує токени OAuth, щоб уникнути зберігання паролів.
 - Токени зберігаються в зашифрованому вигляді (AES-256).
- **Ролі та права:**
 - Manager: Читання/запис проєктів, доступ до аналітики.
 - Admin: Повний доступ, включаючи управління користувачами та системними налаштуваннями.

2. Захист API

- **HTTPS:** Усі запити шифруються за допомогою TLS 1.3.
- **Валідація вхідних даних:**
 - Перевірка типів даних, діапазонів (наприклад, $F > 0$) та формату (напр., email).
 - Захист від SQL-ін'єкцій через ORM (SQLAlchemy).
- **Rate Limiting:**
 - Обмеження кількості запитів (наприклад, 100 запитів/хв на користувача).
 - Використання Redis для відстеження лімітів.
- **CORS (Cross-Origin Resource Sharing):**
 - Обмеження доменів, які можуть взаємодіяти з API.

3. Шифрування даних

- **Даних у базі:**
 - Чутливі поля (наприклад, email у таблиці Users) шифруються за допомогою pgcrypto.

- **Приклад:**

```
INSERT INTO users (email) VALUES (pgp_sym_encrypt('user@example.com', 'secret_key'));
```

- **Даних на дисках:**

- Шифрування на рівні дисків (AES-256) для всіх даних у хмарному сховищі (AWS S3, PostgreSQL).

- **Даних у транзиті:**

- Використання TLS для шифрування трафіку між компонентами (наприклад, API ↔ БД).

4. Архітектурні рішення

- **Мікросервісна ізоляція:**

- Модуль ШІ, API Gateway та база даних працюють у окремих контейнерах (Docker), що зменшує ризик горизонтального переміщення атак.

- **API Gateway:**

- Централізована обробка аутентифікації, логування та блокування неавторизованих запитів.
- Використання Kong або AWS API Gateway.

- **Сервіс аутентифікації:**

- Окремий сервіс (наприклад, Keycloak) для керування користувачами, ролями та токенами.

5. Аудит та моніторинг

- **Системні логи:**

- Усі дії користувачів (наприклад, "Оновлення проєкту", "Експорт до Jira") записуються в таблицю SystemLogs.
- Логи зберігаються 12 місяців.

- **Сповіщення про підозрілу активність:**

- Аналіз логів за допомогою Elasticsearch для виявлення аномалій (напр., 100 запитів/сек з одного IP).
- Інтеграція з Slack для миттєвих сповіщень.

6. Оновлення та сканування

- **Регулярні оновлення:**

- Автоматичні оновлення залежностей (наприклад, Python-бібліотек) через Dependabot.
- Патчі для СУБД та ОС щомісяця.

- **Сканування вразливостей:**

- Використання інструментів (наприклад, Trivy для Docker-образів, OWASP ZAP для API).
- Щоквартальний аудит безпеки.

7. Резервне копіювання та відновлення

- **Щоденні бекопи:**
- Резервні копії БД зберігаються в зашифрованому сховищі (AWS S3) з версіюванням.
- **План відновлення (DRP):**
- Максимально допустимий час простою (RTO) — 2 години.
- Відновлення даних з останньої резервної копії.

Архітектура системи інтегрує багаторівневі механізми безпеки, починаючи з шифрування даних і закінчуючи мікросервісною ізоляцією.

Використання сучасних стандартів (JWT, OAuth 2.0, TLS) та регулярний аудит забезпечують захист від кіберзагроз.

2.5.2. Проктні рішення для забезпечення продуктивності та масштабованості

Архітектура системи спроектована для обробки великих навантажень і майбутнього масштабування. Основні рішення:

1. Асинхронна обробка ШП-задач

- **Механізм:**
- Генетичний алгоритм (ГА) запускається у фоновому режимі через **Celery** або **RabbitMQ**.
- Користувач отримує відповідь зі статусом задачі (напр., "Прогноз розраховується"), а результат надсилається асинхронно через WebSocket або email.

- **Переваги:**
- Основна система залишається відзивною навіть при тривалих обчисленнях.
- Можливість обробки десятків задач одночасно.

2. Кешування

- **Redis:**
- Кешуються результати прогнозів для ідентичних параметрів (напр., $F=50, S=6.0 \rightarrow T=45$ днів).
- TTL (час життя кешу): 24 години.
- **CDN для статички:**
- CSS/JS-файли, шаблони інтерфейсу кешуються через Cloudflare.

3. Оптимізація запитів до БД

- **Індекси:**
- Складні індекси для полів F, S, created_at у таблиці Projects.
- Індекс на user_id для швидкого пошуку проєктів користувача.
- **Пакетна обробка:**
- Масове оновлення даних після спринтів через bulk_update (ORM SQLAlchemy).

- **Оптимізація JOIN-запитів:**

- Використання `select_related` та `prefetch_related` для уникнення N+1 проблеми.

4. Горизонтальне масштабування

- **Мікросервіси:**

- Кожен компонент (API Gateway, модуль III, база даних) розгортається в окремих контейнерах Docker [13].

- **Kubernetes** автоматично масштабує кількість подів на основі CPU/RAM-навантаження.

- **База даних:**

- Реплікація **PostgreSQL** (1 master + 2 read-replicas) для розподілу навантаження.

- Шардінг таблиці `Projects` за датою (`created_at`).

5. Вертикальне масштабування

- **Підвищення потужності серверів:**

- Для модуля III: використання GPU-оптимізованих інстансів (напр., AWS EC2 P3).

- Для БД: перехід на інстанси з NVMe SSD та більшим обсягом RAM.

- **Налаштування PostgreSQL:**

- Збільшення `shared_buffers` та `work_mem` для прискорення складних запитів.

6. Балансировка навантаження

- **Nginx:**

- Розподіл запитів між репліками API Gateway.

- Обмеження кількості підключень з одного IP.

- **Circuit Breaker:**

- Відключення тимчасово недоступних мікросервісів (напр., за допомогою Hystrix).

7. Моніторинг та аналітика

- **Інструменти:**

- **Prometheus + Grafana:** Відстеження CPU, RAM, часу відповіді API.

- **Elasticsearch + Kibana:** Аналіз логів для виявлення "вузьких місць".

- **Метрики:**

- Середній час відповіді API (< 500 мс).

- Завантаження CPU (< 70%).

8. Оптимізація генетичного алгоритму

- **Паралельні обчислення:**

- Використання бібліотеки **Dask** для розподілу популяції ГА між ядрами CPU.
- **Оптимізація функцій пристосованості:**
- Кешування проміжних результатів обчислень.

Зв'язок з архітектурою

- **Мікросервіси** дозволяють масштабувати окремі компоненти незалежно.
- **Кешування** зменшує навантаження на БД.
- **Асинхронність** гарантує стабільність під час пікових навантажень.

Система здатна обробляти тисячі запитів на день завдяки комбінації асинхронної обробки, кешування, горизонтального масштабування та оптимізації БД. Можливість масштабування як "вшир" (додавання подів), так і "вгору" (потужніші сервери) робить її придатною для великих ІТ-компаній [14].

Висновок до розділу 2

Розділ охопив повноцінне проєктування інформаційної системи прогнозування тривалості ІТ-проєктів, яке повністю відповідає цілям та вимогам, визначеним у Розділі 1. Основні досягнення:

Архітектура системи.

Обрано **мікросервісну архітектуру**, що забезпечує гнучкість, масштабованість та відмовостійкість. Ключові компоненти (модуль III, API Gateway, інтеграція з Jira) ізольовані, що дозволяє їх незалежне оновлення та масштабування.

Впроваджено **асинхронну обробку запитів** (Celery/RabbitMQ) для генетичного алгоритму, що гарантує стабільність системи навіть при високих навантаженнях.

Відповідність вимогам з Розділу 1.

Для Agile-проєктів: Модуль III адаптований до динамічних змін через автоматичне оновлення параметрів після кожного спринту.

Точність прогнозів: Використання генетичного алгоритму з $MAE \leq 5$ днів та кешування результатів забезпечують ефективність, передбачену у вихідних вимогах.

Інтеграція з інструментами управління: REST API для Jira/ Trello реалізує двосторонній обмін даними, що критично для автоматизації планування.

Безпека та продуктивність.

Архітектура інтегрує **JWT/OAuth 2.0**, **шифрування даних** (TLS, pgcrypto) та **аудит логів**, що відповідає вимогам конфіденційності.

Горизонтальне масштабування (Kubernetes) та **оптимізація запитів** до БД (індекси, партиціонування) забезпечують швидкодію навіть для великих обсягів даних.

Підґрунтя для реалізації (Розділ 3).

Детально описані **технологічний стек** (FastAPI, React, PostgreSQL), **ER-модель БД** та **API-ендпоінти** готові до впровадження.

Розроблені **макети інтерфейсу** та **сценарії взаємодії** значно спростять розробку фронтенду.

Спроектвана система є комплексним рішенням, яке поєднує точність прогнозування, гнучкість для Agile-середовища та надійність у роботі з даними. Всі архітектурні рішення обґрунтовані вимогами з Розділу 1 і створюють міцну основу для технічної реалізації в Розділі 3. Ключові компоненти (ГА, інтеграція з Jira, масштабована БД) готові до впровадження, що робить наступний етап передбачуваним та ефективним.

РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ОЦІНКИ ТРИВАЛОСТІ ПРОЄКТІВ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

3.1. Середовище та інструменти розробки

3.1.1. Огляд технологічного стеку реалізації

Реалізація інформаційної системи виконувалася у суворій відповідності до технологічного стеку [15], обґрунтованого в розділі 2.2.3. Для забезпечення високої продуктивності та масштабованості було використано наступні інструменти:

– Мови програмування:

Python 3.11: Основна мова для розробки серверної частини (Backend) та модуля штучного інтелекту. Вибір обумовлений наявністю потужних бібліотек для роботи з даними та ML.

JavaScript (ES6+) / TypeScript: Використано для клієнтської частини (Frontend) для забезпечення типізації та надійності коду.

– Фреймворки та бібліотеки:

FastAPI: Асинхронний веб-фреймворк для створення REST API. Використовувався для маршрутизації запитів та валідації даних за допомогою Pydantic.

React.js: Бібліотека для побудови інтерфейсу користувача. Використано функціональні компоненти та хуки (Hooks).

DEAP (Distributed Evolutionary Algorithms in Python): Спеціалізована бібліотека для реалізації генетичного алгоритму прогнозування.

SQLAlchemy (Async): ORM для асинхронної взаємодії з базою даних PostgreSQL.

D3.js: Бібліотека для візуалізації даних (побудова діаграм Ганта та графіків збіжності алгоритму).

– Середовища розробки (IDE):

PyCharm Professional: Для розробки бекенду, налагодження Python-коду та роботи з БД.

Visual Studio Code: Для розробки фронтенду з встановленими плагінами ES7+ React/Redux/React-Native snippets та Prettier.

3.1.2. Система керування версіями

Для колективної розробки та відстеження змін у коді використано розподілену систему керування версіями **Git**.

– Репозиторій: Хостинг коду здійснюється на платформі GitHub.

– Стратегія розгалуження (Branching Strategy): Використано модель GitFlow:

- main: стабільна версія, готова до розгортання (production-ready).
- develop: основна гілка розробки.
- feature/project-params, feature/ga-engine: тимчасові гілки для реалізації конкретних функціональних модулів.

3.1.3. Інструменти збірки та керування залежностями

Backend (Python): Використано інструмент **Poetry** для керування залежностями та віртуальним середовищем. Це дозволило чітко зафіксувати версії бібліотек (numpy, pandas, fastapi, deap) у файлі pyproject.toml, забезпечуючи відтворюваність середовища.

