МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії програмного забезпечення, канд. техн. наук, доцент

_____Є. О. Давиденко

«____» _____ 20 ____ р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

АВТОМАТИЗАЦІЯ РОЗПОДІЛЕННЯ РОБОЧОГО НАВАНТАЖЕННЯ ЗА ДОПОМОГОЮ ЗАСОБІВ МАШИННОГО НАВЧАННЯ ПРИ УПРАВЛІННІ ІТ-ПРОЄКТАМИ

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ – 608м.21810804

Здобувач: _____ Д. Р. Бечка
підпис

«__» _____ 20__ р.

Керівник: канд. техн. наук, доцент _____ Є. О. Давиденко
підпис

«__» _____ 20__ р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. Давиденко

«____» _____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи магістра

Видано студенту групи 608м факультету комп'ютерних наук

Бечкі Дмитру Романовичу .

(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи

«Автоматизація розподілення робочого навантаження за допомогою засобів машинного навчання при управлінні IT-проєктами» .

Затверджена наказом по ЧНУ від «10» листопада 2023 р. № 234

2. Строк представлення кваліфікаційної роботи «____»_____2024 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Система для автоматизації процесу розподілення робочого навантаження за допомогою засобів машинного навчання та штучного інтелекту при управлінні IT-проєктами. .

4. Перелік питань, що підлягають розробці

- провести аналіз сучасного стану автоматизації управління проєктами;
- визначити цільові функції системи автоматизації;
- сформувати специфікацію вимог до системи;

– спроєктувати функціональні моделі та архітекру системи;

– розробити програмний застосунок на базі навченої моделі та провести тестування на запропонованих наборах даних, проаналізувати отримані результати.

5. Перелік графічних матеріалів:

Презентація, рисунки, таблиці .

Керівник роботи канд. техн. наук, доцент Давиденко Євген Олександрович
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання

Бечка Дмитро Романович
(прізвище, ім'я, по батькові студента)

_____
(підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: «Автоматизація розподілення робочого навантаження за допомогою засобів машинного навчання при управлінні ІТ-проєктами» .

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРМ	01.09.2023р.	10.10.2023р.	Виконано
2.	Огляд літератури за темою роботи	19.10.2023р.	22.10.2023р.	Виконано
3.	Складання календарного плану КРМ	23.10.2023р.	10.11.2023р.	Виконано
4.	Аналіз предметної області	11.11.2023р.	04.12.2023р.	Виконано
5.	Розробка проєктних рішень	04.12.2023р.	04.01.2024р.	Виконано
6.	Моделювання та конструювання ПЗ	05.01.2024р.	10.01.2024р.	Виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	11.01.2024р.	01.02.2024р.	Виконано
8.	Розробка спеціальної частини з охорони праці	02.02.2024р.	05.02.2024р.	Виконано
9.	Відгук керівника КРМ	05.02.2024	15.02.2024	Виконано
10.	Оформлення КРМ та презентації	07.02.2024	08.02.2024	Виконано
11.	Попередній захист	08.02.2024	08.02.2024	Виконано
12.	Рецензування	16.02.2024	19.02.2024	Виконано
13.	Завершення оформлення КРМ та презентації	01.02.2024	20.02.2024	Виконано
14.	Захист кваліфікаційної роботи	26.02.2024р.	27.02.2024р.	Виконано

Розробив студент Бечка Дмитро Романович _____
(прізвище, ім'я, по батькові студента) (*підпис*)

«____»_____20 __ р.

Керівник роботи канд. техн. наук, доцент Давиденко Євген Олександрович
(посада, прізвище, ім'я, по батькові) (*підпис*)

«____»_____20 __ р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра

«Автоматизація розподілення робочого навантаження за допомогою засобів машинного навчання при управлінні ІТ-проєктами»

Студент 608м гр.: Бечка Дмитро Романович

Керівник: канд. техн. наук, доцент Давиденко Є. О.

У цифрову епоху, коли дані часто називають новою нафтою, ІТ-компанії виявляють, що інструменти для видобутку та вдосконалення цього ресурсу – аналіз даних та машинне навчання – є життєво важливими для їхньої діяльності та зростання. Застосування цих технологій – це не просто тенденція, а фундаментальний зсув у тому, як ІТ-компанії підходять до вирішення проблем та інновацій.

Об’єкт роботи – процес розподілення робочого навантаження при управлінні ІТ-проєктами.

Предмет роботи – програмні засоби та математичні моделі для забезпечення автоматизації розподілення робочого навантаження при управлінні ІТ-проєктами.

Мета – проєктування і впровадження інноваційної системи для автоматизації процесу розподілення робочого навантаження в межах ІТ-проєктів з використанням методів машинного навчання.

Кваліфікаційна робота магістра складається з вступу, чотирьох розділів, спеціального розділу з охорони праці, висновків та додатків.

У вступі визначається актуальність теми, що приймається за мету та невеликий огляд поставленої задачі, предмет дослідження та об’єкт дослідження.

У першому розділі наведено теоретичну частину дослідження, в ході якого було ознайомлено з предметною областю. В ході дослідження проведено аналів системи, які займають провідне місце у ніші. В результаті проведеної роботи сформовано специфікацію вимог.

У другому розділі кваліфікаційної роботи описуються проєктні рішення для розроблюваної системи, відповідно до специфікації вимог до програмного забезпечення виявленої у попередньому розділі.

У третьому розділі кваліфікаційної роботи наведено послідовність проєктування системи на основі отриманих у попередніх розділах вимог. Зокрема, детально описано архітектуру системи, структуру та бази даних та вибір технологій реалізації програмного забезпечення.

У четвертому розділі наведено детальний опис усіх етапів розрообки системи. Включаючи такі етапи як налаштування середовища розробки, розробка моделей машинного навчання, розробка бізнес-логіки та клієнтської частини застосунку.

У висновках проводиться аналіз проведеної роботи та отриманих результатів.

Кваліфікаційна робота бакалавра викладена на 92 сторінок, вона містить 4 розділи, 47 ілюстрацій, 2 таблиць, 24 джерел в переліку посилань.

Ключові слова: *штучний інтелект*, *машинне навчання*, *управління проєктами*, *кредитний скорінг*, *градієнтний бустінг, логістична регресія, автоматизація робочого навантаження*.

ABSTRACT

of the Master's Thesis

«Automating workload distribution using machine learning tools in IT project management»

Student: Bechka Dmitry

Supervisor: Candidate of Technical Sciences (Ph. D.), Associate Professor Davydenko Y. O.

In the digital age, where data is often referred to as the new oil, IT companies are finding that the tools to mine and improve this resource – data analytics and machine learning – are vital to their operations and growth. The use of these technologies is not just a trend, but a fundamental shift in how IT companies approach problem solving and innovation.

The object of the process of workload distribution in IT project management.

Subject of work – software tools and mathematical models to automate the workload distribution in IT project management.

The purpose is implementation an innovative system for automating the process of workload distribution within IT projects using machine learning methods.

A master's thesis consists of an introduction, four chapters, a special section on occupational safety, conclusions and appendices.

The introduction defines the relevance of the topic, which is taken as the goal and a brief overview of the task, the subject of research and the object of research.

The first section of the study was the theoretical part of the research, which provided an overview of the subject area. During the study, we analyzed the systems that occupy a leading position in the niche. As a result, a specification of requirements was formed.

The second section of the qualification work describes the design solutions for the system under development, in accordance with the specification of software requirements identified in the previous section.

The third section of the qualification work presents the sequence of system design based on the requirements obtained in the previous sections. In particular, the system

architecture, structure and databases, and the choice of software implementation technologies are described in detail

The fourth section provides a detailed description of all stages of system development. This includes such stages as setting up the development environment, developing machine learning models, developing business logic, and the client side of the application.

The conclusions analyze the work done and the results obtained.

The qualification work of the bachelor is presented on 92 pages, it contains 4 sections, 47 illustrations, 2 tables, 24 sources in the list of references.

Keywords: *artificial intelligence, machine learning, project management, credit scoring, gradient boosting, logistic regression, workload automation*.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

МН – машинне навчання

ПЗ – програмне забезпечення

ШІ – штучний інтелект

API – application programming interface

ANOVA – ANalysis Of VAriance

CI/CD – continuous integration/continuous deployment

ML – machine learning

MSE – mean squared error

WOE – weight of evidence

ВСТУП

У цифрову епоху, коли дані часто називають новою нафтою, ІТ-компанії виявляють, що інструменти для видобутку та вдосконалення цього ресурсу – аналіз даних та машинне навчання – є життєво важливими для їхньої діяльності та зростання. Застосування цих технологій – це не просто тенденція, а фундаментальний зсув у тому, як ІТ-компанії підходять до вирішення проблем та інновацій.

Потік даних, що генеруються ІТ-проєктами, може бути приголомшливим, але в ньому криються закономірності та прогнози, які чекають свого часу, щоб їх розкрити. Аналіз даних виконує функцію компаса, який спрямовує компанії крізь ці величезні інформаційні простори, допомагаючи розробляти стратегії, що ґрунтуються на емпіричних фактах. Машинне навчання робить ще один крок далі, пропонуючи прогностичні інсайти, які здатні передбачити ринкові тенденції, поведінку споживачів і потенційні збої в роботі системи ще до того, як вони проявлять себе, що дає лідерам змогу ухвалювати проактивні рішення.

Аналіз даних лежить в основі прийняття обґрунтованих рішень, забезпечуючи статистичний фундамент, на якому ІТ-компанії можуть будувати свої стратегії. ML розширює ці можливості до сфери прогнозування, уможливлюючи передбачення розвитку тенденцій і поведінки системи, що дозволяє діяти на випередження. Ці технології уможливлюють перехід від реактивної до проактивної позиції у стратегічному плануванні.

По суті, аналіз даних і машинне навчання реінжинірингують структуру операцій ІТ-компаній, каталізуючи прогрес, який виходить за рамки звичних кордонів. Вони створюють середовище постійного вдосконалення та інновацій, а завдяки предиктивній аналітиці та складній інтерпретації даних ІТ-компанії можуть не лише передбачати майбутні виклики, але й розробляти надійні рішення, забезпечуючи стійкість та стабільне зростання у секторі, що швидко розвивається.

Актуальність теми зумовлена тим, що в IT-компаніях, де гнучкість і стратегічне управління ресурсами мають вирішальне значення для виживання і зростання, інтеграція машинного навчання та аналітики даних змінює правила гри. Використовуючи ці технології, такі компанії можуть значно покращити планування проєктів та оптимізувати розподіл ресурсів, тим самим підвищуючи ефективність та конкурентну перевагу.

Аналізуючи історичні дані проєкту, алгоритми ML можуть прогнозувати часові рамки проєкту з більшою точністю, допомагаючи таким чином встановлювати більш реалістичні проміжні етапи. Така предиктивна аналітика поширюється і на управління ризиками, де потенційні перешкоди на шляху проєкту передбачаються за допомогою аналізу тенденцій минулих проєктів, що дозволяє планувати на випередження.

Управління задачами також виграє від ML. Декомпозиція проєктів на окремі задачі та їх розумна послідовність зменшує кількість потенційних проблем. Така оптимізація не лише впорядковує робочі процеси, але й дозволяє невеликим IT-компаніям виконувати проєкти в стислі терміни, що є критично важливим фактором для задоволення та утримання клієнтів.

Користь ML також проникає в управління людськими ресурсами, де він допомагає розшифровувати складні закономірності в кадрових даних, що призводить до вдосконалення стратегій підбору персоналу та підвищення рівня утримання працівників. Прогнозуючи плинність кадрів і визначаючи змінні, що впливають на задоволеність роботою, алгоритми машинного навчання допомагають формувати більш стабільну і залучену робочу силу. Відповідність навичок удосконалюється за допомогою аналізу даних, гарантуючи, що потрібні люди з потрібними навичками працюють над завданнями, які відповідають їхньому досвіду. Таке стратегічне узгодження стає можливим завдяки глибокому вивченню даних про ефективність роботи працівників та їх оцінці відповідно до вимог проєкту, що не лише забезпечує успіх проєкту, але й підвищує залученість та продуктивність працівників.

Об’єкт дослідження – процес розподілення робочого навантаження при управлінні ІТ-проєктами.

Предмет дослідження – програмні засоби та математичні моделі для забезпечення автоматизації розподілення робочого навантаження при управлінні ІТ-проєктами.

Метою кваліфікаційної роботи є проєктування і впровадження інноваційної системи для автоматизації процесу розподілення робочого навантаження в межах ІТ-проєктів з використанням методів машинного навчання.

Досягнення поставленої мети обумовлює необхідність вирішення наступних **завдань**:

- провести аналіз сучасного стану автоматизації управління проєктами;
- визначити цільові функції системи автоматизації;
- сформувати специфікацію вимог до системи;
- спроєктувати функціональні моделі та архітекру системи;
- розробити програмний застосунок на базі навченої моделі та провести тестування на запропонованих наборах, проаналізувати отримані результати.

1 АНАЛІЗ СТАНУ ПРОБЛЕМИ АВТОМАТИЗАЦІЇ РОЗПОДІЛЕННЯ РОБОЧОГО НАВАНТАЖЕННЯ В УПРАВЛІННІ ІТ-ПРОЄКТАМИ

Системи автоматизації управління проєктами є набором інструментів, програмного забезпечення та методологій, призначених для оптимізації процесів управління проєктами [1]. Вони спрощують планування, виконання, моніторинг та звітність проєктів, а також підвищують ефективність комунікацій між учасниками проєкту. Основна мета таких систем — забезпечити своєчасне та ефективне виконання проєктів відповідно до заданих бюджетів і термінів.

Автоматизація розподілення робочого навантаження є однією з найбільш актуальних та складних проблем управління ІТ-проєктами. За умов швидкого розвитку технологій та зростання обсягів даних, необхідність ефективно розподіляти завдання між членами команди, враховуючи їхні компетенції, досвід та поточне навантаження, стає ключовою для успішної реалізації проєктів.

Серед основних викликів автоматизації розподілення робочого навантаження є нестача гнучкості в існуючих системах управління проєктами, які часто не враховують змінювані умови проєкту та індивідуальні особливості працівників [2]. Більшість підходів до управління робочим навантаженням засновані на ручному розподілі задач, що може призводити до нерівномірного навантаження на працівників, зниження мотивації та ефективності роботи команди.

1.1 Аналіз існуючих засобів автоматизації в системах управління проєктами

Для того, щоб отримати повне уявлення про поточний стан інструментів управління ІТ-проєктами, дуже важливо ретельно проаналізувати наявні засоби автоматизації. Цей аналіз повинен охоплювати не лише функціональні можливості та обмеження цих інструментів, але й їхній вплив на управління проєктами. Вивчення існуючих систем допомагає визначити найбільш підходящі

інструменти та програмне забезпечення для управління проєктами для конкретних потреб [3].

На ринку наразі присутні два гіганти у сфері управління проєктами – Trello та Jira. В контексті даного дослідження не будуть розглядатися основні функції систем управління проєктами, натомість увагу буде зосереджено саме на методах автоматизації в даних системах.

Trello

Trello використовує систему дошок, списків та карток для організації завдань і проєктів. Це інтуїтивно зрозумілий інструмент для візуального управління проєктами, який дозволяє користувачам легко переміщувати завдання між різними стадіями виконання. Trello особливо підходить для невеликих команд і простих проєктів.

Автоматизація з Butler в Trello є потужним інструментом, що дозволяє користувачам автоматизувати рутинні задачі та процеси в рамках своїх дошок Trello. Butler був створений як сторонній плагін, але після його успіху та популярності був інтегрований безпосередньо в платформу Trello як вбудована функція. Він працює на основі простих або складних команд, які можна налаштувати для автоматизації різних завдань, спрощуючи управління проєктами та підвищуючи продуктивність команди.

Загалом автоматизація в Trello має невеликий набір функцій, але при бажані можна створювати дасить потужні команди, які дуже добре показують себе у роботі.

Таблиця 1.1 – Засоби автоматизації системи управління проєктами Trello

Автоматизація з Butler	
Правила	Можливість створювати специфічні правила для автоматизації робочих процесів на основі тригерів та дій, які задаються. Наприклад, «Коли картка переміщується в список 'Виконано', додайте коментар 'Завдання завершено».

Кінець таблиці 1.1

Кнопки	Butler дозволяє створювати налаштовані кнопки, які можна розміщувати на картках або дошках для миттєво виклику автоматизованих дій.
Розклад	Можливість налаштування періодичних завдань, таких як щоденне оновлення статусів або тижневе нагадування про звіти, які виконуються автоматично за розкладом.
Приклади використання Butler	
Автоматичне призначення завдань	Коли створюється нова картка в певному списку, Butler може автоматично призначити її відповідальним особам.
Управління термінами виконання	Автоматичне оновлення дат виконання на основі прогресу роботи або нагадування про майбутні дедлайни.
Оптимізація робочих процесів	Автоматичне переміщення карток до відповідних списків, коли завдання досягає певного етапу виконання

На зображенні (рис. 1.1) представлено скріншот інтерфейсу Butler. Це вікно налаштування автоматичної команди, яка запускається за розкладом. У розділі «Trigger» зазначено, що команда виконуватиметься щодня о 8:10 ранку.

У розділі «Actions» показано доступні дії, які ще не додано до команди. Користувачеві пропонується вибрати дію, наприклад, додавання картки, переміщення карток, роботу зі списками, сортування та інші дії, включно з інтеграцією з Jira і Slack. В даному випадку обрано дію «Перемістити картку».

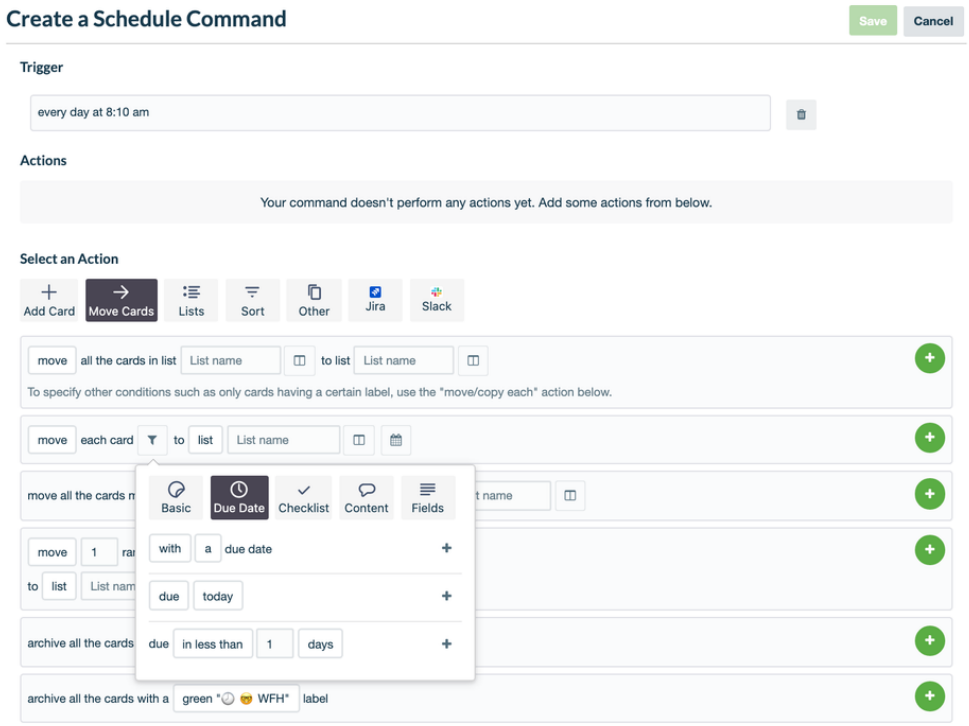


Рисунок 1.1 – Процес створення задачі за розкладом в Butler

Автоматизація з Butler в Trello дозволяє командам мінімізувати час, витрачений на рутинні та повторювані завдання, зосереджуючись більше на стратегічній роботі та креативному процесі.

Jira

Jira від Atlassian є одним з найбільш відомих інструментів для управління проєктами, особливо популярним у середовищі розробників програмного забезпечення. Серед сильних сторін можна виділити наступні:

– Дозволяє налаштовувати робочі процеси під конкретні потреби проєкту, що робить її ідеальною для різних методологій управління, включаючи Scrum і Kanban.

– Велика кількість інтеграцій з іншими інструментами та сервісами, такими як Confluence, Bitbucket та численні зовнішні API для розширення функціональності.

– Потужні інструменти для відстеження помилок та проблем, які дозволяють командам ефективно управляти виправленнями.

Таблиця 1.2 – Засоби автоматизації системи управління проєктами Jira

Автоматизація з Jira Automation	
Автоматизація робочих процесів (Workflows)	Можливість створити налаштовані робочі процеси, які автоматично переводять задачі між різними статусами, в залежності від визначених подій або умов.
Правила автоматизації (Automation Rules)	Jira дозволяє створювати складні правила автоматизації, що включають тригери, умови та дії. Наприклад, ви можете налаштувати правило для автоматичного призначення задачі новому користувачеві, коли попередній виконавець її закриває.
Скриптинг	Для більш складних сценаріїв можна використовувати скриптові розширення (наприклад, ScriptRunner для Jira), які дозволяють писати власні скрипти на Groovy для створення дуже специфічних і потужних автоматизацій.
Приклади використання автомаитзації	
Автоматичне призначення задач	Можна налаштувати правило, яке автоматично призначатиме нові задачі конкретним членам команди або розподілить їх залежно від певних критеріїв, таких як тип задачі.
Оновлення статусів	Правила можуть автоматично оновлювати статуси задач, коли вони рухаються через робочий процес. Наприклад, коли всі підзадачі завершені, основна задача може автоматично перейти в статус «Готово».

Кінець таблиці 1.2

Запуск інтегрованих інструментів	Можна налаштувати правила для запуску зовнішніх скриптів або інтеграції з іншими інструментами, такими як CI/CD пайплайни, коли задача досягає певного статусу.

На скріншоті (рис 1.2) зображено інтерфейс налаштування автоматизації в системі управління завданнями (ймовірно, Jira). У розділі «Automation» (Автоматизація) відображається правило з деталями:

Подія «When: Issue transitioned» (Коли: Перехід задачі) вказує, що автоматизація спрацьовує при зміні статусу задачі з «Open» (Відкрито) на «In Progress» (У роботі).

Умова «Issue Type equals Bug» (Тип завдання дорівнює Bug) означає, що автоматична дія застосовуватиметься тільки до завдань типу «Bug» (Помилка).

Праворуч від цього блоку розташована форма «Issue fields condition» (Умова по полях завдання), яка дає змогу налаштувати умову для поля завдання:

Це означає, що автоматизація застосовуватиметься для завдань, у яких тип вказано як «Помилка». Внизу форми розташовані кнопки «Cancel» (Скасування) і «Save» (Зберегти), імовірно для скасування або збереження налаштованого правила.

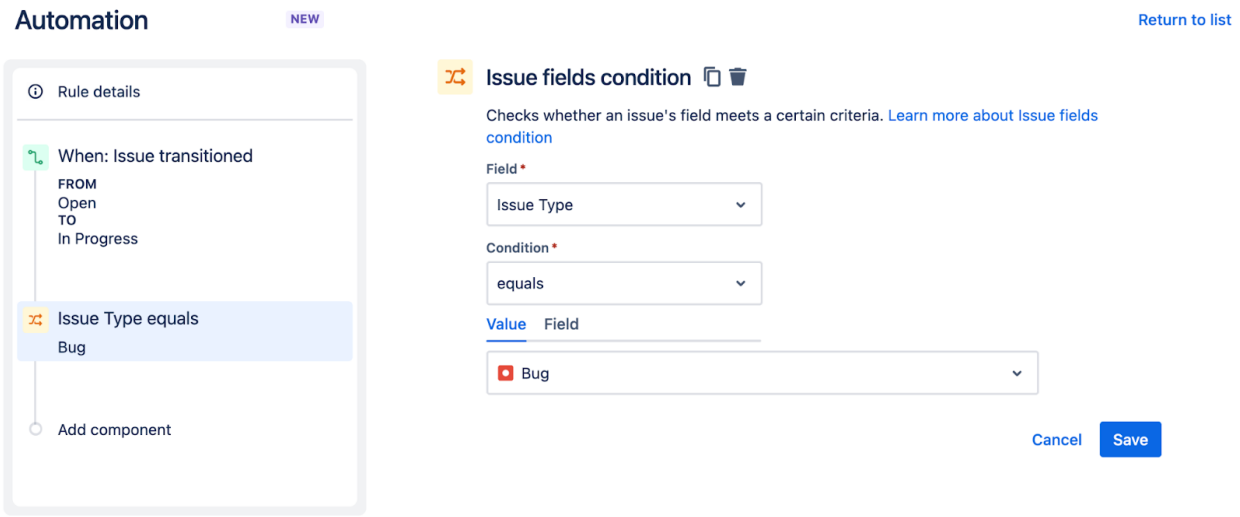


Рисунок 1.2 – Приклад створення автоматизації в Jira

В плані автоматизації Jira набагато розвинута аніж Trello. Дана система більш гнучка, дозволяє використовувати більш широкий спектр функцій. Але основна проблема всього цього різноманіття механік автоматизації – складна крива навчання. Для більшості новачків дуже важко розібратися навіть у базових можливостях Jira Automation.

Розглянуті системи є самодостатніми і створення подібної системи, але з розширеним функціоналом просто не має сенсу. Основна причина – з даними системами неможливо конкурувати. Однак, є багато можливостей для покращення цих систем, шляхом використання штучного інтелекту для автоматизації робочих процесів. Це може включати в себе використання машинного навчання для прогнозування результатів проєктів, алгоритми розподілу завдань з урахуванням вмінь та навичок кожного учасника команди, а також використання інтелектуальних асистентів для підтримки комунікації та співпраці між учасниками проєкту. Такі нові можливості можуть відкрити шлях до використання більш ефективних та продуктивних методів управління проєктами.

1.2 Потенціал машинного навчання в управлінні проєктами

Машинне навчання відкриває нові горизонти у покращенні та оптимізації процесів управління ІТ-проєктами, пропонуючи інноваційні підходи до аналізу даних, прогнозування та автоматизації [4]. Машинне навчання відкриває безліч можливостей для впровадження нових технологій та покращення ефективності управління проєктами. Воно дозволяє аналізувати велику кількість даних та знаходити складні залежності, що допомагає в прийнятті кращих рішень. Більше того, за допомогою машинного навчання можна зробити точні прогнози та автоматизувати багато рутинних задач.

Автоматизація та оптимізація рутинних завдань

МН може автоматизувати рутинні та часозатратні задачі в управлінні проєктами, такі як розподіл ресурсів, відстеження прогресу завдань і управління

змінами. Завдяки цьому, менеджерам проєктів відкривається можливість витратити більше часу на аналіз та стратегічне планування, що дозволяє зосередитися на більш важливих аспектах управління проєктами. Новий рівень автоматизації забезпечує більш ефективне використання ресурсів та оптимізацію процесів управління, що збільшує продуктивність і результативність проєктів.

Поліпшення процесу прийняття рішень

Застосування алгоритмів машинного навчання для аналізу великих обсягів даних є важливим інструментом, який дозволяє глибше досліджувати тенденції та закономірності, що можуть мати великий вплив на проєкт [5]. При використанні цих алгоритмів, ми можемо більш детально виявити потенційні ризики, оцінити вплив змін та забезпечити більш точні прогнози. Це дозволяє нам зробити кращі та обґрунтованіші рішення, що покращують весь процес прийняття рішень.

