МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут неперервної освіти
Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО
«___»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **«МАГІСТР»**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ»

Тема: «Система підтримки прийняття рішень в управлінні проектною командою**»**

Виконавець: студентка групи УС-201Мз Погрібна Інна Юріївна

Керівник: к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер:_____ Ігор РАЙЧЕВ

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Навчально-науковий інститут неперервної освіти*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 «Інформаційні технології», 122 «Комп'ютерні науки», «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____Аліна САВЧЕНКО
«_____»_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студентки

Погрібній Інні Юріївні
(прізвище, ім'я, по батькові)

1. Тема роботи: «Система підтримки прийняття рішень в управлінні проектною командою» затверджена наказом ректора від «10» жовтня 2023 р. за №2074/ст.

2. Термін виконання роботи: 02.10.2023 – 31.12.2023р.

3. Вихідні дані до роботи: середовище розробки IntelliJ IDEA, фреймворки Angular, NestJS, D3.js, мови програмування Java, JavaScript, TypeScript, база даних MongoDB.

4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, методи, алгоритми та моделі для побудови та оптимізації дерев рішень, структури даних та моделі для представлення та управління деревами рішень, інструменти та технології для візуалізації дерев рішень, розробка прототипу системи підтримки прийняття рішень, аналіз виконаного проектування, висновки.

5. Перелік обов'язкового ілюстративного матеріалу: структура даних в деревах прийняття рішень, архітектура розробленої системи, візуалізація дерев рішень приклади побудови дерев прийняття рішень, всі інші зображення вікон редакторів та їх компонентів.

6. Календарний план-графік

№ п/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1.	Проаналізувати літературу та джерела за темою дипломної роботи	02.10.23 – 08.10.23р.	
2.	Розроблення та затвердження плану дипломної роботи	09.10.23 – 11.10.23р.	
3.	Привести консультації з науковим керівником щодо створення першого розділу	12.10.23 – 16.10.23р.	
4.	Розробка розділу 1	17.10.23 – 28.10.23р.	
5.	Розробка розділу 2	29.10.23 – 19.11.23р.	
6.	Розробка розділу 3	20.11.23 – 01.12.23р.	
8.	Висновки та оформлення пояснювальної записки дипломної роботи	02.12.23 – 13.12.23р.	
9.	Підписання необхідних документів у встановленому порядку	14.12.22 –19.12.23р.	
10.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломної роботи	20.12.23 – 24.12.23р.	

7. Дата видачі завдання: «02» жовтня 2023 р.

Керівник дипломної роботи _____ Тетяна ХОЛЯВКІНА
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Інна ПОГРІБНА
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Система підтримки прийняття рішень в управлінні проектною командою» містить 110 сторінок, 29 рисунків, 2 таблиці, 20 бібліографічних джерел, 4 додатки.

Об'єктом дослідження **є**: процеси управління в IT-проектах.

Предметом дослідження є: система управління проектами в розробці програмного забезпечення.

Мета роботи: розробка системи підтримки прийняття рішень управління проектом в розробці програмного забезпечення.

Методи дослідження включають у себе:

- Інтелектуальний аналіз даних (Data Analytics):
- Нейронні мережі (Neural Networks):
- Машинне навчання (Machine Learning):
- Обробка природних мов (Natural Language Processing, NLP):

Ключові слова: СИСТЕМА УПРАВЛІННЯ ПРОЕКТАМИ, ПІДТРИМКА ПРИЙНЯТТЯ РІШЕНЬ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, IT-ПРОЕКТИ, АНАЛІТИКА ДАНИХ, НЕЙРОННІ МЕРЕЖІ, МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНИХ МОВ, МОДЕЛЮВАННЯ ПРОЕКТІВ, ЕФЕКТИВНЕ УПРАВЛІННЯ ПРОЕКТАМИ, ДАНІ ПРОЕКТУ, ОПТИМІЗАЦІЯ РІШЕНЬ, ІНТЕЛЕКТУАЛЬНІ РІШЕННЯ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, ПЛАНУВАННЯ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

СУП – Система Управління Проектами.

СППР – Система Підтримка Прийняття Рішень.

РПЗ – Розробка Програмного Забезпечення.

ІТ – Інформаційні Технології.

АД – Аналітика Даних.

НМ – Нейронні Мережі.

МН – Машинне Навчання.

ОПНМ – Обробка Природних Мов.

МП – Моделювання Проектів.

ЕУП – Ефективне Управління Проектами.

ДП – Дані Проекту.

ОР – Оптимізація Рішень.

ІР – Інтелектуальні Рішення.

КІ – Користувацький Інтерфейс.

ПП – Планування Проекту.

ВСТУП

Актуальність теми полягає в тому, що система підтримки прийняття рішень (СППР) в управлінні проектною командою в розробці програмного забезпечення є невід'ємною складовою успішного виконання проектів. У сфері розробки програмного забезпечення, де складність завдань і вимоги зростають, необхідність у швидкому та точному прийнятті рішень стає все важливішою.

Управління проектною командою вимагає здатності аналізувати складність завдань, визначати пріоритети, враховувати ризики та координувати роботу багатьох учасників проекту. В такому контексті СППР стає незамінним інструментом, що допомагає керівникам та учасникам команди в прийнятті обґрунтованих та оптимальних рішень.

СППР в управлінні проектною командою в програмній розробці забезпечує набір алгоритмів, моделей та методів, що допомагають в аналізі, оцінці та виборі стратегій дій, вирішенні конфліктів і визначенні оптимального шляху для досягнення цілей проекту. Від побудови дерев рішень та байєсовських мереж до використання агільних методологій та генетичних алгоритмів – існує широкий спектр підходів, які допомагають управляти процесом прийняття рішень з використанням раціонального та систематичного підходу.

У цьому контексті, розуміння та використання СППР в управлінні проектною командою в розробці програмного забезпечення відіграє важливу роль у забезпеченні успішності проектів, збільшенні продуктивності та зниженні ризиків. Використання цих систем допомагає забезпечити ефективність, точність та обґрунтованість прийнятих рішень, що, в свою чергу, сприяє досягненню більшої якості та успішності в розробці програмного забезпечення.

Метою роботи є розробка системи підтримки прийняття рішень управління проектом в розробці програмного забезпечення для покращення

процесів розробки ПЗ та забезпечення більшої точності та ефективності в прийнятті стратегічних рішень в управлінні проектами.

Для досягнення поставленої мети необхідно виконати **наступні завдання**:

1. Аналіз потреб і вимог:

- Вивчення особливостей процесів управління в IT-проектах.

- Визначення вимог до системи підтримки прийняття рішень в контексті розробки програмного забезпечення.

2. Вибір технологій та методологій:

- Вибір інструментів і технологій для розробки системи.

- Визначення методології управління проектом, яка буде використовуватися в системі.

3. Розробка функціоналу системи:

- Розробка модулів для збору, аналізу і візуалізації даних проекту.

- Створення інструментів для прийняття рішень з урахуванням проектних параметрів.

4. Інтеграція з існуючими системами (якщо це потрібно):

- Інтеграція розроблюваної системи з існуючими ПЗ та інструментами, які використовуються в IT-проектах.

5. Тестування та валідація:

- Проведення тестів для перевірки функціональності і надійності системи.

- Валідація системи на практиці в реальних умовах проектів.

6. Впровадження та супровід:

- Впровадження системи в робочий процес управління проектами в IT-сфері.

- Забезпечення підтримки та супроводу після впровадження.

7. Моніторинг та аналіз ефективності:

- Встановлення метрик і засобів моніторингу для оцінки ефективності системи.

- Постійний аналіз результатів і можливість оптимізації системи.

В процесі роботи були використані такі ***методи дослідження:***

1. Інтелектуальний аналіз даних (Data Analytics):

- Збір і аналіз історичних даних проектів розробки ПЗ для виявлення шаблонів і тенденцій у процесі управління проектами.

- Використання статистичних методів для оцінки ризиків і можливостей проекту.

- Розробка моделей прогнозування, які можуть допомогти приймати рішення на основі аналізу даних.

2. Нейронні мережі (Neural Networks):

- Використання нейронних мереж для автоматичного аналізу та класифікації даних проекту, таких як вимоги, часові рамки і ризики.

- Розробка моделей глибокого навчання для прогнозування можливих проблем у проекті та оптимізації управлінських рішень.

- Використання нейронних мереж для автоматичного моніторингу ходу проекту і реагування на зміни в реальному часі.

3. Машинне навчання (Machine Learning):

- Застосування методів машинного навчання для побудови моделей, які можуть прогнозувати ризики та рекомендації з управління проектом.

- Розробка системи рекомендацій, яка може пропонувати оптимальні варіанти дій команді проекту на основі аналізу даних.

4. Обробка природних мов (Natural Language Processing, NLP):

- Використання NLP для аналізу текстової інформації, такої як логи, коментарі та звіти проекту, для виявлення ключових проблем та тенденцій.

Розробка інтерфейсу користувача, який може сприймати та обробляти текстову інформацію для підтримки прийняття рішень.

В роботі досліджується процеси управління проектами в розробці програмного забезпечення. Наведено теоретичні відомості щодо існуючих методів та підходів в управлінні IT-проектами, виявлено їх переваги та

недоліки. Розроблено систему підтримки прийняття рішень управління проектом командою в розробці програмного забезпечення по сучасній методології Kanban. Програмний продукт розроблено відповідно сучасним стандартам та самим новим технологіям, таких як Docker, Node.js, MongoDB, Angular та інші.

Основна **новизна роботи** полягає в інтеграції різноманітних методів та технологій для вдосконалення системи підтримки прийняття рішень в управлінні проектною командою. Робота включає в себе дослідження та застосування методів побудови та оптимізації дерев рішень, а також використання інтелектуальних методів, таких як нейронні мережі, машинне навчання та обробка природньої мови, для аналізу даних проектів і прийняття управлінських рішень.

Крім того, робота містить розробку прототипу системи підтримки прийняття рішень з використанням дерев рішень, що передбачає створення інтерактивного інтерфейсу для користувачів. Такий комплексний підхід дозволяє поєднати традиційні методи управління проектами з інноваційними інтелектуальними підходами, що робить роботу унікальною та значущою для покращення управління проектами в умовах сучасного бізнесу.

РОЗДІЛ 1
МЕТОДИ, АЛГОРИТМИ ТА МОДЕЛІ ДЛЯ ПОБУДОВИ ТА ОПТИМІЗАЦІЇ ДЕРЕВ РІШЕНЬ

1.1. Огляд цілей та методології дослідження

У сучасному світі, де прийняття ефективних та обґрунтованих рішень є ключовим фактором успіху, системи підтримки прийняття рішень знаходять все більше застосувань. Одним з потужних інструментів, що використовуються для аналізу та прийняття рішень, є дерева рішень. Вони забезпечують структурований підхід до моделювання та розуміння складних проблем, допомагаючи управляти невизначеністю та навантаженнями даних.

В цьому розділі розглянемо методи, алгоритми та моделі, що використовуються для побудови та оптимізації дерев рішень. Метою є розкриття основних підходів та їх застосування у сфері прийняття рішень. Для досягнення цієї мети, будуть розглянуті такі ключові аспекти:

1.1.1. Побудова дерев рішень

Дерева рішень є структурою, яка використовується для моделювання прийняття рішень в умовах невизначеності та комплексності. Вони дозволяють представити проблему у вигляді деревоподібної структури, де кожен вузол є рішенням або умовою, а кожне гілля становить можливі варіанти розвитку подій[1].

Розглянемо такі алгоритми та методи для побудови дерев рішень:

1. ID3 (Iterative Dichotomiser 3): Цей алгоритм базується на інформаційній ентропії та виграші інформації для вибору оптимальних

розділів даних на кожному кроці побудови дерева. ID3 шукає найбільш інформативний атрибут для розбиття даних та продовжує рекурсивно будувати дерево досягнення листових вузлів.

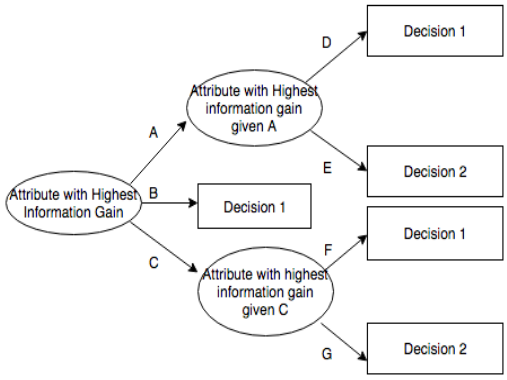


Рис.1.1. ID3 (Iterative Dichotomiser 3)

2. C4.5: Це покращена версія алгоритму ID3, яка дозволяє обробляти неповні або відсутні дані. C4.5 використовує міру виграшу інформації та інші фактори для вибору оптимальних розділів та побудови дерева.

3. CART (Classification and Regression Trees): Цей алгоритм може використовуватись для побудови дерев класифікації та дерев регресії. CART використовує критерій Джині для вибору оптимальних розділів та мінімізації помилки прогнозування.

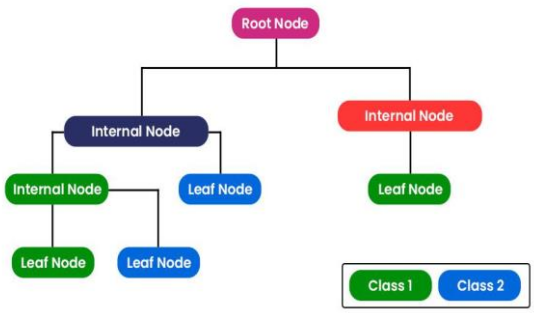


Рис.1.2. Classification and Regression Trees

1.1.2. Оптимізація дерев рішень

Розглянемо методи та підходи до оптимізації дерев рішень. Оптимізація дерев рішень включає в себе вибір оптимальної структури дерева, оптимальне розбиття вузлів та управління складністю моделі. Деякі з методів оптимізації, які будуть розглянуті, включають:

- Обрізка дерева (Tree Pruning): Цей метод використовується для зменшення складності моделі та уникнення перенавчання шляхом видалення незначущих гілок або злиття вузлів.
- Регуляризація (Regularization): Це підходи, що додають штрафні члени до функції втрати, щоб управляти складністю моделі та зменшити ймовірність перенавчання.
- Ансамбль дерев (Tree Ensembles): Ансамбль методів, таких як Random Forests та Gradient Boosting, використовує декілька дерев рішень, щоб отримати більш точні й стійкі прогнози. Ці методи комбінують декілька дерев рішень та використовують різні стратегії для покращення точності та зменшення перенавчання.

Використання дерев рішень у системах підтримки прийняття рішень для управління проектною командою в розробці програмного забезпечення може допомогти зробити обґрунтовані та ефективні рішення на основі аналізу даних та управління невизначеністю. У наступних розділах звіту будуть розглянуті практичні приклади застосування дерев рішень у сфері управління проектною командою в розробці програмного забезпечення[2].

1.2. Структури даних та моделі для представлення та управління деревами рішень

У попередньому розділі ми розглянули методи та алгоритми для побудови та оптимізації дерев рішень. Однак, для ефективного використання дерев рішень у системах підтримки прийняття рішень, необхідно також мати

ефективні структури даних та моделі для їх представлення та управління. У цьому розділі ми розглянемо різні структури даних та моделі, що використовуються для цілей представлення та управління деревами рішень.