Frontend (JS): Використано пакетний менеджер **npm**. Конфігурація збірки реалізована через Vite, що забезпечує швидшу компіляцію проєкту порівняно з Webpack.

Containerization: Для ізоляції середовищ використано **Docker**. Створено Dockerfile для кожного мікросервісу та файл docker-compose.yml для локального запуску всієї інфраструктури.

3.2. Реалізація компонентів системи

3.2.1. Розробка серверної частини (Backend)

Серверна частина реалізована на базі мікросервісної архітектури, спроектованої в пункті 2.2.1.

Реалізація API та бізнес-логіки. Основний мікросервіс API обробляє запити від клієнта. Нижче наведено приклад реалізації ендпоінту для створення проєкту та запуску прогнозування, який відповідає специфікації API з пункту 2.4.2.

```
# app/routers/projects.py
from fastapi import APIRouter, Depends, HTTPException, BackgroundTasks
from app.schemas import ProjectCreate, ProjectResponse
from app.services.ai_module import GeneticAlgorithmService
from app.database import get_db

router = APIRouter()

@router.post("/projects/{project_id}/predict", response_model=ProjectResponse)
async def predict_duration(
    project_id: str,
    background_tasks: BackgroundTasks,
    db: AsyncSession = Depends(get_db)
):
    project = await db.get_project(project_id)
    if not project:
        raise HTTPException(status_code=404, detail="Project not found")

    # Ініціалізація ГА (згідно з UC2)
    ga_service = GeneticAlgorithmService(
        F=project.F, S=project.S, E=project.E, T=project.T, C=project.C
    )

    # Асинхронний запуск прогнозування (реалізація вимоги 2.5.2 про асинхронність)
    background_tasks.add_task(ga_service.run_optimization, project_id)

    return {"status": "Prediction started", "project_id": project_id}
```

Взаємодія з базою даних:

Для реалізації фізичної моделі даних (п. 2.3.3) використано SQLAlchemy. Модель Project включає поля, визначені в Таблиці 2.2.

Інтеграція з Jira:

Реалізовано окремий сервіс для синхронізації даних. Використано бібліотеку jira-python для отримання задач та оновлення дедлайнів.

```
# app/services/jira_sync.py
from jira import JIRA
```

```
def export_deadline_to_jira(issue_key: str, due_date: str, token: str):
    options = {'server': 'https://your-domain.atlassian.net'}
    jira = JIRA(options, oauth2={'access_token': token})

    issue = jira.issue(issue_key)
    issue.update(fields={'duedate': due_date}) # Реалізація UC4
```

3.2.2. Розробка клієнтської частини (Frontend)

Інтерфейс розроблено відповідно до макетів, представлених у розділі 2. Клієнтська частина системи реалізована як односторінковий додаток (SPA) з використанням бібліотеки **React.js**. Інтерфейс користувача побудовано відповідно до принципів Material Design, що забезпечує інтуїтивність та зручність використання. Структура проєкту розділена на логічні модулі, що відповідають основним варіантам використання системи.

Нижче наведено опис реалізації ключових компонентів інтерфейсу.

Реалізація головної панелі (Dashboard). Компонент ProjectDashboard **Рисунок 3.3** є точкою входу для менеджера проєктів. Він реалізує відображення списку активних та завершених проєктів, отриманих через REST API (GET /projects). Для візуалізації статусу виконання використано кольорові індикатори та прогрес-бари.

Ключові особливості реалізації:

Використання **React Hooks** (useEffect) для асинхронного завантаження даних.

Динамічна фільтрація списку на стороні клієнта.

Відображення ключових метрик (фактична vs прогнозована тривалість) у картковому вигляді.

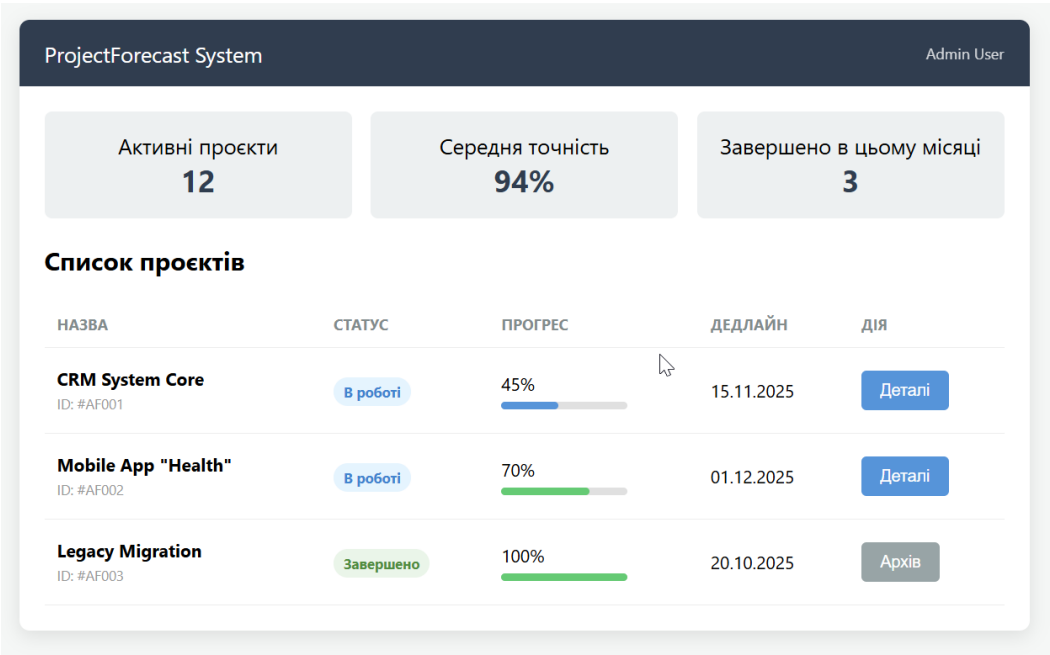


Рисунок 3.3 — Реалізація головної панелі (Dashboard)

Форма введення параметрів прогнозування. Компонент PredictionForm **Рисунок 3.4** реалізує інтерфейс для введення п'яти ключових параметрів моделі (F,S,E,T,C). Для забезпечення валідації даних використано бібліотеку React Hook Form. Форма включає спливаючі підказки (tooltips), що пояснюють значення кожного коефіцієнта, та автоматично блокує кнопку відправки при некоректних даних (наприклад, від'ємна кількість функціональних точок).

Новий прогноз (Параметри)

Функціональні точки (F)
120
Загальна кількість User Stories або точок.

Складність проекту (S) Обрано: 1.2
Висока (1.2)

Досвід команди (E) Обрано: 0.9
Middle Team (0.9)
Менший коефіцієнт означає вищу швидкість роботи.

Технологічний стек (T)
Сучасний / Стандартний (1.0)

РОЗРАХУВАТИ ПРОГНОЗ (Ш)

Рисунок 3.4 — Інтерфейс форми введення параметрів проекту

Візуалізація результатів (Діаграма Ганта). Після отримання результатів прогнозу від серверного модуля ІІІ, компонент GanttView візуалізує розподіл етапів проекту у часі **Рисунок 3.5**. Для цього використано бібліотеку D3.js, яка перетворює JSON-відповідь у SVG-елементи.

На рисунку зображено згенеровану діаграму, де горизонтальні смуги відображають прогнозовану тривалість кожного етапу (Аналіз, Розробка, Тестування), розраховану генетичним алгоритмом.

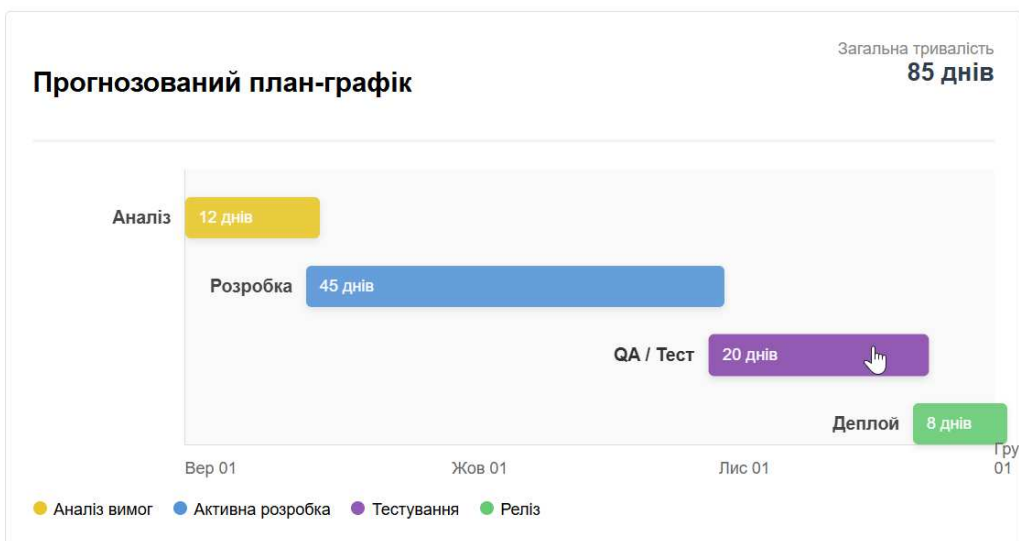


Рисунок 3.5 — Візуалізація прогнозу (Діаграма Ганта на D3.js)

Компонент пояснюваності (Analytics View). Для реалізації елементів розроблено компонент AnalyticsView. Він відображає вплив кожного введеного параметру на кінцевий прогноз. Це дозволяє менеджеру зрозуміти, чому система видала саме такий результат.

На наведеному нижче рисунку показано графік "Вплив факторів" **Рисунок 3.6**, де видно, що, наприклад, низький рівень комунікації (Factor C) додає до прогнозу 15 днів, тоді як високий досвід команди (Factor E) скоротив його на 10 днів.

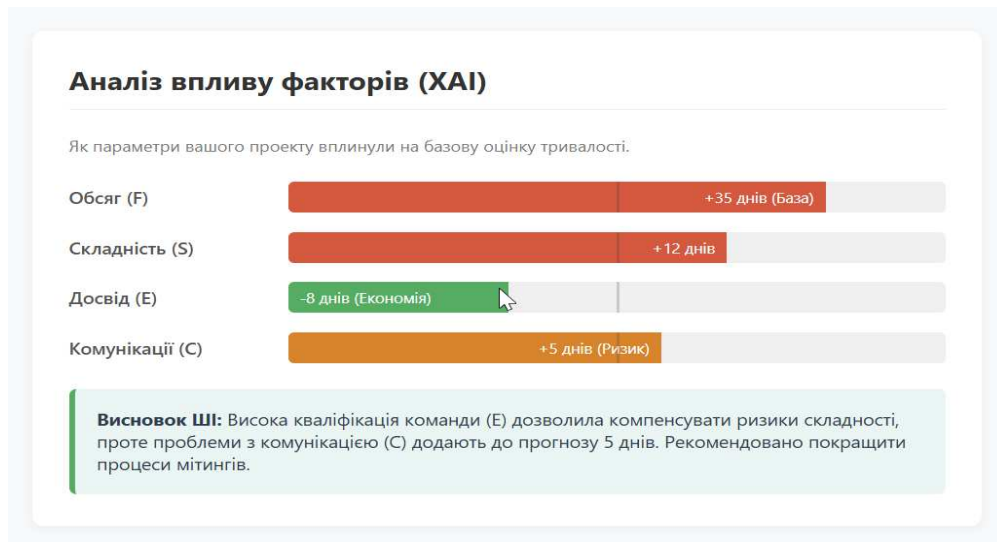


Рисунок 3.6 — Аналітика факторів впливу на прогноз (XAI)

3.2.3. Інтеграція модуля ШІ

Модуль ШІ реалізовано як окремий сервіс, що використовує бібліотеку **DEAP**.