Прогнозування проблем і визначення ризиків

МН може бути використаний для прогнозування потенційних проблем і визначення ризиків на ранніх стадіях проєкту. Наприклад, шляхом аналізу даних з попередніх проєктів, можна виявити патерни, які часто призводять до затримок або перевитрат, і заздалегідь прийняти відповідні заходи. Крім того, МН може допомогти ідентифікувати можливі варіанти оптимізації проєкту та прогнозувати ефективність застосування різних стратегій [6]. Таким чином, використання МН є корисним інструментом для покращення управління проєктами і забезпечення успішності їх реалізації.

Оптимізація розподілу ресурсів

Алгоритми машинного навчання можуть допомогти в оптимізації розподілу ресурсів, аналізуючи потреби проєкту та доступність ресурсів. Вони можуть автоматично рекомендувати найбільш ефективне використання персоналу, технологій та фінансів, засноване на історичних даних та поточних трендах. Більше того, алгоритми машинного навчання дозволяють виявляти складні залежності та взаємозв'язки між різними факторами, що сприяє більш точному та цілеспрямованому прийняттю рішень. Використання цих алгоритмів управління

ресурсами може значно покращити ефективність та результативність проєктів, забезпечуючи оптимальне використання обмежених ресурсів та досягнення найкращих результатів [7].

Підвищення якості продукту

Машинне навчання може аналізувати зворотний зв'язок від користувачів та тестові результати, щоб ідентифікувати потенційні проблеми якості в продукті або процесі розробки. Крім того, це дозволяє командам швидше реагувати на проблеми та покращувати продукт на основі реальних даних. Наприклад, використання машинного навчання може допомогти виявити недоліки у продукті, які можуть бути неочевидними для людського спостереження. Також, за допомогою машинного навчання можна автоматизувати процес аналізу даних, що спрощує та прискорює виявлення проблем та вдосконалення продукту.

Підтримка безперервного навчання і поліпшення

МН дозволяє збирати і аналізувати значні обсяги даних про процеси управління проєктами, що сприяє постійному навчанню та постійному покращенню. Використання цієї методології може допомогти в ідентифікації найбільш ефективних практик та їх подальшому впровадженні в майбутні проєкти. Завдяки збору та аналізу великих обсягів даних, МН дозволяє отримати більш детальну картину процесів управління проєктами, що забезпечує більш точні рекомендації та покращує результати. Додатково, це дозволяє здійснювати порівняльний аналіз між різними проєктами та виявляти загальні тенденції, які можуть бути корисні для майбутніх проєктів управління [8].

Застосування машинного навчання в управлінні ІТ-проєктами обіцяє значні переваги, від підвищення ефективності та продуктивності команд до покращення якості кінцевих продуктів. Однак важливо враховувати й потенційні виклики, такі як потреба в чистих, організованих даних та необхідність розуміння можливостей та обмежень МН алгоритмів.

1.3 Специфікація вимог до системи

Специфікація вимог до програмного забезпечення (Software Requirements Specification, SRS) є документом, який повністю описує поведінку системи, яку має бути розроблено. Вона включає в себе набір функціональних і нефункціональних вимог, які програмне забезпечення повинне задовольнити. Цей документ важливий для розробників, тестувальників, проєктних менеджерів і зацікавлених сторін, так як він допомагає забезпечити, що всі розуміють, що саме повинно бути створено.

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначення системи (застосунку), для якої розробляється програмне забезпечення:

Система «Автоматизація розподілення робочого навантаження в управлінні ІТ-проєктами» призначена для оптимізації процесу розподілу завдань між співробітниками команди управління ІТ-проєктами за допомогою методів машинного навчання та аналізу даних.

ЗАГАЛЬНИЙ ОПИС

Загальна структура і склад системи

Система складається з серверної частини, яка відповідає за обробку та аналіз даних, та клієнтської частини, що надає інтерфейс користувача для взаємодії з системою.

Загальні обмеження

– Система має бути доступна для використання в обмеженому мережевому середовищі.

– Інформація, що обробляється та зберігається в системі, повинна бути захищена від несанкціонованого доступу.

ФУНКЦІЇ СИСТЕМИ:

– Система повинна в автоматично отримувати оновлення даних про проєкти із плаформи управління проєктами (Trello/Jira).

– Система повинна автоматично оцінювати складність поставлених задач. Складність задач оцінюється в часі потрібному на виконання або story points.

– Система повинна аналізувати навантаженість членів команди які працюють на проєктах.

– Система повинна прогнозувати успішність виконання проєкту.

– Система повинна автоматично розподіляти завдання між учасниками команди на основі аналізу їхньої кваліфікації, досвіду, пріоритетів та навантаження.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

– Система повинна мати можливість функціонувати на операційних системах Windows, Linux або macOS.

– Наявність браузера для користувачів для доступу до інтерфейсу системи.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Програмна система повинна реалізовувати клієнт-серверну архітектуру, яка повинна містити наступні складові частини:

– сервер, який надає інформацію або інші послуги програмам, які звертаються до нього;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та сервером.

Системне програмне забезпечення

Система повинна бути розроблена на мові програмування Python і мати можливість запуску на серверах, що підтримують виконання Python-скриптів.

Мережне програмне забезпечення

Система повинна підтримувати мережеві протоколи TCP/IP для забезпечення комунікації між клієнтом та сервером.

Програмне забезпечення ведення інформаційної бази

У якості системи управління базами даних повинна виступати PostgreSQL версії 13 і вище.

Мова і технологія розробки ПЗ

Серверна частина застосунку повинна бути реалізована на мові програмування Python. В якості основного фреймворку для розробки моделей машинного навчання повинен бути застосований JAX. Крім цього до розробки потрібно залучити такі бібліотеки як NumPy, Pandas та Scikit-learn.

Клієнтська частина застосунку повинна бути написана на JS та використовувати фреймворк Remix.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Інтерфейс користувача системи повинен відповідати наступним вимогам:

– Графічні інтерфейси мають бути захищені від несанкціонованих та нетипових дій користувача.

– Графічні інтерфейси повинні виводити повідомлення про помилку у разі некоректних або помилкових дій системи.

– Графічні інтерфейси повинні виводити повідомлення про помилку у разі некоректних дій користувача.

– Вважається неприйнятним, якщо будь-якими діями у вебінтерфейсі користувач міг викликати недієздатність системи частково або повністю для інших користувачів системи.

Програмний інтерфейс

Система повинна надавати API для інтеграції з іншими системами.

Комунікаційний протокол

Система повинна використовувати протокол HTTPS для передачі даних між клієнтом та сервером.

Для взаємодії з базою даних система повинна використовувати TCP/IP протокол для віддаленого з’єднання.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Безпека

Система повинна забезпечувати високий рівень безпеки для захисту конфіденційної інформації та персональних даних співробітників.

Ефективність

Система повинна працювати швидко та ефективно навіть при великому обсязі даних та великій кількості одночасних користувачів.

Оптимізація продуктивності та ресурсів повинна бути забезпечена для забезпечення швидкого розподілу завдань.

Легкість використання

Інтерфейс користувача повинен бути інтуїтивно зрозумілим та зручним у використанні для користувачів різного рівня технічної підготовки.

Навчання та підтримка користувачів повинні бути надані для ефективного використання системи.

Надійність

– Система повинна зберігати працездатність у разі відмови або виходу з ладу вебсерверів.

– Необхідно забезпечити збереження всієї накопиченої інформації на момент відмови або виходу з експлуатації одного з дискових накопичувачів.

– У разі повного відключення електроенергії система повинна обробити та після відновлення електроенергії запустити повторно на виконання всі незавершені операції.

– Система має забезпечувати працездатність у разі збою одного зі своїх модулів.

ІНШІ ВИМОГИ

Система не має додаткових вимог, окрім описаних у даному документі.

Висновки до розділу 1

У першому розділі будо проведено теоретичну частину дослідження, в ході якого було ознайомлено з предметною областю.

Також у даному розділі було проаналізовано системи, які займають провідне місце в сфері управління ІТ-проєктами. Слід підкреслити той факт, що системи не дарма мають таку високу популярність в різних компаніях. Однак наразі ці системи не можуть покрити проблеми рішенню яких присвячено дане дослідження, а саме автоматизації управління проєктами з використанням методів машинного навчання. З урахуванням цього було виявлено потенціал МН в даній області та сформовано основні переваги, які можна отримати використовуючи ці технології.

З отриманих результатів можна зробити висновок, що проведення дослідження в даній області є доцільним.

Результатом проведеної у розділі роботи стало виявлення специфікації вимоги до розроблюваної системи автоматизації розподілення робочого навантаження в ІТ-проєктах за допомогою засобів машинного навчання.

2 МОДЕЛЮВАННЯ ФУНКЦІОНАЛЬНОЇ СКЛАДОВОЇ СИСТЕМИ

Розділ присвячений детальному аналізу та моделюванню функціональної складової системи автоматизації розподілу робочого навантаження в управлінні IT-проєктами. Основна увага зосереджена на застосуванні передових методів машинного навчання для вирішення ключових задач, які стоять перед системою, зокрема оцінки задач, прогнозування успішності проєктів та оптимального розподілу задач. Цей розділ має на меті не лише висвітлити теоретичні аспекти використаних алгоритмів, але й демонструвати практичну їх застосовність та ефективність у контексті розроблюваної системи.

2.1 Застосування градієнтного бустингу для оцінки задач

Уміння коректно визначати час, необхідний для виконання завдань, - одна з ключових навичок для успішної командної роботи в IT-проєктах. Компанія McKinsey дослідила сотні проєктів у сфері розробки корпоративного ПЗ. Виявилося, що в 66% проєктів відбувається перевищення наміченого бюджету. Третину цих проєктів завершують із серйозним відставанням від графіка. Ще 20% команд взагалі не досягають намічених цілей і завдань. Більше того, у 17% випадків відставання від графіка і перевитрати настільки критичні, що ці проєкти загрожують існуванню компанії [9].

Дана проблема може бути вирішена за допомогою методів машиного навчання. Серед них, алгоритми класифікації надають можливість розпізнавати та категоризувати об’єкти або події на основі набору вхідних даних, що є фундаментальним для багатьох систем аналізу даних. Проте, коли мова заходить про точне прогнозування складності та часу виконання проєктних задач, знадобляться більш спеціалізовані підходи.

У цьому контексті алгоритм градієнтного бустингу, що є одним із передових методів машинного навчання для розв’язання задач регресії та класифікації, виходить на передній план. Він дозволяє з високою точністю

аналізувати історичні дані про виконання задач і їх характеристики для прогнозування оцінок.

Градієнтний бустинг — це метод, що послідовно конструює ансамбль слабких моделей прогнозування, таких як дерева рішень, з метою утворення однієї сильної прогностичної моделі [10]. Ключова особливість градієнтного бустингу полягає у використанні градієнтного спуску для мінімізації помилок, що дозволяє алгоритму ефективно виправляти свої помилки та адаптуватися до складних даних.

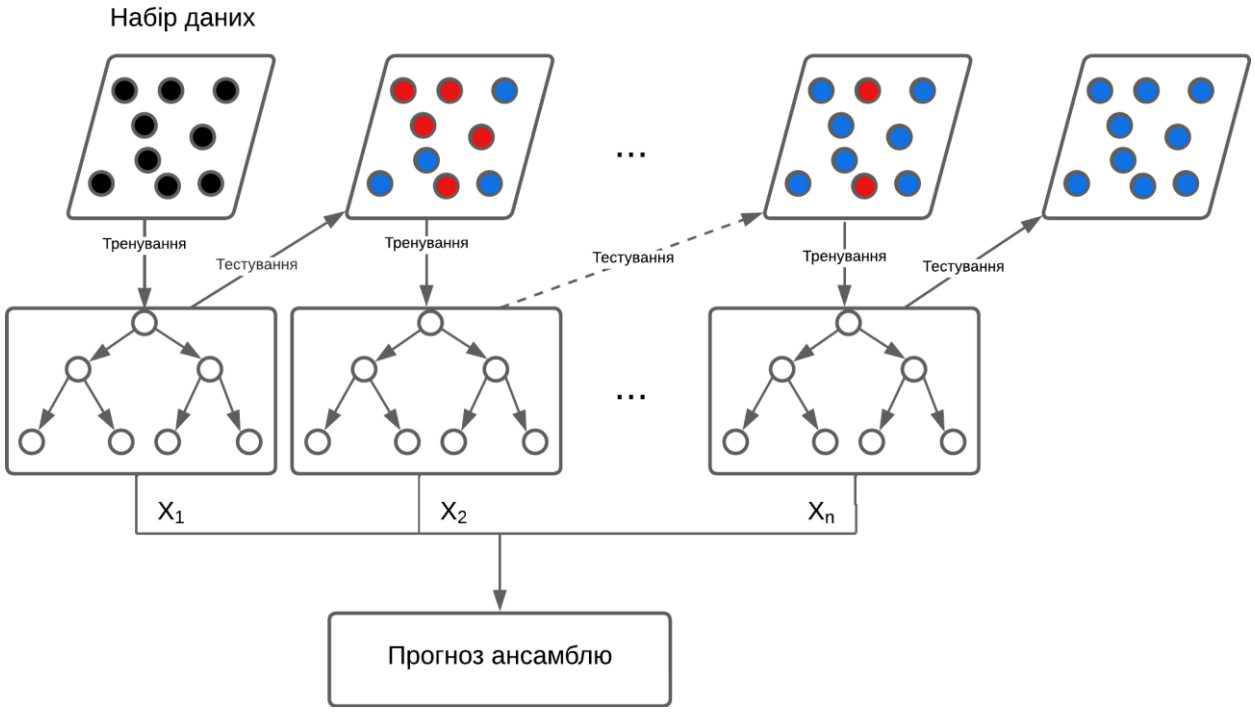


Рисунок 2.1 – Алгоритм навчання градієнтного бустингу з ансамблем дерев

Для реалізації алгоритму градієнтного бустингу в контексті оцінки складності та часу виконання задач у проєктному менеджменті, спочатку визначимо математичну модель та пройдемося по кроках її реалізації з використанням формул і конкретних прикладів. Спочатку потрібно визначити задачі – нехай у нас є набір даних

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}, \quad (2.1)$$

де x_i представляє набір ознак для і-ої задачі,

y_i – час виконання або складність цієї задачі.

Мета – побудувати модель F(x), яка мінімізує загальну помилку прогнозування на цьому наборі даних.

Початкова модель $\mathrm{F}_0(\mathrm{x})$ ініціалізується як константа, наприклад, як середнє значення цільової змінної:

$$\mathrm{F}_0(\mathrm{x}) = \frac{1}{N}\sum_{i=1}^{N} y_i\,, \tag{2.2}$$

Для кожної ітерації $t = 1,2,\ldots,T$:

1. Обчислюємо градієнти втрати для кожного спостереження у наборі даних щодо поточного прогнозу:

$$g_{it} = -\left[\frac{\partial L\big(y_i, F(x_i)\big)}{\partial F(x_i)}\right]_{F(x)=Ft-1(x)}, \tag{2.3}$$

де L – функція втрат, наприклад, середньоквадратична помилка (MSE).

2. Навчаємо дерево рішень *h*t(x) на обчислених градієнтах, тобто модель намагається передбачити градієнт втрат для кожного спостереження.

3. Знаходимо оптимальний коефіцієнт γt для дерева, мінімізуючи втрати при його додаванні до ансамблю:

$$\gamma\mathrm{t} = \arg min_{\gamma}\sum_{i=1}^{N} L\big(y_i, F_{t-1}(x_i) + \gamma h_t(x_i)\big)\,, \tag{2.4}$$

4. Оновлюємо модель, додаючи до неї нове дерево з оптимальним множником:

$$\mathrm{F}_t(\mathrm{x}) = F_{t-1}(x_i) + \gamma_t h_t(x_i), \tag{2.5}$$

Після того, як всі T дерев було додано, кінцева модель $\mathrm{F}_T(x)$ використовується для прогнозування складності або часу виконання нових задач. Після тренування моделі ми можемо використовувати її для прогнозування часу виконання нових задач, враховуючи складність задачі та досвід виконавця.

Однією з найбільших мотивів використання градієнтного бустингу є те, що воно дозволяє оптимізувати визначену користувачем функцію витрат замість функції втрат, яка зазвичай пропонує менший контроль і по суті не відповідає реальним результатам.

2.2 Прогнозування успішності проєктів через логістичну регресію

Сучасні методики прогнозування ризиків варіюються від якісних оцінок до складних кількісних моделей. Логістична регресія була визнана потужним інструментом у різних галузях завдяки своїй здатності опрацьовувати бінарні результати, що цілком підходить для оцінювання ризиків, де результати часто є дихотомічними (успіх або невдача) [11]. У контексті управління проєктами, логістична регресія може використовуватися для прогнозування успішності проєкту (наприклад, успішно завершений чи ні) на основі різних предикторів, таких як оцінки задач, розподіл робочого навантаження, досвід та ефективність команди.

Логістична регресія моделює імовірність $P(Y = 1|X)$ події Y, використовуючи логістичну функцію, що забезпечує вихідні значення між 0 та 1. Математично це можна виразити як:

$$P(Y = 1|X) = \frac{1}{1 + e - (\beta 0 + \beta 1X1 + \beta 2X2 + \cdots + \beta nXn)}, \quad (2.6)$$

де P(Y=1|X) позначає ймовірність того, що проєкт зіткнеться зі значними ризиками за умови X,

$X1$, $X2$,..., Xn представляють прогнозовані змінні (предиктори), коефіцієнти моделі, які визначають вплив кожного предиктора на логіт імовірності події,

$\beta 1$, $\beta 2$,..., βn коефіцієнти моделі, які визначають вплив кожного предиктора на імовірності події.

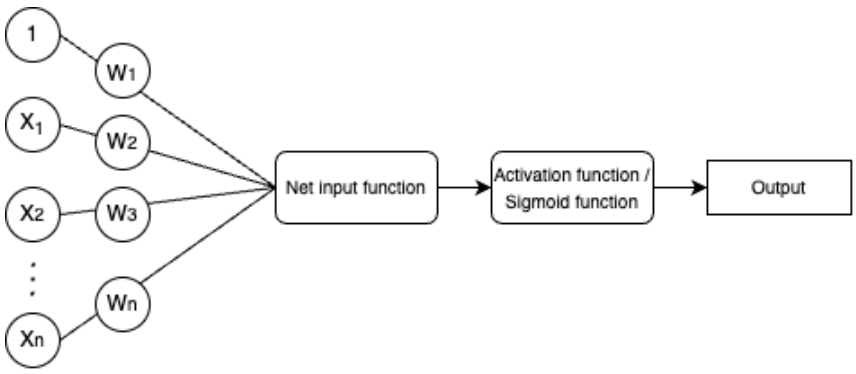


Рисунок 2.2 – Базова модель логістичної регресії

Базова модель досить проста (рис. 2.1). Функція підсумовування, яка множить кожну вхідну характеристику на відповідну вагу, додає всі ці добутки разом, а потім додає зміщення. Це також називається зваженою сумою вхідних даних [12]. Потім зважена сума пропускається через функцію активації, в даному випадку сигмоїдну функцію, яка перетворює вихід у значення між 0 і 1. Це значення інтерпретується як ймовірність того, що вхідні дані належать до позитивного класу (зазвичай позначається як «1»).

Для покращення результатів до цієї моделі потрібно додати порогову функцію. Ця функція перетворює ймовірність на мітку класу (0 або 1). Якщо ймовірність більша або дорівнює певному пороговому значенню (зазвичай 0,5), прогнозована мітка класу дорівнює 1, інакше – 0.

Крім цього слід додати істинну мітку для вхідних даних, яка використовується під час навчання для порівняння з прогнозованою міткою. А також урахування різниці між істинною міткою та передбаченою міткою – похибку. Під час навчання модель використовує цю похибку для коригування ваг і зсуву, зазвичай використовуючи метод градієнтного спуску, щоб мінімізувати цю похибку з часом.

Додатково додамо базис. Це додатковий параметр в моделі, який дозволяє змістити функцію активації вліво або вправо, що може допомогти краще підігнати модель до даних.

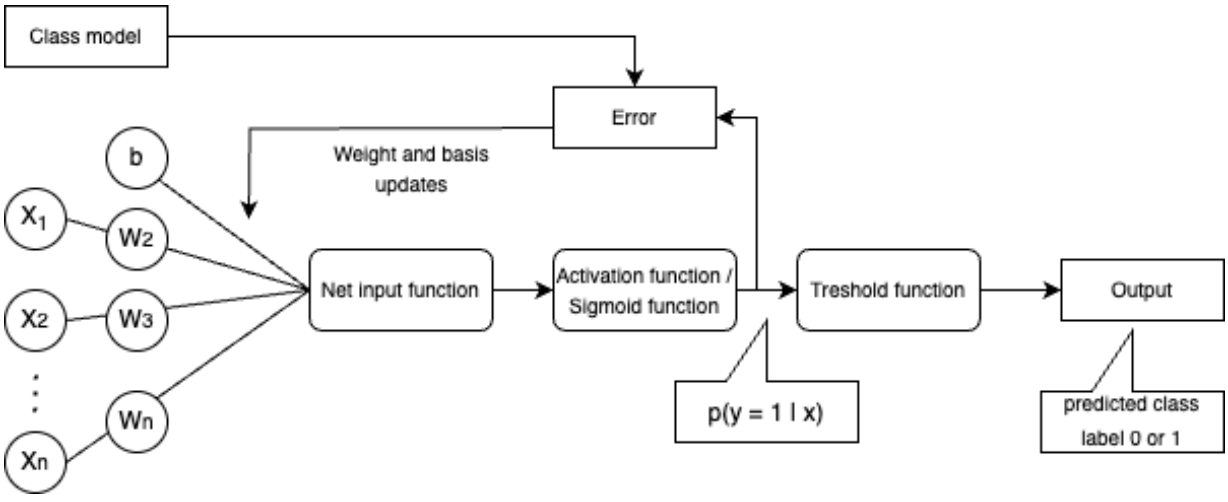


Рисунок 2.3 – Доповнена модель логістичної регресії

Для застосування логістичної регресії до прогнозування успішності проєкту, спочатку необхідно зібрати та підготувати відповідні дані. Це може включати:

– оцінки задач – час, необхідний для виконання кожної задачі;
– розподіл робочого навантаження – сумарний обсяг роботи, розподілений між членами команди;
– досвід команди – рівень кваліфікації та досвіду членів команди;
– ефективність команди – попередні результати команди у подібних проєктах.

Після збору та підготовки даних проводиться навчання моделі логістичної регресії. Після тренування моделі можна використовувати її для прогнозування успішності нових проєктів. Наприклад, якщо значення $P(Y = 1|X)$ перевищує певний поріг (наприклад, 0.5), то проєкт вважається успішним, інакше - невдалим.

Таким чином, логістична регресія може бути потужним інструментом для прогнозування успішності проєктів у сфері управління IT, а також для виявлення впливу різних факторів на цей успіх.

2.3 Адаптація системи кредитного скорінгу при розподілі задач між учасниками проєкту

Кредитний скорінг — це система, яка використовується фінансовими інститутами для оцінки кредитоспроможності позичальників. Цей процес включає аналіз кредитної історії, поточного фінансового стану, доходів, а також інших факторів, щоб визначити ймовірність того, що позичальник виконає свої кредитні зобов'язання. Результатом кредитного скорінгу є кредитний рейтинг або бал, який використовується банками та іншими кредитними організаціями для прийняття рішень про надання кредиту або встановлення умов кредитування.

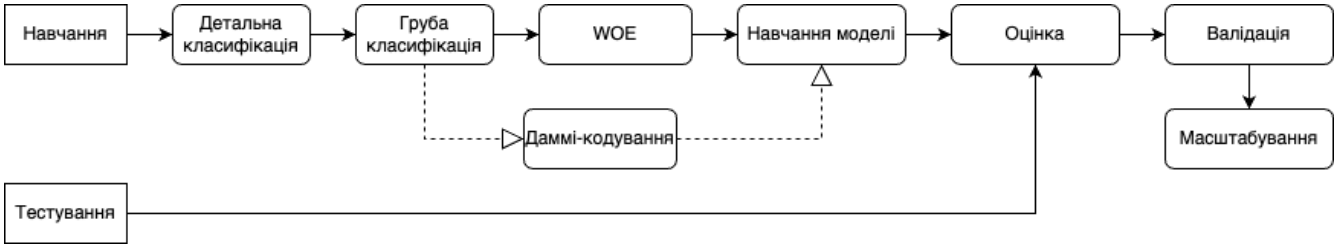


Рисунок 2.4 – Модель кредитного скорінгу

Кредитний скорінг використовує статистичні моделі для аналізу історії платежів, кредитних звітів та інших фінансових даних особи з метою визначення ймовірності своєчасного повернення кредиту. Цей процес включає збір даних, вибір релевантних ознак (наприклад, історія платежів, рівень доходу) та застосування математичних алгоритмів для розрахунку кредитного рейтингу.

Адаптація системи кредитного скорінгу до контексту управління проєктами передбачає використання подібного підходу для оцінки надійності та продуктивності учасників проєктних команд. Замість фінансових показників аналізуються професійні характеристики співробітників, такі як досвід роботи, навички, історія успішно виконаних проєктів, відгуки від колег та менеджерів.

Для створення ефективної системи кредитного скорінгу співробітників у контексті управління проєктами, важливо використовувати кількісні параметри оцінки, які можна точно виміряти.

Рисунок 2.5 – Кругова діаграма параметрів оцінки та їх значущість (%)

Діаграма (рис 1.6) показує основні параметри для розрахунку кредитного скорінгу розподілення задач.

1) Кількість успішно завершених задач – вимірює досвід співробітника та його здатність досягати результатів.

2) Середній час виконання задачі – показує ефективність роботи співробітника порівняно з плановими термінами.

3) Кількість перевищень бюджету проєктів – відображає здатність співробітника управляти ресурсами та дотримуватися фінансових рамок.

4) Кількість вирішених задач за одиницю часу – ілюструє продуктивність та швидкість роботи.

5) Індекс задоволеності клієнтів – враховує відгуки та оцінки клієнтів, що є важливим показником якості роботи.

6) Кількість помилок або дефектів у задачах – показує точність та уважність до деталей при виконанні задач.

7) Відсоток завдань, виконаних без додаткової допомоги – вимірює самостійність та здатність до вирішення проблем.

Кожен параметр має свою вагу при розрахунках (відсоток значущості). Для кожного параметра визначимо нормалізоване значення (наприклад, від 0 до 1 або від 0 до 100, залежно від методики оцінки). Потім загальний кредитний скорінг S можна розрахувати як взважену суму цих нормалізованих значень:

$$S = w_1 \cdot P_1 + w_2 \cdot P_2 + w_3 \cdot P_3 + \cdots + w_n \cdot P_n, \quad (2.7)$$

де P_i – нормалізоване значення і-го параметра,

w_i – значущість і-го параметра,

n – кількість параметрів.

Ця формула дозволяє отримати кількісну оцінку продуктивності та потенціалу співробітника, яку можна використовувати для порівняння між співробітниками та для прийняття обґрунтованих рішень щодо розподілу задач у проєктах.