1.2.1. Структура даних дерева рішень

Одним з найпоширеніших способів представлення дерев рішень є структура даних, що використовується для збереження їх структури та інформації вузлів. Найпростішою формою структури даних дерева рішень є структура зв'язаних об'єктів, де кожен вузол містить посилання на його батьківський вузол та його дочірні вузли (якщо вони є). Кожен вузол також містить інформацію про умову або рішення, що пов'язані з цим вузлом.

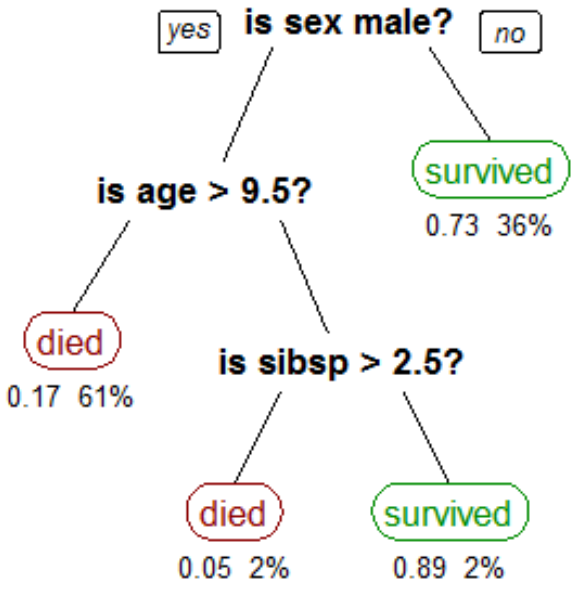


Рис.1.3. Приклад структури даних дерева рішень

Залежно від конкретного завдання та вимог, можуть використовуватись інші структури даних для представлення дерев рішень. Наприклад, дерева

рішень можуть бути представлені у вигляді матриць, таблиць або графів залежностей. Кожен підхід має свої переваги та обмеження, і вибір конкретної структури даних залежить від конкретного контексту та потреб системи[3].

1.2.2. Моделі управління деревами рішень

Крім структур даних, для ефективного управління деревами рішень також необхідні моделі, які дозволяють здійснювати різноманітні операції, такі як додавання та видалення вузлів, оновлення інформації вузлів, пошук шляхів та інші.

Одна з поширених моделей управління деревами рішень – це *модель рекурсивних функцій*, де кожна функція відповідає операції над вузлами дерева. Рекурсивні функції дозволяють легко реалізувати операції, які потребують обходу дерева, такі як обчислення сумарних значень, пошук певного вузла або виконання певної дії на кожному вузлі.

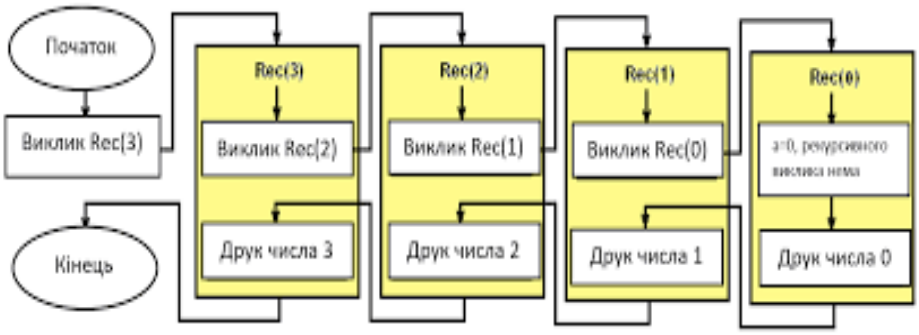


Рис.1.4. Приклад рекурсивної функції

Крім того, існують інші моделі, такі як модель ітераторів, які дозволяють послідовно обходити вузли дерева, або модель подій, яка використовує події та сповіщення для взаємодії з деревом та його вузлами.

1.3. Інструменти та технології для візуалізації дерев рішень

У попередніх розділах ми розглянули методи та моделі для побудови та оптимізації дерев рішень, а також структури даних та моделі для їх представлення та управління. Однак, для ефективного використання дерев рішень у системах підтримки прийняття рішень, важливо мати інструменти та технології, що дозволяють візуалізувати дерева рішень та створювати користувацький інтерфейс для зручної роботи з ними. У цьому розділі ми розглянемо різні інструменти та технології, що використовуються для візуалізації дерев рішень та створення користувацького інтерфейсу[3].

1.3.1. Інструменти візуалізації дерев рішень

Інструменти візуалізації дерев рішень дозволяють графічно відображати структуру та характеристики дерева. Вони надають зручний спосіб представлення інформації та взаємодії з деревом. Деякі з поширених інструментів візуалізації дерев рішень включають:

- Graphviz: Це потужна бібліотека та набір інструментів для візуалізації графів, включаючи дерева рішень. Graphviz надає різноманітні можливості настройки та стилізації візуалізацій, що дозволяє створювати привабливі та інформативні діаграми дерев рішень.
- D3.js: Ця JavaScript-бібліотека дозволяє створювати інтерактивні візуалізації дерев рішень за допомогою веб-технологій. D3.js надає потужні інструменти для маніпулювання даними та створення динамічних та змінних візуалізацій.

1.3.2. Користувацький інтерфейс для роботи з деревами рішень

Крім візуалізації, також важливо мати користувацький інтерфейс, що дозволяє зручно взаємодіяти з деревами рішень. Користувацький інтерфейс

повинен надавати зручні інструменти для перегляду, редагування та аналізу дерев рішень. Деякі з популярних технологій та підходів для створення користувацького інтерфейсу включають:

- HTML/CSS та JavaScript: Використання веб-технологій дозволяє створити інтерактивний інтерфейс, який працює в браузері. HTML/CSS використовується для створення розмітки та оформлення, а JavaScript – для програмної логіки та взаємодії з деревом рішень.
- GUI-фреймворки: Існують різні GUI-фреймворки, такі як Qt, Tkinter, wxWidgets, які дозволяють створювати користувацький інтерфейс з використанням графічних елементів та інструментів.

У цьому розділі ми розглянули різні інструменти та технології для візуалізації дерев рішень та створення користувацького інтерфейсу для роботи з ними. Візуалізація дерев рішень та зручний користувацький інтерфейс є важливими складовими для ефективного використання дерев рішень у системах підтримки прийняття рішень. У наступних розділах звіту будуть розглянуті приклади застосування цих інструментів та технологій у практичних сценаріях роботи з деревами рішень[4].

1.4. Прототип системи підтримки прийняття рішень з використанням дерев рішень

У попередніх розділах ми досліджували методи, алгоритми та моделі для побудови та оптимізації дерев рішень, а також інструменти та технології для їх візуалізації та створення користувацького інтерфейсу. У цьому розділі ми розглянемо створення прототипу системи підтримки прийняття рішень з використанням дерев рішень, використовуючи технології ***Angular, D3.js, MongoDB та NestJS***.

1.4.1. Архітектура системи

Прототип системи підтримки прийняття рішень базується на клієнт-серверній архітектурі. Клієнтська частина розроблена за допомогою Angular, що забезпечить динамічний та інтерактивний користувацький інтерфейс. Серверна частина реалізована за допомогою NestJS, який забезпечить серверну логіку та комунікацію з базою даних MongoDB.

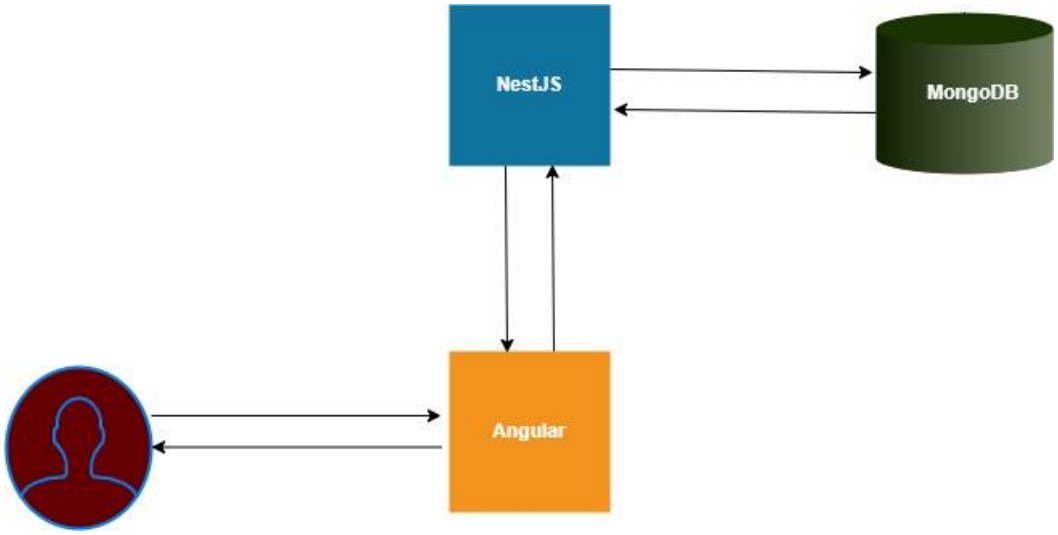


Рис.1.5. Архітектура розробленої системи

1.4.2. Робота з деревами рішень

Для побудови та оптимізації дерев рішень у прототипі системи буде використовуватись алгоритм, який був розглянутий в попередніх розділах звіту. За допомогою цього алгоритму будуть створюватись та модифікуватись дерева рішень відповідно до потреб користувача.

Приклад JavaScript-функцій для створення та модифікації дерева рішень:

```
// Модель вузла дерева рішень
class TreeNode {
```

```
  constructor(data) {
    this.data = data;
    this.children = [];
  }
}
// Створення нового вузла дерева
function createNode(data) {
  return new TreeNode(data);
}
// Додавання дочірнього вузла до батьківського вузла
function addChild(parentNode, childNode) {
  parentNode.children.push(childNode);
}
// Видалення вузла з дерева
function removeNode(rootNode, targetNode) {
  const index = rootNode.children.findIndex(node => node === targetNode);
  if (index > -1) {
    rootNode.children.splice(index, 1);
  } else {
    rootNode.children.forEach(node => removeNode(node, targetNode));
  }
}
// Редагування даних вузла
function editNode(node, newData) {
  node.data = newData;
}
// Приклад використання
// Створення кореневого вузла дерева
const root = createNode("Root");
// Створення дочірніх вузлів
```

```
const node1 = createNode("Node 1");
const node2 = createNode("Node 2");
const node3 = createNode("Node 3");
// Додавання дочірніх вузлів до кореневого вузла
addChild(root, node1);
addChild(root, node2);
addChild(root, node3);
// Видалення вузла node2
removeNode(root, node2);
// Редагування вузла node3
editNode(node3, "Modified Node 3");
console.log(root);
```

Цей код демонструє створення нового дерева рішень з кореневим вузлом "Root" та додавання, видалення та редагування вузлів. Можна використовувати ці функції для створення та модифікації дерева рішень у нашому прототипі.

1.4.3. Візуалізація дерев рішень

Для візуалізації дерев рішень у прототипі використовується D3.js. Ця бібліотека дозволяє створити інтерактивні та привабливі візуалізації, які демонструють структуру та характеристики дерев рішень. Користувачі можуть легко навігувати по дереву, редагувати його та аналізувати рішення.

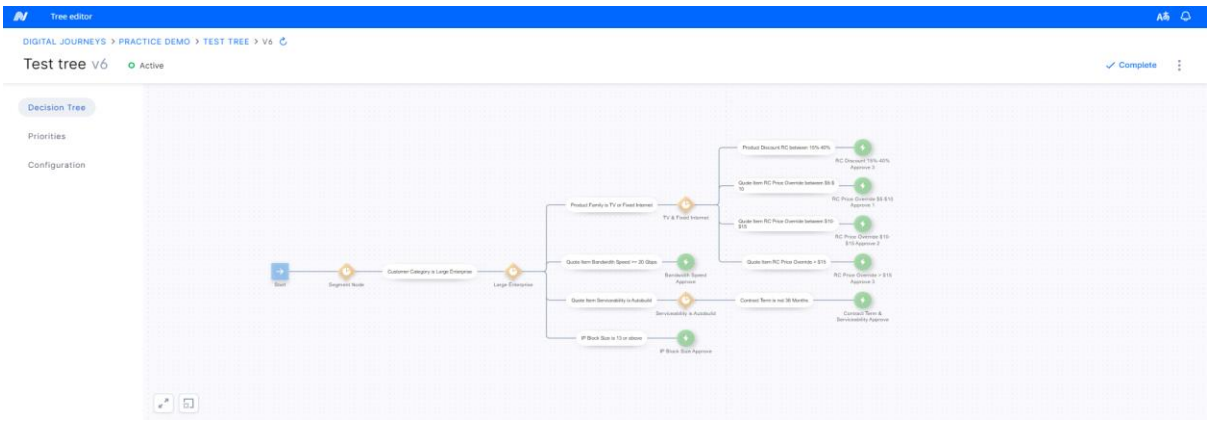


Рис. 1.6. Візуалізація дерев рішень в розробленому прототипі

Сторінка редагування дерева рішень: інтерфейс для редагування структури дерева, додавання та видалення вузлів, встановлення вагових коефіцієнтів тощо.

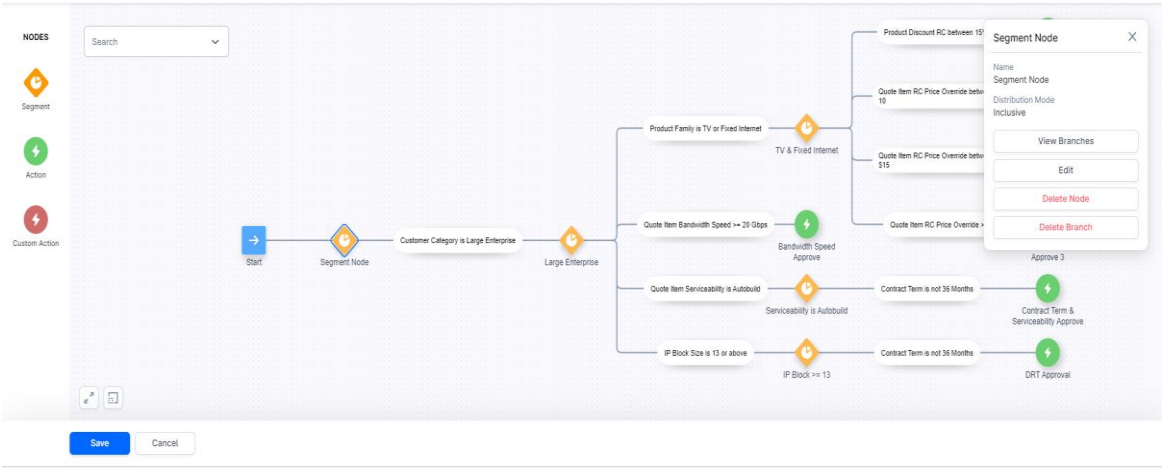


Рис. 1.7. Редагування дерева рішень

1.4.4. Збереження та взаємодія з даними

У процесі розробки системи підтримки прийняття рішень з використанням дерев рішень, велику увагу було приділено аспектам збереження та взаємодії з даними. Забезпечення ефективного та безперебійного збереження дерева рішень та здатність взаємодіяти з ним є критичними функціональними вимогами до системи. Розглянемо деталі та

рекомендації щодо збереження та взаємодії з даними у системі підтримки прийняття рішень. Вибір бази даних: Один з перших кроків полягає в обранні підходящої бази даних для збереження дерева рішень. Рекомендується використовувати нереляційні бази даних, такі як MongoDB, оскільки вони забезпечують гнучкість та швидкий доступ до даних. MongoDB пропонує документно-орієнтований підхід, що дозволяє зберігати дерева рішень у вигляді документів, що містять структуровану інформацію.