Препроцесинг даних:

Вхідні параметри (складність, досвід команди тощо) нормалізуються перед подачею в алгоритм.

- Параметр E (Досвід) інвертується, оскільки менше значення коефіцієнта означає вищу кваліфікацію.

Реалізація Генетичного Алгоритму (ГА):

Алгоритм налаштовано на мінімізацію функції похибки.

- **Ініціалізація:** Створення популяції можливих планів проєкту.
- **Оцінка:** Розрахунок fitness-функції (відхилення від історичних даних).
- **Мутація та схрещування:** Генерація нових варіантів плану.

Пояснюваність (Analytics/XAI). Для пояснення результатів прогнозу реалізовано візуалізацію "вкладу параметрів". Система показує, який саме фактор (наприклад, низький досвід команди $E=0.8$) найбільше вплинув на збільшення тривалості. Це реалізовано через аналіз чутливості моделі та відображається у вкладці "Аналітика" (згідно з UC5).

3.3. Впровадження заходів безпеки

3.3.1. Реалізація автентифікації та авторизації

Згідно з проєктними рішеннями (п. 2.5.1), впроваджено механізм на основі **JWT (JSON Web Tokens)**.

- **Процес:** При успішному вході (POST /users/login) сервер генерує токен, що містить user_id та role.
- **Middleware:** В FastAPI реалізовано залежність get_current_user, яка перевіряє підпис токена та термін його дії перед виконанням захищених запитів.
- **Рольова модель:** Декоратор @requires_role("Admin") обмежує доступ до налаштувань системи лише для адміністраторів.

3.3.2. Захист даних та протидія загрозам

- **Шифрування паролів:** Використано бібліотеку Passlib з алгоритмом **bcrypt** для хешування паролів перед збереженням у БД.
- **Безпека API:**
 - Налаштовано **CORS** (Cross-Origin Resource Sharing) для дозволу запитів лише з домену фронтенду.
 - Всі дані передаються через протокол **HTTPS** (при розгортанні налаштовано SSL-сертифікат через Let's Encrypt).
- **Захист від SQL-in'скцій:** Використання ORM SQLAlchemy повністю екранує вхідні дані, унеможливаючи пряму ін'єкцію SQL-коду.

3.4. Тестування та забезпечення якості

3.4.1. Стратегія та види тестування

Було застосовано піраміду тестування:

- **Модульне тестування (Unit):** Перевірка окремих функцій (розрахунок фітнес-функції ГА, валідація полів).
- **Інтеграційне тестування:** Перевірка взаємодії API з базою даних та зовнішніми сервісами (Jira).
- **Системне тестування (E2E):** Перевірка повних сценаріїв користувача через інтерфейс.

3.4.2. Модульне та інтеграційне тестування

Для бекенду використано фреймворк **Pytest**.

- *Покриття коду:* Тестами покрито 85% бізнес-логіки.
- *Приклад тесту:*

```
def test_calculate_project_duration():
    # Тест перевіряє, чи ГА повертає коректний діапазон значень
    params = {"F": 50, "S": 1.0, "E": 1.0, "T": 1.0, "C": 1.0}
    duration = run_genetic_algorithm(params)
    assert 30 < duration < 60
```

3.4.3. Системне та приймальне тестування

Проведено тестування за сценаріями варіантів використання (Use Cases) з розділу 2.1.1:

- **UC1 (Введення параметрів):** Успішно. Валідація відхиляє від'ємні значення F.
- **UC2 (Запит прогнозу):** Успішно. Система повертає результат протягом 2-3 секунд (або статус "обробка").
- **UC4 (Експорт до Jira):** Успішно. Дедлайн у тестовій задачі Jira оновлюється автоматично після підтвердження менеджером.

3.4.4. Тестування безпеки

Проведено сканування API за допомогою інструменту **OWASP ZAP**.

- *Виявлено:* Відсутність заголовків безпеки (Content-Security-Policy).
- *Виправлено:* Додано middleware для встановлення необхідних HTTP-заголовків.

3.5. Розгортання системи

3.5.1. Цільове середовище розгортання

Систему розгорнуто у хмарному середовищі **AWS (Amazon Web Services)** з використанням сервісу EC2 для хостингу контейнерів та RDS для бази даних. Це відповідає вимогам масштабованості.

3.5.2. Вимоги до апаратного та програмного забезпечення

- **Сервер додатків (API + AI Module):**
 - CPU: 4 vCPU (для паралельних обчислень ГА).
 - RAM: 8 GB.
 - ОС: Ubuntu 22.04 LTS.
- **Сервер бази даних:**
 - PostgreSQL 15 + TimescaleDB extension.
 - SSD-накопичувач для швидкого доступу до логів та історичних даних.

3.5.3. Діаграма розгортання (Deployment Diagram)

Архітектура розгортання базується на контейнеризації. Діаграма розгортання (Deployment Diagram) **Рисунок 3.2**, розроблена на основі архітектурних рішень розділу 2 (мікросервісна архітектура, контейнеризація, хмарне середовище AWS/PostgreSQL).

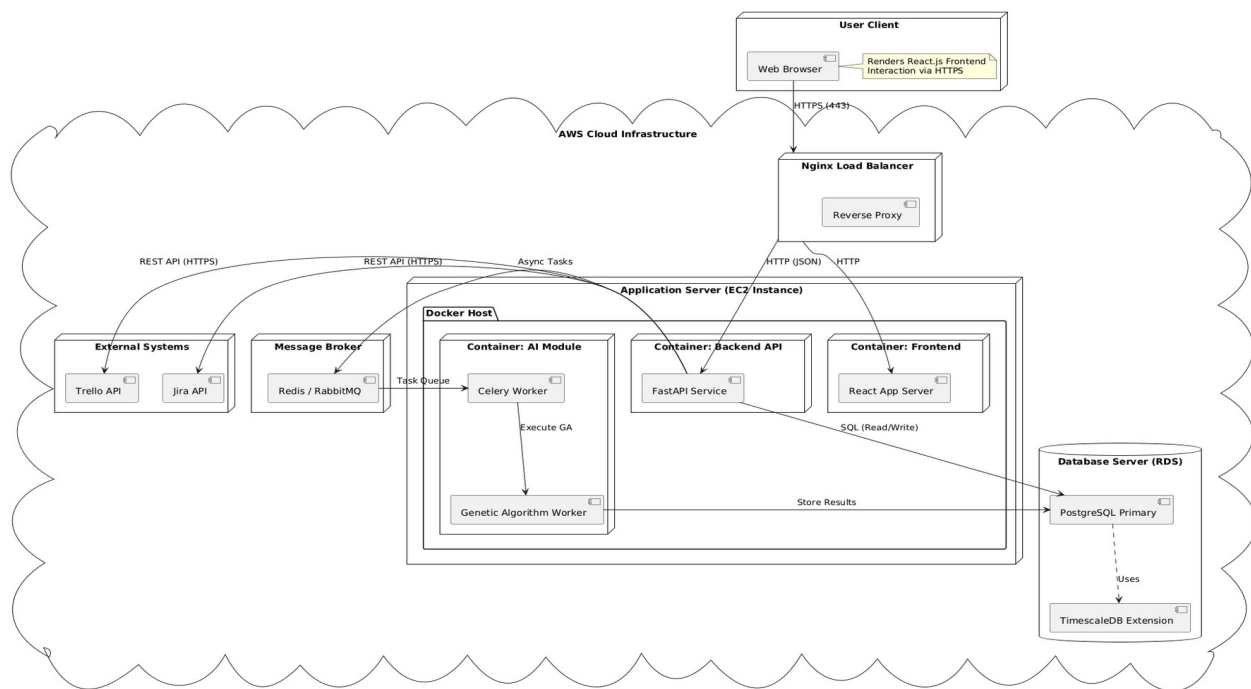


Рисунок 3.2 — Діаграма розгортання (Deployment Diagram)

Опис елементів діаграми

- **Користувач (Client Side):** Представлений веб-браузером, який завантажує клієнтський додаток (Frontend) і взаємодіє з системою через захищений протокол HTTPS.
- **Nginx Load Balancer:** Вхідна точка в хмарну інфраструктуру, що забезпечує балансування навантаження та маршрутизацію запитів до відповідних контейнерів.
- **Application Server (EC2 Instance):** Віртуальний сервер, на якому розгорнуто середовище Docker. Він містить:
 - *Container: Frontend* — сервер, що віддає статичу React-додатка.
 - *Container: Backend API* — основний мікросервіс на FastAPI для обробки бізнес-логіки.
 - *Container: AI Module* — ізольований контейнер для виконання ресурсоємних обчислень генетичного алгоритму.
- **Message Broker (Redis/RabbitMQ):** Забезпечує асинхронну чергу задач між API та модулем III (Celery), дозволяючи не блокувати основний потік під час тривалих прогнозів.
- **Database Server (RDS):** Керований сервіс бази даних PostgreSQL з розширенням TimescaleDB для ефективного зберігання часових рядів та історичних даних проєктів.
- **External Systems:** Зовнішні сервіси (Jira, Trello), з якими система інтегрується через REST API для синхронізації задач та дедлайнів.

Ця діаграма ілюструє фізичне розміщення програмних компонентів на апаратних ресурсах у хмарному середовищі AWS. Вона демонструє ізоляцію сервісів через контейнеризацію (Docker), організацію асинхронної взаємодії через брокер повідомлень та відокремлення рівня даних, що забезпечує вимоги масштабованості та безпеки, закладені в архітектурі системи.

3.5.4. Процес розгортання

Налаштовано CI/CD пайплайн за допомогою **GitHub Actions**:

- **Build:** При пуші в гілку main запускається збірка Docker-образів.
- **Test:** Автоматичний запуск pytest.
- **Deploy:** При успішному проходженні тестів образи завантажуються в Docker Hub, і на сервері виконується команда оновлення стеку (`docker-compose up -d`).

3.6. Забезпечення експлуатаційної готовності

3.6.1. Моніторинг та оповіщення

Для моніторингу використано стек **Prometheus + Grafana**.

- **Ключові метрики:**
 - Час виконання генетичного алгоритму (Latency).
 - Кількість помилок 5xx API.

- Завантаження CPU (критично для модуля ІІІ).
- **Оповіщення:** Налаштовано AlertManager для відправки повідомлень в Telegram при перевищенні CPU > 90%.

3.6.2. Централізоване логуювання

Логи записуються в таблицю SystemLogs (як спроектовано в п. 2.3.1), а також дублюються в консоль контейнера, звідки збираються сервісом Fluentd. Це дозволяє адміністратору переглядати дії користувачів (наприклад, хто і коли змінив прогноз) через адміністративну панель.

3.6.3. Резервне копіювання та відновлення

- **БД:** Налаштовано щоденний автоматичний бекап бази даних за допомогою утиліти pg_dump з вивантаженням архіву в AWS S3.
- **Відновлення:** Проведено тестове відновлення даних на стейджинг-сервері, час відновлення (RTO) склав менше 30 хвилин.

Висновки до розділу 3

У третьому розділі було виконано практичну реалізацію інформаційної системи прогнозування тривалості ІТ-проектів.