Але дана формула має одну проблему, вона не враховує додаткові фактори. Серед таких факторів можуть виступати наприклад наступні фактори:

– Коефіцієнт стресостійкості ($C_{\text{стресостійкость}}$) – визначає, як ефективно співробітник працює під тиском та в стресових ситуаціях.

– Коефіцієнт навчальної активності ($C_{\text{навчання}}$) – визначає, як ефективно співробітник засвоює нові знання.

Додамо також множник важливості для кожного параметра, що дозволить більш гнучко налаштовувати вплив кожного з них на загальний скорінг.

$$S = w_1 \cdot P_1 + w_2 \cdot P_2 + w_3 \cdot P_3 + \cdots + w_n \cdot P_n + \sum_{i=1}^{m} C_i * I_i, \qquad (2.8)$$

де: S – загальний кредитний скорінг співробітника,

P_i – нормалізоване значення і-го кількісного параметра,

w_i – вага, що відображає важливість і-го параметра в загальній оцінці,

C_i – коефіцієнт, що враховує додаткові фактори, які можуть впливати на продуктивність співробітника,

I_i – індекс або показник для і-го додаткового фактора,

n – кількість основних кількісних параметрів,

m – кількість додаткових факторів.

Розширена формула дозволяє більш точно врахувати різноманітні аспекти роботи співробітника та його внесок у проєкти, забезпечуючи комплексну та багатогранну оцінку його продуктивності та потенціалу.

На блок-схемі (рис. 2.5) зображено процес присвоєння оцінки кандидату для назначення на задачу від моменту збору інформації про задачу до прийняття рішення щодо її виконання, включаючи етапи оцінки, схвалення та планування. Перш за все на вхід потрапляють дані про задачу для виконання якої потрібно обрати учасника. Після цього проходить процес оцінки учасника. В разі, якщо оцінка має невелике відхилення – вона буде додатково обчислена, і процес перевірки відбудеться знову. У разі великого відхилення, учасника буде

відкинуто. Якщо оцінка буде достатньою – учасника буде запропоновано на виконання поставленої задачі.

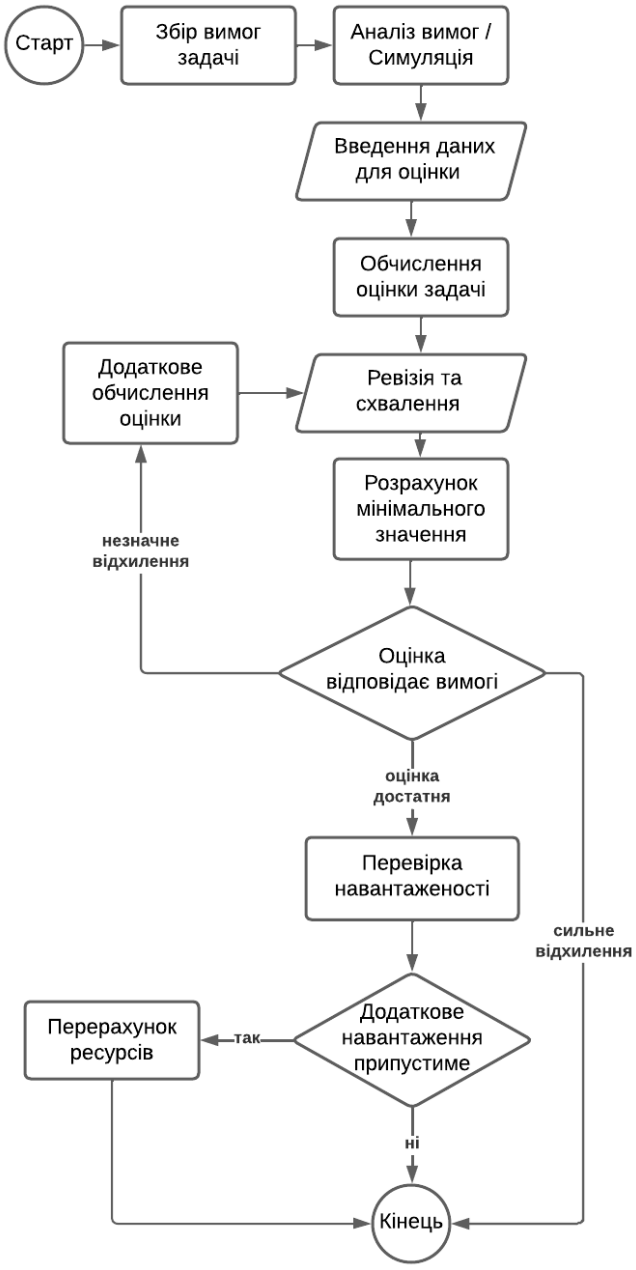


Рисунок 2.5 – Блок-схема застосування кредитного скорінгу

Для більш адаптивного налаштування очікувана оцінка буде передаватися в якості вхідного параметра. Таким чином для різних задач можна вказувати різні мінімально необхідні оцінки для рекомендації кандидатів. Наприклад в тих випадках коли не вдалось виявити учасника при встановлені занадто високого мінімального порогу оцінки.

Висновки до розділу 2

У другому розділі були розроблені проєктні рішення для розроблюваної системи, відповідно до специфікації вимог до програмного забезпечення виявленої у попередньому розділі.

Було проведено моделювання цільових функцій системи. Серед цільових функцій впровадження автоматичного оцінювання задач, прогнозування успішності виконання проєкту, а також адаптація системи кредитного скорінгу для вирішення задачі розподілення задач між учасниками проєкту. Для кожної задачі підібрані найбільш підходящі алгоритми машинного навчання.

Результатом проведеної у розділі є математичні та імітаційні моделі цільових функцій системи.

3 ПРОЄКТУВАННЯ АРХУТЕКТУРИ СИСТЕМИ АВТОМАТИЗАЦІЇ УПРАВЛІННЯ РОБОЧОГО НАВАНТАЖЕННЯ

3.1 Побудова схеми взаємодії користувачів із системою

При проєктуванні системи, перш за все потрібно подбати про зручність використання. Основою для системи є навчені моделі. Модель – це якийсь виконуваний код, якому можна віддати дані в певному форматі (подати на вхід) і отримати відповідь (отримати вихід). Як правило, це масиви чисел: масиви numpy, тензори, спарс-масиви та інші.

Користувачам моделі недостатньо знання про те, який масив їм потрібно віддати вашій моделі машинного навчання. Їм потрібен інтерфейс взаємодії з нею: сайт, телеграм-бот, мобільний застосунок. Тому постає питання – як надати користувачам доступ до використання моделей?

Пряме використання моделі

Найпростіша й очевидна на перший погляд архітектура. Для кожного запиту користувача викликається передбачення моделі.

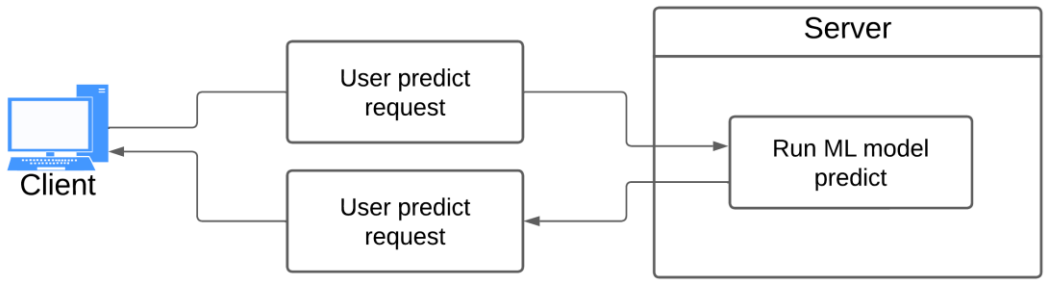


Рисунок 3.1– Пряме використання моделі з клієнту

Моделі машинного навчання, в основному, зосереджені на операціях, що залежать від процесорного часу (CPU bound операції), під час фази передбачення. Це означає, що кожен індивідуальний запит користувача ініціює процес передбачення моделі, який відбувається незалежно від інших запитів, що призводить до паралельних передбачень. Оскільки кожен такий запит на передбачення використовує власні ресурси, загальні ресурси, які модель може

використовувати, будуть множитися для кожного окремого запиту. Це може призвести до того, що ресурси швидко закінчаться, що, в свою чергу, може призвести до того, що застосунок стане нестабільним або навіть зависне.

Ця архітектура найшвидша для передбачення. Вона підходить для випадків: коли модель проста, коли модель швидка або клієнти надсилають мало запитів до неї.

Використання черги та воркеру

У випадку коли ми обмежені ресурсами та не має можливості запустити кілька паралельних передбачень моделі, існує інше рішення – можна використовувати систему черг. Використання систему черг дозволяє моделі машинного навчання обробляти лише одне передбачення (операцію, обмежену процесором) в один конкретний момент часу. Це вирішує проблему з обмеженнями ресурсів, не втрачаючи здатності обробляти запити користувачів. Ці запити будуть оброблені в строгому порядку, відповідно до їхнього надходження, гарантуючи, що кожен користувач отримає відповідь.

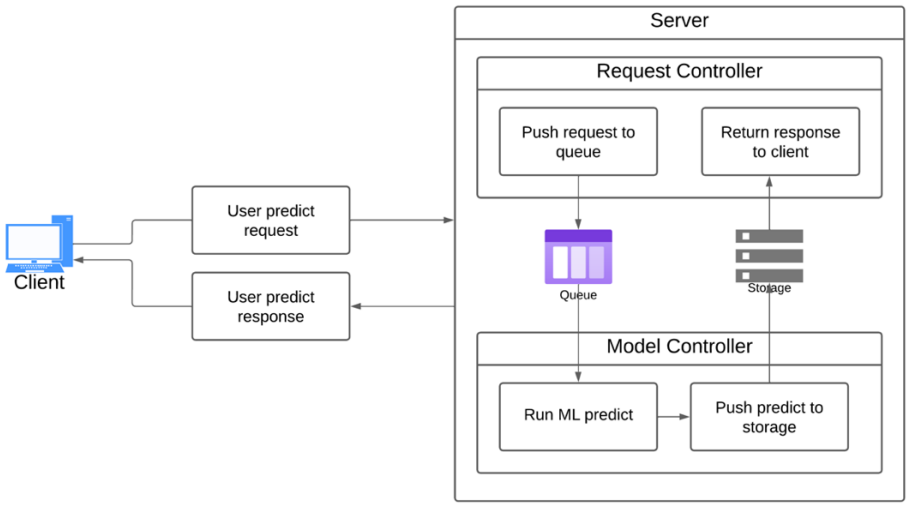


Рисунок 3.2– Використання моделі з чергою та воркером

Ця архітектура підтримує довгі передбачення (long predict) – коли модель машинного навчання обчислює відповідь довше, ніж клієнт може чекати. Клієнт може надіслати запит на передбачення, отримати у відповіді predict_id (uuid) і повернутися за відповіддю пізніше.

Батчинг даних із запитів

Доступ до моделі через чергу не завжди буває ефективним. Архітектура батчингу даних із запитів забезпечує такий самий доступ до моделі через чергу, як архітектура черги ти воркера. Але в модель потрапляють на передбачення вже не один запит у момент часу, а кілька.

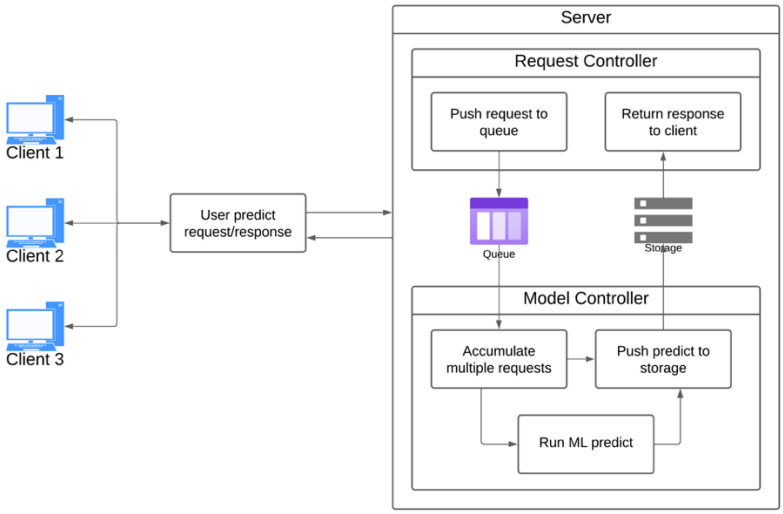


Рисунок 3.3 – Використання батчингу даних

Воркер із моделлю читає чергу запитів, поки не набере певну кількість даних для передбачення (батч). Зібрану пачку даних з декількох запитів відправляє в модель, а результати розкладає в сховище результатів за ключем predict_id. Для цього можна використати пам’ять або залучити сторонні сховища.

3.2 Технології та методи для розробки застосунку

Python

Коли мова заходить про вибір мови програмування для машинного навчання, майже завжди вибір однозначний – мова програмування Python. Вона є оптимальним вибором в даній ситуації, оскільки широко застосовується в області організації машинного навчання. Це великою мірою обумовлено легкістю кодування на цій мові та швидкістю виконання задач. У порівнянні з такими мовами програмування, як C++ або Java, для реалізації схожої функціональності на Python необхідно написати значно меншу кількість рядків коду.

Крім того, мова Python відрізняється простотою своєї синтаксичної структури і наявністю динамічної типізації, що спрощує розробку програм. При роботі з нейронними мережами необхідно проводити велику кількість експериментів, таких як визначення оптимальної архітектури нейронної мережі для конкретної задачі, оцінювання параметрів навчання, та інше.

Враховуючи, що в якості основної мови програмування для розробки застосунку обрано Python, то стек та засоби машинного навчання також будуть реалізовані на цій мові. До основних фреймворків і бібліотек, що застосовуються у машинному навчанні, а також будуть використані у даній роботі належать наступні:

JAX

Бібліотека, розроблена Google, яка дозволяє виконувати точні, швидкі та ефективні наукові обчислення. JAX підтримує автоматичне диференціювання, що робить її ідеальною для розробки складних алгоритмів машинного навчання та глибокого навчання.

Використання JAX для розробки алгоритмів оптимізації, які динамічно розподіляють задачі між членами команди на основі їхніх навичок, завантаження та історії успішності. Моделі машинного навчання, треновані з використанням JAX, можуть швидко адаптуватися до змін у проєкті та оптимізувати процес розподілу задач для підвищення загальної ефективності.

Використання JAX як основного фреймворку надає значні переваги у швидкості розробки та виконання моделей, ефективності використання ресурсів обчислювальних пристроїв та гнучкості у реалізації складних математичних операцій та алгоритмів.

NumPy

NumPy – це головна бібліотека для програмування на мові Python, яка значно полегшує роботу із векторами та матрицями. Вона надає великий набір математичних функцій, що ефективно працюють з багатовимірними масивами. Бібліотека містить готові методи для операцій різних типів, включаючи числові,

статистичні та алгебраїчні операції. Завдяки широкому спектру функціональних можливостей NumPy, розробники можуть легко проводити такі операції, як створення, зміни форми, множення і розрахунок детермінанту матриць. Крім того, вони можуть вирішувати лінійні рівняння, проводити сингулярне розкладання, а також виконувати безліч інших завдань пов'язаних з обробкою та аналізом даних.

Pandas

Pandas є надзвичайно потужною бібліотекою, що забезпечує розробників високоякісними структурами даних і інструментами для їх аналізу. Ця бібліотека виявляється особливо корисною при роботі з неповними, невпорядкованими і немаркованими даними, які як правило, зустрічаються у реальному світі.

Однією з найбільших переваг Pandas є те, що вона дозволяє замінити досить складні операції з даними однією чи двома командами. Бібліотека включає велику кількість готових методів для групування, фільтрації та об'єднання даних, що значно спрощує процес роботи з ними.

Окрім того, Pandas має вбудовану можливість розпізнавання різноманітних джерел даних. Це означає, що ви можете витягувати дані безпосередньо з файлів, без необхідності спочатку організовувати базу даних. Ця особливість значно спрощує процес роботи з даними.

Особливо важливо відзначити, що Pandas має можливість об'єднувати таблиці аналогічно до SQL JOIN. Це надає вам додаткову гнучкість при роботі з різними наборами даних.

Нарешті, однією з найважливіших переваг бібліотеки Pandas є її швидкість. Незалежно від розміру ваших даних, ви можете бути впевнені, що Pandas обробить їх ефективно і швидко.

Scikit-learn

Це основна бібліотека Python для машинного навчання, яка надає широкий набір інструментів для класифікації, регресії, кластеризації та зниження розмірності. Scikit-learn відомий своєю простотою використання, документацією та ефективністю при роботі з середніми наборами даних.

FastAPI

Ще одним не мало важливим аспектом системи є реалізація взаємодії клієнтської та серверної частини застосунку за допомогою API.

FastAPI – це сучасний, швидкий (високопродуктивний) веб-фреймворк для створення API з Python 3.6+ на основі стандартних Python типів. Цей фреймворк забезпечує автоматичну генерацію документації за допомогою Swagger UI та ReDoc, підтримує асинхронне програмування та надає вбудовані інструменти для валідації даних.

FastAPI дозволяє швидко розробляти високопродуктивні API завдяки простоті синтаксису та автоматичній валідації запитів/відповідей на основі Python типів. Асинхронна підтримка фреймворку забезпечує високу швидкість обробки запитів, що критично важливо для систем з великою кількістю користувачів та запитів.

3.3 Проєктування бази даних

Великим проєктам машинного навчання потрібна СУБД, яка може обробити великий обсяг даних, надаючи можливість продуктивно масштабувати їх. У цьому випадку, бази даних, такі як PostgreSQL і MySQL, вирішують ці масштабні проблеми зберігання та обробки великих обсягів даних.

PostgreSQL має вбудовану підтримку JSON/JSONB, що дозволяє зберігати вихідні дані моделей машинного навчання безпосередньо в базі даних. Це може бути корисно для зберігання детальних результатів прогнозів, включаючи імовірності, класифікації або будь-які інші метадані, пов’язані з прогнозом моделі.

Ще однією можливістю PostgreSQL є віконні функції. Віконна функція виконує обчислення для набору рядків таблиці, які якимось чином пов’язані з поточним рядком. Це можна порівняти з типом обчислень, які можна виконати за допомогою агрегатної функції. Вони можуть використовуватися, наприклад, для аналізу робочого навантаження на команду.

```sql
SELECT UserID, SUM(EstimatedTime) OVER (PARTITION BY UserID) AS TotalWorkload
FROM Tasks
WHERE Status = 'in_progress';
```

Рисунок 3.4 – Приклад віконної функції

Цей запит дозволяє визначити загальне навантаження на кожного співробітника, спираючись на оцінку часу для активних задач, що сприяє рівномірному розподілу роботи.

Додаткова та не мешн потужна можливість PostgreSQL – використання триггерів. Тригери використовують для того, щоб сказати рушію PostgreSQL виконати частину коду при настанні певної події. Виходить свого роду каталізатор змін, спусковий гачок, який запускає ланцюг подій.

Тригер має бути пов’язаний із зазначеною таблицею, поданням (псевдотаблицею) або зовнішньою таблицею. Він запускає свою частину коду тільки під час виконання операцій із цією сутністю – INSERT, UPDATE, DELETE або TRUNCATE. Залежно від вимог ми можемо запускати тригер до, після або замість події/операції.

```sql
CREATE OR REPLACE FUNCTION log_task_changes()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO ChangeHistory(TaskID, ChangeDescription, ChangeDate)
    VALUES (NEW.TaskID, 'Task updated', NOW());
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TaskUpdateTrigger
AFTER UPDATE ON Tasks
FOR EACH ROW EXECUTE FUNCTION log_task_changes();
```

Рисунок 3.5 – Приклад створення тригеру на оновлення задач

В даному прикладі відбувається автоматичне створення запису в історії змін при кожному оновленні задачі. Це дозволяє відстежувати всі зміни, спрощуючи управління проєктом та аналіз прогресу.

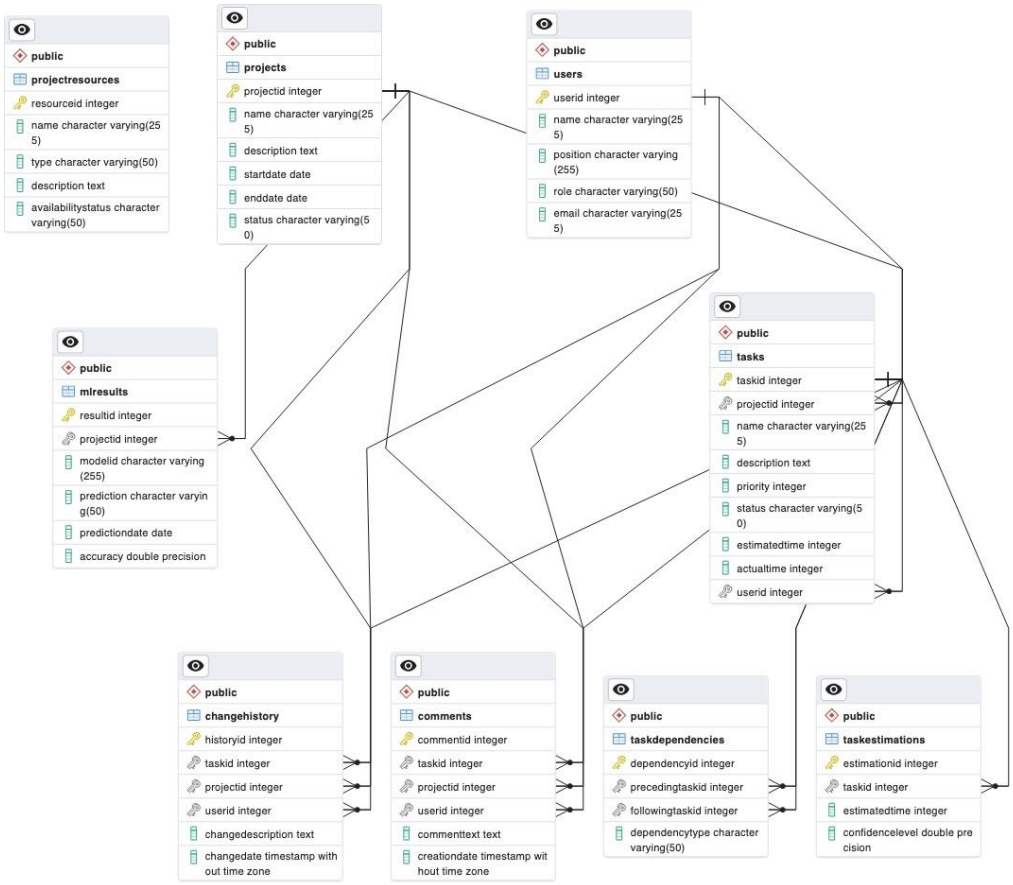


Рисунок 3.6 – Фізична модель бази даних

На фізичній моделі (рис. 3.6) зображені усі сутності та їх атрибути, а також зв’язки між ними у контексті розроблюваної системи. Вона включає кілька ключових таблиць, кожна з яких відіграє важливу роль в організації та аналізі даних, що стосуються проєктів, задач, команди та результатів машинного навчання:

– Проєкти (Projects):

Зберігає інформацію про кожен проєкт, включаючи його унікальний ідентифікатор, назву, опис, дати початку та закінчення, а також статус. Ця таблиця є основою для трекінгу всіх проєктів у системі.

– Задачі (Tasks):

Включає деталі про задачі в рамках проєктів, такі як назва, опис, пріоритет, статус, оцінку часу та фактичний час виконання. Задачі зв’язані з конкретними проєктами та відповідальними виконавцями.

– Користувачі (Users):

Містить інформацію про учасників проєктів, включаючи їх ролі, позиції та контактні дані. Це дозволяє управляти командами та призначати задачі конкретним користувачам.

– Оцінки задач (TaskEstimations):

Зберігає оцінки моделей машинного навчання для кожної задачі, включаючи прогнозований час виконання та рівень впевненості цих оцінок. Ця таблиця важлива для аналізу та оптимізації розподілу робочого навантаження.

– Результати машинного навчання (MLResults):

Відображає результати прогнозів машинного навчання на рівні проєктів, включаючи ідентифікатор моделі, прогноз успішності проєкту, дату прогнозу та точність. Це допомагає оцінити потенційний успіх проєктів.

– Ресурси проєкту (ProjectResources):

Описує доступні ресурси, такі як обладнання та програмне забезпечення, їх тип, статус доступності та іншу важливу інформацію. Управління ресурсами критично важливе для планування та виконання проєктів.

– Залежності між задачами (TaskDependencies):

Моделює залежності між задачами, що дозволяє враховувати послідовність їх виконання. Це критично для ефективного планування та управління проєктами.

– Коментарі та зворотній зв’язок (Comments):

Забезпечує можливість зберігати коментарі та зворотній зв’язок від команди або клієнтів по задачам або проєктам, що сприяє кращій комунікації та управлінню змінами.

– Історія змін задач/проєктів (ChangeHistory):

Відстежує всі зміни, що відбуваються з задачами або проєктами, забезпечуючи повний аудит та можливість аналізу історії проєкту.

Ці таблиці разом формують комплексну структуру бази даних, яка забезпечує зберігання всієї необхідної інформації для ефективного управління IT-проєктами, оптимізації розподілу робочого навантаження та використання

передових методів машинного навчання для покращення процесів прийняття рішень.

3.4 Технології реалізації клієнтської частини системи

Remix.js

В якості основи для клієнтської частини застосунку обрано фреймфорк Remix.js. Remix.js – це сучасний фреймворк для створення веб-додатків, що використовує React. Він призначений для спрощення процесу розробки, забезпечення високої продуктивності додатків і поліпшення користувацького досвіду. Remix.js дає змогу розробникам використовувати React на стороні сервера для попереднього відтворення сторінок, що сприяє швидшому завантаженню та інтерактивності сторінок, а також поліпшенню SEO.

Переваги Remix.js:

– Покращений користувацький досвід і продуктивність – завдяки серверному рендерингу та оптимізації завантаження ресурсів, сторінки завантажуються швидше, що сприяє поліпшенню взаємодії користувача з додатком.

– Оптимізація для SEO – серверний рендеринг покращує індексацію контенту пошуковими системами, оскільки сторінки вже попередньо відмальовані під час їх відвідування пошуковими роботами.

– Спрощене управління даними – Remix.js надає зручні абстракції для роботи з даними на стороні сервера і клієнта, полегшуючи завантаження і мутацію даних без необхідності додавання додаткового коду для управління станом.

– Підвищена безпека – фреймворк заохочує розробку з урахуванням найкращих практик безпеки, наприклад, автоматично опрацьовуючи багато аспектів безпеки, пов’язаних із передачею даних між клієнтом і сервером.

Tailwind CSS

Tailwind CSS є одним з CSS-фреймворків, що призначені для швидкого створення користувацьких інтерфейсів. Цей фреймворк відрізняється від інших популярних бібліотек та фреймворків, що базуються на CSS, таких як Bootstrap, Bulma та Foundation. Однією з особливостей Tailwind CSS є відсутність теми за замовчуванням або вбудованих компонентів інтерфейсу, що зазвичай присутні в інших CSS-фреймворках. Натомість Tailwind CSS пропонує багатий набір попередньо розроблених віджетів, які можна використовувати для створення сайту з нуля. Ці віджети дозволяють користувачам гнучко та ефективно налаштовувати дизайн своїх вебзастосунків, що робить Tailwind CSS привабливим вибором для розробників.