1. ***Модель даних:*** При проектуванні моделі даних для збереження дерева рішень, рекомендується використовувати структури даних, що підтримують ієрархічну структуру, наприклад, деревоподібні структури або вкладені набори даних. Крім того, слід враховувати, які додаткові властивості дерева рішень потрібно зберігати, наприклад, вагові коефіцієнти, статуси рішень тощо.

2. ***Запити до бази даних:*** Для ефективної взаємодії з даними, слід ретельно планувати та оптимізувати запити до бази даних. Запити повинні бути спрямовані на швидкий доступ до потрібних даних, а також на забезпечення консистентності та цілісності даних. Використання індексів та оптимізація запитів можуть допомогти покращити продуктивність системи.

3. ***Забезпечення безпеки даних:*** Важливим аспектом є забезпечення безпеки даних, особливо коли йдеться про конфіденційну інформацію про рішення та проектну команду. Рекомендується використовувати механізми аутентифікації та авторизації, шифрування даних та інші заходи безпеки для захисту даних в системі.

4. ***Резервне копіювання та відновлення даних:*** Забезпечення резервного копіювання та відновлення даних є важливим кроком для збереження дерева рішень в разі випадкового видалення або пошкодження даних. Рекомендується регулярно створювати резервні копії даних та перевіряти їх відновлення для забезпечення безпеки та доступності даних.

Отже були розглянуті основні аспекти, пов'язані з ефективним збереженням та взаємодією з даними в системі підтримки прийняття рішень з

використанням дерев рішень. Використання підходящої бази даних, оптимізація запитів, забезпечення безпеки та резервного копіювання даних є ключовими чинниками для успішної роботи системи.

1.4.5. Створення користувацького інтерфейсу

Створення користувацького інтерфейсу є важливою складовою системи підтримки прийняття рішень з використанням дерев рішень. Ефективний та зручний інтерфейс дозволяє користувачам легко взаємодіяти з системою, робити розрахунки, аналізувати дані та приймати обґрунтовані рішення. У цьому підрозділі будуть розглянуті деталі та рекомендації щодо створення користувацького інтерфейсу для системи підтримки прийняття рішень з використанням дерев рішень.

Навігація та організація інформації: Важливим аспектом створення користувацького інтерфейсу є зручна навігація та організація інформації. Користувач повинен легко знаходити необхідні функції та дані, а також мати доступ до різних розділів системи. Рекомендується використовувати зрозумілу ієрархію меню та вкладок, пошукові функції та можливості фільтрації для полегшення навігації.

Візуалізація дерева рішень: Одним з ключових елементів користувацького інтерфейсу є візуалізація дерева рішень. Забезпечення чіткого та зрозумілого відображення структури дерева, включаючи вузли, гілки, зв'язки та атрибути, є важливим завданням. Рекомендується використовувати графічні елементи, такі як діаграми, дерева, схеми тощо, для наглядного представлення дерева рішень.

1. ***Взаємодія з вузлами дерева:*** Користувач повинен мати можливість додавати нові вузли, видаляти та редагувати існуючі вузли дерева. Для цього можна використовувати контекстне меню, кнопки дій або перетягування вузлів. Забезпечення інтуїтивно зрозумілих та простих способів взаємодії з вузлами сприяє зручності в редагуванні дерева рішень.

2. ***Відображення атрибутів вузлів:*** Кожен вузол дерева рішень може мати певні атрибути, такі як назва, опис, вага тощо. Важливо забезпечити їх відображення у користувацькому інтерфейсі. Можна використовувати форми, таблиці або попап-вікна для введення та відображення атрибутів вузлів. Додаткові можливості, такі як розрахунки або графіки, також можуть бути використані для аналізу та відображення атрибутів.

3. ***Користувацькі налаштування:*** Рекомендується надати користувачам можливість налаштування інтерфейсу залежно від їхніх потреб. Наприклад, вони можуть вибрати тип візуалізації дерева, встановити свої персоналізовані кольорові схеми або налаштувати сповіщення та повідомлення системи.

За допомогою Angular розроблений користувацький інтерфейс прототипу системи. Інтерфейс забезпечує зручну навігацію, редагування дерев рішень, аналіз результатів та інші функції, необхідні для роботи з деревами рішень.

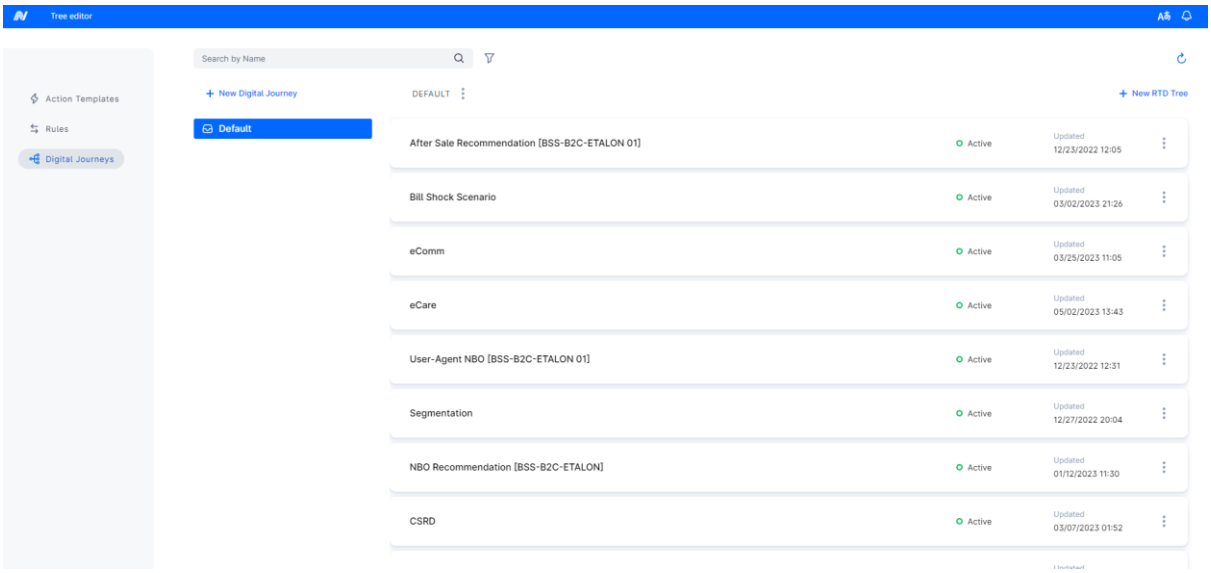


Рис. 1.8. Головна сторінка прототипу

Загальною метою створення користувацького інтерфейсу є забезпечення зручності в роботі з системою підтримки прийняття рішень з використанням дерев рішень. Рекомендується проводити тестування ітераційно та отримувати зворотній зв'язок від користувачів для вдосконалення інтерфейсу та врахування їхніх потреб та пропозицій.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділу на базі результатів з практики по розробці системи підтримки прийняття рішень з використанням дерев рішень, було отримано цінний досвід та результати, які дають змогу зробити наступні висновки та рекомендації щодо подальшого розвитку та вдосконалення системи:

● ***Ефективність дерева рішень:*** Використання дерев рішень у системі підтримки прийняття рішень дозволяє структурувати рішення та їх альтернативи, забезпечуючи зрозумілу та логічну модель для аналізу та вибору оптимального шляху дій.

● ***Гнучкість та розширюваність:*** Система підтримки прийняття рішень з використанням дерев рішень дозволяє легко додавати, видаляти та модифікувати вузли дерева, а також змінювати їх вагові коефіцієнти.

● ***Візуалізація та інтерактивність:*** Використання візуалізаційних засобів, таких як бібліотека D3.js, дозволяє графічно відобразити дерево рішень та його характеристики. Це дозволяє зрозуміло представити взаємозв'язки між рішеннями та спростити процес прийняття рішень. Крім того, створення користувацького інтерфейсу з використанням технології Angular дозволяє створити зручне середовище для роботи з деревом рішень.

На підставі отриманих результатів, рекомендується подальше розвитку та вдосконалення системи підтримки прийняття рішень з використанням дерев рішень за такими напрямками:

1. ***Розширення функціональності:*** Додавання можливостей для аналізу та порівняння різних сценаріїв та альтернатив, розрахунку ризиків та економічної ефективності рішень. Це допоможе забезпечити більш об'єктивне та детальне прийняття рішень.

2. ***Удосконалення візуалізації:*** Розробка більш інтерактивного та зручного інтерфейсу для відображення дерева рішень, включаючи можливість розгортання/згортання гілок, масштабування, додаткових анімаційних ефектів

тощо. Це поліпшить користувацький досвід та зробить систему більш привабливою для користувачів.

3. ***Інтеграція з іншими інструментами:*** Розробка можливостей інтеграції системи підтримки прийняття рішень з іншими інструментами розробки програмного забезпечення, такими як управління проектами, системи контролю версій, комунікаційні платформи тощо. Це сприятиме більш плавному та зручному використанню системи в контексті розробки програмного забезпечення.

Загалом, подальший розвиток та вдосконалення системи підтримки прийняття рішень з використанням дерев рішень дозволить забезпечити більш ефективне та обґрунтоване управління проектною командою в процесі розробки програмного забезпечення.

РОЗДІЛ 2
АНАЛІЗ ТА ВИКОРИСТАННЯ ІНТЕЛЕКТУАЛЬНИХ МЕТОДІВ В СППР УПРАВЛІННЯ ПРОЕКТАМИ

2.1. Інтелектуальний аналіз даних в СППР управління проектами

Інтелектуальний аналіз даних (ІАД) в системах підтримки прийняття рішень (СППР) для управління проектами – це важливий компонент сучасного підходу до ефективного управління проектами. Він поєднує в собі аналітичні методи, алгоритми машинного навчання і обробку даних для підтримки керівників проектів у прийнятті обґрунтованих рішень та оптимізації ресурсів. ІАД дозволяє зрозуміти і прогнозувати різні аспекти проекту, від тривалості до витрат, від ризиків до розподілу ресурсів, що допомагає покращити результати проекту та зменшити його ризики[5].

2.1.1. Збір та аналіз історичних даних проектів для виявлення шаблонів і тенденцій в процесі управління проектами

Збір та аналіз історичних даних проектів є важливою частиною ефективного управління проектами. В цьому підрозділі ми розглянемо процес збору та аналізу історичних даних проектів, а також визначимо важливість цього процесу для вдосконалення управління проектами.

Важливість збору історичних даних

Аналіз історичних даних проектів дозволяє отримувати цінні висновки з попередніх проектів та використовувати їх для покращення якості та результативності майбутніх проектів. Важливість збору історичних даних полягає в наступному:

1. *Виявлення шаблонів та тенденцій:* Аналіз історичних даних дозволяє виявляти шаблони та тенденції в різних аспектах проектів, таких як тривалість виконання, бюджет, ризики, якість та інші параметри. Це допомагає передбачити можливі проблеми та уникнути них у майбутніх проектах.

2. *Покращення процесів:* Аналіз історичних даних надає можливість ідентифікувати найбільш ефективні стратегії та практики, які допомагали досягати успішних результатів. Це сприяє покращенню процесів управління проектами.

3. *Управління ризиками:* Аналіз даних попередніх проектів дозволяє ідентифікувати потенційні ризики та шляхи їх уникнення. Це допомагає зменшити ймовірність негативних наслідків у майбутніх проектах.

Збір історичних даних проектів

Для збору історичних даних проектів використовуються різні джерела, включаючи документацію проектів, звіти, реєстри ризиків, фінансову звітність, інформацію про виконавців та інші джерела. Дані повинні бути систематично зібрані, структуровані та збережені для подальшого аналізу.

Аналіз історичних даних проектів

Аналіз історичних даних включає в себе використання різних методів, таких як статистичний аналіз, візуалізація даних, аналіз часових рядів та інші методи. Цей аналіз допомагає виявити ключові патерни, тенденції та причини, що впливають на результати проектів.

Використання результатів аналізу для управління проектами

Результати аналізу історичних даних використовуються для прийняття рішень щодо покращення процесів управління проектами. Це може включати в себе впровадження нових стратегій, коригування бюджету та графіку, підвищення якості виконання та зменшення ризиків.

Збір та аналіз історичних даних проектів є важливою складовою успішного управління проектами. Цей процес допомагає виявити шаблони, тенденції та вивчені уроки, які сприяють покращенню ефективності та результативності майбутніх проектів.

2.1.2. Використання статистичних методів для оцінки ризиків і можливостей проекту

У цьому підрозділі ми розглянемо важливість використання статистичних методів для об'єктивної оцінки ризиків і можливостей, що виникають під час реалізації проекту. Використання статистичних методів дозволяє ретельно аналізувати ймовірність подій та їх вплив на фінансовий результат проекту.

Використання статистичних методів для ідентифікації ризиків та можливостей

1. *Статистичний аналіз історичних даних:* Для оцінки ризиків та можливостей можна використовувати статистичний аналіз історичних даних проектів[6]. Однією з ключових формул є розрахунок ймовірності ризику (P_r) на основі історичних даних:

$$\boldsymbol{P_r} = \frac{\text{Кількість разів, коли ризик виник, в історичних даних}}{\text{Загальна кількість проектів у історичних даних}} \tag{2.1}$$

Приклад: У історичних даних з 100 проектів 20 разів виникали ризики. Тоді ймовірність ризику $\boldsymbol{P_r} = \frac{20}{100} = 0.2$.

2. *Співвідношення ймовірностей:* Для порівняння ризиків та можливостей можна використовувати співвідношення ймовірностей. Формула для співвідношення (*ROR*) ризику (P_r) та можливості (P_o) така:

$$\boldsymbol{ROR} = \frac{P_r}{P_o} \tag{2.2}$$

Приклад: Якщо ймовірність ризику $P_r = 0.2$, а ймовірність можливості $P_o = 0.4$, то співвідношення $ROR = \frac{0.2}{0.4} = 0.5$.

Аналіз впливу ризиків та можливостей

1. *Метод монте-карло:* Для аналізу впливу ризиків і можливостей на фінансовий результат проекту можна використовувати метод монте-карло. Цей метод передбачає симуляцію великої кількості можливих сценаріїв розвитку подій.

Приклад:

- *Ризик:* Затримка впливає на фінансовий результат на суму 100,000 одиниць з ймовірністю 0.3.
- *Можливість:* Впровадження нової технології впливає на фінансовий результат на суму 50,000 одиниць з ймовірністю 0.2.