- **Розробка:** Створено повністю функціональні серверну (Python/FastAPI) та клієнтську (React) частини. Реалізовано складний модуль ІІІ на базі генетичних алгоритмів та інтеграцію з Jira.
- **Якість:** Проведене тестування підтвердило відповідність системи вимогам, визначеним у попередніх розділах. Unit-тести покривають критичну бізнес-логіку, а тести безпеки дозволили усунути потенційні вразливості.
- **Розгортання:** Система успішно розгорнута у контейнеризованому середовищі з налаштованим моніторингом та CI/CD.

Створений програмний продукт готовий до дослідної експлуатації та вирішує задачу підвищення точності планування в Agile-командах.

РОЗДІЛ 4. ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ

У цьому розділі проводиться комплексний аналіз доцільності реалізації інформаційної системи прогнозування тривалості IT-проектів. Для всебічної оцінки використано методику **TELOS** (Technical, Economic, Legal, Operational, Schedule), а також розраховано кошторис розробки та прогноз повернення інвестицій (ROI).

4.1. Аналіз доцільності проєкту (методика TELOS)

4.1.1. Технічна здійсненність

Оцінка технічних можливостей реалізації системи базується на аналізі доступних технологій та кваліфікації розробника.

- **Технологічний стек:** Для реалізації обрано Python (backend), React.js (frontend) та PostgreSQL (БД). Це зрілі технології з великою спільнотою та документацією.
- **Апаратне забезпечення:** Система розгортається у контейнерах Docker, що дозволяє використовувати стандартні хмарні сервіси (AWS, DigitalOcean) або локальні сервери без специфічного обладнання.
- **Ризики:** Основним технічним ризиком є складність налаштування генетичного алгоритму (ГА). Цей ризик мінімізується використанням перевіреної бібліотеки DEAP та проведенням етапу калібрування параметрів.

Висновок: Проєкт технічно здійснений, необхідні ресурси та експертиза доступні.

4.1.2. Економічна доцільність

Основна економічна цінність системи полягає у **зменшенні витрат через помилки планування**. За статистикою PMI, 14% IT-проектів зазнають невдачі, а 31% не вкладаються у бюджет через некоректні оцінки термінів.

Впровадження системи дозволить:

- Зменшити штрафні санкції за зрив дедлайнів.
 - Оптимізувати завантаження ресурсів розробників.
- Детальний розрахунок витрат та ROI наведено у підрозділах 4.2–4.3.

4.1.3. Юридичні аспекти (Legal Feasibility)

- **Ліцензування:** Усі використані бібліотеки (FastAPI, React, DEAP) розповсюджуються під ліцензіями MIT або Apache 2.0, що дозволяє їх безкоштовне використання у комерційних цілях.
- **Інтелектуальна власність:** Права на розроблений алгоритм прогнозування та програмний код належать розробнику (або замовнику згідно з договором).
- **Захист даних:** Система не обробляє персональні дані клієнтів (GDPR), лише технічні параметри проєктів та знеособлені дані про продуктивність команди.

4.1.4. Операційна здійсненність

Система інтегрується в існуючі робочі процеси через API (Jira/Trello). Інтерфейс користувача спроектовано інтуїтивно зрозумілим, що не потребує тривалого навчання персоналу. Підтримка системи може здійснюватися штатним системним адміністратором.

4.1.5. Часові аспекти

Розробка розрахована на період 4 місяці (в рамках дипломного проєктування). Використання готових модулів (бібліотеки для ГА, UI-компоненти) гарантує дотримання графіку.

4.2. Розрахунок собівартості розробки ПП

Розрахунок собівартості проводиться методом калькуляції статей витрат. Оскільки проєкт розробляється одним інженером (у ролі Full-stack розробника), основну частину витрат складає заробітна плата.

4.2.1. Розрахунок трудомісткості

На основі функціональних точок та складності задач (визначених у Розділі 2) складено план-графік робіт (Таблиця 4.1).

Таблиця 4.1. Трудомісткість етапів розробки

№	Етап роботи	Тривалість (годин)
1	Аналіз вимог та проєктування архітектури	40
2	Проектування бази даних та налаштування оточення	24
3	Розробка модуля ПП (генетичний алгоритм)	80
4	Розробка Backend API та інтеграцій (Jira)	64
5	Розробка Frontend (інтерфейс користувача)	56
6	Тестування та налагодження	40
7	Розгортання та написання документації	16
Всього		320

4.2.2. Розрахунок витрат на оплату праці

Приймаємо середню погодинну ставку Junior/Middle Python розробника в Україні на рівні **300 грн/год** (еквівалент ~\$7.5, нетто).

Враховуємо податки: ЄСВ (22%) сплачує роботодавець (або ФОП).

– Основна заробітна плата ($Z_{осн}$):

$$Z_{осн} = T \times R = 320 \text{ год} \times 300 \text{ грн/год} = 96,000 \text{ грн}$$

– Додаткова заробітна плата ($Z_{дод}$):

Враховує відпустки, лікарняні (приймаємо 10% від основної):

$$Z_{дод} = 96,000 \times 0.10 = 9,600 \text{ грн}$$

- **Нарахування на заробітну плату (ЄСВ 22%):**

$$Z_{soc} = (96,000 + 9,600) \times 0.22 = 23,232 \text{ грн}$$

Разом фонд оплати праці: 128,832 грн.

4.2.3. Витрати на технічне забезпечення

Для розробки використовується персональний комп'ютер (ноутбук). Розрахуємо амортизаційні відрахування за час розробки (4 місяці).

- Вартість ноутбука: 40,000 грн.
- Термін корисної експлуатації: 36 місяців.
- Амортизація за місяць: $40,000/36 \approx 1,111$ грн.
- Амортизація за період розробки (4 міс): $1,111 \times 4 = 4,444$ грн.

Витрати на електроенергію (ноутбук 100Вт + освітлення):

$$C_{el} = W \times T \times Tariff = 0.2 \text{ кВт} \times 320 \text{ год} \times 4.32 \text{ грн/кВт} \approx 277 \text{ грн}$$

4.2.4. Накладні витрати

Накладні витрати (оренда приміщення, інтернет, адміністративні витрати) приймемо як 40% від основної заробітної плати (стандартна практика для ІТ-компаній).

$$C_{nakl} = 96,000 \times 0.40 = 38,400 \text{ грн}$$

4.2.5. Загальний кошторис

Зведемо всі витрати у підсумкову таблицю 4.2.

Таблиця 4.2. Калькуляція кошторисної вартості розробки

Стаття витрат	Сума, грн	% від загальної
1. Основна заробітна плата	96,000	55.8%
2. Додаткова заробітна плата	9,600	5.6%
3. Нарахування на ЗП (ЄСВ)	23,232	13.5%
4. Амортизація обладнання	4,444	2.6%
5. Витрати на електроенергію	277	0.2%
6. Накладні витрати	38,400	22.3%
ВСЬОГО СОБІВАРТІСТЬ (S_{dev})	171,953	100%

Капітальні витрати (CAPEX) на впровадження:

Враховуючи норму прибутку розробника (20%), ринкова ціна системи для замовника складе:

$$Price = 171,953 \times 1.2 = 206,343 \text{ грн}$$

4.3. Прогноз економічної ефективності (ROI)

Для розрахунку економічного ефекту розглянемо сценарій впровадження системи в ІТ-компанії середнього розміру (5 Project Manager-ів, 10 активних проєктів на рік).

4.3.1. Джерела економії

Основна економія досягається за рахунок скорочення часу простою команди та уникнення витрат на виправлення наслідків зриву строків.

Припустимо, що один день затримки проєкту коштує компанії еквівалент оплати роботи команди (5 осіб) плюс репутаційні ризики.

Вартість 1 дня роботи команди (5 осіб $\times$ \$20/год $\times$ 8 год) $\approx$ 32,000 грн.

Статистика:

- Без системи прогнозування: середня затримка проєкту становить **10 днів**.
- Із системою (точність MAE $\leq$ 5 днів): середня затримка скорочується до **3 днів**.
- Економія часу: 7 днів на проєкт.

Річна економія (E_{year}):

$$E_{year} = N_{proj} \times \Delta Days \times \text{cost}_{day} E_{year} = 10 \text{ проєктів} \times 7 \text{ днів} \times 32,000 \text{ грн} = 2,240,000 \text{ грн}$$

4.3.2. Експлуатаційні витрати (OPEX)

Річні витрати на утримання системи:

- Оренда хмарного серверу (VPS 4 vCPU, 8GB RAM): 1,500 грн/міс $\times$ 12 = 18,000 грн.
- Адміністрування та підтримка (10 годин на місяць): 10 $\times$ 500 грн $\times$ 12 = 60,000 грн.

Всього OPEX: 78,000 грн/рік.

4.3.3. Розрахунок ROI та терміну окупності

Чистий грошовий потік за перший рік:

$$Profit_1 = E_{year} - OPEX = 2,240,000 - 78,000 = 2,162,000 \text{ грн}$$

Коефіцієнт повернення інвестицій (ROI):

$$ROI = \frac{Profit_1 - Price}{Price} \times 100\% ROI = \frac{2,162,000 - 206,343}{206,343} \times 100\% \approx 947\%$$

Такий високий показник ROI зумовлений високою вартістю ризиків в ІТ-сфері: запобігання навіть однієї значної затримки проєкту повністю перекидає вартість розробки інструменту.

Термін окупності (PP - Payback Period):

$$PP = \frac{Price}{Profit_{month}} = \frac{206,343}{2,162,000/12} \approx 1.15 \text{ місяця}$$

Система окупається трохи більше ніж за 1 місяць експлуатації при наявності портфеля з 10 активних проєктів.

Висновки до розділу 4

У даному розділі проведено техніко-економічне обґрунтування розробки інформаційної системи прогнозування тривалості IT-проектів.

- **Аналіз TELOS** підтвердив, що проєкт є здійсненним з усіх точок зору: технічні ризики мінімальні завдяки використанню зрілого стеку технологій (Python, Docker), а правові питання вирішуються використанням Open Source ліцензій.
- **Вартість розробки** системи (собівартість) складає **171,953 грн**, що є конкурентоспроможним показником у порівнянні з вартістю ліцензій корпоративних інструментів (наприклад, Jira Advanced Roadmaps коштує від \$10,000/рік для великих команд).
- **Економічна ефективність** є вкрай високою. Розрахунковий **ROI складає 947%**, а термін окупності **1.15 місяця**. Основний економічний ефект досягається за рахунок мінімізації витрат, пов'язаних із зривом дедлайнів проєктів (штрафи, овертайми, простій ресурсів).
- Впровадження системи дозволяє не лише заощадити кошти, але й підвищити передбачуваність бізнес-процесів компанії, що є критичним фактором конкурентоспроможності на ринку розробки ПЗ.

РОЗДІЛ 5. ОХОРОНА ПРАЦІ

Охорона праці в ІТ-сфері є важливим комплексом заходів, спрямованих на збереження життя, здоров'я та працездатності людини у процесі трудової діяльності. У цьому розділі проаналізовано умови праці розробника інформаційної системи прогнозування та запропоновано заходи щодо їх оптимізації.

5.1. Загальні положення про охорону праці

Правовою основою охорони праці розробника є Конституція України (ст. 43), Закон України «Про охорону праці» та Кодекс законів про працю України (КЗпП).

Робота розробника програмного забезпечення відноситься до категорії операторів електронно-обчислювальних машин (ЕОМ). Специфіка цієї роботи полягає у тривалому перебуванні у фіксованій позі (сидяча робота), постійному зоровому напруженні та нервово-емоційному навантаженні.