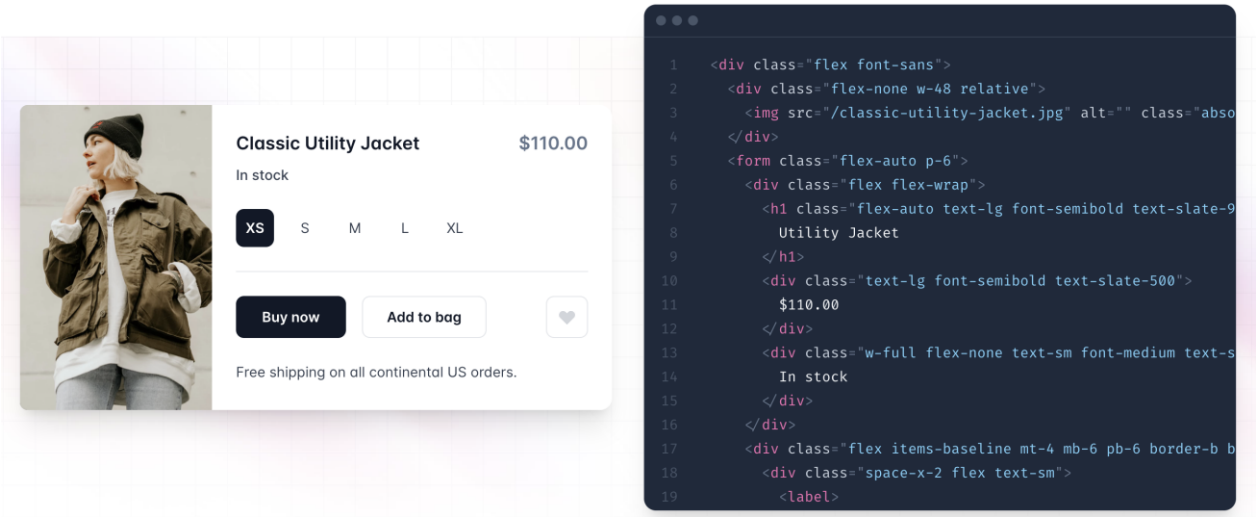


Рисунок 3.7 – Приклад компоненту створеного за допомогою Tailwind

Refine

Refine (раніше відомий як React-Admin) являє собою фреймворк для швидкого розроблення адміністративних інтерфейсів, що працює поверх React. Він призначений для спрощення процесу створення адміністративних панелей, CRM-систем, панелей управління контентом (CMS) та інших видів веб-додатків, що вимагають управління даними.

Фреймворк полегшує інтеграцію з різними типами API (REST, GraphQL, SOAP та іншими), надаючи абстракції, що спрощують запити до даних і обробку відповідей.

Завдяки модульній архітектурі та використанню React, Refine дає змогу розробникам легко налаштовувати й розширювати функціональність додатків, додавати власні компоненти й інтегруватися з іншими бібліотеками та сервісами.

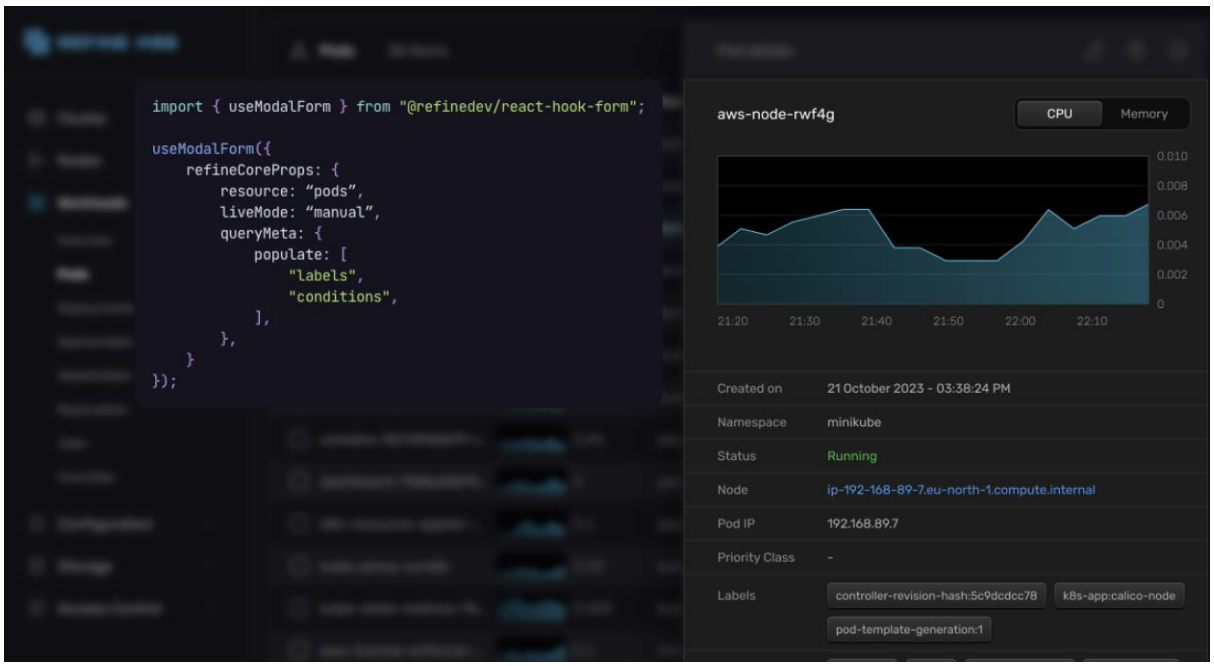


Рисунок 3.8 – Приклад компоненту Refine

Після того, як основний стек технологій для реалізації клієнтської частини системи вибрано, потрібно розробити структуру проєкту. Оскільки в якості базового фреймворку використовується Remix, базова файлова структура буде запозичена саме у нього, але з модифікацією урахувуючи вимоги до розроблюваної системи.

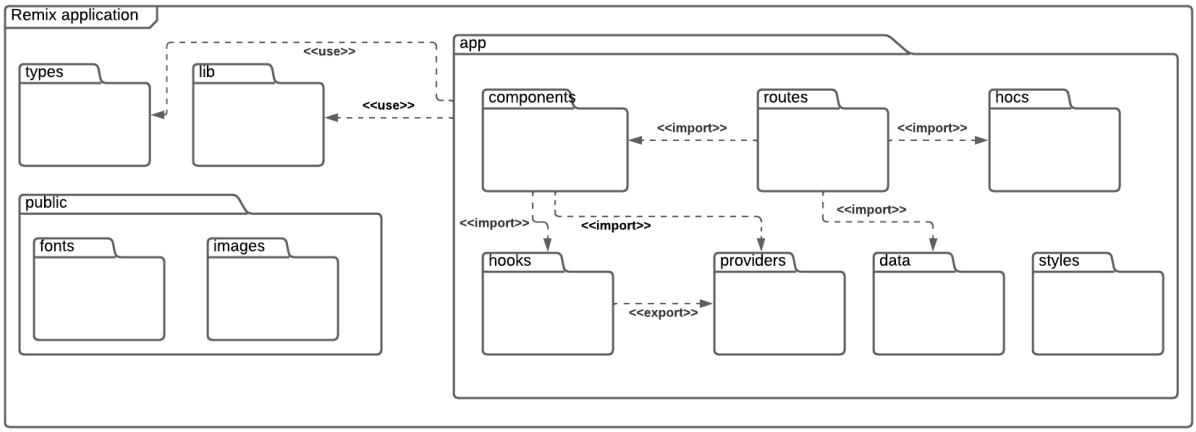


Рисунок 3.9 – Діаграма пакетів вебзастосунку

На діаграмі пакетів (рис. 3.9), відображаються основні пакети для розроблюваного вебзастосунку. На діаграмі враховані особливості архітектури Remix. Структура включає використання роутінгу, управління станом, використання API, а також декомпозицію шаблонів для можливості повторного використання окремих компонентів.

3.5 Загальний огляд архітектури системи

Останнім кроком проєктування архітектури застосунку є створення загально є створення загальної схеми архітектури проєкту. Потрібно візуально відобразити, як саме різні частини застосунку будуть взаємодіяти між собою (рис. 3.8).

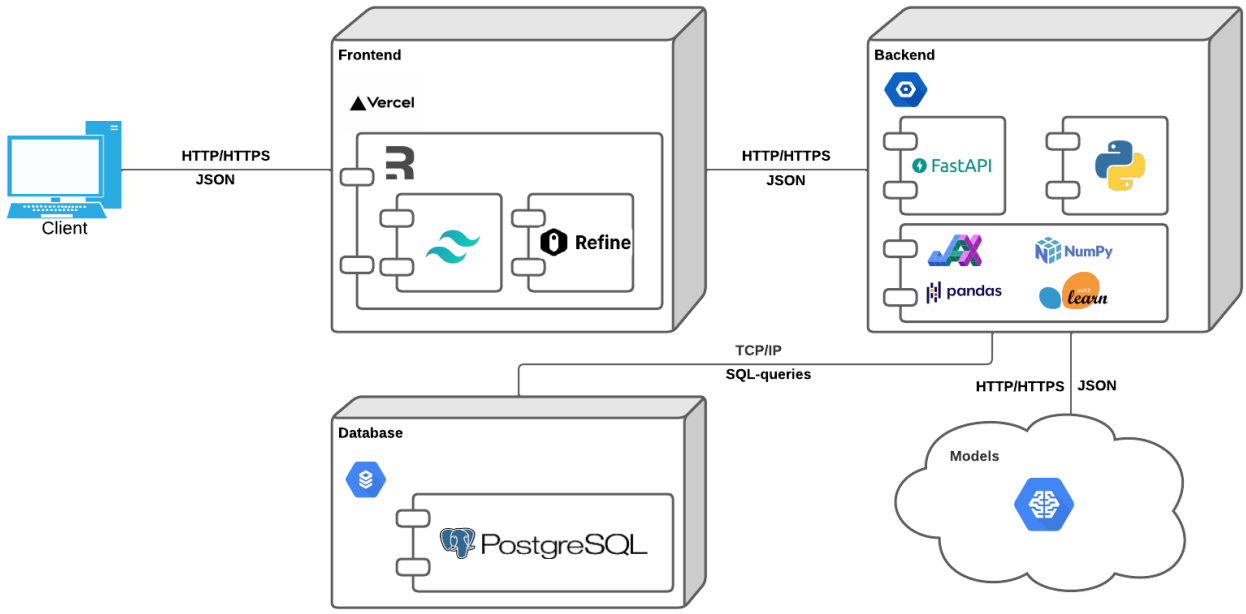


Рисунок 3.9 – Загальна архітектура системи

На діаграмі розгортання (рис. 3.9) зображено основні компоненти системи. Фронтенд, розгорнутий на платформі Vercel, забезпечує користувачам інтуїтивно зрозумілий інтерфейс для управління проєктами, задачами та перегляду аналітики. Бекенд, який виступає як середник між фронтендом, базою даних та моделями машинного навчання, обробляє бізнес-логіку застосунку, включаючи запити на створення, оновлення та видалення проєктів та задач.

База даних PostgreSQL, розгорнута на сервісі Google Cloud SQL, забезпечує надійне сховище для всіх даних системи. Це включає інформацію про проєкти, задачі, користувачів та результати аналізу моделей машинного навчання. Моделі машинного навчання, розроблені за допомогою JAX, розгорнуті як на окремих високопродуктивних обчислювальних ресурсах, а саме Google AI Platform Models.

Ця архітектура забезпечує гнучке розгортання, високу доступність та масштабованість системи, дозволяючи легко адаптуватися до змінних обсягів завантаження та вимог користувачів.

Висновки до розділу 3

У третьому розділі було детально спроєктовано архітектуру застосунку. Виявлено і обґрунтовано вибір типу архітектури, визначено основні складові частини системи, проаналізовано ключові компоненти, та підібрані ефективні інструменти для розробки на різних етапах.

В якості основної мови програмування було обрано Python, а основного фреймворку для машинного навчання JAX. Окрім цього, також було обрано набір бібліотек та інструментів для машинного навчання.

В якості СКБД було обрано PostgreSQL. Спроєктовано фізичну модель бази даних та детально описано кожну таблицю.

Додатково було обрано набір інструментів та фреймворків для розробки клієнтської частини застосунку.

Проєктування архітектури системи є ключовим етапом у розробці програмного забезпечення. Його використання дозволяє значно спростити процес розробки, чітко розуміти кінцевий результат і функціонування всіх функцій. Архітектура проєкту сприяє створенню якісного програмного забезпечення і зменшує витрати на його подальше обслуговування та утримання.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗАЦІЇ УПРАВЛІННЯ РОБОЧОГО НАВАНТАЖЕННЯ

4.1 Налаштування середовища розробки

Перед тим як перейти до безпосередньої програмної реалізації застосунку, невід'ємним етапом є налаштування середовища розробки. Цей процес включає в себе ряд важливих кроків, які повинні бути виконані для забезпечення гладкого і ефективного процесу розробки. Деякі технічні деталі, такі як встановлення спеціалізованого програмного забезпечення для розробки та налаштування залежностей для застосунку, в даному розділі упущено. Причина цього полягає в тому, що ці налаштування є досить базовими і, як правило, не потребують додаткового опису.

4.1.1 Конфігурація контейнерізації Docker

Docker спрощує розгортання застосунків. Контейнери можуть бути швидко створені, зупинені, видалені або переміщені між середовищами. Це також полегшує горизонтальне масштабування застосунків, оскільки додавання додаткових інстансів сервісу стає тривіальним завданням.

Docker ідеально підходить для розробки мікросервісних архітектур, дозволяючи кожному сервісу бути упакованим і розгорнутим незалежно. Це допомагає в розбитті монолітних застосунків на менші, легко управляємі частини.

Оскільки ми маємо планувати розгортання застосунку на різних сервісах хмарних обчислень (Google Cloud та Vercel), інтегрувати базу даних PostgreSQL і моделі машинного навчання з Google AI, а також використовувати різні технології для розробки фронтенду та бекенду, Docker може бути ідеальним інструментом для створення середовища розробки, що може бути легко перенесене та масштабоване.

Для серверної частини створено файл Dockerfile.backend (рис. 4.1). Цей Dockerfile починає з базового образу Python версії 3.9-slim, створює робочу директорію /app, встановлює змінну середовища PYTHONPATH, копіює файл

requirements.txt для установки залежностей, виконує оновлення та установку пакетів за допомогою pip, копіює файли додатку у вказану робочу директорію, викладає порт 8000, а після цього запускає сервер FastAPI за допомогою UVicorn на адресі 0.0.0.0 і порту 8000 з автоматичним перезавантаженням при зміні файлів.

```
FROM python:3.9-slim

RUN mkdir -p /app
WORKDIR /app
ENV PYTHONPATH=/app
COPY ./requirements.txt ./requirements.txt
RUN pip install --upgrade pip \
    && pip install -r requirements.txt
COPY ./app /app
EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--reload", "--port", "8000"]
```

Рисунок 4.1 – Вміст файлу Dockerfile.backend

Для клієнтської частини застосунку також створено окремий Dockerfile (рис. 4.2). Даний файл використовує базовий образ Node.js версії 16 на базі Alpine Linux. Після цього встановлюється глобально інструмент командного рядка Remix за допомогою npm. Далі, файли фронтенду копіюються у внутрішню директорію /app в контейнері. Потім встановлюється робоча директорія /app. Нарешті, запускається команда remix dev, яка запускає сервер фронтенду з режимом розробки.

```
FROM node:16-alpine

RUN npm install -g @remix-run/cli
COPY ./frontend /app
WORKDIR /app
CMD ["remix", "dev"]
```

Рисунок 4.2 – Вміст файлу Dockerfile.frontend

Після цього можна додати docker-config.yaml файл який буде відправною точною застосунку. Файл містить конфігурацію для трьох сервісів:

– **backend**. Спочатку збирається контейнер за допомогою Dockerfile.backend, після чого встановлюються порти для з'єднання із зовнішніми додатками. Директива volumes монтує локальний каталог ./app до /app у контейнері, щоб забезпечити гнучкість у розробці. environment встановлює змінні середовища, необхідні для підключення до бази даних PostgreSQL, а depends_on гарантує, що контейнер db буде запущений перед запуском контейнера web. Також використовується файл .env для завантаження змінних середовища.

– **frontend**. Контейнер фронтенду також збирається за допомогою власного Dockerfile.frontend і встановлює порти для з'єднання.

– **db**. Використовується образ PostgreSQL версії 13, налаштований змінними середовища, які завантажуються з файлу .env. Додатково, створюється том postgres_data, який монтується у директорію /var/lib/postgresql/data, забезпечуючи збереження даних між перезапусками контейнера.

4.1.2 Підготовка бази даних до роботи

Ще одним етапом налаштування є підготовка бази даних. Правильне налаштування бази даних дозволяє забезпечити постійний доступ до даних вашого застосунку. Налаштування параметрів реплікації, резервного копіювання та відновлення допоможе уникнути втрати даних у разі виникнення проблем.

Для початку потрібно створити Google Cloud та включити сервіс Cloud SQL. Після цього можна створити базу даних за допомогою інтерфейсу керування Google Cloud або скористатися інструментами командного рядка (рис. 4.3).

```
den_bechka@cloudshell:~ (pm-forcaster)$ gcloud sql instances create pm-forcaster-db --database-version=POSTGRES_13 --tier=db-custom-1-3840 --
region=us-central1
Creating Cloud SQL instance for POSTGRES_13...done.

Created [https://sqladmin.googleapis.com/sql/v1beta4/projects/pm-forcaster/instances/pm-forcaster-db].
NAME: pm-forcaster-db
DATABASE_VERSION: POSTGRES_13
LOCATION: us-central1-b
TIER: db-custom-1-3840
PRIMARY_ADDRESS: 34.172.167.98
PRIVATE_ADDRESS: -
STATUS: RUNNABLE
den_bechka@cloudshell:~ (pm-forcaster)$
```

Рисунок 4.3 – Створення екземпляру PostgreSQL в Google Cloud за допомогою командного рядка

Ця команда створює новий екземпляр PostgreSQL у Google Cloud SQL з використанням типу тарифування «custom», яке дозволяє налаштувати власні ресурси для бази даних. Параметр **--database-version=POSTGRES_13** вказує на використання PostgreSQL версії 13. Параметр **--tier=db-custom-1-3840** вказує на використання рівня «custom» з одним ядром і обсягом пам'яті 3,840 MB. Нарешті, параметр **--region=us-central1** визначає регіон, де буде розгорнута база даних.

Наступним кроком потрібно створити нову базу даних. Для цього можна використати команду **gcloud sql databases create.**

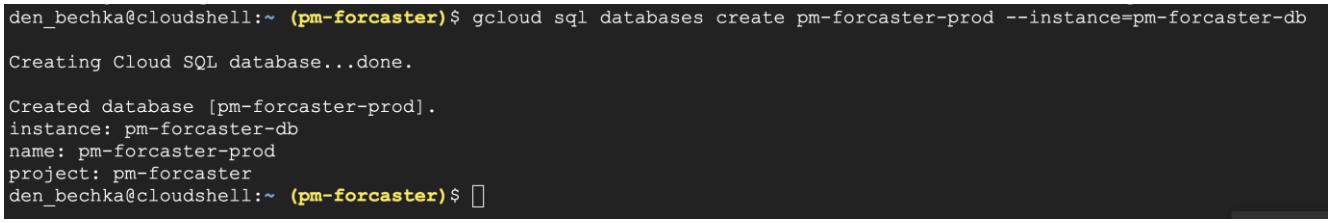


Рисунок 4.4 – Створення бази даних в Google Cloud за допомогою командного рядка

Ця команда gcloud sql databases create використовується для створення нової бази даних з ім'ям **pm-forcaster-prod** у вказаному екземплярі бази даних **pm-forcaster-db** в Google Cloud SQL. Після виклику цієї команди нова база даних буде створена, якщо обліковий запис, з якого вона викликається, має відповідні дозволи.

Далі потрібно додати до файлу .env змінні оточення для підключення до бази даних (рис 4.5).

```
# Database
POSTGRESS_USER=postgres
POSTGRES_PASSWORD=admin
POSTGRES_CONNECTION=pm-forcaster:us-central1:pm-forcaster-db
POSTGRES_DB=pm-forcaster-prod
POSTGRES_HOST=34.172.167.98
POSTGRES_PORT=5432
```

Рисунок 4.5 – Додавання змінних оточення для підключення до Google Cloud

Після цього потрібно перевірити чи все правильно налаштовано. Для цього можна під’єднатися до бази даних, та перевірити активні з’єднання.

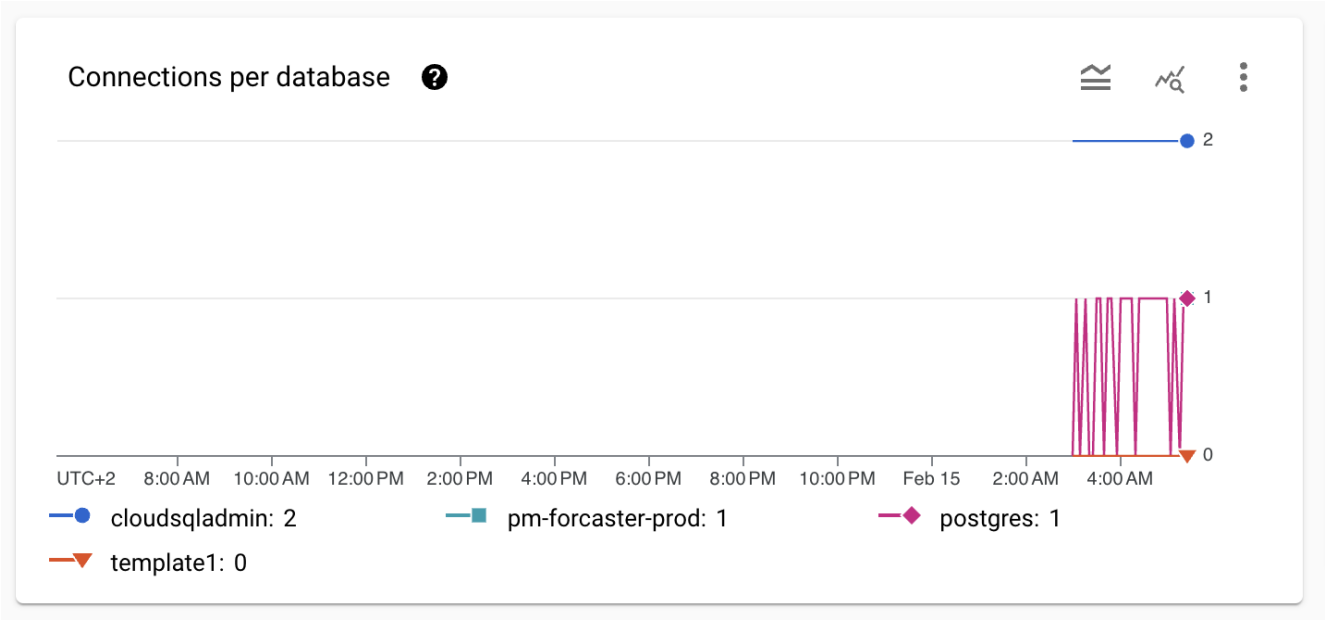


Рисунок 4.6 – Перевірка активних з’єднань в Google Cloud

На діаграмі (рис. 4.6) можна побачити, що база даних доступна та має одне активне підключення.

4.2 Розробка моделей машинного навчання

Наступним етапом є розробка моделей машинного навчання. Цей процес включатиме збір та підготовку відповідних даних, реалізація змодельованих алгоритмів та функцій системи відповідно до специфікації вимог створеної у минулих розділах, а також безпосереднє навчання та оцінку моделей.

4.2.1 Збір та підготовка даних

Перш ніж навчати модель, потрібно зрозуміти характеристики та формат даних, що є важливими для задачі. Це включає в себе збір даних з різних джерел, вирішення проблем зі структурою даних, очищення від аномалій та пропущених значень, кодування категоріальних змінних, а також можливо, стандартизацію та масштабування ознак. Важливо виконати цей етап уважно, оскільки якість даних має прямий вплив на ефективність та надійність моделей машинного навчання.

З міркувань безпеки та конфіденційності інформації, для навчання доцільно використовувати фейкові датасети. Для цієї цілі можна використати інструменти на основі ШІ для генерації фейкових наборів даних. В контексті дослідження використовується сервіс Mockaroo. Це інтернет-сервіс для генерації випадкових даних, який широко використовується для тестування програмного забезпечення, заповнення тестових баз даних або просто для створення випадкових наборів даних для потреб розробки. Mockaroo дозволяє користувачам налаштовувати різні типи даних (такі як імена, адреси, номери телефонів, адреси електронної пошти тощо), а також вказувати кількість записів, які потрібно згенерувати. Цей інструмент дуже зручний для швидкого створення тестових наборів даних без необхідності вручну вводити або заповнювати ці дані.

В Mockaroo наведено велику кількість типів даних, як стандартних так і специфічних для різних предметних областей (рис 4.7).

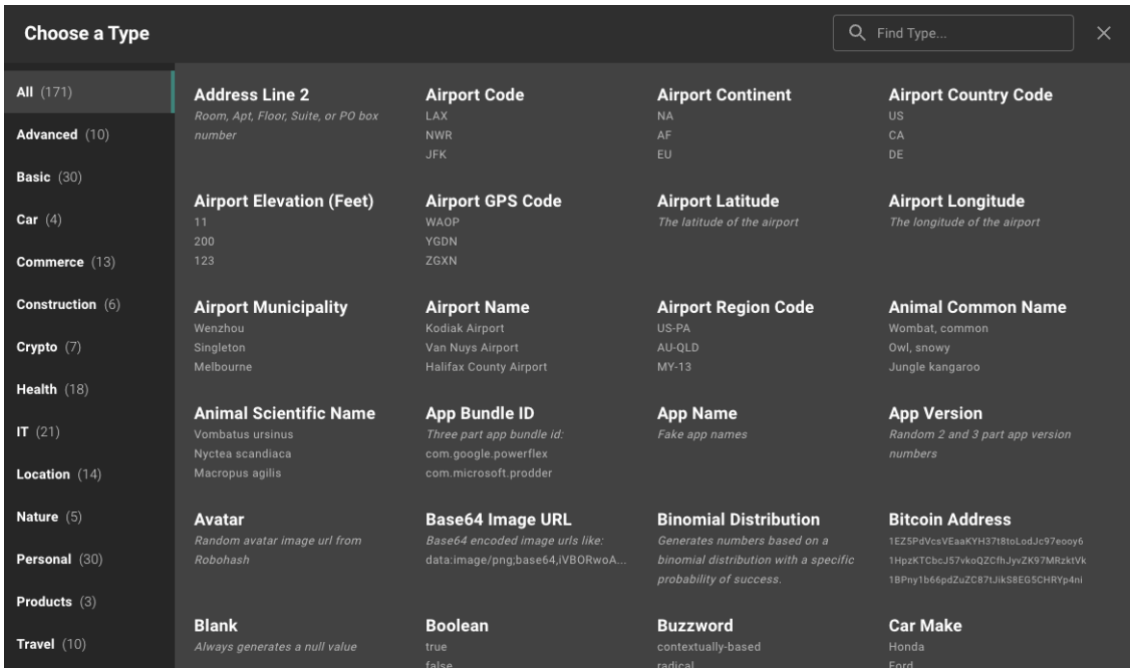


Рисунок 4.7 – Список типів даних в Mockaroo

Не дивлячись на широкий спектр доступних варіацій типів даних, деякі специфічні для предметної області типи даних відсутні. Наприклад опис та статус виконання проєкту. Але на такий випадок система дозволяє створити специфічний тип за допомогою ШІ.