Проведемо симуляцію 1,000 можливих сценаріїв і обчислимо фінансовий результат (*NPV*) для кожного:

$NPV = 0.7 \cdot 100{,}000 - 0.8 \cdot 50{,}000$

Розрахунок середнього фінансового результату (*ExpectedNPV*) на основі симуляції:

$$ExpectedNPV = \frac{1}{1000}\sum_{i=1}^{1000} NPVi \qquad (2.3)$$

2. *Аналіз варіацій:* Використання аналізу варіацій для ідентифікації чинників, які найбільше впливають на ризики та можливості проекту. Це допомагає зосередити зусилля на ключових аспектах управління ризиками та можливостями. Формула для аналізу варіацій включає в себе обчислення варіацій у впливі різних факторів на проект.

Використання статистичних методів для оцінки ризиків та можливостей проекту дозволяє об'єктивно визначити ймовірні наслідки подій та приймати обгрунтовані рішення щодо управління проектом. Цей підхід допомагає зменшити невизначеність та покращити результативність управління проектами.

2.1.3. Розробка моделей прогнозування, для прийняття рішення на основі аналізу даних

У цьому підрозділі ми розглянемо важливість розробки моделей прогнозування на основі аналізу даних для прийняття обґрунтованих рішень в IT проектах. Моделі прогнозування дозволяють передбачати розвиток подій і вибирати оптимальні стратегії.

Розробка моделей прогнозування для IT проекту

1. *Вибір моделі прогнозування:* Перший крок – вибір відповідної моделі прогнозування для конкретного IT проекту. Це може бути модель часового ряду, класифікації, регресії або інша в залежності від завдання.

2. *Збір та підготовка даних:* Для розробки моделі потрібні дані. Збережіть, очистіть та підготуйте дані для подальшого аналізу.

3. *Побудова моделі:* На основі підготовлених даних розробляється модель прогнозування. Вибрана модель може мати власну структуру та параметри.

Приклад: Прогнозування завантаженості серверів

Розглянемо приклад прогнозування завантаженості серверів в IT проекті. Ми хочемо побудувати модель, яка допоможе передбачати завантаженість серверів на основі історичних даних та інших факторів, таких як обсяг запитів та час доби.

Вибір моделі: У цьому випадку, ми можемо вибрати модель регресії для прогнозування завантаженості серверів, оскільки ми сподіваємося знайти залежність між факторами та завантаженістю.

Зібрання та підготовка даних: Збираємо дані про завантаженість серверів, обсяги запитів, час доби та інші важливі параметри. Потім очищаємо та підготовлюємо дані для аналізу[7].

Побудова моделі: Модель регресії може бути виражена формулою:

$$ServerLoad = \beta_0 + \beta_1 \cdot RequestVolume + \beta_2 \cdot TimeOfDay + \epsilon \quad (2.4)$$

Де:

ServerLoad – прогнозована завантаженість сервера.

RequestsVolume – обсяг запитів.

TimeOfDay – час доби.

$\beta_0, \beta_1, \beta_2$ – коефіцієнти регресії, які визначаються під час навчання моделі.

ϵ – помилка моделі.

Після побудови моделі і навчання її на історичних даних, ми можемо використовувати її для прогнозування завантаженості серверів у майбутньому на основі значень обсягів запитів та часу доби.

Цей приклад демонструє процес розробки моделей прогнозування для IT проекту, де формули та приклади використовуються для побудови моделі прогнозування та прийняття рішень на основі аналізу даних. Моделі прогнозування можуть бути корисними інструментами для оптимізації ресурсів та планування дій в IT-сфері.

2.2. Нейронні мережі в системах управління проектами

Сучасне управління проектами вимагає інноваційних підходів та інструментів, щоб впоратися зі складними завданнями та викликами. Одним з найбільш потужних інструментів, що став важливим у сфері управління проектами, є нейронні мережі. Ці штучні інтелектуальні системи дозволяють аналізувати дані, робити прогнози та приймати рішення на основі великих обсягів інформації.

Нейронні мережі, інспіровані біологічною структурою людського мозку, здатні до навчання на великій кількості даних і виявлення складних залежностей. Вони здійснюють аналіз, розпізнавання патернів і прогнозування на рівні, недосяжному для класичних методів[9].

Цей розділ присвячений вивченню застосування нейронних мереж у системах управління проектами. Ми дослідимо, як ці інтелектуальні системи можуть покращити управління проектами, зробити прогнози та допомогти в

прийнятті рішень. При цьому будемо розглядати основи нейронних мереж, методи їхнього навчання, а також конкретні сценарії використання у сфері управління проектами.

2.2.1. Використання нейронних мереж для автоматичного аналізу та класифікації даних проекту

Нейронні мережі – це потужний інструмент для автоматичного аналізу та класифікації даних в управлінні IT проектами. Вони можуть допомогти вирішити різноманітні завдання, такі як прогнозування термінів завершення проектів, виявлення ризиків та здатність аналізувати великі обсяги даних для прийняття управлінських рішень. Ось кілька прикладів використання нейронних мереж в управлінні IT проектами:

Таблиця 2.1

Приклади використання нейронних мереж в управлінні IT проектами

Задача	Модель	Приклад застосування
Прогнозування термінів завершення проектів	Рекурентні НМ	Прогнозування дати завершення на основі історичних даних та параметрів проекту.
Виявлення ризиків	Глибокі НМ	Аналіз текстових звітів та виявлення ключових слів, що вказують на ризики.
Автоматична класифікація завдань та пріоритетів	Моделі NLP	Класифікація завдань на основі текстового опису та визначення їх пріоритетів.
Управління ресурсами та розподіл завдань	Алгоритми RL	Розподіл завдань між командами або ресурсами для оптимізації використання.
Аналіз великих обсягів даних для стратегічного управління	Глибокі НМ	Аналіз даних для виявлення тенденцій та можливостей у стратегічному плануванні.

Приклади які наведені в таблиці вказують на широкий спектр можливостей, які надають нейронні мережі в управлінні ІТ проектами, від покращення передбачення до аналізу текстових даних та оптимізації ресурсів. Успішне впровадження таких рішень може покращити ефективність та продуктивність ІТ проектів, зменшити ризики та допомогти приймати більш обґрунтовані управлінські рішення.

2.2.2. Розробка моделі глибокого навчання для прогнозування можливих проблем у проекті та оптимізації управлінських рішень

Управління проектами набуває нові риси завдяки використанню глибокого навчання. У цьому підрозділі ми розглянемо, як розробити модель глибокого навчання для прогнозування можливих проблем у проекті та оптимізації управлінських рішень[9].

Розробка моделі глибокого навчання

Збір та підготовка даних: Початковий етап включає в себе збір та підготовку даних проекту. Це можуть бути дані про попередні проекти, дані про ресурси, строків, витрати, інформація про комунікацію та інші важливі параметри.

Вибір архітектури моделі глибокого навчання: Важливим аспектом є вибір відповідної архітектури глибокого навчання. Це може бути зворотнє поширення (feedforward), рекурентні мережі (RNN), згорткові мережі (CNN) або сучасні архітектури, такі як глибокі нейронні мережі (DNN) та глибокі згорткові мережі (DCNN).

Навчання моделі: Після вибору архітектури моделі, проводиться процес навчання на історичних даних проекту. Під час навчання модель намагається виявити патерни та залежності в даних, які можуть вказувати на можливі проблеми в майбутньому.

Прогнозування можливих проблем у проекті

Виявлення аномалій та ризиків: Навчана модель може автоматично виявляти аномалії та потенційні ризики у проекті. Для цього вона аналізує зміни в патернах даних і ідентифікує негативні відхилення.

Прогнозування строків та витрат: Модель може бути використана для прогнозування строків завершення проекту та очікуваних витрат. Це дозволяє оптимізувати ресурси та здійснювати своєчасне втручання для попередження можливих затримок.

Оптимізація управлінських рішень: На основі прогнозів модель допомагає оптимізувати управлінські рішення та приймати обґрунтовані вибори для забезпечення успішного завершення проекту.

Приклад: Прогнозування строків проекту

Розглянемо приклад прогнозування строків завершення проекту з використанням глибокого навчання. Ми маємо дані про попередні проекти, включаючи їхня завершення та витрати. Ми використовуємо глибоку рекурентну мережу (RNN) для аналізу цих даних та прогнозування строків завершення нового проекту на основі зібраних даних.

Формула для прогнозування строків завершення:

$$\gamma = f(X) \tag{2.5}$$

γ – прогнозований строк завершення проекту.

X – вхідні дані проекту, такі як історичні дані, витрати, ресурси тощо.

f – функція глибокої рекурентної мережі.

Нехай у нас є проект з історичними даними, де попередні проекти схожої природи зазвичай завершувалися за 12 місяців. Модель глибокого навчання аналізує ці дані та враховує інші фактори, такі як обсяг ресурсів та витрати. Після навчання модель може надавати прогноз, що проект завершиться через 14 місяців з урахуванням усіх вхідних параметрів.

Цей приклад ілюструє, як модель глибокого навчання може бути використана для прогнозування можливих проблем у проекті та оптимізації управлінських рішень.

Переваги використання моделей глибокого навчання

- Здатність до виявлення складних залежностей у даних.
- Підвищення точності прогнозів та рішень.
- Можливість автоматизації процесу виявлення ризиків та оптимізації управлінських рішень.

Використання моделей глибокого навчання для прогнозування можливих проблем у проекті та оптимізації управлінських рішень є правильною стратегією у сфері управління проектами. Ця технологія допомагає виявляти передбачувані проблеми та приймати обґрунтовані рішення для досягнення успішних результатів у проекті[9].

2.2.3. Використання нейронних мереж для автоматичного моніторингу ходу проекту і реагування на зміни в реальному часі.

Розробка програмного забезпечення є складним та багатоетапним процесом, який вимагає уваги до деталей, ресурсів та відповідальності. У контексті розробки програмного забезпечення, ефективний моніторинг та управління проектом має вирішальне значення для досягнення поставлених цілей і уникнення можливих проблем.

Моніторинг ходу проекту та реагування на зміни в реальному часі – це ключова складова успішного завершення розробки програмного забезпечення. Для досягнення цієї мети, ми можемо використовувати сучасні методи штучного інтелекту, зокрема нейронні мережі, які допомагають аналізувати та прогнозувати хід проекту.

У цьому контексті, мета цієї моделі – створити систему, яка на основі історичних даних та поточних відомостей про проект, зможе прогнозувати терміни виконання завдань, витрати та ідентифікувати можливі ризики. Дана модель використовує нейронні мережі для забезпечення точності та надійності прогнозів та рекомендацій[10].

Давайте розглянемо загальну архітектуру такої нейронної мережі та її компоненти.

Підготовка даних

Збір даних: Зібрати історичні дані про проекти, включаючи інформацію про завдання, графіки, витрати, ресурси, ризики та інші відомості.

Нормалізація даних: Нормалізувати дані для забезпечення однакового масштабу. Зазвичай це виконується шляхом перетворення даних на інтервал [0, 1] або за допомогою середнього значення та стандартного відхилення.

Архітектура нейронної мережі

Вхідний шар: Вхідний шар мережі містить дані про проект, такі як інформація про завдання, графіки тощо.

Прихований шар: Прихований шар містить нейрони, які виконують обробку вхідних даних і відповідають за аналіз та прогнозування ходу проекту.

Функції активації: Кожен нейрон у прихованому шарі може використовувати функції активації для обробки вхідних даних. Один із прикладів – ReLU (Rectified Linear Unit):

$$f(x) = \max(0, x) \tag{2.6}$$

Вихідний шар: Вихідний шар містить нейрони, які генерують прогнози та рекомендації. Він може включати нейрони для прогнозування термінів виконання, витрат, ідентифікації ризиків тощо.

Функція втрат

Функція втрат вимірює різницю між прогнозованими значеннями та дійсними даними про проект. Один із прикладів – середньоквадратична помилка (MSE):

$$\boldsymbol{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{Y}_i - \widehat{\boldsymbol{Y}}_i)^2 \tag{2.7}$$

Де N – кількість прикладів, $\boldsymbol{Y}_i$ – дійсний результат, а $\widehat{\boldsymbol{Y}}_i$ – прогнозований результат для прикладу i.

Навчання нейронної мережі

Після визначення архітектури та функції втрат, мережу можна навчити на історичних даних про проекти.

Для навчання використовуються методи оптимізації, такі як стохастичний градієнтний спуск *(SGD)* або більш розвинені алгоритми, такі як *Adam*.

Процес навчання включає в себе подачу даних через мережу, обчислення функції втрат і оновлення ваг нейронів для зменшення втрат.

Реагування на зміни в реальному часі

Після навчання, мережа може використовуватися для прогнозування ходу проекту на основі поточних даних.

Якщо мережа виявляє зміни або потенційні ризики, система може автоматично реагувати, наприклад, пропонуючи *виправлення графіка робіт* або *перерозподіл ресурсів*.

Така модель нейронної мережі вимагає належної настройки, великого обсягу даних та може бути застосована для моніторингу та управління проектами розробки програмного забезпечення.

2.3. Машинне навчання в системах прийняття рішень

Машинне навчання (Machine Learning, ML) в системах прийняття рішень відіграє важливу роль у вирішенні проблем, де потрібно приймати рішення на основі великих обсягів даних або складних вхідних факторів. Цей підхід використовується в різних галузях, таких як фінанси, медицина, логістика, маркетинг, інженерія та багато інших[11].

Основні питання, які вирішуються за допомогою машинного навчання в системах прийняття рішень, включають:

Прогнозування: ML алгоритми допомагають передбачити майбутні події на основі історичних даних. Наприклад, в фінансах це може включати прогнозування цін на акції або валюту, а в медицині – прогнозування розвитку хвороб.

Класифікація: ML може допомагати визначати класи об'єктів на основі їх характеристик. Наприклад, класифікація електронних листів як спаму або не спаму, або визначення патологій на знімках медичних зображень.

Оптимізація: Машинне навчання може бути використане для вибору найкращих рішень серед багатьох альтернатив. Це може включати вибір оптимального маршруту для логістичних задач або оптимізацію портфеля інвестицій.

Рекомендації: Системи рекомендацій використовують ML для пропозиції користувачам рекомендацій на основі їхніх інтересів та попереднього поведінки. Це може бути застосовано в інтернет-магазинах, медіа-платформах і соціальних мережах.

Автоматизація рішень: Машинне навчання може бути використане для автоматичного прийняття рішень в реальному часі. Це може бути корисним у системах, де потрібно швидко реагувати на зміни, такі як управління торгівлею на фондовому ринку.

Усі ці завдання вимагають обробки та аналізу великих обсягів даних, а також створення та налаштування моделей машинного навчання. Це може бути виконано за допомогою різних методів, таких як нейронні мережі, дерева рішень, метод опорних векторів та багато інших. Важливо правильно підбирати методи та моделі в залежності від конкретного завдання і доступних даних.

Машинне навчання в системах прийняття рішень стає все більше популярним, оскільки воно допомагає покращити точність та ефективність процесу прийняття рішень в різних галузях[11].