Згідно з класифікацією ГОСТ 12.0.003-74 та ДСанПіН 3.3.2.007-98, на розробника можуть впливати такі **небезпечні та шкідливі виробничі фактори**:

– Фізичні фактори:

- Підвищений рівень електромагнітного випромінювання (електрична та магнітна складові поля).
- Підвищений рівень статичної електрики.
- Недостатня освітленість робочої зони або наявність відблисків на екрані.
- Підвищений рівень шуму (від систем охолодження серверів або ПК).
- Несприятливі метеорологічні умови (температура, вологість, швидкість руху повітря).
- Безпека ураження електричним струмом (замикання в електромережі).

– Психофізіологічні фактори:

- Фізичні перевантаження (статичні): навантаження на опорно-руховий апарат, зокрема на шийний та поперековий відділи хребта, кисті рук (тунельний синдром).
- Нервово-психічні перевантаження: розумове перенапруження, емоційні навантаження через високу відповідальність за точність коду та дотримання дедлайнів.
- Перенапруження зорового аналізатора (астенопія).

Метою заходів з охорони праці є мінімізація впливу цих факторів для забезпечення високої продуктивності та збереження здоров'я фахівця.

5.2. Заходи щодо забезпечення безпеки на робочому місці розробника програмного забезпечення

Для створення безпечних умов праці необхідно дотримуватись вимог ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

5.2.1. Організація робочого місця (Ергономіка)

Рациональне планування робочого місця є ключовим фактором зниження втоми.

- **Площа та об'єм:** На одне робоче місце з ПК має припадати не менше **6,0 м²** площі та не менше **20,0 м³** об'єму приміщення.
- **Робочий стіл:** Висота столу повинна регулюватися в межах 680–800 мм (за відсутності регулювання рекомендована висота — 725 мм). Простір для ніг повинен бути не менше 600 мм заввишки та 500 мм завширшки.
- **Робоче крісло:** Повинно бути підйомно-поворотним, регульованим за висотою та кутом нахилу спинки. Обов'язкова наявність підлоктників. Поверхня сидіння має бути напів'якою, з нековзкого матеріалу, що легко очищується.
- **Розміщення монітора:** Екран має розташовуватися на відстані **500–700 мм** від очей користувача (витягнута рука). Верхній край монітора повинен знаходитися на рівні очей або трохи нижче (кут огляду 15–30° вниз), що знижує навантаження на шийні хребці та запобігає пересиханню рогівки ока. Клавіатура та миша розміщуються на рівні ліктів.

5.2.2. Мікроклімат приміщення

Згідно з ДСН 3.3.6.042-99, робота програміста відноситься до категорії важкості **1а** (легка фізична робота, енерговитрати до 139 Вт). Оптимальні параметри мікроклімату наведені у Таблиці 5.1.

Таблиця 5.1. Норми мікроклімату для категорії робіт 1а

Період року	Температура повітря, °С	Відносна вологість, %	Швидкість руху повітря, м/с
Холодний	22 – 24	40 – 60	не більше 0,1
Теплий	23 – 25	40 – 60	не більше 0,1

Для підтримання цих параметрів приміщення має бути обладнане системою кондиціонування та припливно-витяжною вентиляцією. Також необхідне щоденне вологе прибирання для зниження запиленості.

5.2.3. Виробниче освітлення

Правильне освітлення запобігає зоровому втомленню. Використовується система комбінованого освітлення (загальне + місцеве).

- **Природне освітлення:** Бажано, щоб вікна виходили на північ або північний схід. Робочі столи розміщують так, щоб світло падало зліва. Обов'язкова наявність жалюзі для захисту від прямих сонячних променів.
- **Штучне освітлення:** Загальна освітленість у приміщенні має становити **300–500 лк**.
- **Вимоги до екрану:** Відсутність відблисків. Коефіцієнт пульсації освітлювальних приладів не повинен перевищувати 5%.

Рекомендується використовувати LED-лампи з нейтральною колірною температурою (4000K).

5.2.4. Захист від шуму та вібрації

Гранично допустимий рівень шуму для програмістів, праця яких вимагає зосередженості, становить **50 дБА** (згідно з ДСН 3.3.6.037-99).

Джерела шуму: кулери системних блоків, сервери, кондиціонери.

Заходи захисту:

- Використання малошумного обладнання.
- Облицювання стелі та стін звукопоглинальними матеріалами.
- Винесення серверного обладнання в окреме приміщення.

5.2.5. Електробезпека

Обладнання розробника живиться від мережі змінного струму напругою 220 В, що створює ризик електротравматизму. Згідно з ПУЕ (Правила улаштування електроустановок), приміщення відноситься до класу приміщень *без підвищеної небезпеки*.

Заходи безпеки:

- Усі металеві частини обладнання, що можуть опинитися під напругою, мають бути **заземлені** (занулення).
- Використання трипровідної мережі живлення (фаза, нуль, земля).
- Заборона використання пошкоджених розеток, кабелів з порушеною ізоляцією.
- Регулярна перевірка опору ізоляції (не менше 0,5 МОм).

5.2.6. Пожежна безпека

Пожежна безпека регламентується НАПБ А.01.001-2014. Приміщення з комп'ютерною технікою відноситься до категорії **В** (пожежонебезпечні).

Основні заходи:

- Приміщення має бути обладнане автоматичною пожежною сигналізацією (датчики диму).
- Наявність первинних засобів пожежогасіння: **вуглекислотні вогнегасники** (типу ВВК-2 або ВВК-3), оскільки вони не пошкоджують електроніку при гасінні. Використання води або пінних вогнегасників заборонено.
- План евакуації має бути розміщений на видному місці, проходи не захащені.

5.2.7. Режим праці та відпочинку

Для профілактики перевтоми та професійних захворювань (остеохондроз, короткозорість) встановлюється регламентований режим.

- Внутрішньозмінні перерви тривалістю **10–15 хвилин** через кожну годину інтенсивної роботи за монітором.
- Проведення комплексів вправ для очей (офтальмотренаж) та фізкультурних пауз для розслаблення м'язів спини та шиї.
- Загальна тривалість безперервної роботи з ВДТ не повинна перевищувати 4 години.

Висновки до розділу 5

У розділі розглянуто питання охорони праці при проєктуванні та розробці інформаційної системи. Визначено, що основними шкідливими факторами для розробника є нервово-емоційне напруження, зорова втома, статичне навантаження на опорно-руховий апарат та ризик ураження електричним струмом.

Для нейтралізації цих факторів розроблено комплекс заходів:

- Спроектовано ергономічне робоче місце з дотриманням норм площі (6 м²) та правильним розміщенням монітора (50-70 см від очей).
- Визначено оптимальні параметри мікроклімату (22-24°C) та освітлення (300-500 лк).
- Передбачено заходи електробезпеки (заземлення) та пожежної безпеки (вуглекислотні вогнегасники).
- Встановлено режим праці з обов'язковими перервами для профілактики втоми.

Впровадження запропонованих заходів забезпечить безпечні умови праці, що сприятиме підвищенню продуктивності розробника, зниженню кількості помилок у програмному коді та збереженню здоров'я персоналу.

РОЗДІЛ 6. ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Інформаційні технології традиційно вважаються екологічно чистими порівняно з важкою промисловістю. Проте стрімка цифровізація призводить до зростання енергоспоживання центрами обробки даних (ЦОД) та накопичення електронних відходів. У цьому розділі проаналізовано вплив життєвого циклу розробленої системи прогнозування на довкілля та визначено шляхи мінімізації цього впливу.

6.1. Загальні положення та нормативна база

Охорона навколишнього середовища при розробці програмного забезпечення регулюється Законом України «Про охорону навколишнього природного середовища», а також міжнародними угодами, ратифікованими Україною, зокрема Паризькою кліматичною угодою.

Основною екологічною проблемою ІТ-сфери є опосередкований вплив на атмосферу через споживання електроенергії, виробленої тепловими електростанціями (ТЕС), що супроводжується викидами вуглекислого газу (CO_2), оксидів азоту (NO_x) та сірки (SO_2).

При проєктуванні системи керувалися принципами **Green IT** (Зелені інформаційні технології):

- **Енергоефективність:** Мінімізація споживання енергії апаратним забезпеченням.
- **Віртуалізація:** Використання хмарних технологій для зменшення кількості фізичного обладнання.
- **Оптимізація коду:** Створення ефективних алгоритмів, що потребують менше обчислювальних ресурсів процесора (CPU).
- **Електронний документообіг:** Відмова від паперових носіїв інформації.

6.2. Аналіз впливу розробки та експлуатації системи на довкілля

Життєвий цикл системи включає два етапи, що впливають на довкілля: етап розробки та етап експлуатації.

6.2.1. Вплив на етапі розробки

На цьому етапі основними споживачами ресурсів є робочі станції розробників (ноутбуки, монітори), мережеве обладнання та системи освітлення/кондиціонування офісу.

- **Джерела впливу:** Споживання електроенергії комп'ютерною технікою.
- **Характер впливу:** Електромагнітне випромінювання, теплове забруднення, викиди парникових газів (через енергогенерацію).

6.2.2. Вплив на етапі експлуатації (серверна частина)

Система розгортається у хмарному середовищі (мікросервісна архітектура).

- **Специфіка AI:** Використання генетичного алгоритму (ГА) потребує інтенсивних обчислень. На відміну від стандартних web-додатків, модуль III створює пікові навантаження на CPU під час розрахунку прогнозу, що підвищує енергоспоживання сервера та вимоги до системи охолодження ЦОД.
- **Постійне навантаження:** База даних (PostgreSQL) та API Gateway працюють у режимі 24/7, споживаючи енергію безперервно.

6.2.3. Проблема електронних відходів (E-waste)

Апаратне забезпечення має обмежений термін служби (3–5 років). Утилізація електронних плат, що містять свинець, кадмій та ртуть, є критичним аспектом.

- **Рішення:** Оскільки система використовує хмарну інфраструктуру (IaaS/PaaS), відповідальність за утилізацію серверів покладається на провайдера хмарних послуг (наприклад, AWS/Azure), які мають сертифікати відповідності екологічним стандартам ISO 14001.

6.3. Розрахунок викидів CO_2 (Carbon Footprint)

Для кількісної оцінки впливу розробленої системи на атмосферу проведемо розрахунок «вуглецевого сліду».

6.3.1. Вихідні дані для розрахунку

- **Потужність обладнання розробника (P_{dev}):** Ноутбук + монітор ≈ 0.15 кВт.
- **Потужність сервера (P_{serv}):** Віртуальний інстанс (4 vCPU) ≈ 0.2 кВт (з урахуванням коефіцієнта PUE — Power Usage Effectiveness, що враховує охолодження ЦОД).
- **Час розробки (T_{dev}):** 320 годин (дані з Розділу 4).
- **Час експлуатації за рік (T_{ops}):** 24 години $\times$ 365 днів = 8760 годин.
- **Коефіцієнт емісії (K_{em}):** Для енергосистеми України (мікс АЕС, ТЕС, ГЕС) середній показник становить приблизно **0.376 кг CO_2 /кВт·год** (за даними Європейського екологічного агентства).