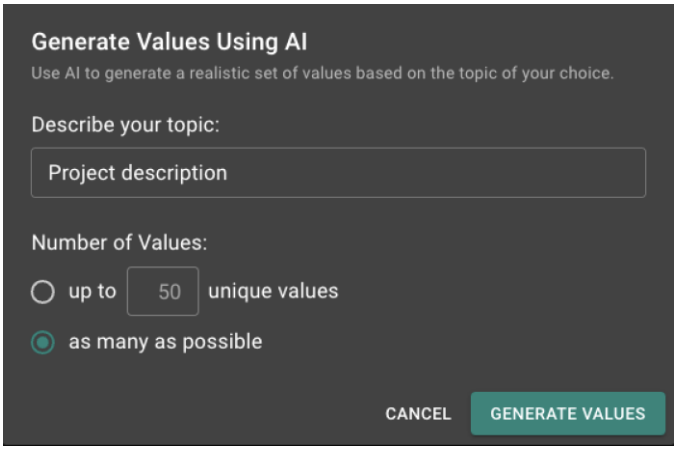


Рисунок 4.8 – Створення типу за допомогою штучного інтелекту

Додатково можна вказати формулу розрахунку для поля. Це можна застосувати наприклад для перевірки дати завершення проєкту, основним правилом в такому разі є те що дата завершення не повинна бути меншою за дату початку (рис 4.9).

```
field('start_date') + seconds(random(1,59)) + days(random(1,30)) + month(random(1,12)) + year(random(1,8))
```

Рисунок 4.9 – Створення розраховуваного поля для дати кінця проєкту

На прикладі сутності «Проєкт» буде розглянуто створення датасету за допомогою Mockaroo. Для початку потрібно визначити всі необхідні поля та типи даних для заповнення.

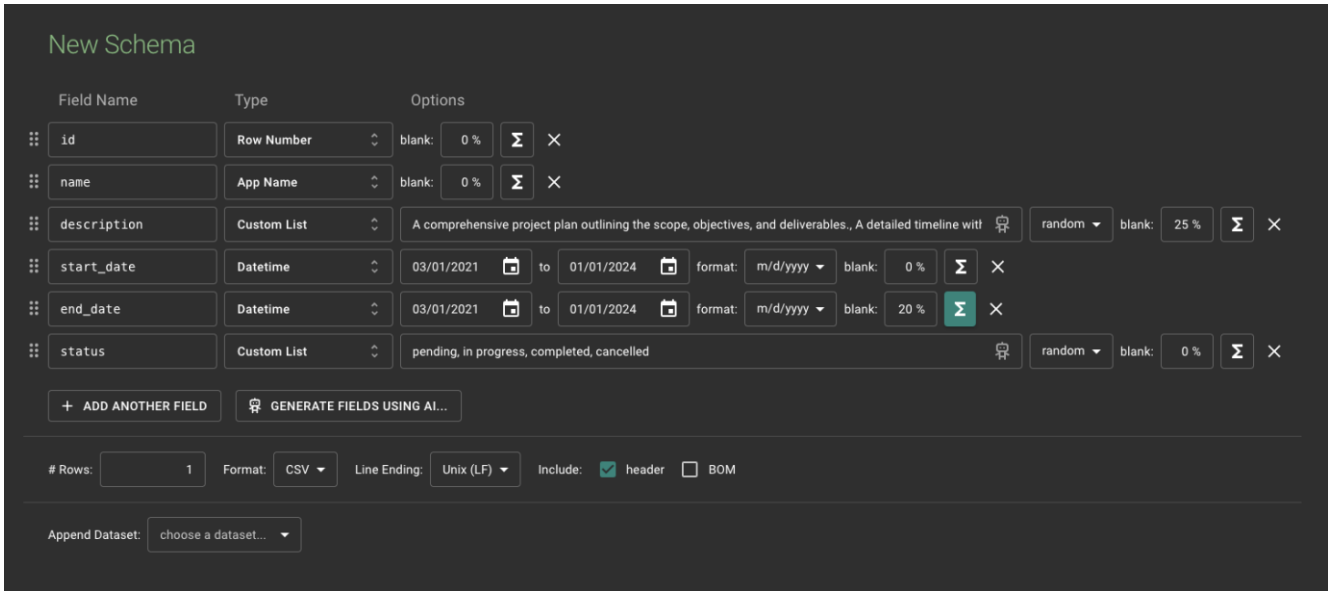


Рисунок 4.10 – Налаштування схеми для генерації датасету в Mockaroo

На рисунку 4.10 зображено процес створення схеми для датасету. Для схеми визначено назви полів, типи полів, опції для генерації даних а також відсоток незаповненості. За допомогою останнього можна вказати який відсоток даних для поля буде пустим. Це також відіграє свою роль в імітації реальних даних, та допоможе в розробці більш стійкої до відмови системи.

Після цього можна вибрати розмір вибірки та перевірити формат даних, який буде згенеровано сервісом (рис. 4.11).

TABLE RAW

id	name	description	start_date	end_date	status
1	Tampflex	Create a mobile app for task management	2/17/2022	8/22/2024	cancelled
2	Flexidy		11/1/2021	2/10/2025	cancelled
3	Alpha	Develop a software for data analysis	10/11/2022		cancelled
4	Konklab		10/6/2023		cancelled
5	Viva	Design a user interface for a booking system	8/8/2023	11/12/2031	pending
6	Cardify	Create a social media strategy for a fashion brand	2/21/2023	1/27/2025	completed
7	Holdlamis		12/1/2022	1/4/2028	cancelled
8	Domainer		7/10/2022	11/24/2027	in progress
9	Hatity	Develop a game for mobile devices	11/23/2021	3/28/2029	pending
10	Fix San	Design a website for an e-commerce store	12/6/2022	1/18/2027	completed

Рисунок 4.11 – Вибірка згенерованих даних

Наступним кроком буде генерація вибірки та запис в csv файл. Для тренування моделей згенеровано вибірку на тисячу рядків. Цього об’єму даних буде достатньо для первинного тренування на них моделей машинного навчання. Те ж саме потрібно зробити і для інших сутностей.

Далі отриманий датасет потрібно підготувати до подальшого використання, а саме потрібно позбутися від аномалій та пропущених значень. В випадку проєктів потрібно врахувати наступні пункти:

– заповнити пропущені значення для поля **description**;

– очистити рядки в яких дата початку проєкту більша за дату завершення;

– очистити рядки в яких поле статус **completed** та відсутня дата завершення.

Наступний код здійснює підготовку даних за заданими умовами:

```
import pandas as pd
from datetime import datetime

data = pd.read_csv('projects.csv')

data['description'].fillna('', inplace=True)

data['start_date'] = pd.to_datetime(data['start_date'], errors='coerce')
data['end_date'] = pd.to_datetime(data['end_date'], errors='coerce')

data = data[(data['start_date'] < data['end_date'])]

valid_statuses = ['pending', 'in progress', 'cancelled', 'completed']

data = data[data['status'].isin(valid_statuses)]
data = data[~((data['status'] == 'completed') & data['end_date'].isnull())]

today = datetime.today().date()
data = data[(data['end_date'].isnull()) | (data['end_date'].dt.date <= today)]

data.to_csv('cleaned_projects.csv', index=False)
```

У цьому скрипті використовується бібліотека pandas для обробки даних з файлу CSV, який містить інформацію про проєкти. Починаючи з завантаження даних, за допомогою методу read_csv(). Очищені дані зберігаються в новому файлі CSV за допомогою методу to_csv(). Цей скрипт дозволяє ефективно очищувати та фільтрувати дані проєктів, використовуючи різноманітні умови та операції обробки даних.

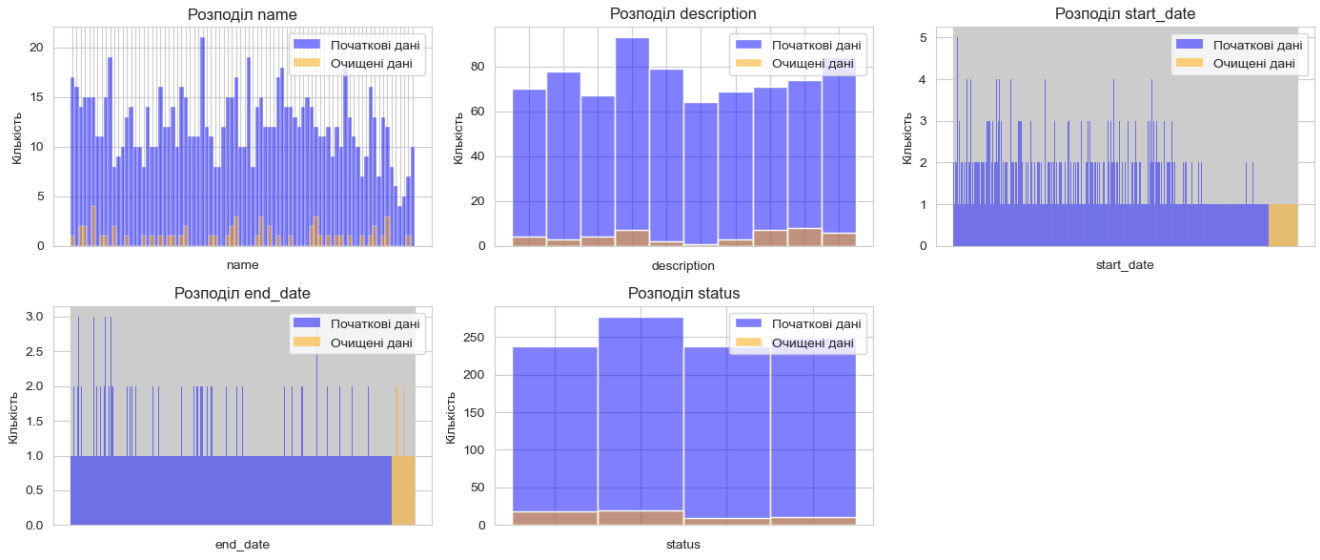


Рисунок 4.12 – Графіки розподілу ознак

На рисунку 4.12 зображено графіки розподілу для кожної ознаки для початкових та очищених даних. Графіки містять гістограми, які показують розподіл значень кожної ознаки в обох наборах даних. З графіку можна побачити, що після очищення залишилась лиш частина вхідних значень, а це означає, що більша частина даних не придатна до тренування. Саме тому процес підготовки даних є досить важливим.

В результаті очищення даних, отраманий датасет не підійде для тренування моделі через свій розмірі. Тому потрібно повторити процес генерації та очищення даних декілька разів для того, щоб створити великий датасет достатній для тренування.

Далі теж саме потрібно виконати для інших сутностей. Лістинг коду для очищення датасету для задач наведено в (Додаток А). Після цього дані можна використовувати для тренування моделей.

4.2.2 Застосування NLP для виявлення ознак

В контексті предметної області більша частина інформації представлена у вигляді тексту – назви та опис задач, описи проєктів та інші дані. Але текстові дані не зовсім підходять для тренування моделей. Для цього можна застосувати методи NLP. Основна задача на даному етапі виявити ознаки з тексту, які можуть бути застосовані для тренування моделей [16]. Наприклад описи задач допоможуть класифікувати задачі, які виконували різні члени команди для подальшого використання у кредитному скорингу для визначення учасника, чиї навички найбільше підійдуть для виконання конкретних задач.

Можна виділити основні етапи обробки даних за допомогою NLP:

1) Токенізація – розбиваємо текст на окремі слова або токени.

2) Чистка тексту – видаляємо непотрібні символи, знаки пунктуації, стоп-слова (якщо потрібно).

3) Лематизація або стемінг – зменшуємо слова до їх основного слова (леми або кореня).

4) Векторизація – перетворюємо оброблені слова в числові вектори, наприклад, за допомогою методу TF-IDF або векторизації слів Word2Vec.

Для виконання даної задачі, використано метод Word2Vec. Word2Vec – це техніка векторизації слів, яка представляє слова у векторному просторі, де семантично близькі слова розташовані близько одне до одного. Це може бути корисним для виявлення семантичних зв'язків між словами та розуміння контексту.

Наступний код виконує всі вказані етапи для датасету projects:

```
# Завантаження даних з CSV файлу
data = pd.read_csv('cleaned_projects.csv')

# Ініціалізація засобів NLTK
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

data['description'] = data['description'].fillna('')

# Токенізація
tokenized_descriptions = [word_tokenize(str(description)) for description in
data['description']]

# Видалення стоп-слів
stop_words = set(stopwords.words('english'))
filtered_descriptions = [[word for word in description if word.lower() not in
stop_words] for description in tokenized_descriptions]

# Лематизація
lemmatizer = WordNetLemmatizer()
lemmatized_descriptions = [[lemmatizer.lemmatize(word) for word in description]
for description in filtered_descriptions]

# Навчання моделі Word2Vec
model = Word2Vec(lemmatized_descriptions, vector_size=100, window=5, min_count=1,
workers=4)

# Отримання векторів слів
word_vectors = model.wv
```

У цьому коді проводиться обробку текстових даних з файлу CSV, що містить інформацію про проєкти. Після завантаження даних використовується бібліотека NLTK для токенізації описів проєктів, щоб розбити текст на окремі слова. Потім видаляються стоп-слова, які не несуть суттєвої інформації, та лематизація слів, щоб зменшити розмірність даних і уніфікувати слова з однаковим значенням. Після попередньої обробки тексту використовується

модель Word2Vec для перетворення слів в вектори фіксованої довжини у векторному просторі, зберігаючи семантичні зв'язки між ними. Це дозволяє використовувати ці вектори для подальшого аналізу тексту та виявлення семантичних зв'язків між словами.

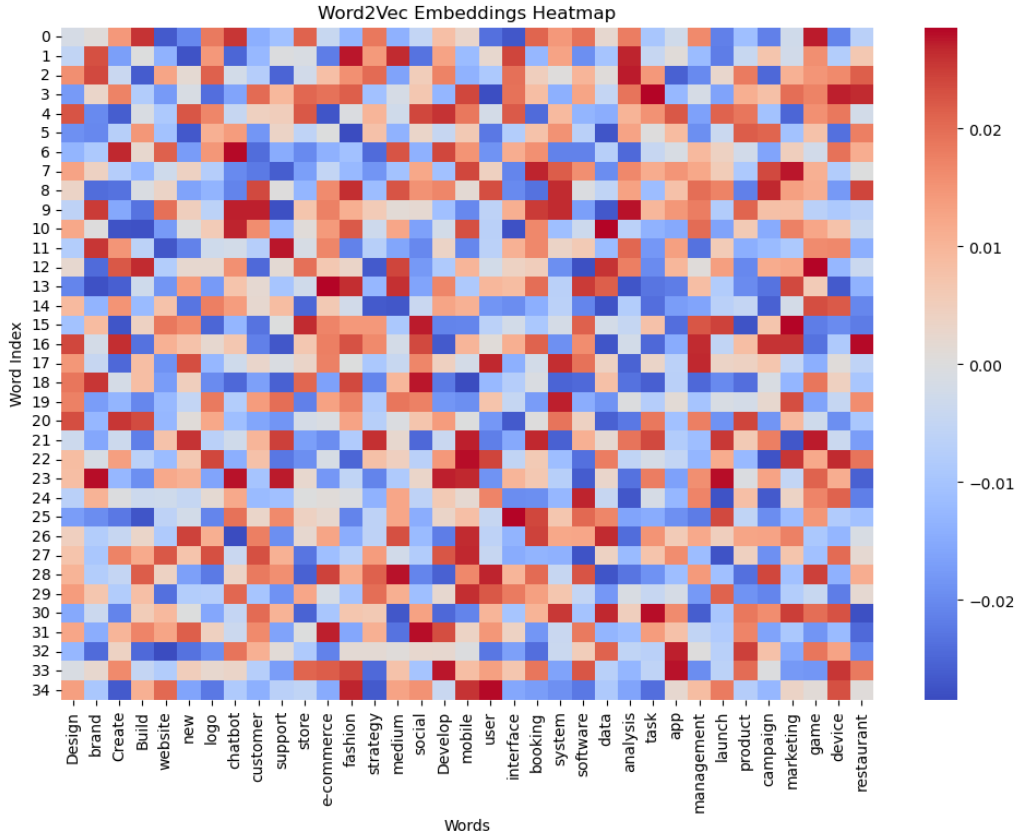


Рисунок 4.13 – Теплова карта розподілення слів в описі проєктів

На тепловій карті для TF-IDF матриці (рис 4.13) зображена частота використання слів в описі для вибірки проєктів. Кожен рядок у матриці відповідає одному проєкту, а кожний стовпчик – слову. Кольори на карті відображають значення TF-IDF для кожного слова у кожному проєкті: чим темніше колір, тим менше значення TF-IDF, а яскравіші кольори вказують на вищі значення.

4.2.3 Тренування моделей машинного навчання

Після того як дані для тренування зібрані, очищені та нормалізовані можна переходити до етапу розробки самих моделей. На даному кроці будуть розроблятися моделі для вирішення задач цільових функцій системи.

Для початку буде розглянуто процес розробки моделі для оцінки задач з використанням градієнтного бустингу. Першим кроком потрібно отримати дані з бази та розділити дані на тренувальні та тестові:

```
conn = psycopg2.connect(
    host=host,
    database=database,
    user=user,
    password=password
)

projects_query = "SELECT * FROM projects;"
tasks_query = "SELECT * FROM tasks;"
projects_data = pd.read_sql(projects_query, conn)
tasks_data = pd.read_sql(tasks_query, conn)

merged_data = pd.merge(tasks_data, projects_data, on='project_id', how='left')

X_train, X_test, y_train, y_test = train_test_split(features, labels,
test_size=0.2, random_state=42)
```

Розроблений код використовує бібліотеку psycopg2 для підключення до бази даних PostgreSQL та зчитування даних з таблиць «projects» і «tasks». Параметри підключення до бази даних передаються як аргументи функції psycopg2.connect(). Після підключення виконується SQL-запит до кожної таблиці, результати зчитуються в об’єкти DataFrame за допомогою функції pd.read_sql(). Потім дані об’єднуються за допомогою pd.merge(). Далі можуть бути виконані операції очистки та підготовки даних, наприклад, видалення зайвих стовпців або обробка пропущених значень. Нарешті, дані розділяються на тренувальний та тестовий набори за допомогою функції train_test_split().

Далі наведено лістинг коду класу для з використанням градієнтного бустингу для оцінки задач:

```
class GradientBoosting:
    def __init__(self, num_trees=100, learning_rate=0.1, max_depth=3):
        self.num_trees = num_trees
        self.learning_rate = learning_rate
        self.max_depth = max_depth

    def _initialize_params(self):
        return {'trees': [DecisionTree(max_depth=self.max_depth) for _ in
range(self.num_trees)], 'weights': jnp.ones(self.num_trees)}

    def _update_weights(self, y, y_pred, weights):
        return weights * jnp.exp(-self.learning_rate * (y_pred - y))

    def fit(self, x, y):
```

```
        params = self._initialize_params()
        for i in range(self.num_trees):
            y_pred = self.predict(params, x)
            gradient = 2 * (y_pred - y)
            for j in range(i + 1):
                params['trees'][j].fit(x, gradient)
            params['weights'] = self._update_weights(y, y_pred, params['weights'])
        self.params = params

    def predict(self, params, x):
        return jnp.sum(params['weights'][i] * params['trees'][i].predict(x) for i
in range(self.num_trees))
```

Даний код використовує клас дерев рішень лістинг коду якого наведено в (Додаток Б). Клас **GradientBoosting** реалізує алгоритм градієнтного бустингу з використанням дерев рішень. Під час ініціалізації він приймає параметри, такі як кількість дерев (num_trees), швидкість навчання (learning_rate) та максимальну глибину дерев (max_depth). Метод **fit** навчає модель за допомогою градієнтного спуску, використовуючи ансамбль дерев рішень. Метод **predict** здійснює прогнози за допомогою навченої моделі, а параметри моделі зберігаються для подальшого використання.

Приклад використання градієнтного бустингу наведено нижче:

```
project_data, task_data = generate_synthetic_data(num_projects,
num_tasks_per_project, num_features)
x, y = task_data[:, :-1], task_data[:, -1]

model = GradientBoosting(num_trees=num_trees, learning_rate=learning_rate,
max_depth=max_depth)
model.fit(x, y)

y_pred = model.predict(model.params, x)
mse = jnp.mean((y_pred - y) ** 2)
```

У цьому коді спочатку генеруються синтетичні дані для проєктів та завдань за допомогою функції **generate_synthetic_data**. Потім дані для вхідних ознак x та вихідної змінної y виділяються з отриманих даних. Наступна, ініціалізується модель градієнтного бустингу з вказаними параметрами та навчається на отриманих даних. Після тренування моделі здійснюється прогноз **y_pred** на основі вхідних даних **x**, після чого обчислюється середньоквадратична помилка (MSE) між прогнозованими та фактичними значеннями y.

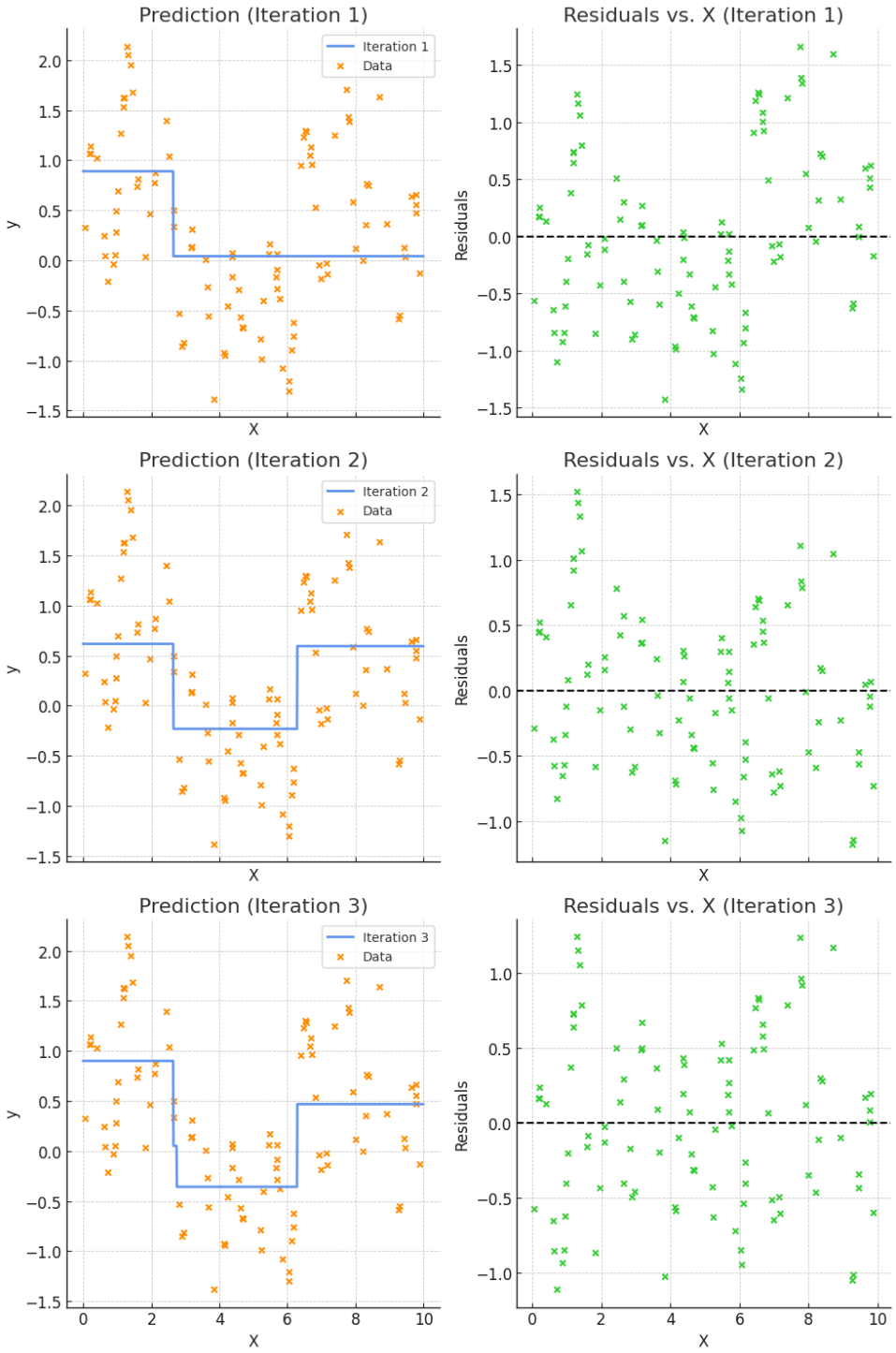


Рисунок 4.14 – Результат виконання градієнтного бустингу

На рисунку 4.14 можна побачити результат виконання трьох ітерацій градієнтного бустингу. На лівій стороні кожного ряду відображено прогнозовані значення (синій графік) у порівнянні з реальними даними (помаранчеві точки). На правій стороні показано залишки, тобто різницю між реальними та прогнозованими значеннями на кожній ітерації. З кожною ітерацією модель

намагається скоректувати свої помилки, зменшуючи залишки і покращуючи прогноз. Лише в деяких місцях графіку гіпотеза підтверджена. Тому додатково потрібно провести оптимізацію алгоритму, потрібно збільшити кількість дерев, глибину, а також зменшити швидкість навчання.