2.3.. Застосування методів машинного навчання для побудови моделі прогнозування ризиків та рекомендації в управлінні проектом

Застосування методів машинного навчання для побудови моделі прогнозування ризиків та рекомендацій в управлінні ІТ-проектами може бути корисним для підвищення ефективності і зменшення ризиків в ході виконання проекту. Нижче наведено приклади можливих методів і підходів до вирішення цієї задачі.

Прогнозування часу завершення проекту

Модель прогнозування часу завершення проекту може бути побудована на основі різних методів машинного навчання, таких як лінійна регресія, дерева рішень або нейронні мережі. Для цього потрібно мати наступні дані:

Історичні дані про подібні проекти: Вони включають в себе інформацію про обсяг роботи, терміни, ресурси, види робіт та інші фактори для подібних проектів.

Вхідні фактори: Для прогнозування термінів завершення проекту потрібно визначити вхідні фактори, такі як обсяг роботи, досвід команди, кількість доступних ресурсів тощо.

Вихідні дані: Термін завершення проекту – це вихідні дані моделі.

Модель лінійної регресії для прогнозування термінів завершення проекту може виглядати так:

$$Y = a + b_1X_1 + b_2X_2 + \cdots + b_nX_n \quad (2.8)$$

Де:

Y – прогнозований час завершення проекту.

X_1, X_2,…, X_n – вхідні фактори.

a – попередня оцінка часу.

b_1, b_2, … , b_n – коефіцієнти, які визначаються методом найменших квадратів.

Модель може бути натренована на історичних даних, і потім використовувати для прогнозування часу завершення нового проекту.

Прогнозування ризиків

Модель для прогнозування ризиків може використовувати методи класифікації, такі як логістична регресія або дерева рішень. Для цього потрібно наступні дані:

Історичні дані про ризики: Вони містять інформацію про ризики, які вже виникали в подібних проектах, включаючи їх характеристики та наслідки.

Вхідні фактори: Визначення факторів, що впливають на виникнення ризиків, такі як обсяг роботи, види робіт, досвід команди тощо.

Вихідні дані: Ймовірність виникнення ризику або класифікація ризику (наприклад, "низький", "середній", "високий").

Модель логістичної регресії для прогнозування ризиків може виглядати так:

$$P(Risk) = \frac{1}{1+e^{-(a+b_1X_1+b_2X_2+\cdots+b_nX_n}} \quad (2.9)$$

Де:

P(Risk) – ймовірність виникнення ризику.

$X_1, X_2, \dots, X_n$ – вхідні фактори.

a – попередня оцінка ймовірності.

$b_1, b_2, \dots, b_n$ – коефіцієнти, які визначаються методом логістичної регресії.

Модель може бути натренована на історичних даних про ризики і використовувати для прогнозування ймовірності виникнення ризику в новому проекті[11].

Рекомендації для управління ризиками

Після прогнозування ризиків можна використовувати методи машинного навчання для надання рекомендацій щодо управління цими ризиками. Один з підходів – це використання *алгоритмів кластеризації* для групування ризиків за їхнім характером. Наприклад, можна створити кілька груп ризиків, таких як "технічні ризики", "проектні ризики", "внутрішні ризики" і надавати рекомендації для кожної групи.

Додатково, можна використовувати *методи асоціаційних правил* для виявлення взаємозв'язків між ризиками та пропозиції щодо мінімізації їх наслідків.

Процес застосування машинного навчання для управління IT-проектами включає в себе наступні кроки:

- *Збір та підготовка даних:* Збирання історичних даних про проекти та ризики, а також визначення вхідних факторів для моделей.

- *Вибір методів машинного навчання:* Визначення методів, що найкраще підходять для конкретних завдань прогнозування та класифікації.

- *Тренування моделей:* Натренування моделей на історичних даних і налаштування їх параметрів.

- *Тестування моделей:* Перевірка точності та ефективності моделей на тестових даних.

- *Впровадження:* Реалізація моделей в систему управління проектами.

- *Моніторинг та підтримка:* Постійний моніторинг роботи моделей та їх оновлення відповідно до змін в умовах проекту.

Це загальний огляд застосування машинного навчання для прогнозування ризиків та рекомендацій в управлінні IT-проектами. Конкретні підходи та методи будуть залежати від характеристик проекту та доступних даних.

2.3.2. Автоматизоване планування IT проекту за допомогою генеративно-змагальної мережі (GAN)

Генеративно-змагальні мережі (GAN) – це глибокі нейронні мережі, що використовуються для генерації нових даних на основі навчального набору даних. GAN складаються з двох основних компонентів: генератора і дискримінатора[12]. Генератор намагається створити дані, які схожі на навчальний набір, тоді як дискримінатор намагається відрізнити справжні дані

від сгенерованих. Такий процес створює генеративну модель, яка постійно покращує якість згенерованих даних.

Алгоритм роботи GAN:

1. Ініціалізація моделей:

Генератор (G): Генератор починає з випадкового шуму і створює зразок даних.

Дискримінатор (D): Дискримінатор починає з вагами, які ініціалізуються випадковим чином.

2. Тренування GAN:

Подача даних: Дійсні дані з навчального набору і сгенеровані дані від генератора подаються на вхід дискримінатора.

Оцінка втрат: Дискримінатор оцінює, наскільки вони справжні (від 1 до 0) для кожного вхідного зразка даних.

Формула для втрат дискримінатора (L_D):

$$L\_D = -[\log(D(\text{дійсні дані})) + \log(1 - D(G(\text{шум})))] \quad (2.10)$$

Оцінка втрат генератора: Генератор оцінює, наскільки сгенеровані дані близькі до справжніх, за допомогою відповіді дискримінатора на сгенеровані дані.

Формула для втрат генератора (L_G):

$$L\_G = -\log(D(G(\text{шум}))) \quad (2.11)$$

Оновлення ваг: Генератор і дискримінатор оновлюють свої ваги за допомогою градієнтного спуску, щоб зменшити свої втрати.

3. Повторення: Повторювати кроки тренування, поки модель не досягне бажаного рівня якості генерації.

Генеративно-змагальні мережі широко використовуються для завдань, таких як генерація зображень, створення тексту, перетворення даних та багато інших. Вони допомагають створювати реалістичні дані, які можуть бути корисні в багатьох сферах, а саме при плануванні виконання ІТ проекту.

Розробка системи автоматизованого планування ІТ проекту на основі генеративних засобів *(GANs – Generative Adversarial Networks)* може бути

складним завданням, але така система може виявитися дуже потужною для покращення процесів управління проектом та планування.

Нижче наведено загальний огляд кроків, які потрібно виконати для розробки такої системи:

Збір та підготовка даних: Перш за все, потрібно мати велику кількість даних про IT-проекти. Ці дані повинні містити інформацію про ресурси, строки, завдання, завдання, ризики, інші параметри проекту.

Вибір архітектури GANs: Вибір архітектури GANs, яка найкраще підходить для нашої задачі. Можливо, знадобиться користуватися варіантами GANs, спеціально призначеними для роботи з послідовними даними, такими як часові ряди.

Підготовка генератора: Генератор GANs відповідає за створення планів проекту. Він має навчатися генерувати прийнятні плани на основі навчальних даних. Можемо використовувати рекурентні нейронні мережі (RNNs) або LSTM для створення послідовностей задач і строків.

Підготовка дискримінатора: Дискримінатор GANs відповідає за оцінку якості планів проекту, що генеруються генератором. Він навчається розрізняти реальні плани від сгенерованих.

Тренування GANs: Проведемо тренування GANs на підготовлених даних, забезпечуючи взаємодію між генератором і дискримінатором. Під час тренування GANs мають збалансовувати зусилля генератора і дискримінатора так, щоб генератор навчився створювати реалістичні плани проекту.

Оцінка та валідація: Після завершення тренування оцінка якості згенерованих планів. Можна використовувати метрики, такі як точність, ризики, ресурси та інші для оцінки планів.

Інтеграція в управління проектом: Якщо система планування показує гарні результати, і ми впевнені в її якості, то інтегруємо її в процес управління IT-проектом. Вона може надавати рекомендації з покращення планування та оптимізації ресурсів.

Постійне навчання: Регулярно оновлюємо та навчаємо систему на нових даних, щоб забезпечити її актуальність та ефективність у реальних умовах.

Розробка системи планування IT проекту на основі GANs може покращити точність та ефективність процесу планування, але вимагає дбайливого підходу та ресурсів для успішної реалізації[12].

2.4. Обробка природних мов (Natural Language Processing, NLP) для аналізу текстової інформації

Обробка природних мов (Natural Language Processing, NLP) в контексті управління проектами включає в себе використання комп'ютерних алгоритмів для аналізу, розуміння та витягнення інформації з людської мови. В управлінні проектами NLP може бути використана для автоматичного аналізу текстової інформації, що створюється під час виконання проекту, такою як звіти, електронні листи, комунікації в проєктних системах, тощо. Методи обробки природньої мови в управлінні проектами включають в себе такі аспекти:

1. Автоматичне розпізнавання та класифікація текстової інформації.
2. Аналіз тональності текстів для виявлення ставлення учасників проекту до певних аспектів його реалізації.
3. Витягування ключової інформації з текстових документів та звітів проектів.
4. Автоматичний аналіз та структурування текстової інформації для автоматизованого формування звітів та аналітичних матеріалів.

Використання методів обробки природної мови в управлінні проектами може значно спростити та прискорити аналіз великих обсягів текстової інформації, допомагаючи в ідентифікації ключових проблем, ризиків та можливостей у проекті, а також у покращенні процесів управління комунікаціями та прийняття рішень на основі аналізу текстових даних[12].

2.4.1. Використання NLP для витягування ключової інформації з текстових документів та звітів проектів

В останні роки в сучасному управлінні проектами велике значення набуває автоматизація та оптимізація процесів аналізу текстової інформації, що міститься в різноманітних документах та звітах проектів. Одним із потужних інструментів для досягнення цієї мети є обробка природної мови (NLP), яка використовує комп'ютерні алгоритми та моделі для розуміння та обробки людської мови.

У цьому контексті, NLP виявляється важливим інструментом для витягування ключової інформації з текстових документів та звітів проектів. Від розпізнавання іменованих сутностей та визначення тематичного спрямування до аналізу настроїв та екстракції ключових термінів, NLP дозволяє автоматизувати та полегшити процеси обробки великої кількості текстової інформації, забезпечуючи швидше та більш точне прийняття рішень в управлінні проектами.

У цьому контексті відбувається злиття передових технологій NLP та інструментів управління проектами, що відкриває нові можливості для підвищення продуктивності, покращення аналітичних можливостей та ускладнених стратегічних рішень в управлінні проектами. У цьому вступі розглянемо різноманітні аспекти використання NLP в контексті витягування ключової інформації, вивчаючи методи, алгоритми та реальні застосування цієї технології у сучасному управлінні проектами[13].

Розглянемо приклад використання NLP для витягування ключової інформації з текстових документів та звітів проектів:

Токенізація та витягування іменованих сутностей

Для витягування ключової інформації з текстових документів та звітів проектів за допомогою токенізації та витягування іменованих сутностей (NER), можна використовувати такі кроки:

Токенізація:

```python
import spacy

# Завантаження моделі spaCy для англійської мови
nlp = spacy.load("en_core_web_sm")

# Припустимий текст звіту про проект
text = "The project, named ProjectX, is led by John Doe and located in

# Токенізація
tokens = [token.text for token in nlp(text)]

print("Токени:", tokens)
```

Рис. 2.1. Приклад токенізації

Витягування іменованих сутностей (NER):

```python
# Витягування іменованих сутностей
named_entities = [(ent.text, ent.label_) for ent in nlp(text).ents]


print("Іменовані сутності:", named_entities)
```

Рис. 2.2. Приклад витягування іменованих сутностей

Токенізація та витягування іменованих сутностей (NER) є важливими етапами обробки природної мови, особливо при витягуванні ключової інформації з текстових документів та звітів проектів[13]. У вищеприведеному коді ми використали бібліотеку spaCy для виконання цих завдань.

Токенізація допомагає розбити текст на окремі токени, що робить його більш доступним для подальшого аналізу. Список токенів може бути використаний для визначення структури тексту та важливих слів.

Витягування іменованих сутностей (NER) визначає та класифікує іменовані об'єкти у тексті, такі як імена осіб, назви організацій, тощо. Це особливо корисно при аналізі звітів про проекти, де імена учасників, місця та

інші ключові інформаційні елементи можуть відігравати важливу роль у розумінні тексту.

Об'єднуючи обидва етапи, ми отримуємо повний інструментарій для аналізу та витягування ключової інформації з текстових документів та звітів проектів. Це дозволяє автоматизувати процеси управління проектами, роблячи їх більш ефективними та зрозумілими для виконавців та керівників проекту.

2.4.2. Екстракція семантичних зв'язків між різними частинами тексту

В управлінні проектом визначення семантичних зв'язків, зокрема причинно-наслідкових зв'язків та відносин часу, може бути критичним для ефективного планування та виконання завдань. Використання обробки природної мови (NLP) може значно полегшити цей процес. Нижче подано кілька способів, які можна використовувати для екстракції семантичних зв'язків в контексті управління проектами:

1. Визначення залежностей між завданнями:

Аналіз синтаксичних залежностей:

Використання NLP для аналізу синтаксичних структур речень та визначення зв'язків між різними завданнями чи подіями в проекті.

```
import spacy

nlp = spacy.load("en_core_web_sm")

# Приклад тексту з описом завдань
text = "The completion of Task A depends on the successful execution of

# Аналіз синтаксичних залежностей
doc = nlp(text)
for token in doc:
    print(token.text, token.dep_, token.head.text)
```

Рис. 2.3. Приклад аналізу синтаксичних залежностей

Використання семантичного аналізу:

Застосування методів семантичного аналізу для визначення схожості чи відмінностей між завданнями та їхніми сутностями.

2. Визначення відносин часу:

Використання виразів часу:

Використання NLP для визначення виразів часу та їх використання для визначення відносин часу між подіями.

```
# Приклад визначення виразів часу
time_expressions = ["before", "after", "during", "next", ...]

text = "The project should be completed before the end of the month."

# Перевірка наявності виразів часу
if any(expression in text for expression in time_expressions):
    print("Визначено відносини часу.")
```

Рис. 2.4. Приклад визначення виразів часу та їх використання

3. Визначення зв'язків з ресурсами та учасниками:

Витягування іменованих сутностей (NER):

Використання NLP для витягування іменованих сутностей, таких як імена учасників проекту, ресурси чи місця, що можуть взаємодіяти в рамках проекту.

```
# Приклад витягування іменованих сутностей
named_entities = [(ent.text, ent.label_) for ent in nlp(text).ents]
print("Іменовані сутності:", named_entities)
```

Рис. 2.5. Приклад витягування іменованих сутностей

Використання NLP у вищезазначених випадках допомагає автоматизувати та полегшити аналіз тексту в управлінні проектами, забезпечуючи більш глибоке розуміння семантичних зв'язків між різними частинами інформації[14]. Це може полегшити управління завданнями, визначення пріоритетів та виявлення можливих ризиків у проекті.