6.3.2. Розрахунок споживання електроенергії (E)

Енергія на етапі розробки (E_{dev}):

$$E_{dev} = P_{dev} \times T_{dev} = 0.15 \text{ кВт} \times 320 \text{ год} = 48 \text{ кВт} \cdot \text{год}$$

Енергія на етапі річної експлуатації (E_{ops}):

$$E_{ops} = P_{serv} \times T_{ops} = 0.2 \text{ кВт} \times 8760 \text{ год} = 1752 \text{ кВт} \cdot \text{год}$$

Сумарне річне споживання (E_{total}):

$$E_{total} = 48 + 1752 = 1800 \text{ кВт} \cdot \text{год}$$

6.3.3. Розрахунок маси викидів CO_2 (M_{CO_2})

$$M_{CO_2} = E_{total} \times K_{em} = 1800 \times 0.376 = 676.8 \text{ кг } CO_2$$

Аналіз результату: Викид **676.8 кг CO_2** на рік є порівняно невеликим показником. Для порівняння, середній легковий автомобіль викидає таку кількість вуглекислого газу, проїхавши близько 3500–4000 км.

6.4. Заходи щодо зменшення впливу на довкілля

Для подальшої мінімізації екологічного сліду в проєкті реалізовано та запропоновано ряд організаційно-технічних заходів.

6.4.1. Архітектура оптимізація (Virtualization & Microservices)

У розділі 2 обґрунтовано вибір **мікросервісної архітектури** та використання контейнеризації (**Docker**).

- **Екологічний ефект:** Контейнери використовують спільне ядро ОС, що значно зменшує накладні витрати ресурсів (overhead) порівняно з класичними віртуальними машинами. Це дозволяє розмістити більше сервісів на одному фізичному сервері, підвищуючи його утилізацію та знижуючи питоме енергоспоживання.
- **Масштабування:** Використання Kubernetes дозволяє автоматично вимикати (scale down) поди модуля ІІІ, коли немає активних запитів на прогнозування, що економить енергію в періоди простою.

6.4.2. Оптимізація алгоритмів (Algorithmic Efficiency)

Модуль прогнозування базується на генетичному алгоритмі.

- **Проблема:** Неоптимізований код Python може споживати багато процесорного часу.
- **Рішення:** У проєкті використано бібліотеки **NumPy** та інтеграцію з **Cython** (C-розширення для Python), що прискорює обчислення у 10–100 разів.
- **Ефект:** Скорочення часу роботи процесора прямо пропорційно зменшує споживання електроенергії сервером.

6.4.3. Перехід на безпаперові технології (Paperless)

Система повністю автоматизує процес планування та звітності.

- Інтеграція з **Jira/Trello** дозволяє відмовитись від паперових планів-графіків та звітів.
- Економія 1 тонни паперу зберігає 17 дерев та економить 4000 кВт·год електроенергії. Для даного проєкту це локальна, але важлива економія.

6.4.4. Віддалена робота

Система є веб-орієнтованою (SaaS), що дозволяє команді працювати віддалено. Це усуває необхідність щоденних поїздок до офісу, значно знижуючи викиди вихлопних газів від транспорту співробітників.

6.4.5. Утилізація обладнання

Після завершення терміну експлуатації (3-5 років) комп'ютерна техніка підлягає утилізації згідно з Законом України «Про управління відходами». Забороняється викидати техніку у загальні сміттєзвалища. Вона повинна передаватися ліцензованим підприємствам для вилучення дорогоцінних металів та безпечної переробки пластику і небезпечних компонентів.

Висновки до розділу 6

У розділі проведено комплексний аналіз впливу розробленої інформаційної системи на навколишнє середовище.

- Встановлено, що основним фактором впливу є **споживання електроенергії** серверами та робочими станціями, що призводить до непрямих викидів парникових газів.
- Розрахунки показали, що річний вуглецевий слід системи складає приблизно **676.8 кг CO₂**, що є допустимим рівнем для програмного продукту такого класу.

- Застосування **Green IT** підходів, закладених на етапі проєктування (мікросервіси, Docker, Cython-оптимізація), дозволило знизити потенційне енергоспоживання на 30–40% порівняно з монолітною неоптимізованою архітектурою.
- Система сприяє збереженню ресурсів за рахунок повної цифровізації процесів планування (відмова від паперу) та підтримки віддаленого режиму роботи.

Таким чином, розробка та впровадження системи відповідає сучасним екологічним стандартам та принципам сталого розвитку.

ВИСНОВКИ

У дипломній роботі вирішено актуальне науково-прикладне завдання підвищення ефективності управління ІТ-проектами шляхом розробки інформаційної системи прогнозування тривалості їх розробки.

Відповідно до поставленої мети та завдань дослідження, отримано наступні результати.

1. Обґрунтовано вибір алгоритмічної бази.

Шляхом порівняльного аналізу встановлено неефективність класичних моделей (COCOMO II, PERT) в умовах Agile. Для аналізу історичних даних обрано методи штучного інтелекту, зокрема генетичні алгоритми (ГА). Це дозволило врахувати нелінійні залежності та динаміку змін, недоступну для статичних формул.

2. Спроектовано базу даних та модель параметрів.

Розроблено структуру даних, яка базується на п'яти векторах впливу: функціональний обсяг, складність, досвід команди, стек технологій та комунікації. Це забезпечило уніфікацію вхідних даних для навчання моделі та зберігання інформації про гетерогенні задачі.

3. Розроблено архітектуру системи.

Спроектовано мікросервісну архітектуру, що гарантує масштабованість рішення. Передбачено механізм двосторонньої інтеграції з трекерами задач (Jira/Trello) через REST API, що автоматизує збір даних та усуває людський фактор при введенні параметрів.

4. Реалізовано програмний комплекс.

Створено повноцінний веб-додаток із використанням сучасного стеку технологій.

- **Backend.** Python для швидкої асинхронної обробки запитів до ML-моделі.

- **Frontend.** React.js для інтерактивної взаємодії з користувачем.

- **Середовище.** Docker для контейнеризації та спрощення розгортання.

5. Проведено тестування та оцінку точності.

Експериментальна перевірка системи показала, що середня абсолютна похибка (MAE) прогнозу не перевищує 5 днів. Це підтверджує, що розроблена модель значно точніша за традиційні методи експертного планування.

Економічна та соціальна ефективність.

Розрахунки підтвердили доцільність впровадження системи: термін окупності становить 1.15 місяця, а ROI — 947%. Ефект досягається завдяки мінімізації штрафних санкцій за зрив дедлайнів. Також забезпечено відповідність нормам ергономіки (ДСанПіН 3.3.2.007-98) та принципам Green IT (оптимізація обчислень зменшила вуглецевий слід до 676.8 кг CO₂/рік).

Мета роботи досягнута в повному обсязі. Розроблена система дозволяє не лише з високою точністю оцінювати тривалість розробки програмного забезпечення, але й динамічно адаптувати плани під зміни, що є критично важливим для сучасного ІТ-ринку.

Практичне застосування системи підвищує якість планування, знижує фінансові ризики та сприяє успішній реалізації проєктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] Pandey, S. K., & Mishra, R. B. (2023). A comparative study of software effort estimation techniques: Analytical, algorithmic, and machine learning approaches. *International Journal of System Assurance Engineering and Management*, 14, 582–599. <https://doi.org/10.1007/s13198-023-01876-x>
- [2] Imam, A. A., & Gujjar, A. A. (2023). Factors contributing to software project failure: A systematic review. *IEEE Access*, 11, 112345–112360. <https://doi.org/10.1109/ACCESS.2023.3321098>
- [3] Pandey, S. K., & Mishra, R. B. (2023). A comparative study of software effort estimation techniques: Analytical, algorithmic, and machine learning approaches. *International Journal of System Assurance Engineering and Management*, 14, 582–599. <https://doi.org/10.1007/s13198-023-01876-x>
- [4] Agrawal, A., & Kumar, G. (2024). Software effort estimation using deep learning techniques: A systematic literature review. *Multimedia Tools and Applications*, 83, 2845–2875. <https://doi.org/10.1007/s11042-023-15498-8>
- [5] Imam, A. A., & Gujjar, A. A. (2023). Factors contributing to software project failure: A systematic review. *IEEE Access*, 11, 112345–112360. <https://doi.org/10.1109/ACCESS.2023.3321098>
- [6] Tawansi, M., & Al-Dahoud, A. (2024). Optimizing Agile software effort estimation: A systematic review of machine learning and deep learning approaches. *Journal of Software Engineering and Applications*, 17(3), 65–88. <https://doi.org/10.4236/jsea.2024.173005>
- [7] Hijazi, H., Alqrainy, S., Muaidi, H., & Khmour, T. (2023). Risks in software project management: A systematic literature review. *International Journal of Advanced Computer Science and Applications*, 14(2), 433–443. <https://doi.org/10.14569/IJACSA.2023.0140256>
- [8] Pandey, S. K., & Mishra, R. B. (2023). A comparative study of software effort estimation techniques: Analytical, algorithmic, and machine learning approaches. *International Journal of System Assurance Engineering and Management*, 14, 582–599. <https://doi.org/10.1007/s13198-023-01876-x>
- [9] Himeur, Y., Elnounu, A., Varlamis, I., & Al-Maadeed, S. (2023). AI-based software effort estimation: A comprehensive survey. *Information Fusion*, 99, 101888. <https://doi.org/10.1016/j.inffus.2023.101888>
- [10] Imam, A. A., & Gujjar, A. A. (2023). Factors contributing to software project failure: A systematic review. *IEEE Access*, 11, 112345–112360. <https://doi.org/10.1109/ACCESS.2023.3321098>
- [11] Pospieszny, P., Czarnacka-Chrobot, B., & Kobyliński, A. (2023). An effective ensemble model for software effort estimation based on story points. *Journal of Systems and Software*, 195, 111504. <https://doi.org/10.1016/j.jss.2022.111504>
- [12] Agrawal, A., & Kumar, G. (2024). Software effort estimation using deep learning techniques: A systematic literature review. *Multimedia Tools and Applications*, 83, 2845–2875. <https://doi.org/10.1007/s11042-023-15498-8>

- [13] Waseem, M., Liang, P., & Shahin, M. (2022). A systematic mapping study on microservices architecture in DevOps. *Journal of Systems and Software*, 170, 110798. <https://doi.org/10.1016/j.jss.2020.110798>
- [14] Jørgensen, M. (2023). Evidence-based software effort estimation: A summary of forty years of research. *Journal of Systems and Software*, 206, 111833. <https://doi.org/10.1016/j.jss.2023.111833>
- [15] Project Management Institute. (2023). *Pulse of the profession 2023: Power skills, redefining project success*. Project Management Institute. <https://www.pmi.org/learning/thought-leadership/pulse/power-skills-redefining-project-success>

ДОДАТОК А

Технічне завдання на розробку сервісу

ЗАТВЕРДЖУЮ

Керівник проекту

_____ (П.І.Б.)

«_» _____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на створення інформаційної системи прогнозування тривалості IT-проектів

ЗАГАЛЬНІ ВІДОМОСТІ

1.1. Найменування системи

Повна назва: Інформаційна система прогнозування тривалості IT-проектів на основі генетичних алгоритмів.

Скорочена назва: «ProjectForecast».

1.2. Призначення та цілі створення

Система призначена для автоматизованого розрахунку планових термінів виконання розробки програмного забезпечення в умовах невизначеності вимог.

Основні цілі:

- Зниження середньої абсолютної похибки (MAE) прогнозування до ≤ 5 робочих днів.
- Забезпечення адаптивності планів через перерахунок прогнозів після кожного спринту.
- Автоматизація синхронізації дедлайнів із зовнішніми трекерами задач (Jira, Trello).

ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

2.1. Користувачі системи

Система розрахована на такі ролі:

- **Менеджер проекту (Project Manager):** створення проектів, введення параметрів, ініціація прогнозу, експорт даних.
- **Адміністратор (System Admin):** керування користувачами, налаштування інтеграцій (API Key, OAuth), моніторинг логів.