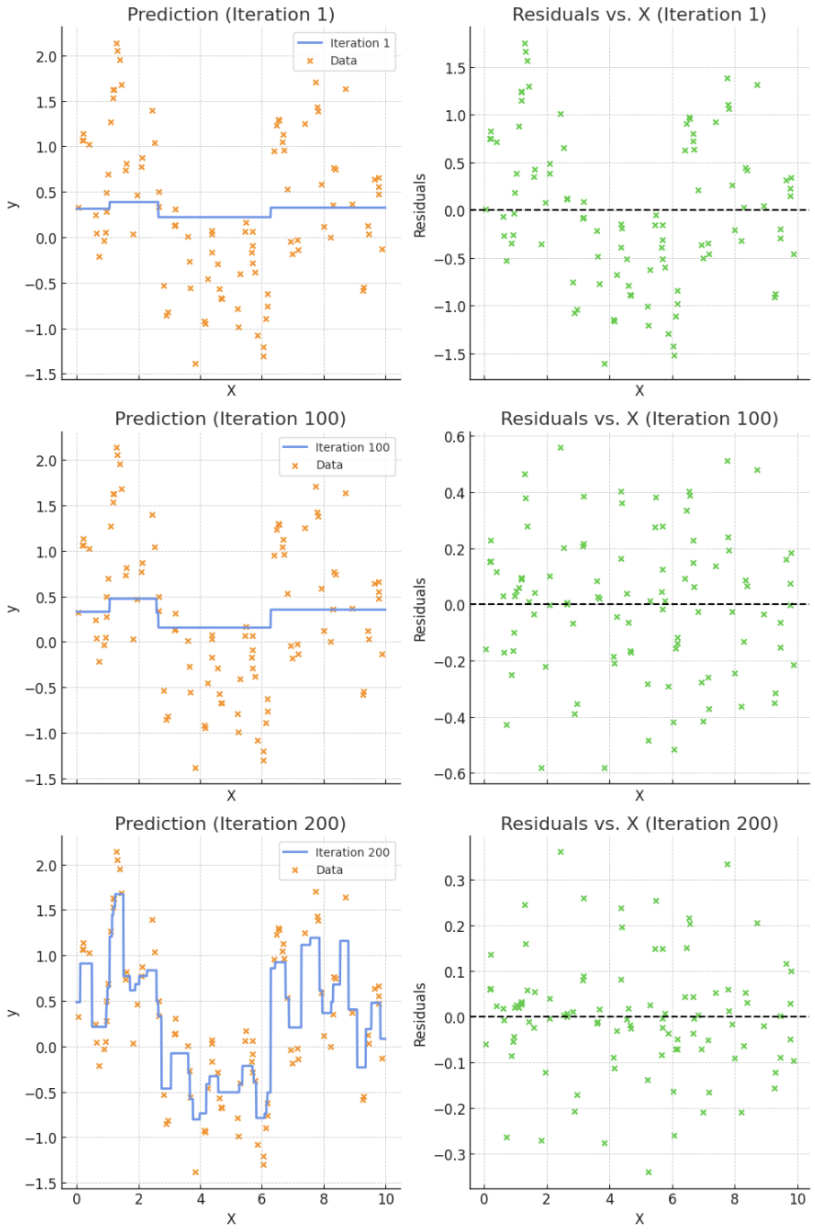


Рисунок 4.15 – Результат виконання градієнтного бустингу після корегування

На рисунку 4.15 відображені прогнозовані значення для 1-ї, 100-ї та 200-ї ітерацій. Після корегування можна побачити що точність алгоритму значно виросла. В даному випадку додано кількість дерев (200 дерев). З кожною ітерацією модель стає точнішою, і залишки зменшуються, свідчачи про здатність

моделі точно апроксимувати дані.. Процес підвищення точності можна повторити до досягнення необхідних результатів, але в контексті задачі, цієї точності достатньо.

Процес тренування інших моделей не відрізняється, тому детальний опис буде упущено. Лістинг коду моделі для прогнозування наведено в (Додаток В). Лістинг коду для тренування моделі кредитного скорінгу для автоматичного розподілу задач між учасниками наведено в (Додаток Г).

4.2.4 Збереження моделей на Google AI Platform Models

Google AI Platform, як передова хмарна платформа, надає широкі можливості для створення, збереження та розгортання моделей, що дозволяє ефективно впроваджувати інтелектуальні рішення в різноманітних галузях та проєктах.

Підготовка моделі JAX для відправки та розгортання у Google Vertex AI включає кілька важливих кроків. Оскільки Vertex AI прямо не підтримує моделі JAX, тому потрібно використати контейнеризацію для розгортання моделі. Перш ніж створити Docker контейнер, потрібно зберегти параметри моделі, які можуть бути використані для відновлення моделі під час виконання в контейнері. Це можна зробити за допомогою серіалізації параметрів моделі до файлу:

```
import flax
import jax.numpy as jnp

with open('gradient_busting_task_score_params.pkl', 'wb') as f:
    flax.serialization.to_bytes(params).tofile(f)
```

Після цього потрібно створити Dockerfile та додати до нього інструкції для створення образу Docker, який включає модель та код для її використання:

```
FROM python:3.8-slim

RUN pip install jax jaxlib flax numpy
COPY gradient_busting_task_score_params.pkl
/app/models/pretrained/gradient_busting_task_score_params.pkl
COPY model.py /app/models/model.py

WORKDIR /app

CMD ["python", "model.py"]
```

Далі потрібно додати функцію для виконання прогнозу з використанням зображення:

```
def predict(image):
    logits = gradient_boost.apply({'params': params}, image)
    return jnp.argmax(logits, axis=1)
```

Наступним кроком потрібно зібрати образ Docker. Це можна зробити за допомогою команди **docker build -t gradient-boosting-task-score:v1** [20].

```
(venv) den.bechkagmail.com@MacBook-Pro-Dmitro lead-forecaster % docker build -t gradient-boosting-task-score:v1 .
[+] Building 1187.9s (10/10) FINISHED
```

Рисунок 4.16 – Створення образу Docker для моделі gradient-boosting-task-score

Після цього потрібно відкрити застосунок Docker Desktop та перевірити коректність створення образу (рис. 4.17).

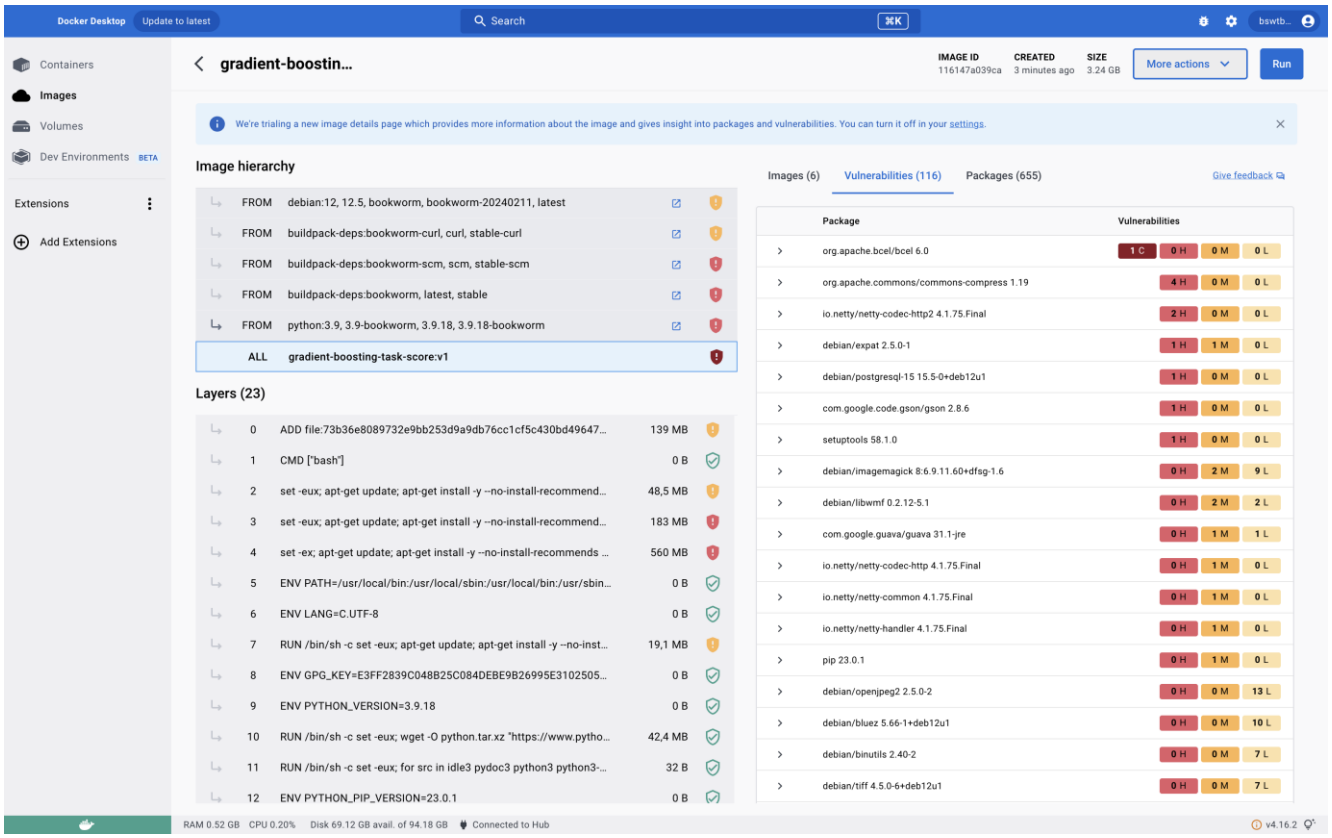


Рисунок 4.17 – Перевірка створеного образу в Docker Desktop

Далі зібраний образ потрібно завантажити до репозиторію контейнерів. Це можна зробити через застосунок або за допомогою командного рядка (рис. 4.18).

Для цього потрібно використати команду **docker push** та вказати адресу репозиторію.

```
(venv) den.bechkagmail.com@MacBook-Pro-Dmitro lead-forecaster % docker push gcr.io/pm-forcaster/gradient-boosting-task-score:v1
The push refers to repository [gcr.io/pm-forcaster/gradient-boosting-task-score]
5f70bf18a086: Pushed
a1993ba84ea3: Pushed
b5cecc989ac6: Pushed
04bd1961f5d7: Pushed
3b48824bd4fd: Pushed
5589e8997c0c: Pushed
5f895c7ab7df: Pushed
9ce63ba53cb8: Pushed
d9c6bbb693ea: Pushed
a974964b27e5: Pushed
973599cf2dad: Pushed
b10a49b17ae6: Pushed
v1: digest: sha256:26a421b467772ab4bc6c871d503643b11132f5084174b085789583658ac4c27c size: 2840
(venv) den.bechkagmail.com@MacBook-Pro-Dmitro lead-forecaster %
```

Рисунок 4.18 – Завантаження образу до репозиторію

Тепер можна перевірити образ на платформі Google Artifact Registry (рис. 4.19).

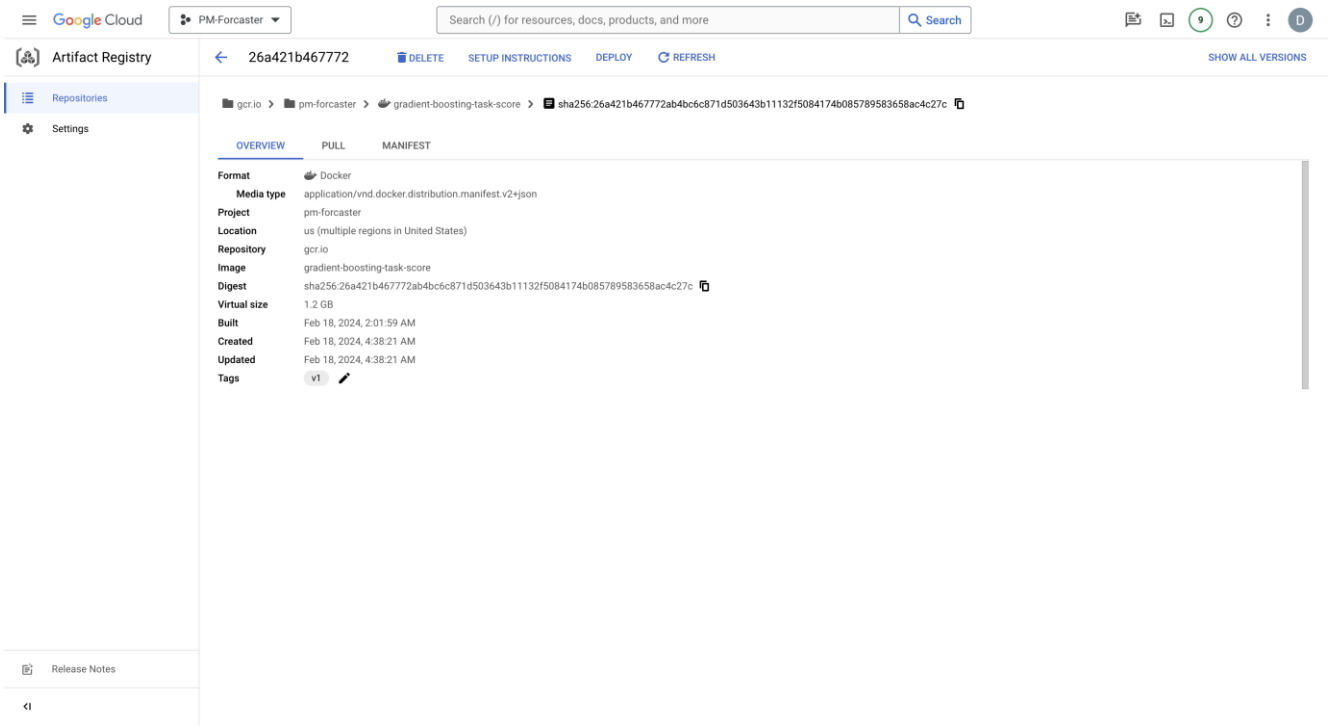


Рисунок 4.19 – Завантажений образ в Google Artifact Registry

Після завантаження образу, його можна розгорнути на Google Vertex AI. Перед тим, як розгорнути модель, потрібно створити саму модель у Vertex AI. Це можна зробити через Google Cloud Console або використовуючи **gcloud** команди.

```
den_bechka@cloudshell:~ (pm-forcaster)$ gcloud ai models upload --region="us-central1"  --display-name="gradient-boosting-task-score"  --proj
ect="pm-forcaster" --container-image-uri="gcr.io/pm-forcaster/gradient-boosting-task-score@sha256:26a421b467772ab4bc6c871d503643b11132f508417
4b085789583658ac4c27c"
Using endpoint [https://us-central1-aiplatform.googleapis.com/]
Waiting for operation [7090319410249859072]...done.
den_bechka@cloudshell:~ (pm-forcaster)$
```

Рисунок 4.20 – Створення моделі в Google Vertex AI

На рисунку 4.20 зображено результат виконання команди для створення нової моделі з завантаженого раніше образу. Команда **gcloud ai models upload** використовується для створення та завантаження нової моделі в Google Cloud Vertex AI, вказуючи конкретний Docker образ, який містить модель, для розгортання.

Далі потрібно створити точку доступу. Створення точки доступу (endpoint) у Google Cloud Vertex AI дозволяє розгортати моделі машинного навчання та надавати доступ до них через інтернет. Точка доступу слугує як інтерфейс, через який користувачі можуть надсилати запити на прогнозування до вашої моделі.

```
den_bechka@cloudshell:~ (pm-forcaster)$ gcloud ai endpoints create \
  --region="us-central1" \
  --display-name="Gradient boosting task score" \
  --project="pm-forcaster"
Using endpoint [https://us-central1-aiplatform.googleapis.com/]
Waiting for operation [4086418458793738240]...done.
Created Vertex AI endpoint: projects/390086106017/locations/us-central1/endpoints/1215226430506401792.
den_bechka@cloudshell:~ (pm-forcaster)$
```

Рисунок 4.21 – Створення точки доступу до моделі в Google Vertex AI

Після створення точки доступу (endpoint) в Google Cloud Vertex AI можна перевірити модель (рис. 4.22).

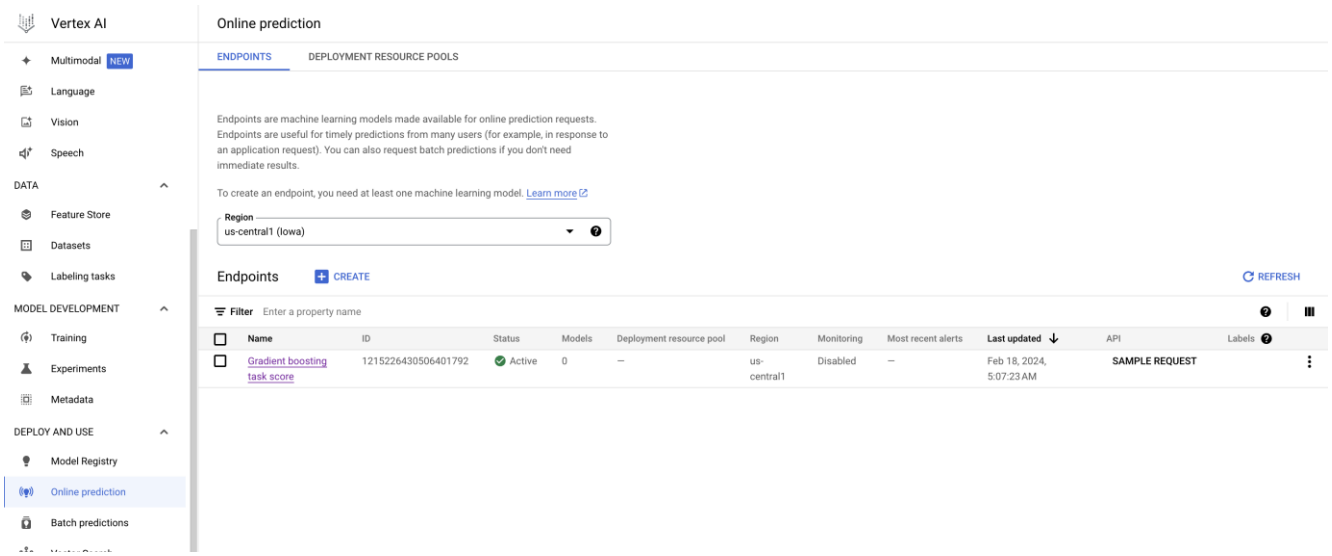


Рисунок 4.22 – Перевірка точки доступу в Google Vertex AI

Далі потрібно повторити всі етапи для кожної моделі. Після цього до моделей можна отримати доступ з основної частини застосунку.

4.3 Розробка бізнес логіки застосунку

4.3.1 Створення міграцій та ORM моделей

Першочерговим завданням бекенд частини будь-якого застосунку є робота з даними. Для цього використовуються моделі та міграції. Міграції – це сутності для маніпулювання процесом створення та видалення таблиць у базі даних. Вони дозволяють керувати процесом створення та внесення змін у схему таблиць бази даних шляхом написання програмного коду на Python [21]. Таким чином не залежно від типу бази даних, створення таблиць для застосунку будуть виглядати майже однаково для всіх.

Моделі в свою чергу надають зручний інтерфейс доступу до даних в базі. Вони спрощують роботу з даними, оскільки надають можливість працювати з даними, використовуючи об’єктно-орієнтовані підходи, такі як спадковість, методи класу та інші.

Спочатку потрібно для кожної таблиці ми можемо створити модель даних у Python за допомогою SQLAlchemy ORM. Нижче наведено лістинг коду, який визначає схему бази даних за допомогою SQLAlchemy ORM для трьох сутностей: «Project», «User» і «Task»:

```
from sqlalchemy import Column, Integer, String, Date, Text, Float, TIMESTAMP,
ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship

Base = declarative_base()

class Project(Base):
    __tablename__ = 'projects'

    id = Column(Integer, primary_key=True)
    name = Column(String(255), nullable=False)
    description = Column(Text)
    start_date = Column(Date)
    end_date = Column(Date)
    status = Column(String(50))
```

```
class User(Base):
    __tablename__ = 'users'
ƒ
    user_id = Column(Integer, primary_key=True)
    name = Column(String(255), nullable=False)
    position = Column(String(255))
    role = Column(String(50))
    email = Column(String(255))

class Task(Base):
    __tablename__ = 'tasks'

    id = Column(Integer, primary_key=True)
    project_id = Column(Integer, ForeignKey('projects.id'))
    name = Column(String(255), nullable=False)
    description = Column(Text)
    priority = Column(Integer)
    status = Column(String(50))
    estimated_time = Column(Integer)
    actual_time = Column(Integer)
    user_id = Column(Integer, ForeignKey('users.id'))

    project = relationship('Project', backref='tasks')
    user = relationship('User', backref='tasks')
```

Кожна сутність представлена відповідним класом, який наслідує базовий клас «Base». Визначені стовпці бази даних відображаються як атрибути класів. Взаємозв'язки між сутностями реалізовані за допомогою **ForeignKey** та **relationship**. Інші сутності системи створюються подібним чином.

Після цього можна переходити до створення міграцій. Для роботи з міграціями використовується Alembic. Alembic – це потужний інструмент для керування міграціями баз даних в середовищі Python. Він дозволяє автоматизувати процес створення, виконання і відкату міграцій для структурних змін у базі даних, таких як створення нових таблиць, додавання або видалення стовпців, зміна типів даних тощо [22]. Alembic забезпечує консистентність структури бази даних у різних середовищах розробки, дозволяючи легко керувати версіями схеми бази даних та забезпечуючи безпеку при впровадженні змін. Він інтегрується з SQLAlchemy, що робить його ідеальним інструментом для роботи з ORM в середовищі Python.

Перш за все потрібно створити файл конфігурації **alembic.ini** з наступним вмістом:

```
[alembic]
script_location = migrations
```

Наступним кроком потрібно виконати команду **alembic init**. Дана команда створить директорію **migrations** з базовою конфігурацією для проведення міграції. Але оскільки база даних зберігається на хмарному сервісі Google Cloud SQL, конфігурацію потрібно налаштувати для роботи з коннектором google cloud.

```
config_path = os.path.join(os.getcwd(), 'alembic.ini')
target_metadata = Base.metadata
config = context.config
config.set_main_option('sqlalchemy.url', DATABASE_URL)

connector = Connector()
cloud_sql_connection = connector.connect(
    DATABASE_CONNECTION,
    "pg8000",
    user=DATABASE_USER,
    password=DATABASE_PASSWORD,
    db=DATABASE_NAME,
    ip_type=IPTypes.PUBLIC,
)


def run_migrations_offline():
    context.configure(
        url=config.get_main_option("sqlalchemy.url"),
        target_metadata=target_metadata,
        literal_binds=True,
        dialect_opts={"paramstyle": "named"},
    )

    with context.begin_transaction():
        context.run_migrations()

def run_migrations_online():
    connectable = engine_from_config(
        config.get_section(config.config_ini_section),
        prefix="sqlalchemy.",
        poolclass=pool.NullPool,
        creator=lambda: cloud_sql_connection,
    )

    with connectable.connect() as connection:
        context.configure(
            connection=connection,
            target_metadata=target_metadata
        )

        with context.begin_transaction():
            context.run_migrations()

if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

Наведений код використовує бібліотеку Alembic для керування міграціями бази даних у вашому проєкті. Він розділений на дві основні функції:

run_migrations_offline() та **run_migrations_online()**. Перша функція **run_migrations_offline()** призначена для виконання міграцій у відсутності активного з'єднання з базою даних. Вона налаштовує контекст Alembic і запускає міграції, використовуючи метадані цільової бази даних **target_metadata**. Друга функція **run_migrations_online()** призначена для виконання міграцій в режимі онлайн, коли існує активне з'єднання з базою даних. Вона налаштовує з'єднання з базою даних, використовуючи конфігурацію з файлу **alembic.ini**, та виконує міграції так само, як і у функції **run_migrations_offline()**. Загальна логіка полягає у тому, що контекст Alembic налаштовується відповідно до режиму роботи (онлайн або офлайн), а потім виконує міграції у відповідності з цим контекстом. Крім того, використовується об'єкт **Connector** для з'єднання з базою даних Google Cloud SQL, а отримані з'єднання передаються у функцію **engine_from_config()** для виконання міграцій в режимі онлайн.

Після додаткової конфігурації можна запустити команди для ініціалізації міграції (рис. 4.22).

```
(venv) den.bechkagmail.com@MacBook-Pro-Dmitro lead-forecaster % alembic revision --autogenerate -m 'initial'
  Generating /Users/den.bechkagmail.com/Work/Personal/lead-forecaster/app/migrations/versions/c7d6374604fd_initial.py ...  done
(venv) den.bechkagmail.com@MacBook-Pro-Dmitro lead-forecaster %
```

Рисунок 4.22 – Запуск ініціалізації міграції за допомогою alembic

Після створення міграцій потрібно застосувати їх до бази даних за допомогою команди **alembic upgrade head**.

Тепер потрібно перевірити коректність проведення міграції. Для цього можна створити тестовий запит та вивести інформацію про таблиці:

```
@router.get("/tables")
def get_tables():
    tables = []
    tabs = ['projects', 'users', 'tasks']
    for tab in tabs:
        table_details = {"table_name": tab, "columns": []}
        query = sqlalchemy.text(
            "SELECT column_name, data_type, character_maximum_length, is_nullable,
column_default "
            "FROM information_schema.columns "
            "WHERE table_name = :tab"
        )
        with pool.connect() as db_conn:
            result = db_conn.execute(query, {'tab': tab})
            columns = []
```

```
            for row in result:
                column_details = {
                    "column_name": row[0],
                    "data_type": row[1],
                    "max_length": row[2],
                    "nullable": row[3],
                    "default": row[4],
                }
                columns.append(column_details)
            table_details["columns"] = columns
        tables.append(table_details)
    return {"tables": tables}
```

Після цього можна звернутися до кінцевої точки **/tables** та перевірити результат (рис. 4.23).

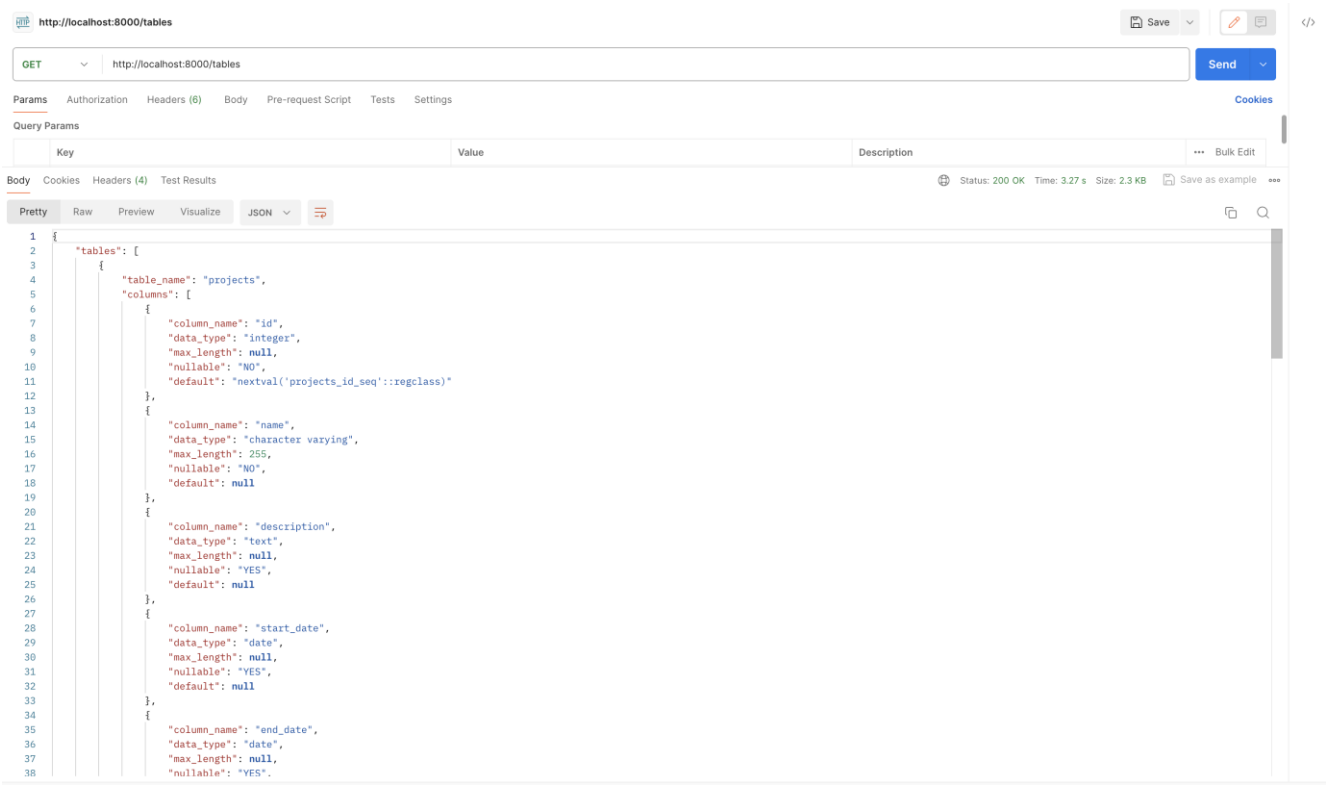


Рисунок 4.23 – Результат виконання тестового запиту для отримання інформації про таблиці в середовищі Postman

Після того як моделі для роботи з даними та міграції для створення структури бази даних створені та протестовані, можна переходити до наступного кроку розробки системи – програмування API для реалізації можливості клієнтам взаємодіяти з системою.

4.3.2 Реалізація API

Створення API є не найголовнішою, але ключовою частиною системи. Саме ця складова застосунку буде виконувати роль звязуючого елемента для бази даних, моделей машинного навчання та клієнта.

Для початку потрібно створити маршрути для роботи з основними сутностями системи. Цей процес включає в себе створення Pydantic класів для валідації даних, визначення основних маршрутів та створення CRUD опепацій.

Pydantic – це бібліотека Python, яка надає можливість визначення схеми даних (data schema) за допомогою створення класів, які відображають структуру даних та їх типи [23]. Ці класи використовуються для валідації та серіалізації даних, що дозволяє встановити правильність формату та типів даних, які отримуються або надсилаються через API або інші джерела даних. Pydantic забезпечує зручний і простий спосіб робити операції валідації та конвертації даних, що допомагає у підтримці надійності та стабільності програмного забезпечення [24].

Наступний код демонструє приклад створення таких класів для сутності Project:

```
class ProjectBase(BaseModel):
    name: str
    description: Optional[str] = None
    start_date: Optional[date] = None
    end_date: Optional[date] = None
    status: Optional[str] = None

class ProjectCreate(ProjectBase):
    pass

class ProjectUpdate(ProjectBase):
    pass

class ProjectInDBBase(ProjectBase):
    id: int

    class Config:
        orm_mode = True
```

Клас **ProjectBase** містить основні поля проєкту, такі як назва, опис, початкова та кінцева дата та статус, які обов’язкові для створення нового проєкту.

Класи **ProjectCreate** та **ProjectUpdate** наслідуються від **ProjectBase** і не містять додаткових полів, що дозволяє їх використання для створення та оновлення проєктів. Клас **ProjectInDBBase** також наслідується від **ProjectBase**, але включає додаткове поле **id**, яке є унікальним ідентифікатором проєкту в базі даних, та використовує параметр **orm_mode=True**, щоб надати можливість повертати дані напряму з об'єктів ORM моделі SQLAlchemy.

Наступним кроком потрібно визначити набір функцій для проведення CRUD операцій над сутністю Project:

```
def get_project(db: Session, project_id: int):
    return db.query(models.Project).filter(models.Project.id ==
project_id).first()

def get_projects(db: Session, skip: int = 0, limit: int = 10):
    return db.query(models.Project).offset(skip).limit(limit).all()

def create_project(db: Session, project: schemas.ProjectCreate):
    db_project = models.Project(**project.dict())
    db.add(db_project)
    db.commit()
    db.refresh(db_project)
    return db_project

def update_project(db: Session, project_id: int, project: schemas.ProjectUpdate):
    db_project = db.query(models.Project).filter(models.Project.id ==
project_id).first()
    for var, value in vars(project).items():
        setattr(db_project, var, value) if value else None
    db.commit()
    db.refresh(db_project)
    return db_project

def delete_project(db: Session, project_id: int):
    db_project = db.query(models.Project).filter(models.Project.id ==
project_id).first()
    db.delete(db_project)
    db.commit()
```

Функції отримують об'єкт сесії бази даних **db** (типу Session) та використовують моделі та схеми Pydantic для взаємодії з даними. Функція **get_project** повертає проєкт за його ідентифікатором, **get_projects** повертає список проєктів з можливістю пропуску та ліміту, **create_project** створює новий проєкт у базі даних, **update_project** оновлює існуючий проєкт та **delete_project** видаляє проєкт. Для взаємодії з базою даних використовується об'єкт сесії db, який передається функціям як аргумент.

Далі потрібно визначити ключові маршрути. Лістинг повного коду

```
# Підключення до бази даних для кожного запиту
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# Маршрут для створення нового проєкту
@app.post("/projects/", response_model=schemas.ProjectInDBBase)
async def create_project(project: schemas.ProjectCreate, db: Session =
Depends(get_db)):
    return crud.create_project(db=db, project=project)

# Маршрут для отримання всіх проєктів
@app.get("/projects/", response_model=list[schemas.ProjectInDBBase])
async def read_projects(skip: int = 0, limit: int = 10, db: Session =
Depends(get_db)):
    projects = crud.get_projects(db, skip=skip, limit=limit)
    return projects

# Маршрут для отримання конкретного проєкту за його ID
@app.get("/projects/{project_id}", response_model=schemas.ProjectInDBBase)
async def read_project(project_id: int, db: Session = Depends(get_db)):
    project = crud.get_project(db, project_id)
    if project is None:
        raise HTTPException(status_code=404, detail="Project not found")
    return project

# Маршрут для оновлення інформації про проєкт
@app.put("/projects/{project_id}", response_model=schemas.ProjectInDBBase)
async def update_project(project_id: int, project: schemas.ProjectUpdate, db:
Session = Depends(get_db)):
    updated_project = crud.update_project(db, project_id, project)
    if updated_project is None:
        raise HTTPException(status_code=404, detail="Project not found")
    return updated_project

# Маршрут для видалення проєкту
@app.delete("/projects/{project_id}", status_code=204)
async def delete_project(project_id: int, db: Session = Depends(get_db)):
    crud.delete_project(db, project_id)
    return
```

В даному прикладі використовується функцію **get_db()** для підключення до бази даних для кожного запиту. Кожен маршрут відповідає певній операції CRUD: створення нового проєкту, отримання всіх проєктів, отримання конкретного проєкту за його ID, оновлення інформації про проєкт та видалення проєкту. Кожен маршрут також використовує схему Pydantic для валідації та форматування вхідних та вихідних даних.

Далі потрібно реалізувати маршрути доступу до функцій моделей машинного навчання. Спочатку потрібно створити допоміжну функцію для підключення до Google Vertex AI:

```
def call_remote_ai(project_data, endpoint):
    endpoint = aiplatform.Endpoint(endpoint)

    instance_dict = project_data.dict()
    instances = [instance_dict]

    response = endpoint.predict(instances=instances)
    return response.predictions
```

Ця функція призначена для виклику віддаленої моделі штучного інтелекту (AI) за допомогою Google Vertex AI. Вона приймає дані про проєкт, користувача та задачу у форматі об'єкта ProjectData та адресу (endpoint) моделі. Функція перетворює ці дані в словник, підготовлює їх для передачі до моделі, викликає віддалену модель AI за вказаною адресою і повертає результати прогнозу, отримані від моделі.

Далі потрібно створити маршрут:

```
@app.post("/credit-score-predict/")
async def predict(project_data: ProjectData):
    try:
        prediction_result = call_remote_ai(project_data,
CREDIT_SCORE_PREDICT_ENDPOINT)
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

    minimal_expression = prediction_result.get("minimal_expression")
    return {"result": prediction_result}
```

У цьому коді створено маршрут POST з адресою «/credit-score-predict/», який приймає дані проєкту у форматі ProjectData. Дані передаються до функції predict, яка викликає функцію **call_remote_ai** для прогнозування кредитного рейтингу на основі цих даних. Якщо виникає помилка під час прогнозування, вона обробляється, і відповідь містить відповідне повідомлення про помилку. У випадку успішного прогнозування результат відправляється назад у відповіді.

Схожим чином створюються маршрути для інших моделей машинного навчання. На даному етапі розробка серверної частини застосунку закінчується.

4.4 Клієнтський дашборд для управління проєктами

Розробка клієнтської частини застосунку по більшій мірі складається з використання вже готових компонентів та IU бібліотек, таких як Refine та Tailwind. Крім цього розробка клієнтського інтерфейсу не є основною частиною даного дослідження, вона представляє собою лише один із методів взаємодії клієнта з системою. Тому на процес розробки користувацького вебінтерфейсу не буде виділено особливої уваги. Увага в даному розділі буде виділена саме на опис клієнтського інтерфейсу та функцій, які він надає клієнтам.

Головна сторінка

Головна сторінка застосунку представляє з себе інформаційну панель. На ній відображена інформація про кількість проєктів, кількість учасників (членів команди) та загальну кількість задач. На скіншоті (рис 4.24) зображений зовнішній вигляд головної сторінки.

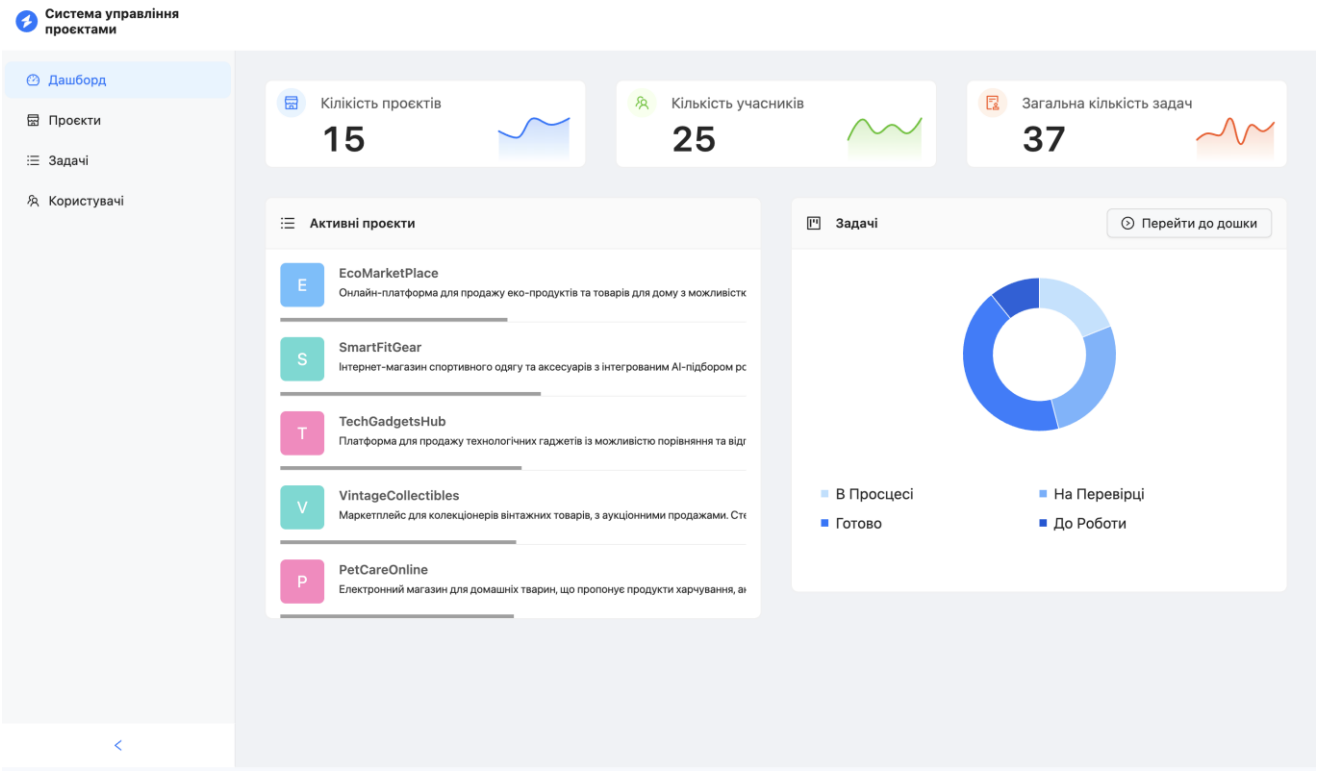


Рисунок 4.24 – Головна сторінка застосунку

Нижче представлений список активних проєктів, а також кругова діаграма з інформацію про загальний статус задач. Задачі розділені на чотири статуси:

«Готові для роботи», «В процесі», «Не перевірці» та «Готові» тобто виконані (рис. 4.25).

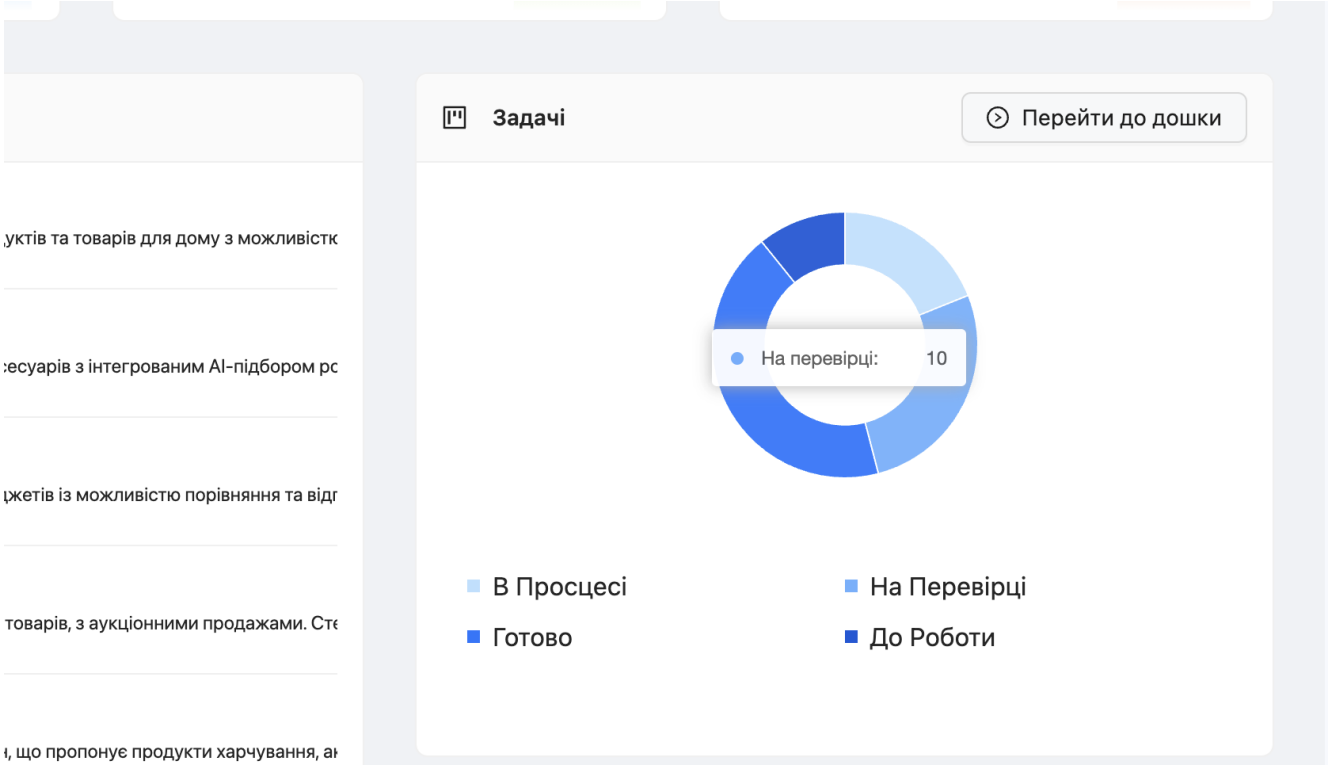


Рисунок 4.25 – Кругова діаграма статусів задач

Крім цього віджет також має кнопку «Перейти до дошки», при кліці на якій відбувається перехід на дошку з задачами.

Сторінка учасників

На сторінці відображається таблиця з усіма учасниками команди (рис. 4.26). Кожен рядок таблиці відповідає окремому учаснику. Таблиця містить інформацію про ім’я, посаду та роль учасника у команді. Серед основних ролей можна відмітити розробника (Developer), тестувальника (QA), проєктного менеджера (Project Manager), а також дизайнера (Designer).

В верхній частині таблиці відображається загальна кількість учасників. В нижній частині знаходиться пагінація.

В таблиці наявна можливість проведення пошуку учасників за ім’ям та посадою. Поля пошуку знаходяться біля заголовків відповідних колонок. Крім того також є можливість фільтрації учасників за роллю. Поле фультрації знаходиться біля заголовку колонки «Роль».

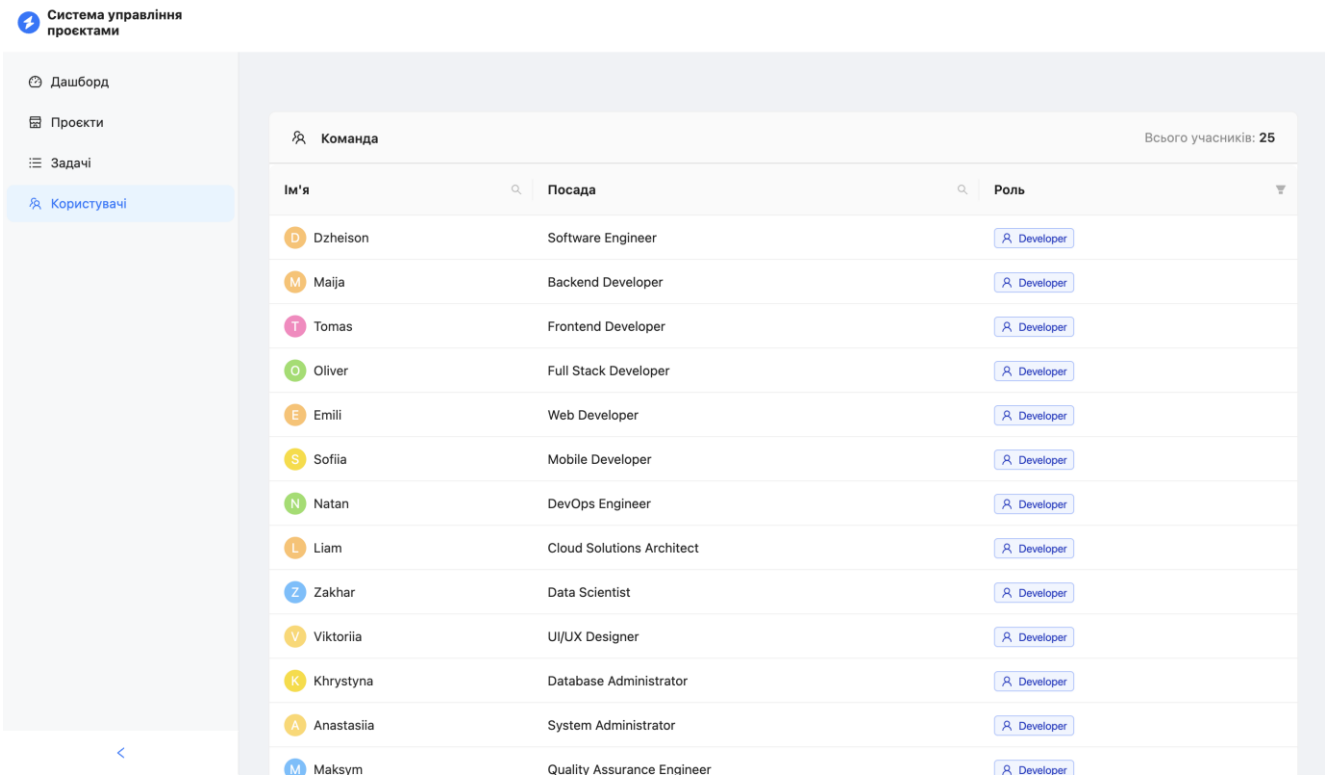


Рисунок 4.26 – Сторінка учасників системи

Сторінка проєктів

На даній сторінці відображається список проєктів (рис. 4.27). Можна виділити активні проєкти та завершені. Завершені проєкти відображається на сторінці темнішим кольором.

На сторінці знаходиться рядок пошуку проєктів за назвою, а також кнопки зміни представлення списку. Для відображення доступно два моди: відображення у вигляді сітки карток, та відображення у вигляді таблиці.

Список складається з карток проєктів. Кожна картка містить назву, зображення (якщо наявне), інформацію про учасників а також менеджера, закріплений за проєктом. Крім того кожна картка має контексте меню у верхньому правому куті у вигляді трьох точок. В контекстному меню знаходяться пунки «Переглянути деталі проєкту» та «Видалити проєкт» для переходу на сторінку з деталями проєкта та видалення відповідно.

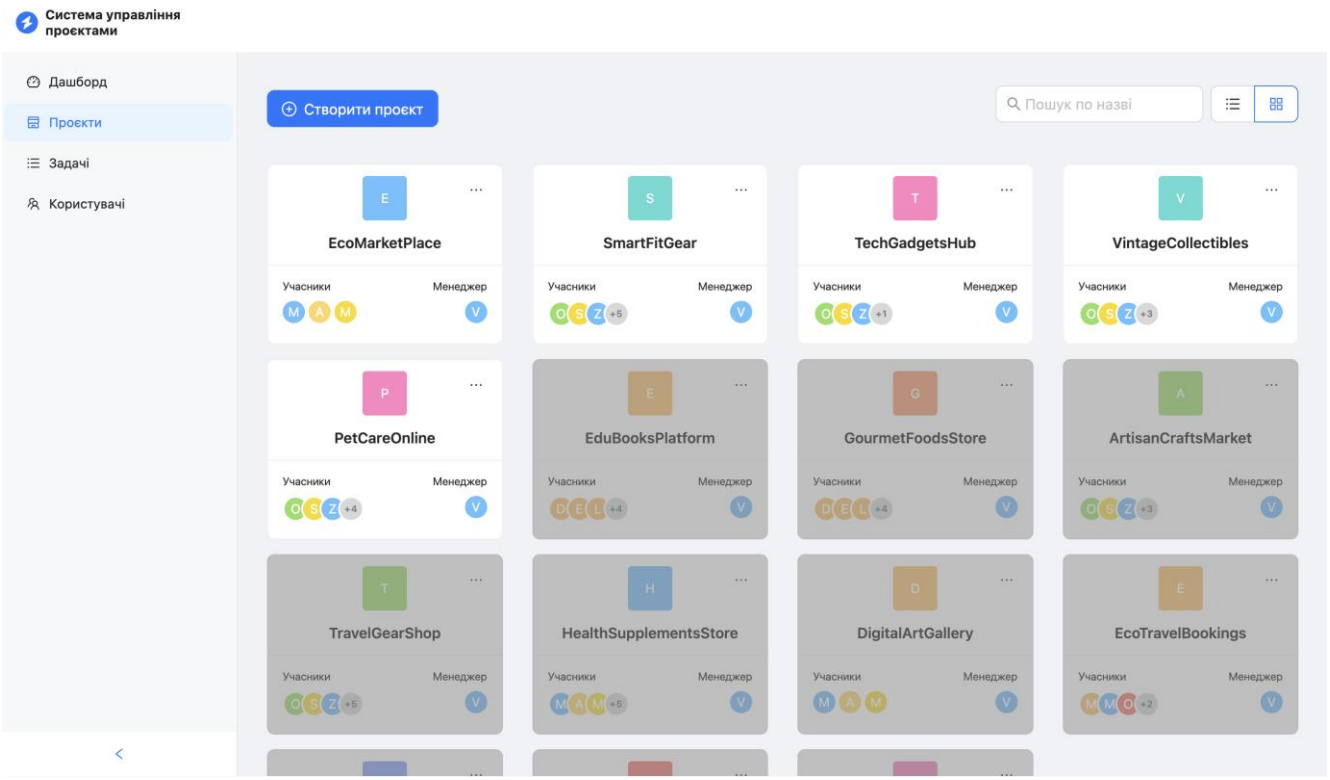


Рисунок 4.27 – Сторінка проєктів

У верхній частині сторінки розташована кнопка «Створити проєкт». При натисканні на якій відображається попап з формою для створення нового проєкту (рис. 4.28).

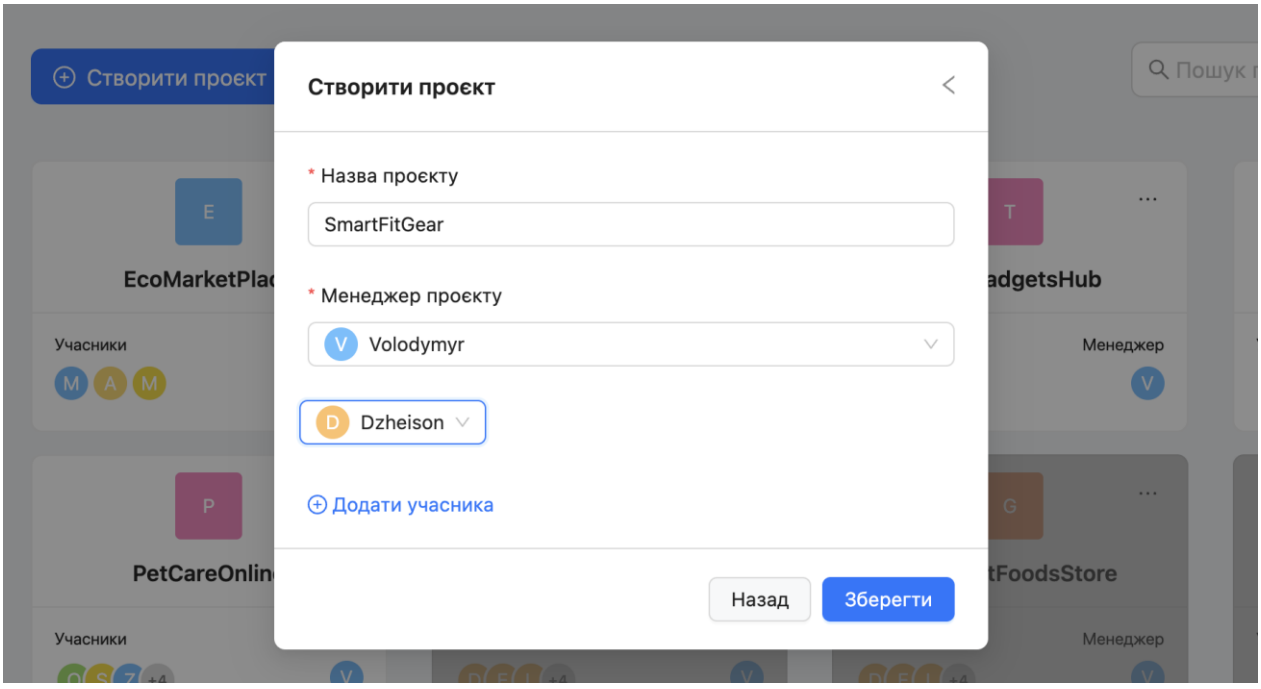


Рисунок 4.28 – Форма створення нового проєкту

Сторінка проєкту

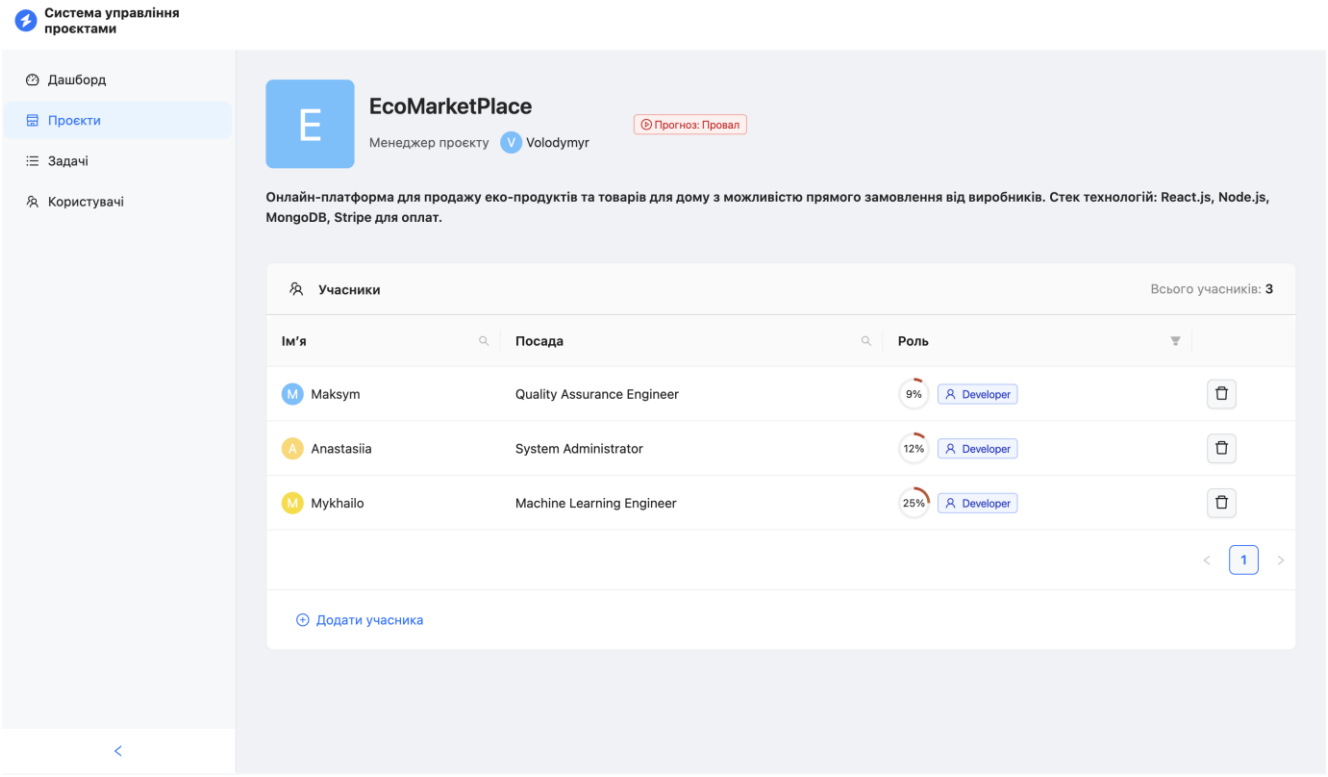


Рисунок 4.29 – Сторінка детальної інформації про проєкт

На даній сторінці представлена детальна інформація про проєкт (рис. 4.29). У верхній частині відображається назва, закріплений менеджер та опис проєкту. Кожне з цих полів редагується на місці. При натисканні на відповідне поле, воно перетворюється на поле для вводу, а також додаються кнопки «Назад» та «Зберегти».

Біля інформації про проєкт відображається бадж з результатом прогнозу для проєкту. Прогноз здійснюється за допомогою моделі машинного навчання на основі логістичної регресії. Резальтат прогнозу може бути «Провал» (червоний бадж) та «Успіх» (зелений бадж).

Нижче розташована таблиця з учасниками проєкту. Дана таблиця має схожий формат з таблицею на сторінці учасників, за виключенням того, що до неї можна додавати та прибирати учасників. Окрім цього для кожного учасника розраховується оцінка (кредитний скоринг). Ця оцінка вказує на скільки підходящим є учасник для цього проєкту. На скриншоті (рис. 4.29) можна

побачити що оцінки прогнозування різних моделей збігаються для конкретного прикладу. З цього можна зробити висновок, що проєкту прогнозується провал через непідходящий під нього набір учасників.

Сторінка задач

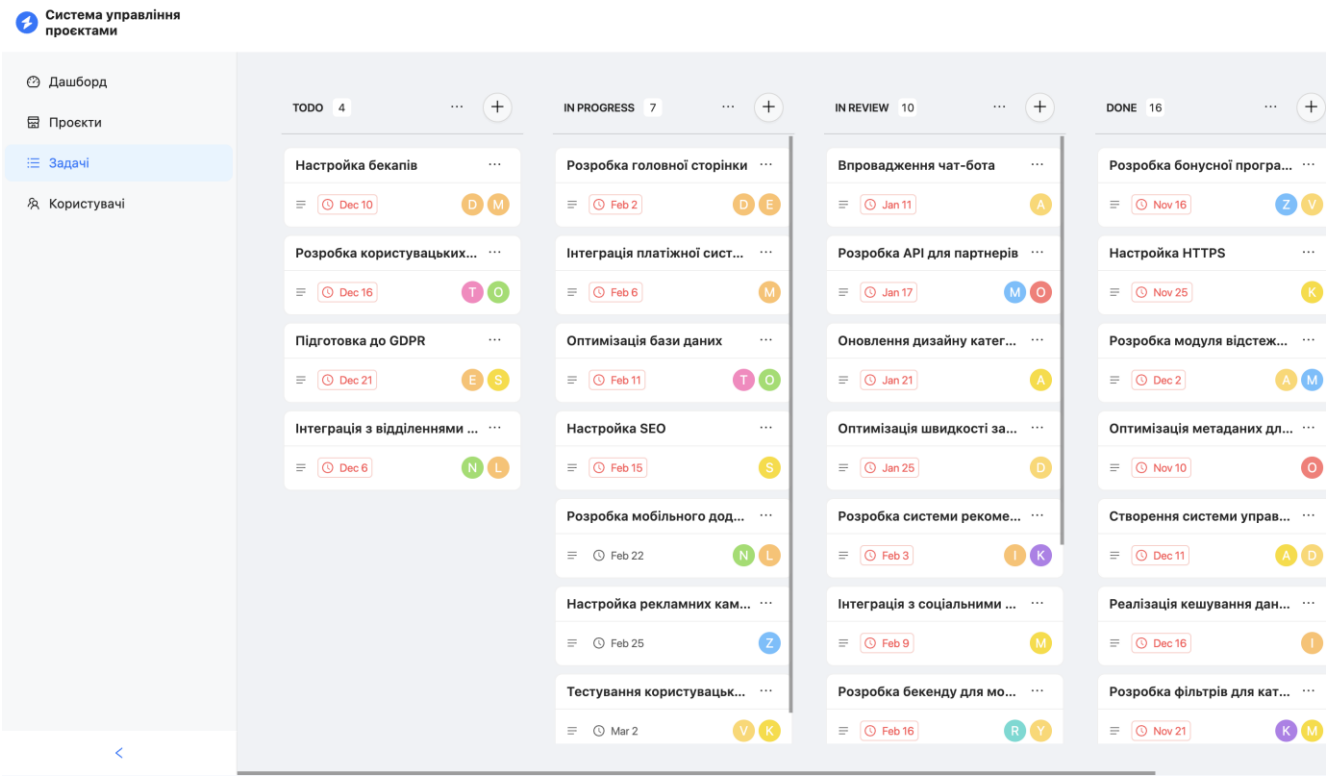


Рисунок 4.30 – Сторінка дошки з задачами

Сторінка представляє собою канбан дошку стандартного вигляду. Задачі розділені на кілька основних стовпців, які відповідають статусу виконання задач (рис. 4.30). Картки можна пересувати між різними стовпцями міняючи їх статус. Для кожного стовпця є можливість додавати нову картку, редагувати назву, видалити стовпець.

Кожна картка містить інформацію про назву задачі, строк виконання а також про учасників, які закріплені на задачі. Картки мають контексте меню, при натисканні на три крапки в верхньому правому куті з’являється список доступних функцій які можна застосувати до неї. А саме «Переглянути детальну інформацію» та «Видалити картку».

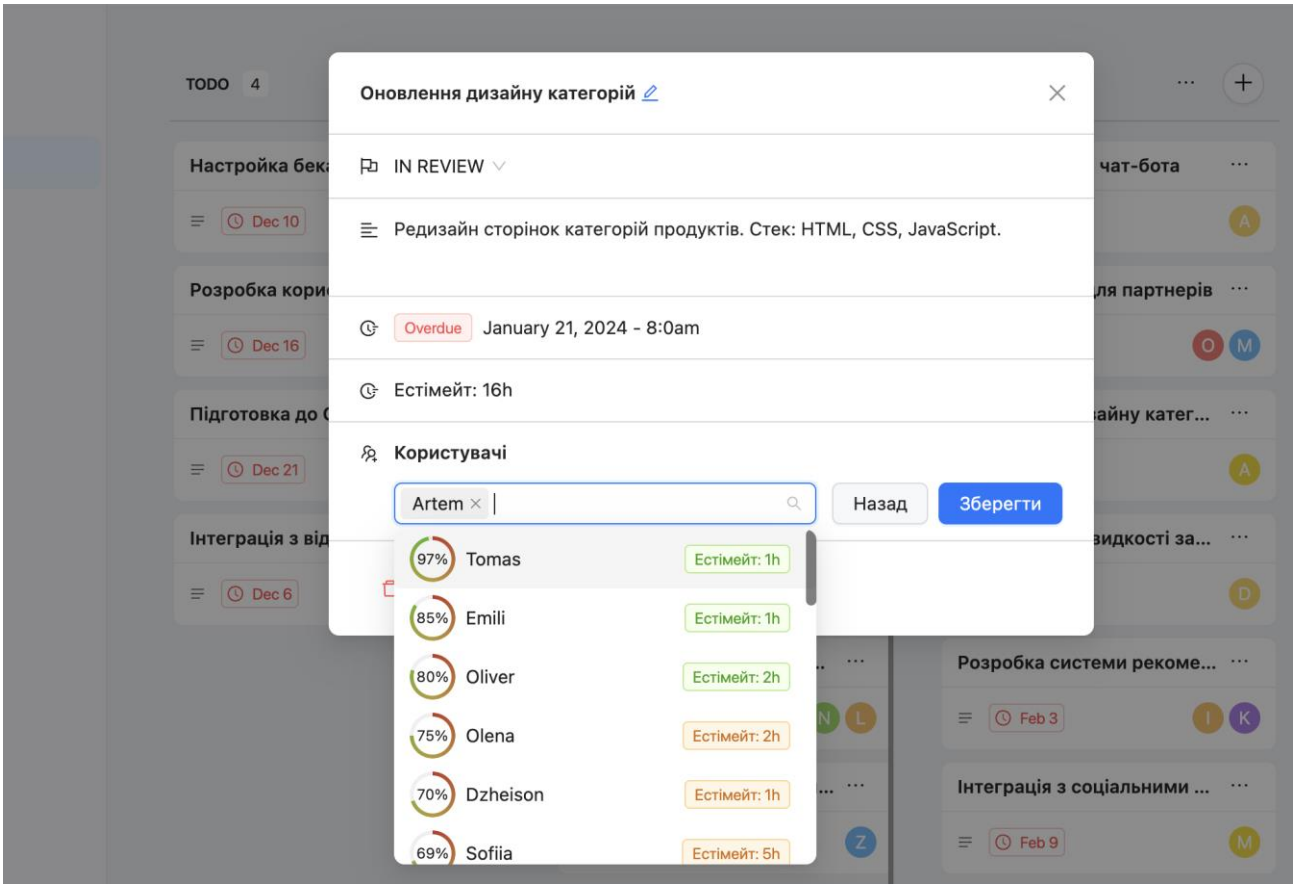


Рисунок 4.31 – Детальна інформація про задачу

На скріншоті (рис. 4.31) представлений попап з детальною інформацією про задачу. В ньому відображається назва задачі, статус, опис, орієнтована дата завершення, естімейт, а також учасники, які закріплені за задачею. Кожне з цих полів можна відредагувати на місці, окрім поля естімейт. Дане значення розраховується автоматично в залежності від учасників, які закріплені за задачею.

Також при редагуванні призначених учасників, в списку з'являється інформація про оцінку кожного учасника, а також розрахований естімейт. Учасники в списку відсортовані у порядку зниження оцінки. Таким чином система підбирає учасників, які найкраще підходять для виконання тієї чи іншої задачі.

Як можна побачити зі скриншоту, при знижені оцінки учасника збільшується прогнозований естімейт на виконання задачі. З цього можна зробити висновок, різні моделі машинного навчання прогнозують приблизно однаковий результат.

Висновки до розділу 4

В ході виконання четвертого розділу, було детально описано етап програмної реалізації системи. Програмна реалізація включає в себе налаштування середовища розробки, розробка моделей машинного навчання, розробка бізнес-логіки та клієнтської частини застосунку.

Також у розділі було продемонстровано користувацький інтерфейс системи. Опис зовнішнього вигляду вебзастосунку подано у вигляді керівництва користувача з описанням основних сторінок та UI елементів.

Результатом виконання даного розділу стало реалізоване програмне забезпечення для автоматизації управління розподілу робочого навантаження при управлінні ІТ-проєктами. Розроблене програмне забезпечення відповідає усім поставленим у першому розділі вимогам, та створено відповідно із спроєктованими моделями розробленими у другому та третьому розділах.

ВИСНОВКИ

При підготовці кваліфікаційної роботи магістра було встановлено основну мету роботи – проєктування і впровадження інноваційної системи для автоматизації процесу розподілення робочого навантаження в межах ІТ-проєктів з використанням методів машинного навчання.

Для досягнення поставленої мети було вирішено ряд задач пов’язаних з аналізом, проєктуванням та розробкою:

– проведено аналіз сучасного стану автоматизації управління проєктами;

– визначено цільові функції системи автоматизації;

– сформовано специфікацію вимог до системи;

– спроєктовано функціональні моделі та архітекту системи;

– розроблено програмний застосунок на базі навченої моделі та провести тестування на запропонованих наборах, проаналізувати отримані результати.

В ході виконання кваліфікаційної роботи було проаналізовано предметну область та систем, які надають функціонал для управління проєктами. Серед проаналізованих систем виступили Trello та Jira. Основним їх недоліком є те, що вони не впроваджують методи машинного навчання для автоматизації процесів управління проєктами. Тому впровадження цих методів в процес управління ІТ-проєктами є доцільним.

Після виявлення специфікації вимог до системи, було створено моделі основних функцій та методів машинного навчання, які система має впровадити. Для цільових функцій було виявлено найбільш підходящі підходи та алгоритми машинного навчання. Серед них автоматизація оцінки складності задач з використанням градієнтного бустингу, прогнозування успішності проєкту за допомогою логістичної регресії, а також адаптація системи кредитного скорингу для автоматизації розподілення задач між учасниками проєкту.

Для системи виявлені найбільш оптимальну архітектуру для з використанням різних сервісів для виконання різних задач. Зокрема розділення клієнтської частини та бізнес логіки, використання Google Cloud SQL для доступу до бази даних, а також використання Google AI Platform Models для зберігання навчених моделей.

Результатом виконання кваліфікаційної роботи магістра є вебзастосунок для автоматизації управління розподілу робочого навантаження при управлінні ІТ-проєктами.

Перевагою розробленого застосунку є те, що для автоматизації рутинних процесів в управлінні проєктами застосовано технології та методи в області машинного навчання, чого в свою чергу не мають аналогічні системи.

В перспективі функціонал системи може бути вдосконалений шляхом додавання нових засобів автоматизації, вдосконалення точності розроблених моделей, інтеграція з іншими великими системами управління, таких як Trello та Jira. Що стосується останнього, інтеграція наддасть такі можливості, як двостороння синхронізація проєктів та задач.

.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Creswell J. W. Research design: qualitative, quantitative, and mixed methods approaches. Sage Publications, Inc, 2002. 246 p.

2. Kuhn T. S. Structure of scientific revolutions. University of Chicago Press, 2009.

3. Automation overview | Trello | Atlassian Support. Atlassian Support. URL: https://support.atlassian.com/trello/docs/butler-overview/(дата звернення: 20.11.2023).

4. Jira Software Automation: Basics & Common Use Cases. Atlassian. URL: https://www.atlassian.com/software/jira/guides/automation/overview (дата звернення: 20.11.2023).

5. Kerzner H. Project management: a systems approach to planning, scheduling, and controlling. 6th ed. New York : Van Nostrand Reinhold, 1998. 1180 p.

6. Shtub A., Rosenwein M. Project management: processes, methodologies, and economics. Pearson, 2016. 736 p.

7. Brynjolfsson E., Mcafee A. Second machine age: work, progress, and prosperity in a time of brilliant technologies. W. W., 2016. 336 p.

8. Fawcett T., Provost F. Data science for business: what you need to know about data mining and data-analytic thinking. O'Reilly Media, Incorporated, 2013.

9. Moon G. E. Parallel algorithms for machine learning. 2019.

10. Dennis A. Systems analysis design. second edition. see notes for publisher info, 2003. 544 p.

11. Robertson S., Robertson J. Complete systems analysis: the workbook, the textbook, the answers. 2-ге вид. Dorset House Publishing Company, Incorporated, 1998. P. 624

12. Захарченко, К., Захарченко, Н., Рудніченко, М., Шибаєва, Н., & Отрадська, Т. (2021). ЗАСТОСУВАННЯ ГЛИБИННОГО НАВЧАННЯ ДЛЯ ПРОГНОЗУВАННЯ ФІНАНСОВИХ ЧАСОВИХ РЯДІВ. *InterConf*, (93), 524-530. https://doi.org/10.51582/interconf.21-22.12.2021.055

13. Андрусенко Ю. О. Аналіз основних моделей прогнозування часових

рядів. *Збірник наукових праць Харківського національного університету Повітряних Сил*. 2020. № 3(65),. С. 91–96. URL: https://doi.org/10.30748/zhups.2020.65.14 (дата звернення: 11.11.2023).

14. Russell S. J., Norvig P. Artificial intelligence: a modern approach. Pearson Publishing, 2021. 1170 p.

15. Hearty J. Advanced machine learning with python. Packt Publishing - ebooks Account, 2016. 278 p.

16. Burns S. S. Python machine learning: machine learning and deep learning with python, scikit-learn, and tensorflow. Independently Published, 2019.

17. Machine learning with pytorch and scikit-learn: develop machine learning and deep learning models with python / D. Dzhulgakov et al. Packt Publishing, Limited, 2022.

18. Bisong E. Introduction to scikit-learn. Building machine learning and deep learning models on google cloud platform. Berkeley, CA, 2019. P. 215–229.

19. Lazzeri F. Machine learning for time series forecasting with python. Wiley & Sons, Incorporated, John, 2020. 224 p.

20. Scopatz A., Huff K. D. Effective computation in physics: field guide to research with python. O'Reilly Media, 2015. 552 p.

21. Мілл І., Хобсон Сейєрс Е. Docker на практиці. Shelter Island: Manning Publications, 2019. 516 с.

22. Moroney L. AI and machine learning for coders: a programmer's guide to artificial intelligence. O'Reilly Media, Incorporated, 2020. 300 p.

23. Jurney R. Agile data science: building data analytics applications with hadoop. Sebastopol, CA : O'Reilly Media, 2013. 178 p.

24. Voron F. Building data science applications with fastapi: develop, manage, and deploy efficient machine learning applications with python. Packt Publishing, Limited, 2021.

25. Peralta J. H. Microservice apis: using python, flask, fastapi, openapi and more. Manning Publications Co. LLC, 2023.

ДОДАТОК А

Лістинг коду для нормалізації датасету задач