2.4.3. Аналіз можливостей розвитку та вдосконалення використання NLP у сучасних системах управління проектами

В сучасному світі системи управління проектами зазнають значних змін і вдосконалень завдяки застосуванню новітніх технологій. Однією з ключових інновацій, яка вносить суттєвий вклад у цю галузь, є застосування Natural Language Processing (NLP) – області штучного інтелекту, що вивчає взаємодію між комп'ютерами та людською мовою. Аналіз можливостей розвитку та вдосконалення використання NLP у сучасних системах управління проектами стає важливим завданням для підвищення ефективності та точності управління проектами.

Зростання важливості NLP в управлінні проектами

Системи управління проектами, оснащені технологіями NLP, можуть значно полегшити аналіз великого обсягу текстової інформації, яка генерується під час роботи над проектами. За допомогою NLP можливо не лише автоматизувати обробку даних, але й розуміти контекст та семантичні зв'язки між різними частинами тексту. Це відкриває широкі можливості для вдосконалення процесів управління проектами та прийняття обґрунтованих рішень.

Ключові напрямки розвитку NLP в управлінні проектами

Розширення функціональності мовних моделей: Подальший розвиток мовних моделей, зокрема застосування передових архітектур, дозволить досягти більшого рівня контекстуальності та точності у розумінні текстової інформації.

Автоматизована класифікація та аналіз сутностей: Поліпшення систем класифікації подій та екстракції іменованих сутностей (NER) дозволить точніше ідентифікувати ключові елементи у тексті логів проектів.

Взаємодія з іншими технологіями: Інтеграція NLP з іншими інтелектуальними технологіями, такими як машинне навчання та аналіз даних, створить комплексні підходи до управління проектами та допоможе виявляти складні патерни[14].

Розробка інтерфейсів та візуалізацій: Створення зручних інтерфейсів для відображення результатів аналізу NLP допоможе керівникам проектів та командам зручніше сприймати та аналізувати інформацію.

Використання NLP в системах управління проектами представляє собою значущий крок у напрямку автоматизації та оптимізації робочих процесів. Розвиток цієї технології в управлінні проектами відкриває перед нами величезні можливості вдосконалення аналізу, планування та контролю проектів. За допомогою розвинених інструментів NLP, команди зможуть ефективніше взаємодіяти з великим обсягом текстової інформації, вчасно виявляти проблеми та приймати обґрунтовані рішення, що сприятиме підвищенню ефективності та успішності проектів.

ВИСНОВОК ДО РОЗДІЛУ 2

В розділі, присвяченому аналізу та використанню інтелектуальних методів у системах підтримки прийняття рішень для управління проектами, виявлено ряд ключових аспектів. Проведено детальний розгляд інтелектуального аналізу даних, зокрема збору та аналізу історичних даних проектів з метою виявлення шаблонів та тенденцій. Оцінено використання статистичних методів для оцінки ризиків і можливостей проекту, а також розроблено моделі прогнозування для підтримки процесу прийняття рішень.

Детально розглянуто застосування нейронних мереж у системах управління проектами, включаючи їхнє використання для автоматичного аналізу та класифікації даних, розробку моделей глибокого навчання для прогнозування проблем та моніторингу ходу проекту в реальному часі. Досліджено можливості машинного навчання в прийнятті рішень у контексті побудови моделей прогнозування ризиків та автоматизованого планування IT проектів за допомогою генеративно-змагальної мережі (GAN).

Окремий акцент приділено використанню обробки природних мов (NLP) для аналізу текстової інформації. Вивчено застосування NLP для витягування ключової інформації з текстових документів, екстракцію семантичних зв'язків та обговорено перспективи вдосконалення цього напрямку в сучасних системах управління проектами.

Загальний висновок розділу свідчить про те, що інтелектуальні методи виявляються потужними інструментами для підвищення якості управління проектами. Вони забезпечують точні прогнози, автоматизують аналіз та прийняття рішень, а також сприяють ефективному використанню інформації, отриманої з історичних даних проектів.

РОЗДІЛ 3

АРХІТЕКТУРА ТА РОЗРОБКА СППР УПРАВЛІННЯ ІТ-ПРОЕКТОМ

У цьому розділі розглянута ключова тема архітектури та розробки Системи Підтримки Прийняття Рішень (СППР) в контексті управління ІТ-проектами. Зосередимо увагу на структурі та компонентах таких систем, які визначають їх функціональність та допомагають в прийнятті виважених та ефективних управлінських рішень.

Першочергово, розглянемо архітектурні аспекти СППР управління ІТ-проектом, звертаючи увагу на взаємодію між різними компонентами та їхню взаємозалежність. Детально розглянемо архітектурні шари та модулі, що визначають функціональний стан системи та взаємодію з користувачем.

Далі в розділі буде висвітлено питання розробки СППР для управління ІТ-проектами. Проаналізуємо вибір технологічних стеків, методів програмування та інструментів розробки, які сприяють оптимальній реалізації функціональності системи.

Крім того, у цьому розділі буде надано огляд архітектурних рішень, які підтримують ефективність та масштабованість СППР управління ІТ-проектами, зокрема у відповіді на зростаючі вимоги та зміни в інформаційному середовищі.

Усі ці аспекти спільно допоможуть у формуванні вичерпного уявлення про архітектуру та розробку СППР управління ІТ-проектами, що, в свою чергу, буде корисним для подальшого розгортання та оптимізації подібних систем у реальних умовах управління проектами в галузі інформаційних технологій.

3.1. Огляд вимог до системи підтримки прийняття рішень в управлінні IT-проектом

У даному розділі проводиться огляд вимог до системи підтримки прийняття рішень в контексті управління IT-проектом. Описуються функціональні вимоги, необхідні для забезпечення ефективного управління проектом, а також вимоги до інформаційної безпеки та конфіденційності даних, що важливі для забезпечення безпеки та захисту важливої інформації в контексті управління IT-проектами.

Функціональні вимоги можуть включати в себе можливості моделювання процесів управління проектом, автоматизоване аналізування даних, генерацію звітів та аналітичних матеріалів, можливості прогнозування ризиків та можливостей проекту, а також функції планування та моніторингу IT-проекту[15].

Успішна реалізація системи підтримки прийняття рішень в управлінні IT-проектом також ґрунтується на необхідності врахування вимог щодо інформаційної безпеки. Це охоплює забезпечення конфіденційності, цілісності та доступності даних, відповідності нормативним вимогам щодо захисту персональних даних та забезпечення захисту від кіберзагроз.

Даний огляд вимог буде включати аналіз потреб користувачів системи, технічних вимог до системи, а також вимог щодо забезпечення безпеки та конфіденційності даних, що формує основу для подальшої розробки архітектури системи підтримки прийняття рішень в управлінні IT-проектом.

3.1.1. Визначення функціональних вимог до системи підтримки прийняття рішень

Визначення функціональних вимог – це складний і важливий процес, що передбачає ідентифікацію конкретних функцій та можливостей, які система повинна надати, забезпечуючи ефективне управління IT-проектами. Для

системи підтримки прийняття рішень в управлінні ІТ-проектом такі вимоги можуть включати, але не обмежуються:

- *Моделювання процесів управління проектами:* система повинна забезпечувати можливість моделювання процесів управління проектами, включаючи планування, виконання, моніторинг, контроль та аналіз.
- *Автоматизований аналіз даних:* система має забезпечувати можливість автоматизованого збору, обробки, аналізу та візуалізації даних проекту для виявлення патернів та трендів.
- *Генерація звітів та аналітичних матеріалів:* система повинна мати можливість генерувати звіти та аналітичні матеріали на основі оброблених даних проекту.
- *Прогнозування ризиків та можливостей проекту:* система має включати модулі для прогнозування можливих ризиків та можливостей, що дозволить приймати обґрунтовані управлінські рішення.
- *Планування та моніторинг ІТ-проекту:* система повинна надати можливість автоматизованого планування та моніторингу ІТ-проекту з відображенням актуальних даних.

Визначення цих функціональних вимог забезпечить належне функціонування системи підтримки прийняття рішень в управлінні ІТ-проектом, а також дозволить досягти поставлених цілей ефективного управління проектами[15].

3.1.2. Вимоги до інформаційної безпеки та конфіденційності даних у контексті управління ІТ-проектами

У контексті управління інформаційними технологіями, забезпечення інформаційної безпеки та конфіденційності даних є критично важливим аспектом. Враховуючи це, система підтримки прийняття рішень в управлінні ІТ-проектами повинна відповідати вимогам щодо захисту інформації.

Це охоплює наступні вимоги:

- *Захист від несанкціонованого доступу:* Гарантування, що лише авторизовані користувачі мають доступ до системи та конфіденційної інформації.
- *Захист від кібератак:* Система повинна мати вбудовані заходи для запобігання кібер атакам, таким як DDoS атаки, витік даних та злам системи.
- *Конфіденційність даних:* Забезпечення шифрування конфіденційної інформації, щоб забезпечити захист від несанкціонованого доступу до даних.
- *Відповідність нормативам безпеки:* Врахування вимог і стандартів щодо безпеки даних та конфіденційності, таких як GDPR, HIPAA, PCI DSS тощо.
- *Аудит доступу:* Забезпечення можливості ведення журналу подій для відстеження доступу до системи та даних.

Ці вимоги допоможуть побудувати систему, що забезпечить найвищий рівень захисту для конфіденційної інформації у контексті управління IT-проектами.

3.2. Архітектурні принципи системи підтримки прийняття рішень в управлінні IT-проектом

Для більш детального розгляду архітектурних принципів системи підтримки прийняття рішень в управлінні IT-проектом, розглянемо ключові аспекти такої системи[16].

1. Модульна архітектура

Система повинна мати модульну архітектуру, що дозволяє розширювати та модифікувати функціонал системи без значних змін у цілісності. Кожен модуль може виконувати певну функцію, таку як обробка та аналіз даних, візуалізація результатів, або виконання конкретних методів прийняття рішень.

2. Використання мікросервісної архітектури

Для підтримки гнучкості та розподіленості, можна використовувати мікросервісну архітектуру, де кожен функціональний блок системи представлений як окремий мікросервіс. Це дозволяє використовувати автономні сервіси, робити їх незалежними один від одного та забезпечувати гнучкість у виборі технологій реалізації.

3. Використання хмарних технологій

З врахуванням розподіленості проектних команд та обмежених обсягів ресурсів для розгортання та підтримки серверів, систему можна розгорнути в хмарних сервісах. Це дозволить забезпечити високу доступність, масштабованість та безпеку даних.

4. Використання принципів CI/CD

Для швидкого впровадження нового функціоналу та корекції помилок використовуються принципи Continuous Integration/Continuous Deployment (CI/CD), що забезпечують автоматизований процес злиття та тестування змін у коді, а також автоматичне розгортання змінених версій системи.

5. Застосування адаптивних інтерфейсів

Система повинна мати адаптивний користувацький інтерфейс, що дозволяє команді управління проектом зручно взаємодіяти з системою на різних пристроях та різними способами, забезпечуючи ефективне використання рішень, що були прийняті[16].

Ці архітектурні принципи допоможуть забезпечити ефективну та гнучку систему підтримки прийняття рішень в управлінні IT-проектами.

3.2.1. Вибір архітектури системи та її компонентів

Вибір архітектури системи та компонентів дуже важливий етап проектування системи підтримки прийняття рішень. Це визначає як система буде функціонувати, як взаємодіяти з користувачем та іншими системами.

Один з підходів до вибору архітектури системи – це використання моделей вибору архітектури, таких як "4+1 вигляд" (logical view, process view, development view, physical view, use case view), або "Model-Driven Architecture" (MDA), які дозволяють краще розуміти вимоги до системи та вибрати найбільш підходящий варіант[16].

При виборі компонентів системи, варто врахувати такі фактори, як їхню спроможність працювати разом, можливість розширення та модифікації, швидкість взаємодії, надійність та безпеку. Для цього можна провести порівняльний аналіз різних архітектур та компонентів з використанням матриць прийняття рішень або аналізу альтернатив.

Також, варто врахувати відкритість системи для інтеграції з іншими СППР та іншими системами управління проектами, що може вимагати використання стандартів та протоколів зв'язку, таких як REST або SOAP для взаємодії з іншими системами.

Такий підхід до вибору архітектури системи та компонентів дозволить забезпечити оптимальну працездатність та ефективність системи підтримки прийняття рішень в управлінні проектами.

З урахуванням наведених архітектурних принципів, найбільш підходящою здатна бути **мікросервісна архітектура** для системи підтримки прийняття рішень в управлінні ІТ-проектами.

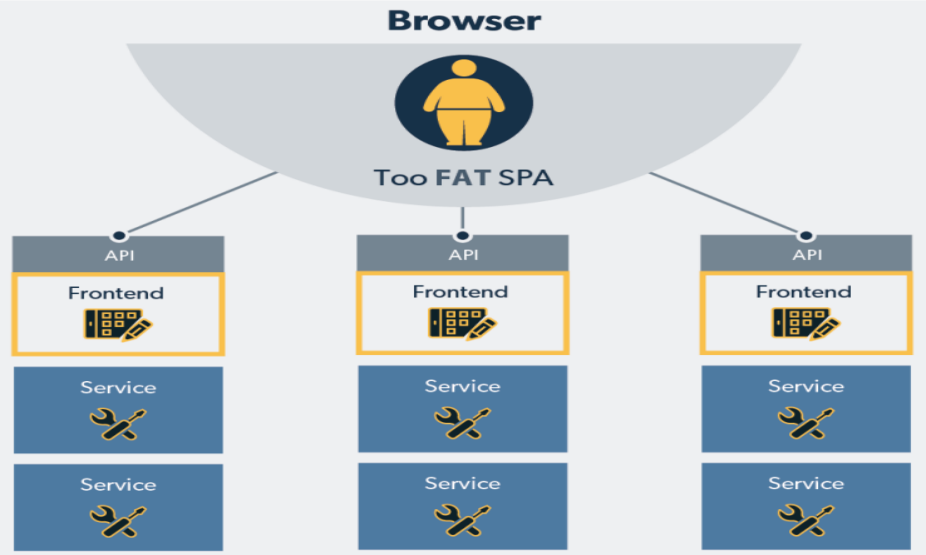


Рис. 3.1. Мікросервісна архітектура системи

Мікросервісна архітектура дозволить розділити функціонал системи на окремі сервіси, які будуть реалізовані як невеликі, автономні та змінювані незалежно один від одного. Кожен сервіс може бути відповідальним за конкретну функцію, таку як обробка та аналіз даних, візуалізація результатів, робота з деревами рішень, інтеграція з іншими системами тощо.

Перевагою мікросервісної архітектури є те, що вона відповідає модульній архітектурі, дозволяючи розширювати та модифікувати функціонал без серйозних змін у цілісності системи. Крім того, використання хмарних технологій буде сприяти масштабованості та доступності системи, а принципи CI/CD дозволять швидко впроваджувати новий функціонал та коригувати помилки.

Для забезпечення адаптивності інтерфейсів, можна реалізувати окремі мікросервіси, відповідальні за різні типи інтерфейсів (Web, мобільні додатки, API тощо) з можливістю швидкого розвитку та автоматизації розгортання змін.

Отже, мікросервісна архітектура, враховуючи всі вказані принципи, найбільше відповідає вимогам ефективної та гнучкої системи підтримки прийняття рішень в управлінні IT-проектами[17].