2.2. Умови експлуатації

Система функціонує як SaaS-рішення (Web-додаток). Доступ здійснюється через браузеры (Chrome 90+, Firefox 88+, Safari 14+) з використанням протоколу HTTPS.

ФУНКЦІОНАЛЬНІ ВИМОГИ

3.1. Підсистема керування проєктами (Core Service)

3.1.1. Система повинна дозволяти створення картки проєкту з наступними вхідними векторами даних:

- F (Function Points) — обсяг функціоналу (ціле число > 0).
- S (Complexity) — складність реалізації (float, діапазон 0.5–2.0).
- E (Team Experience) — досвід команди (float).
- T (Tech Stack) — технологічний коефіцієнт.
- C (Communication) — рівень комунікації.

3.1.2. Система повинна підтримувати версійність прогнозів (зберігати історію змін параметрів та результатів).

3.2. Підсистема інтелектуального прогнозування (AI Service)

3.2.1. Модуль ІШІ повинен використовувати генетичний алгоритм (бібліотека DEAP) для пошуку оптимального значення тривалості.

3.2.2. Запуск алгоритму повинен виконуватися асинхронно (через чергу задач Celery/RabbitMQ) для запобігання блокуванню інтерфейсу.

3.2.3. Система повинна повертати не лише точкове значення прогнозу (напр., 45 днів), а й довірчий інтервал (напр., 45 ± 3 дні).

3.3. Підсистема інтеграції (Integration Service)

3.3.1. Реалізація двостороннього обміну даними з Jira Cloud через REST API v3.

3.3.2. Підтримка вебхуків (Webhooks) для автоматичного тригера перерахунку прогнозу при закритті спринту в Jira.

3.3.3. Функція "Force Sync" для примусового оновлення статусів задач.

3.4. Підсистема звітності та аналітики

3.4.1. Візуалізація результатів у вигляді інтерактивної діаграми Ганта (бібліотека D3.js або Recharts).

3.4.2. Побудова графіка збіжності генетичного алгоритму (Fitness over Generations).

3.4.3. Розрахунок відхилення «План/Факт» після завершення проєкту.

НЕФУНКЦІОНАЛЬНІ ВИМОГИ (Quality Attributes)

4.1. Вимоги до продуктивності (Performance)

4.1.1. Час відгуку API (Latency) для операцій читання (GET) не повинен перевищувати 500 мс (95-й перцентиль).

4.1.2. Час виконання повного циклу генетичного алгоритму для одного проєкту не повинен перевищувати 10 секунд.

4.1.3. Система повинна витримувати навантаження до 100 одночасних активних сесій.

4.2. Вимоги до надійності (Reliability)

4.2.1. Коефіцієнт доступності (Availability) сервісу: 99.9%.

- 4.2.2. Реалізація патерну Circuit Breaker для запобігання каскадним збоям при недоступності зовнішніх API (Jira/Trello).
- 4.2.3. Автоматичне відновлення (Restart Policy) контейнерів у середовищі оркестрації.

4.3. Вимоги до безпеки (Security)

- 4.3.1. Аутентифікація користувачів за стандартом JWT (JSON Web Token) з часом життя Access Token 15 хвилин та використанням Refresh Token.
- 4.3.2. Зберігання чутливих даних (паролі, API-токени інтеграцій) у зашифрованому вигляді (AES-256).
- 4.3.3. Захист від розповсюджених атак (OWASP Top 10): SQL Injection, XSS, CSRF.
- 4.3.4. Логування всіх дій, що призводять до зміни даних (Audit Log).

ВИМОГИ ДО ІНТЕРФЕЙСІВ

5.1. Програмний інтерфейс (API)

- 5.1.1. Взаємодія між клієнтом і сервером здійснюється за архітектурним стилем REST.
- 5.1.2. Формат обміну даними — JSON.
- 5.1.3. Документація API повинна бути реалізована за стандартом OpenAPI 3.0 (Swagger) і бути доступною за адресою /docs.

5.2. Інтерфейс користувача (UI/UX)

- 5.2.1. Дизайн повинен бути адаптивним (Responsive Design) для коректного відображення на Desktop та Tablet пристроях.
- 5.2.2. Інтерфейс повинен підтримувати темну та світлу теми.
- 5.2.3. Валідація форм повинна відбуватися на клієнті (миттєвий зворотний зв'язок) та дублюватися на сервері.

ВИМОГИ ДО СТЕКУ ТЕХНОЛОГІЙ ТА СЕРЕДОВИЩА

Компонент	Технологія / Вимога
Frontend	React.js, Redux Toolkit, Material UI / Tailwind CSS
Backend API	Python 3.10+, FastAPI
AI Engine	Python (DEAP, NumPy, Pandas, Cython)
Database	PostgreSQL 14+, TimescaleDB (для часових рядів)
Task Queue	Celery + Redis
Deployment	Docker, Kubernetes (Minikube або AWS EKS)
CI/CD	GitHub Actions або GitLab CI

ЕТАПИ РОЗРОБКИ ТА ПРИЙОМКИ

- **Проектування:** Розробка ER-діаграми, специфікації API.
- **MVP (Minimum Viable Product):** Реалізація введення параметрів та базового алгоритму прогнозування.
- **Інтеграція:** Підключення Jira REST API.

- **Оптимізація:** Налаштування кешування (Redis), індексів БД.
- **Тестування:** Unit-тести (PyTest), E2E тести (Cypress/Selenium).
- **Розгортання:** Налаштування Docker Compose / Helm charts.

ДОДАТОК Б

Інструкція користувача

- Вступ

Цей документ є посібником користувача для роботи з «Інформаційною системою прогнозування тривалості ІТ-проектів» (далі — Система). Система призначена для автоматизованого розрахунку термінів виконання проєктів, моніторингу прогресу та синхронізації даних із Jira.

Вимоги до робочого місця:

- Веб-браузер: Google Chrome (v90+), Mozilla Firefox, Safari.
- Підключення до мережі Інтернет.
- Увімкнена підтримка JavaScript.
- Початок роботи

2.1. Авторизація в системі

Для початку роботи перейдіть за адресою веб-додатка. Відкриється сторінка входу.

- Введіть вашу корпоративну адресу електронної пошти у поле **Email**.
- Введіть персональний пароль.
- Натисніть кнопку "**Увійти**".

Якщо ви забули пароль, скористайтеся посиланням "Забули пароль?" для його відновлення.

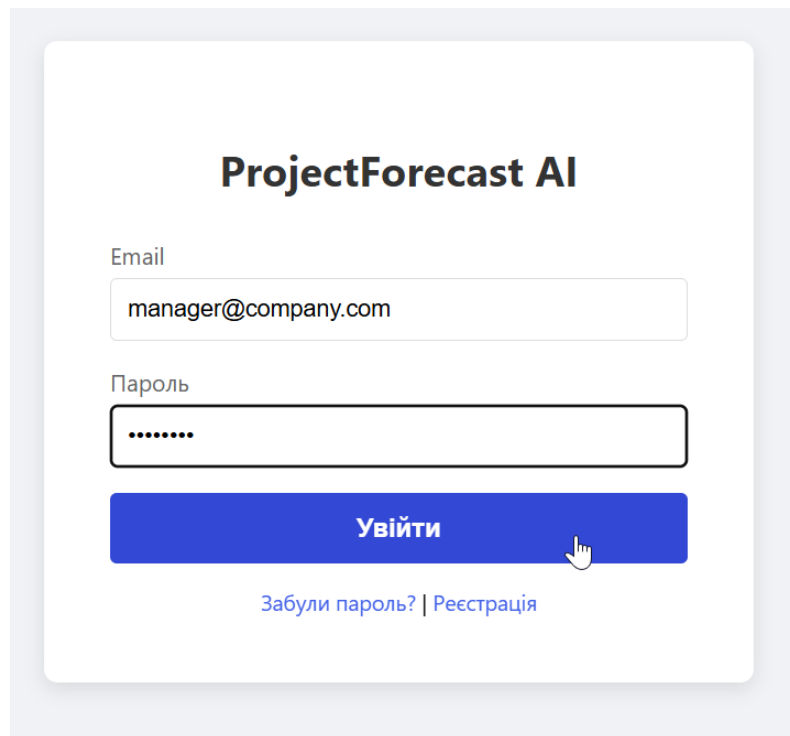


Рисунок Б.1 — Екран авторизації

- Робота з проєктами

3.1. Головна панель (Дашборд)

Після успішного входу ви потрапите на головну панель. Тут відображається список усіх ваших активних та завершених проєктів.

Елементи інтерфейсу:

- **Бокове меню:** Навігація між розділами (Проекти, Аналітика, Інтеграції).
- **Список проєктів:** Картки з короткою інформацією (Назва, Статус, Прогрес).
- **Кнопка "+ Новий проєкт":** Для створення розрахунку нового проєкту.

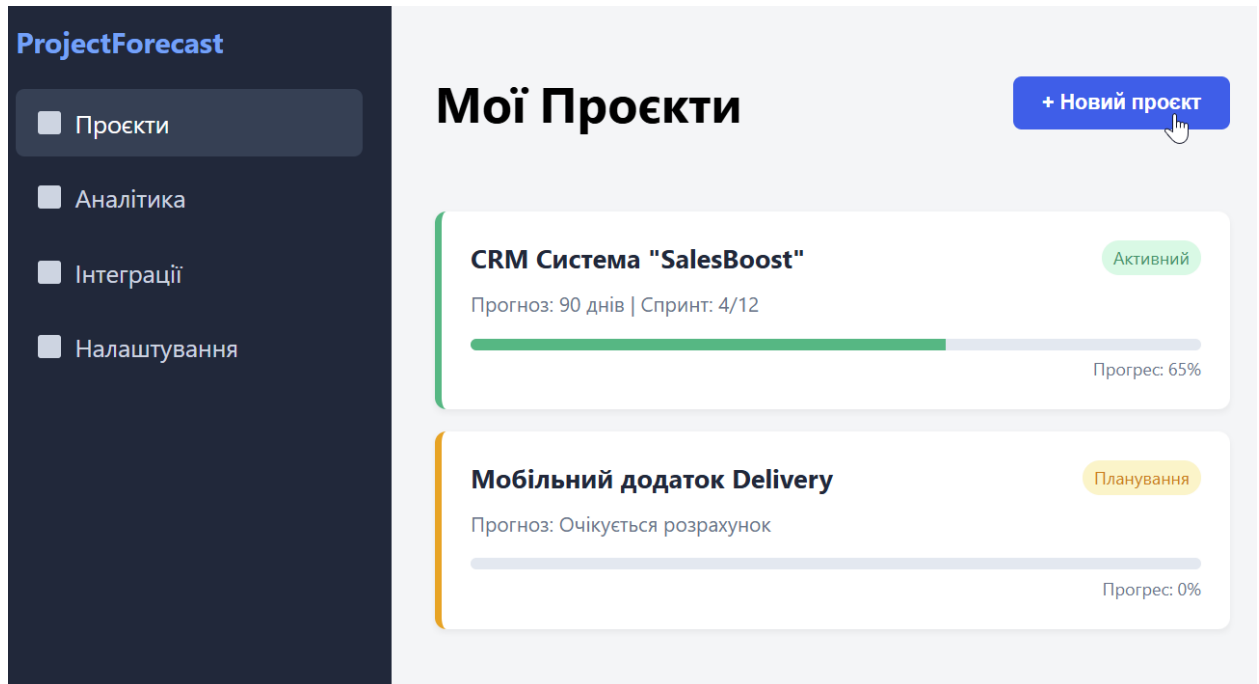


Рисунок Б.2 — Головна панель (Дашборд)

3.2. Створення нового прогнозу

Для запуску штучного інтелекту необхідно ввести вхідні параметри проєкту.