```
def clean_and_normalize_tasks(dataset):
    if dataset.empty:
        raise ValueError("The data set is empty. Check the input data.")

    if dataset['project_id'].nunique() != len(dataset):
        raise ValueError("Project IDs are not unique. Check the data.")

    dataset['description'].fillna('', inplace=True)
    dataset['start_date'] = pd.to_datetime(dataset['start_date'],
errors='coerce')
    dataset['end_date'] = pd.to_datetime(dataset['end_date'], errors='coerce')

    dataset = dataset[(dataset['start_date'] < dataset['end_date'])]

    if dataset.empty:
        raise ValueError("All projects have incorrect dates. Check the data.")

    valid_statuses = ['pending', 'in progress', 'cancelled', 'completed']
    dataset = dataset[dataset['status'].isin(valid_statuses)]

    if dataset.empty:
        raise ValueError("All projects have an unacceptable status. Check the
data.")

    dataset = dataset[~((dataset['status'] == 'completed') &
dataset['end_date'].isnull())]

    today = datetime.today().date()
    dataset = dataset[(dataset['end_date'].isnull()) |
(dataset['end_date'].dt.date <= today)]

    if dataset.empty:
        raise ValueError("All projects have incorrect completion dates. Check
the data.")

    normalized_dataset = dataset.copy()
    normalized_dataset['normalized_user'] = (normalized_dataset['user_id'] -
normalized_dataset['user_id'].min()) / (normalized_dataset['user_id'].max() -
normalized_dataset['user_id'].min())
    normalized_dataset['normalized_project'] =
(normalized_dataset['project_id'] - normalized_dataset['project_id'].min()) /
(normalized_dataset['project_id'].max() -
normalized_dataset['project_id'].min())
    normalized_dataset.drop(['user_id', 'project_id'], axis=1, inplace=True)

    return normalized_dataset
```

ДОДАТОК Б

Лістинг коду класу DecisionTree

```
class DecisionTree:
    def __init__(self, max_depth=3):
        self.max_depth = max_depth

    def _split(self, x, y, feature_index, threshold):
        left_mask = x[:, feature_index] < threshold
        right_mask = ~left_mask
        return x[left_mask], y[left_mask], x[right_mask], y[right_mask]

    def _best_split(self, x, y):
        best_loss = float('inf')
        best_feature = None
        best_threshold = None
        for feature_index in range(x.shape[1]):
            thresholds = jnp.unique(x[:, feature_index])
            for threshold in thresholds:
                x_left, y_left, x_right, y_right = self._split(x, y,
feature_index, threshold)
                loss = self._loss(y_left, y_right)
                if loss < best_loss:
                    best_loss = loss
                    best_feature = feature_index
                    best_threshold = threshold
        return best_feature, best_threshold

    def _loss(self, y_left, y_right):
        return jnp.mean((jnp.mean(y_left) - y_left) ** 2) +
jnp.mean((jnp.mean(y_right) - y_right) ** 2)

    def _build_tree(self, x, y, depth=0):
        if depth >= self.max_depth or jnp.var(y) == 0:
            return {'leaf': jnp.mean(y)}

        feature_index, threshold = self._best_split(x, y)
        x_left, y_left, x_right, y_right = self._split(x, y, feature_index,
threshold)

        return {
            'feature_index': feature_index,
            'threshold': threshold,
            'left': self._build_tree(x_left, y_left, depth + 1),
            'right': self._build_tree(x_right, y_right, depth + 1)
        }

    def fit(self, x, y):
        self.tree = self._build_tree(x, y)

    def _traverse_tree(self, tree, sample):
        if 'leaf' in tree:
            return tree['leaf']
        if sample[tree['feature_index']] < tree['threshold']:
            return self._traverse_tree(tree['left'], sample)
        else:
            return self._traverse_tree(tree['right'], sample)

    def predict(self, x):
        return jnp.array([self._traverse_tree(self.tree, sample) for sample in x])
```

ДОДАТОК В

Лістинг коду моделі для прогнозування успішності проєкту за допомогою логістичної регресії

```
class LogisticRegression:
    def __init__(self, learning_rate=0.1, num_epochs=100):
        self.learning_rate = learning_rate
        self.num_epochs = num_epochs
        self.params = None

    def logistic_regression(self, params, X):
        return expit(jnp.dot(X, params))

    def loss_fn(self, params, X, y):
        preds = self.logistic_regression(params, X)
        return -jnp.mean(y * jnp.log(preds) + (1 - y) * jnp.log(1 - preds))

    def train(self, X, y):
        params = random.normal(random.PRNGKey(0), (X.shape[1],))
        for epoch in range(self.num_epochs):
            gradient = grad(self.loss_fn)(params, X, y)
            params = params - self.learning_rate * gradient
            if epoch % 10 == 0:
                print(f"Epoch {epoch}, Loss: {self.loss_fn(params, X, y)}")
        self.params = params

    def predict(self, X):
        return self.logistic_regression(self.params, X)

    def evaluate(self, X, y):
        y_pred = self.predict(X)
        accuracy = jnp.mean((y_pred > 0.5) == y)
        return accuracy
```

ДОДАТОК Г

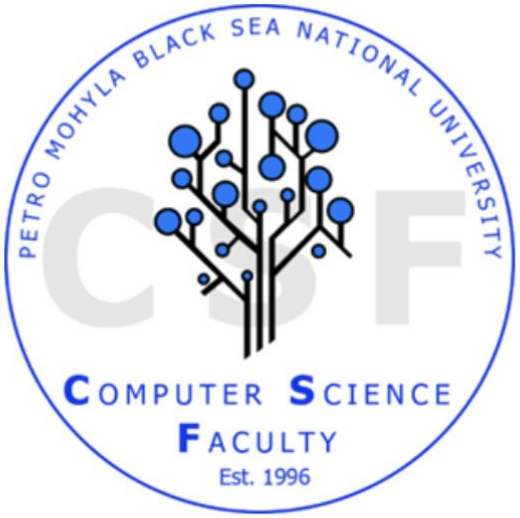
Лістинг коду моделі кредитного скорінгу з використанням градієнтного бустингу

```
class CreditScoringModel:
    def __init__(self, learning_rate=0.1, num_trees=100, max_depth=3):
        self.learning_rate = learning_rate
        self.num_trees = num_trees
        self.max_depth = max_depth
        self.trees = []

    def _logistic_regression(self, params, X):
        return expit(jnp.dot(X, params))

    def _loss_fn(self, params, X, y):
        preds = self._logistic_regression(params, X)
        return -jnp.mean(y * jnp.log(preds) + (1 - y) * jnp.log(1 - preds))

    def _grad_loss_fn(self, params, X, y):
        return grad(self._loss_fn)(params, X, y)

    def _tree_fit(self, X, y):
        params = PRNGKey(0)
        for _ in range(self.num_trees):
            params, subkey = PRNGKey(0), PRNGKey(0)
            X, y = shuffle(subkey, X, y)
            gradient = self._grad_loss_fn(params, X, y)
            params = params - self.learning_rate * gradient
            self.trees.append(params)

    def fit(self, X, y):
        self._tree_fit(X, y)

    def predict(self, X):
        def _predict_single_tree(params, x):
            return self._logistic_regression(params, x)

        def _predict_all_trees(x):
            return jnp.mean(jnp.array([_predict_single_tree(params, x) for
params in self.trees]), axis=0)

        return vmap(_predict_all_trees)(X)
```

ДОДАТОК Д

Апробація кваліфікаційної роботи

Міністерство освіти і науки України
Чорноморський національний університет
імені Петра Могили



«Інформаційні технології та інженерія»

Всеукраїнська науково-практична конференція молодих вчених, аспірантів і студентів

ТЕЗИ

31 січня – 2 лютого 2024 року

Миколаїв – 2024

Методи і засоби програмної інженерії

Вебтехнології та вебдизайн

Інформаційні технології у навчальному процесі