3.2.2. Забезпечення масштабованості та продуктивності системи в умовах управління IT-проектами

Забезпечення масштабованості та продуктивності системи в умовах управління IT-проектами є ключовим аспектом. Для цього можна використовувати різноманітні техніки та підходи. Ось кілька можливих шляхів забезпечення масштабованості та продуктивності системи:

1. Вертикальне та горизонтальне масштабування:

Вертикальне масштабування полягає у збільшенні потужності окремих серверів або баз даних шляхом додавання більш потужних ресурсів (наприклад, пам'яті, процесорів).

Горизонтальне масштабування передбачає розділення навантаження на кілька серверів (фізичних або віртуальних), що може бути досягнуто за допомогою розподілення функцій на окремі мікросервіси.

2. Використання кешування:

Використання кешування може суттєво покращити продуктивність системи шляхом збереження та повторного використання результатів попередніх запитів до деяких даних чи операцій.

3. Оптимізація запитів до баз даних:

Ефективне використання індексів, зменшення обсягу данних, використання оптимізованих запитів, може допомогти забезпечити продуктивність системи.

4. Використання хмарних сервісів та мікросервісної архітектури:

Застосування хмарних сервісів для розгортання інфраструктури може дозволити автоматично масштабувати обсяг ресурсів в залежності від потреб системи.

Мікросервісна архітектура дозволить гнучко масштабувати окремі компоненти системи, що також сприятиме продуктивності та масштабованості.

5. Використання кеш-пам'яті на рівні додатків:

Використання кеш-пам'яті для швидкого доступу до даних чи обчислення результатів певних операцій може покращити продуктивність системи.

Застосування цих підходів, а також систематичний моніторинг продуктивності системи та виявлення bottleneck'ів, дозволять забезпечити масштабованість та продуктивність системи в умовах управління IT-проектами.

3.2.3. Аналіз існуючих рішень для розміщення мікросервісного додатку в хмарному середовищі

При аналізі існуючих рішень для розміщення мікросервісного додатку в хмарному середовищі рекомендується порівнювати різні хмарні платформи та провайдерів хмарних послуг.

Критерії для порівняння можуть включати:

Масштабованість: Можливість автоматичного масштабування ресурсів для відповіді на змінні навантаження.

Доступність та надійність: Рівень гарантованої доступності та надійності.

Безпека: Заходи забезпечення безпеки даних та управління доступом.

Продуктивність: Швидкодія та продуктивність обчислень та роботи з базами даних.

Ціна: Витрати на використання ресурсів та послуг.

Підтримка мікросервісної архітектури: Функціональні можливості, що сприяють створенню та управлінню мікросервісами.

Інструменти моніторингу та керування: Наявність засобів для моніторингу та керування мікросервісами.

Після зібрання цієї інформації можна провести аналіз та порівняти різні хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, IBM Cloud тощо, а також аналізувати можливості

спеціалізованих хмарних сервісів, таких як AWS Lambda для безсерверних рішень, або Kubernetes для оркестрації контейнерів.

Найбільш оптимальний вибір буде залежати від конкретних потреб проекту, бюджетних обмежень та вимог до продуктивності, безпеки та інших параметрів[17].

Отже розглянемо порівняння різних хмарних рішень для розміщення мікросервісів у вигляді таблиці, враховуючи кілька основних аспектів:

Таблиця 3.1

Порівняння хмарних рішень

Критерії / Рішення	**AWS**	**Azure**	**GCP**	**Docker Swarm**	**Kubernetes**	**Heroku**	**IBM Cloud**
Масштабованість	Висока	Висока	Висока	Середня	Висока	Середня	Висока
Доступність та надійність	Висока	Висока	Висока	Середня	Висока	Висока	Висока
Безпека	Висока	Висока	Висока	Середня	Висока	Висока	Висока
Продуктивність	Висока	Висока	Висока	Середня	Висока	Середня	Висока
Ціна	Залежить від використання	Залежить від використання	Залежить від використання	Низька	Низька	Низька	Залежить від використання
Підтримка мікросервісної архітектури	Так	Так	Так	Так	Так	Так	Так
Інструменти моніторингу та керування	Так	Так	Так	Ні	Так	Обмежено	Так

Це загальне порівняння, і важливо враховувати, що конкретні вимоги та сценарії використання можуть впливати на вибір найбільш підходящого рішення для проекту. Також, різні сервіси та інструменти можуть змінюватися з часом, тому рекомендується періодично оновлювати аналіз в залежності від нових можливостей та змін в середовищі хмар.

Вибір **Kubernetes** для розробки системи підтримки прийняття рішень в управлінні проектною командою може бути обґрунтований декількома ключовими перевагами, які особливо цінні в контексті управління проектами та мікросервісною архітектурою:

Оркестрація та управління контейнерами:

Kubernetes забезпечує ефективну оркестрацію та управління контейнерами. Це особливо важливо в мікросервісному середовищі, де різні компоненти додатку можуть бути упаковані в контейнери. Kubernetes дозволяє легко масштабувати, керувати життєвим циклом та розподіляти навантаження між контейнерами.

Масштабованість та гнучкість:

Kubernetes надає високий рівень масштабованості, що важливо для управління проектами різної складності та обсягу. Він легко адаптується до змін у вимогах та обсягу роботи.

Інструменти доставки та оновлення:

Kubernetes має розгортання з можливістю відкату (rolling deployment) та інші інструменти для ефективної доставки та оновлення мікросервісів. Це полегшує процес впровадження нових функцій чи виправлення помилок.

Стандартизація та відкритість:

Kubernetes є відкритим джерелом, що сприяє стандартизації та спільнотному розвитку. Це забезпечує велику кількість інтеграцій з іншими інструментами та сервісами, що полегшує розширення та модифікацію системи.

Декларативний підхід:

Kubernetes використовує декларативний підхід до конфігурації, що дозволяє визначити бажаний стан системи та дозволяє Kubernetes забезпечити відповідність стану системи до визначеного стану. Це полегшує управління конфігурацією та розгортанням.

Спрощена управлінська консоль:

Kubernetes надає веб-консоль та інтерфейс командного рядка для управління та відстеження стану системи. Це дозволяє команді швидко отримувати доступ до важливих даних та моніторингу.

Система моніторингу та журналювання:

Kubernetes має вбудовану систему моніторингу та журналювання, яка дозволяє відстежувати та аналізувати стан системи, виявляти проблеми та вчасно реагувати на них.

Розширюваність та екосистема:

Kubernetes має широкий екосистему, що включає в себе різноманітні інструменти, додатки та сервіси. Це дозволяє легко інтегрувати його з іншими засобами та використовувати додаткові функціональності.

Всі ці аспекти роблять Kubernetes потужним інструментом для розробки системи підтримки прийняття рішень в управлінні проектною командою, особливо в умовах мікросервісної архітектури та динамічного середовища проектів[17].

3.3. Моделювання процесів управління IT-проектом у системі підтримки прийняття рішень

Моделювання процесів управління IT-проектом у системі підтримки прийняття рішень (СППР) є важливою складовою для ефективного управління проектом. Моделі процесів можуть допомогти в оцінці ризиків, плануванні ресурсів, визначенні критеріїв прийняття рішень та інших аспектах управління проектом. Ось кілька підходів до моделювання процесів управління IT-проектом у СППР:

Діаграми потоку процесів (Flowcharts): Діаграми потоку процесів використовуються для візуалізації послідовності кроків та комунікації між різними членами команди проекту. Діаграми потоку допомагають виділити загальну структуру процесу та розуміти взаємодію між окремими кроками.

Діаграми Ганта (Gantt charts): Діаграми Ганта зображують розклад проекту у вигляді горизонтальної панелі, де по вертикалі відображаються різні завдання або етапи проекту, а по горизонталі – тривалість кожного етапу. Діаграми Ганта допомагають візуалізувати схему виконання проекту та визначити критичні шляхи.

Блок-схеми (Block diagrams): Блок-схеми використовуються для представлення блоків або елементів процесу, їх взаємозв'язків та впливу одного елементу на інший. Блок-схеми допомагають дослідникам та учасникам проекту зрозуміти логіку та послідовність виконання процесу.

Аналітичні моделі: Аналітичні моделі використовуються для числової оцінки параметрів процесу управління проектом. Наприклад, моделі прогнозування можуть бути використані для оцінки тривалості проекту, вартості робіт та ресурсів, ризиків та очікуваних результатів.

Системи додаткової реалізації (Add-on Systems): Деякі системи підтримки прийняття рішень дозволяють створювати власні моделі процесів, використовуючи спеціальні моделюючі інструменти або розширювані функціональні можливості. Системи можуть забезпечити створення та налагодження власних моделей процесів, які найкраще відповідають конкретним потребам управління проектом.

Ці підходи до моделювання процесів управління IT-проектом у СППР допоможуть зрозуміти послідовність дій, взаємозв'язки та вплив різних параметрів на результати проекту. Таке моделювання сприятиме подальшому управлінню, аналізу та прийняттю рішень у проекті[17].

3.3.1. Використання BPMN для моделювання процесів управління IT-проектом

Використання Business Process Model and Notation (BPMN) для моделювання процесів управління IT-проектом є ефективним і популярним підходом. BPMN – це стандартна нотація для моделювання бізнес-процесів,

яка дозволяє візуалізувати послідовності дій, взаємозв'язки та потоки роботи між різними елементами процесу.

Основні елементи BPMN, які можуть бути використані при моделюванні процесів управління IT-проектом, включають:

- ***Завдання (Tasks):*** Позначають конкретні дії, які повинні бути виконані в рамках процесу. Наприклад, аналіз вимог, планування проекту, розробка програмного забезпечення.
- ***Схвалення (Gateways):*** Вказують рішення та вибір шляху, що треба прийняти в залежності від певних умов. Наприклад, визначення альтернативних шляхів розвитку проекту.
- ***Події (Events):*** Позначають початок, закінчення або зміни в процесі. Наприклад, запуск проекту, завершення фази, ризикові події.
- ***Потоки (Flows):*** Вказують послідовність дій, з'єднання між елементами процесу. Наприклад, потік завдань в рамках проекту або потік комунікації між членами команди.
- ***Басейни та доріжки (Pools and Lanes):*** Використовуються для організації та групування активностей за командами, відділами або учасниками проекту.

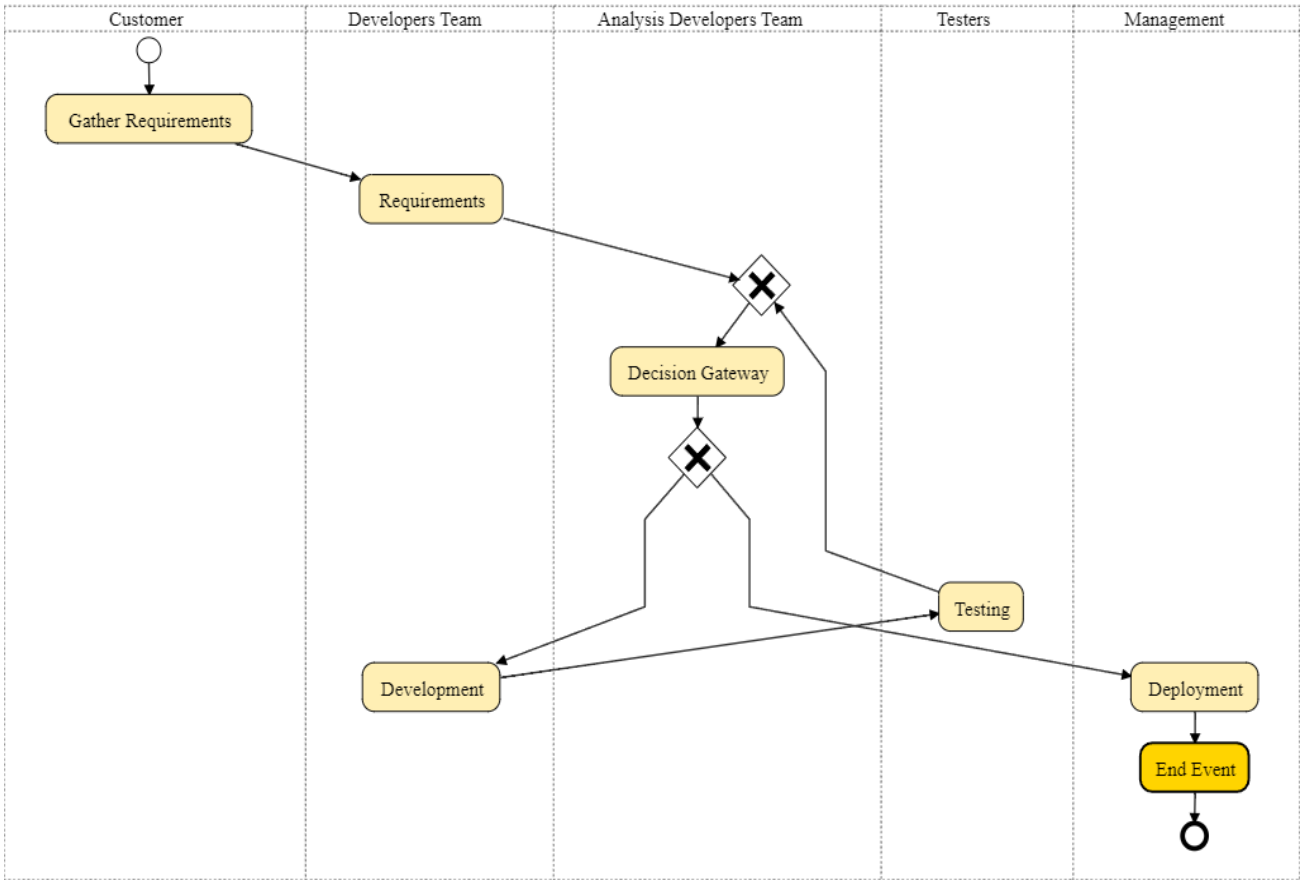


Рис. 3.2. Приклад BPMN діаграми процесу розробки програмного забезпечення

На рис 3.2 наведено модель процесу управління розробкою програмного забезпечення (Software Development) з використанням BPMN елементів:

1. Завдання:
   - Збір вимог (Gather Requirements)
   - Аналіз вимог (Requirements Analysis)
   - Розробка (Development)
   - Тестування (Testing)
   - Впровадження (Deployment)
2. Схвалення:
   - Прийняття рішення щодо переходу до наступного етапу (Decision Gateway)

- Вибір альтернативного шляху (Exclusive Gateway)

3. Події:
   - Початок процесу (Start Event)
   - Завершення процесу (End Event)
   - Ризикова подія (Risk Event)
4. Потоки:
   - Потік завдань в рамках проекту (Task Flow)
   - Потік комунікації між членами команди (Communication Flow)
5. Басейни та доріжки:
   - Басейн "Команда розробників" (Developers Team)
   - Доріжка "Аналіз вимог" в басейні "Команда розробників" (Requirements Analysis Lane)
   - Басейн "Тестувальники" (Testers)
   - Басейн "Менеджмент" (Management)

Ця модель може бути використана для візуалізації та управління процесом розробки програмного забезпечення. Кожен елемент BPMN представляє конкретну дію або стан у процесі, а потоки вказують послідовність з'єднання між елементами. Басейни та доріжки допомагають організувати активності за командами чи виконавцями.