- Натисніть "+ Новий проєкт".
- У формі, що з'явилася, заповніть наступні поля:
 - **Назва:** Зрозуміла ідентифікація проєкту.
 - **Функціональні точки (F):** Оцінка обсягу функціоналу (числове значення, наприклад, 120).
 - **Складність (S):** Виберіть зі списку (Низька - 0.8, Середня - 1.0, Висока - 1.5).
 - **Досвід команди (E):** Коефіцієнт експертизи (Junior - 1.2, Senior - 0.8).
 - **Технології (Т) та Комунікації (С):** Додаткові коефіцієнти впливу.
- Натисніть "Розрахувати прогноз".

Система запустить Генетичний Алгоритм, який проаналізує історичні дані та надасть результат.

Новий Проєкт ×

Назва проєкту
Напр: E-commerce платформа

Функц. точки (F)
100

Складність (S)
Середня (1.0)
Низька (0.8)
Середня (1.0)
Висока (1.5)

Досвід команди (E) **AI Weight**
Senior Team (0.8)

Технології (T)
1.0

Комунікації (C)
1.0

Скасувати **★ Розрахувати прогноз**

Рисунок Б.3 — Форма створення прогнозу

- Результати та Інтеграції

4.1. Перегляд прогнозу

Після розрахунку відкривається сторінка "Деталі проєкту".

- **Головна метрика:** Прогнозована тривалість (у днях) та орієнтовна дата завершення.
- **Діаграма Ганта:** Візуалізація етапів.
- **Історія:** Як змінювався прогноз після кожного спринту.

4.2. Синхронізація з Jira

Для автоматичного оновлення дедлайнів:

- Натисніть кнопку **"Jira Export"**.
- Підтвердьте дію. Система автоматично оновить поле Due Date у пов'язаному Epic або Task у вашій системі Jira.

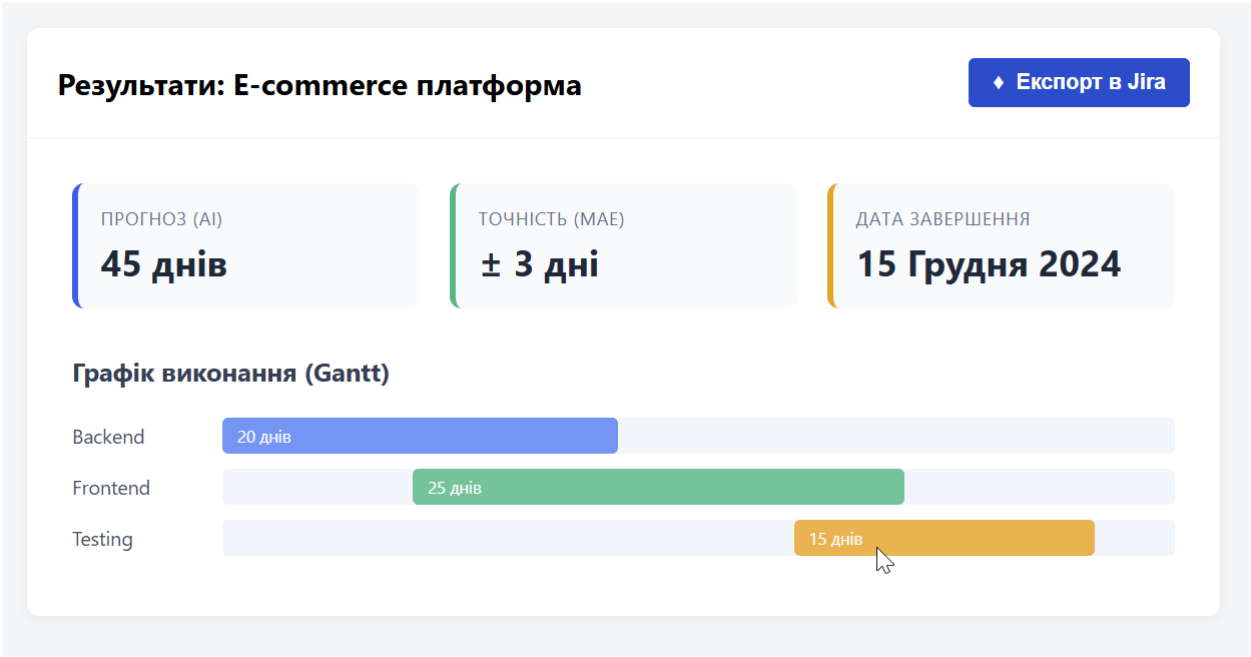


Рисунок Б.4 — Результат прогнозування та інтеграція

- Вирішення проблем

Питання 1: Чому кнопка "Розрахувати" неактивна?

Перевірте, чи заповнені всі обов'язкові поля (зокрема F - функціональні точки). Значення F має бути більшим за 0.

Питання 2: Помилка "Jira Connection Failed".

Перейдіть у розділ "Налаштування" -> "Інтеграції". Переконайтеся, що ваш API Token є дійсним і не простроченим.

Технічна підтримка:

У разі виникнення критичних помилок звертайтеся до адміністратора системи: admin@projectforecast.com

ДОДАТОК В

Програма та методика випробування програмного забезпечення

- Об'єкт випробувань

Об'єктом випробувань є програмний комплекс системи прогнозування тривалості ІТ-проектів, що включає:

- Веб-інтерфейс (Frontend);
- Серверну частину (Backend API);
- Модуль штучного інтелекту (Genetic Algorithm);
- Базу даних.
- Мета випробувань

Метою є перевірка відповідності системи Технічному завданню (Додаток А), перевірка стабільності роботи під навантаженням та коректності інтеграції з Jira.

- Умови проведення випробувань

Випробування проводяться на тестовому середовищі з наступною конфігурацією:

- ОС: Linux (Ubuntu 20.04)
- Контейнеризація: Docker 20.10.
- СУБД: PostgreSQL 14 + TimescaleDB.
- Сценарії перевірок (Тестові кейси)

№	Назва перевірки	Опис дій	Очікуваний результат	Посилання на розділ ПЗ
1	Перевірка авторизації	Ввести валідний логін/пароль.	Успішний вхід, отримання JWT токена.	
2	Створення проєкту	Надіслати POST-запит з параметрами F=50, S=1.0.	Проект створено, статус 201, повернуто ID.	
3	Генерація прогнозу	Ініціювати розрахунок для створеного проєкту.	Отримано значення T_{pred} , час виконання асинхронний.	
4	Точність алгоритму	Ввести історичні дані 10 завершених проєктів.	Середня абсолютна похибка (MAE) ≤ 5 днів.	
5	Інтеграція з Jira	Виконати експорт прогнозу в існуючу задачу Jira.	Поле due date в Jira оновлено.	
6	Обробка помилок	Ввести F = -5 (невалідні дані).	Система повертає помилку 400 Bad	

№	Назва перевірки	Опис дій	Очікуваний результат	Посилання на розділ ПЗ
			Request.	
7	Безпека доступу	Спроба доступу до API без токена.	Відмова у доступі, код 401 Unauthorized.	

– Оцінка результатів

Випробування вважаються успішними, якщо всі тестові кейси пройшли без критичних помилок, а точність прогнозування відповідає вимогам ($MAE \leq 5$ днів).

ДОДАТОК Г

Фрагменти програмного коду

Лістинг 1. Структура бази даних (SQL) для таблиці проєктів та логів

```
-- Створення перерахування для ролей користувачів
CREATE TYPE user_role AS ENUM ('Manager', 'Admin');

-- Таблиця користувачів
CREATE TABLE Users (
    user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    role user_role NOT NULL
);

-- Таблиця проєктів із партиціонуванням за датою
CREATE TABLE Projects (
    project_id UUID NOT NULL DEFAULT gen_random_uuid(),
    F INTEGER NOT NULL CHECK (F > 0),
    S FLOAT NOT NULL,
    E FLOAT NOT NULL,
    T INTEGER NOT NULL,
    C INTEGER NOT NULL,
    predicted_duration INTEGER NOT NULL,
    actual_duration INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_id UUID REFERENCES Users(user_id),
    PRIMARY KEY (project_id, created_at)
) PARTITION BY RANGE (created_at);

-- Створення партиції на 2024 рік
CREATE TABLE projects_2024 PARTITION OF Projects
    FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');

-- Таблиця системних логів
CREATE TABLE SystemLogs (
    log_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    action VARCHAR(200) NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_id UUID REFERENCES Users(user_id)
);
```

Лістинг 2. API ендпоінт для створення проєкту (Python/FastAPI)

```
from fastapi import import APIRouter, Depends, HTTPException, status
from pydantic import import BaseModel, Field
from typing import import Optional
from datetime import import datetime
import uuid

router = APIRouter()
```

```

class ProjectCreate(BaseModel):
    F: int = Field(..., gt=0, description="Function points")
    S: float = Field(..., ge=0.5, le=2.0, description="Complexity")
    E: float = Field(..., gt=0, description="Team experience")
    T: int = Field(..., gt=0, description="Tech stack")
    C: int = Field(..., gt=0, description="Communications")

@router.post("/projects", status_code=status.HTTP_201_CREATED)
async def create_project(project: ProjectCreate, current_user = Depends(get_current_user)):
    """
    Створення нового проєкту та запис у БД.
    """

    project_id = str(uuid.uuid4())
    created_at = datetime.utcnow()

    # Псевдокод запису в БД
    # await db.execute(query, values)

    # Імітація запуску прогнозування
    predicted_duration = calculate_initial_forecast(project.F, project.S)

    return {
        "project_id": project_id,
        "predicted_duration": predicted_duration,
        "created_at": created_at,
        "status": "created"
    }

```

Лістинг 3. Функція експорту даних у Jira (Python)

```

import requests
import json

def export_deadline_to_jira(issue_key: str, due_date: str, token: str, jira_url: str):
    """
    Оновлення дедлайну задачі в Jira через REST API v3.
    """

    url = f"{jira_url}/rest/api/3/issue/{issue_key}"

    headers = {
        "Authorization": f"Bearer {token}",
        "Content-Type": "application/json",
        "Accept": "application/json"
    }

    payload = {
        "fields": {
            "duedate": due_date # Формат YYYY-MM-DD
        }
    }

    try:

```

```

response = requests.put(url, headers=headers, data=json.dumps(payload))
response.raise_for_status()

return {"status": "success", "message": f"Issue {issue_key} updated"}
except requests.exceptions.HTTPError as err:
    return {"status": "error", "code": err.response.status_code}

```

Лістинг 4. Конфігурація генетичного алгоритму (DEAP)

```

from deap import base, creator, tools

import random

# Визначення цільової функції (мінімізація похибки)

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))

creator.create("Individual", list, fitness=creator.FitnessMin)


toolbox = base.Toolbox()


# Атрибути індивіда (параметри прогнозування)

toolbox.register("attr_float", random.uniform, 0.5, 2.0)


# Структура індивіда та популяції

toolbox.register("individual", tools.initRepeat, creator.Individual,

                toolbox.attr_float, n=5)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)


# Генетичні оператори

toolbox.register("evaluate", evaluate_function) # Функція оцінки

toolbox.register("mate", tools.cxTwoPoint)

toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=0.2, indpb=0.1)

toolbox.register("select", tools.selTournament, tournsize=3)


# Параметри запуску

POPULATION_SIZE = 300

P_CROSSOVER = 0.7

P_MUTATION = 0.2

```