BPMN дозволяє гнучко візуалізувати та аналізувати процеси управління IT-проектом, розуміти їх послідовність та взаємозв'язки, інтегрувати бізнес-правила та автоматизувати їх виконання. Визначені в процесі моделі можуть бути використані для оптимізації робочих процесів, забезпечення контролю якості та порядку дій, а також для прогнозування та управління ризиками управління проектами[18].

Таким чином, BPMN є потужним інструментом для моделювання та управління процесами управління IT-проектами у СППР.

3.3.2. Інтеграція процесів управління проектом у систему підтримки прийняття рішень

Для розробки інтеграції та взаємодії між системою управління проектом та системою прийняття рішень на основі BPMN моделі можна використовувати такі кроки:

Аналіз потреб користувачів: Важливо зрозуміти потреби користувачів системи управління проектом та системи прийняття рішень. Це допоможе визначити необхідні функції та можливості інтерфейсу.

Визначення взаємодії між системами: Розробка інтерфейсу передбачає взаємодію між системою управління проектом та системою прийняття рішень на основі BPMN моделі. Важливо визначити, яка інформація потрібна передавати між цими системами та яким чином це має відбуватись.

Розробка візуальної компоненти для відображення BPMN моделі: Інтерфейс має включати візуальну компоненту для відображення BPMN моделі. Це може бути діаграма, яка показує елементи моделі, їх послідовність та зв'язки між ними.

Реалізація взаємодії між системами: Необхідно реалізувати механізм передачі даних між системами управління проектом та системою прийняття рішень на основі BPMN моделі. Це може здійснюватись через передачу файлів, використання API чи інші методи взаємодії.

Розробка функціональних елементів для прийняття рішення: Інтерфейс також має включати функціональні елементи для прийняття рішення на основі BPMN моделі. Це можуть бути елементи для оцінки та аналізу ризиків, планування проекту, прийняття важливих рішень тощо.

Тестування та вдосконалення інтерфейсу: Після розробки інтерфейсу важливо провести його тестування та отримати відгуки користувачів. Це допоможе виявити можливі проблеми та удосконалити інтерфейс для забезпечення ефективної взаємодії між системами[18].

Розробка інтерфейсу для взаємодії між системою управління проектом та системою прийняття рішень на основі BPMN моделі є складним завданням, яке потребує ретельного планування та аналізу потреб користувачів. Використання методології розробки програмного забезпечення та тестування допоможе забезпечити якість та ефективність інтерфейсу. Розглянемо один із екранів розробленої системи:

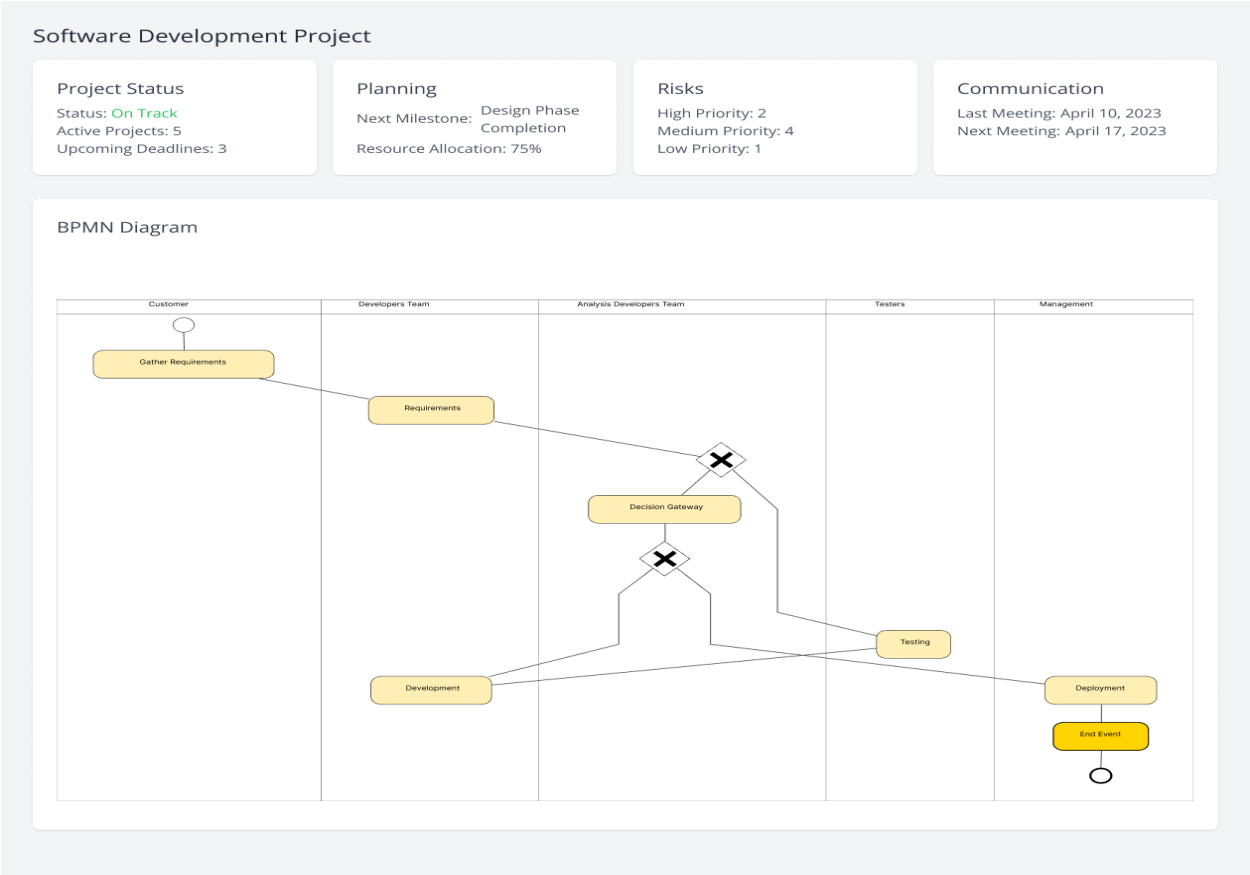


Рис. 3.3. Екран деталей проекту розробленої системи

На рис. 3.3 зображено один із екранів інтерфейсу для взаємодії між системою управління проектом та системою прийняття рішень на основі BPMN моделі. В межах кожного проекту можуть бути окремі сторінки з детальною інформацією про проект.

На цих сторінках можуть бути показані різні елементи BPMN моделі, такі як завдання, схвалення, події та потоки.

Кожен вузол моделі може бути натиснутий для відображення деталей процесу, стану завдань та взаємозв'язків.

Інші екрани розробленої системи:

Головна сторінка

- На головній сторінці інтерфейсу можуть бути відображені загальні дані про проекти, такі як стан проектів, планування, ризики та комунікація.

- Використовуючи діаграму BPMN, користувач може відразу бачити структуру та стан проектів.

Управління ризиками

- Інтерфейс може містити окремий розділ для управління ризиками в проекті.

- Користувачі можуть додавати, оцінювати та моніторити ризики на основі BPMN моделі.

- Можливість визначення вірогідності ризиків, впливу та управління ними забезпечує команду проекту необхідною інформацією для прийняття рішень.

Вибір альтернативних шляхів

- Інтерфейс може представляти можливі альтернативні шляхи розвитку проекту на основі BPMN моделі.

- За допомогою візуальної діаграми користувач може переглянути різні шляхи розвитку та їх вплив на проект.

- Користувач може прийняти рішення щодо вибору конкретного шляху та побачити сценарії розвитку проекту залежно від вибору.

Моніторинг та звітність

- Інтерфейс може мати можливість моніторингу та звітності на основі даних з системи управління проектом та системи прийняття рішень на основі BPMN моделі.

- Користувач може переглядати різні графіки, діаграми та статистику, що допомагає оцінити стан проекту та приймати рішення на основі цих даних.

Цей приклад показує, як інтерфейс може забезпечити взаємодію між системою управління проектом та системою прийняття рішень на основі BPMN моделі. Він показує, як візуальне представлення проекту, управління ризиками, вибір альтернативних шляхів та звітність можуть бути інтегровані для забезпечення ефективного управління проектами та прийняття обґрунтованих рішень.

3.4. Розробка та реалізація системи підтримки прийняття рішень в управлінні IT-проектом

В сучасному світі розробка програмного забезпечення є складним завданням, що вимагає не тільки високих технічних знань, але й ефективного управління. Однією з ключових галузей управління є впровадження систем прийняття рішень в управлінні IT-проектами. Ці системи відіграють важливу роль у забезпеченні ефективного прийняття стратегічних та тактичних рішень на різних етапах життєвого циклу проекту[18].

У контексті розробки програмного забезпечення, особливу увагу заслуговує розгляд систем прийняття рішень, побудованих на основі мікросервісної архітектури. Мікросервіси стали потужним інструментом для створення розподілених, гнучких та легко масштабованих систем. В їх основі – ідея про розбиття функціоналу на невеликі та автономні блоки, які можуть функціонувати незалежно один від одного.

Розробка системи підтримки прийняття рішень на основі мікросервісів вимагає глибокого розуміння технічних, бізнесових та управлінських аспектів. Вона повинна враховувати особливості взаємодії мікросервісів, ефективно використовувати дані та надавати інструменти для прийняття обґрунтованих рішень на кожному етапі проекту.

У цьому контексті важливо розглядати такі аспекти, як розподіл функціоналу, комунікація та інтеграція, безпека, моніторинг та оптимізація, спрощення впровадження та оновлення, аналіз та звітність. Ці елементи

допомагають забезпечити ефективність системи підтримки прийняття рішень у контексті розробки програмного забезпечення.

У наш час, коли ринок програмного забезпечення стрімко розвивається, використання передових технологій та ефективних методів управління стає визначальним фактором успіху. Розробка системи підтримки прийняття рішень на базі мікросервісної архітектури є важливим кроком у напрямку створення інноваційних та конкурентоздатних IT-проектів.

Розроблена система складається з декількох модулів, які виконують різні функції. У нас є модуль збору даних, який отримує дані про IT-технології з різних джерел, таких як веб-сторінки або API. Цей модуль розроблений як окремий мікросервіс, що забезпечує його незалежність і перевикористання в інших системах.

Також у нас є модуль аналізу даних, який виконує різні аналітичні алгоритми для визначення найкращих варіантів IT-технологій. Цей модуль також реалізований як окремий мікросервіс, який може використовуватись незалежно від інших компонентів системи.

Крім того, ми маємо модуль відображення результатів, який дозволяє візуалізувати інформацію про найкращі варіанти IT-технологій та надає користувачам зрозуміле представлення результатів аналізу. Цей модуль також реалізований як окремий мікросервіс зі своїм власним інтерфейсом користувача.

Кожен з цих мікросервісів можна розробляти, тестувати та розгортати незалежно від інших. Вони можуть комунікувати між собою за допомогою API, що дозволяє їм взаємодіяти та обмінюватись даними. Така архітектура дозволяє нам гнучко розвивати та змінювати окремі компоненти системи без впливу на решту системи, а також масштабувати їх за необхідністю.

Загалом, розробка цієї системи підтримки прийняття рішень в управлінні IT-проектом на основі мікросервісної архітектури дозволяє нам розподілити функціональність на окремі компоненти, які є незалежними та

можуть комунікувати між собою. Це забезпечує гнучкість, швидкість впровадження та масштабованість системи.

3.4.1. Вибір технологій та інструментів для розробки системи

Система підтримки прийняття рішень в управлінні IT-проектом є складною та багатофункціональною системою, яка вимагає використання сучасних технологій та інструментів для її розробки.

Запропонований стек технологій та інструментів – **Java, Angular, PostgreSQL, Kubernetis, Docker і Git**, надає потужні інструменти для швидкого розробки, масштабування та розгортання системи.

Java є мовою програмування, яка є дуже популярною у галузі розробки підприємницьких додатків. Завдяки своїй універсальності та потужності, Java дозволяє розробити потужний та безпечний бекенд для системи. Spring Framework – це один з найпопулярніших фреймворків для розробки на Java, який надає готові рішення для керування веб-додатками та інтеграції з базами даних.

Angular – це популярний фреймворк JavaScript для розробки користувацького інтерфейсу веб-додатків. Він надає потужні засоби для створення динамічних та інтерактивних клієнтських додатків.

PostgreSQL – це потужна та надійна база даних з відкритим кодом. Вона надає високий рівень безпеки, масштабованість та розширюваність.

Kubernetes – це платформа для керування контейнеризованими додатками у розподіленому середовищі. Використання Kubernetes дозволяє автоматизувати розгортання, масштабування та керування додатками, забезпечуючи високу доступність та надійність системи.

Docker – це інструмент для контейнеризації додатків, що дозволяє упакувати додатки та їх залежності в незмінні контейнери. Використання Docker полегшує розгортання та управління додатками, забезпечуючи єдинообразність та відтворюваність середовища[19].

Git – це система контролю версій, яка дозволяє ефективно керувати та спільно розвивати кодову базу. Git дозволяє команді розробників працювати над одним проектом, використовуючи версію кожного змінного набору файлів і автоматизує процес злиття змін з різних гілок.

Крім того, для автоматизації процесу збирання, тестування та розгортання можна використовувати інструменти, такі як Jenkins або GitLab CI/CD, що дозволяють створювати та керувати CI/CD-пайплайнами.

Запропонований стек технологій та інструментів надає потужні та гнучкі засоби для розробки та управління системою підтримки прийняття рішень в управлінні IT-проектом. Використання цих технологій допоможе створити високоякісну та масштабовану систему, яка забезпечить ефективне управління вашим IT-проектом.

3.4.2. Проектування бази даних системи

Для проектування бази даних для нашої системи підтримки прийняття рішень для управління IT-проектом, ми можемо розглянути наступні сутності та взаємозв'язки:

- Сутність "IT-Технології" – ця сутність буде містити інформацію про різні IT-технології, такі як назва, опис, категорія, характеристики тощо.
- Сутність "Джерело Даних" – ця сутність буде містити інформацію про джерела, з яких ми отримуємо дані про IT-технології, такі як назва, тип (веб-сторінка, API тощо), URL тощо.
- Сутність "Модуль Збору Даних" – ця сутність буде містити інформацію про модулі збору даних, які використовуються для отримання даних про IT-технології з різних джерел. Вона буде містити відомості про назву модуля, опис, автора, джерела даних, з яких він отримує дані.
- Сутність "Аналітичний Алгоритм" – ця сутність буде містити інформацію про алгоритми аналізу даних, які використовуються для

визначення найкращих варіантів ІТ-технологій. Вона буде містити відомості про назву алгоритму, опис, автора тощо.

- Сутність "Модуль Аналізу Даних" – ця сутність буде містити інформацію про модулі аналізу даних, які використовуються для виконання аналітичних алгоритмів. Вона буде містити відомості про назву модуля, опис, автора, аналітичні алгоритми, які він використовує.

- Сутність "Результат Аналізу" – ця сутність буде містити інформацію про результати аналізу даних, такі як ІТ-технології, що були визначені як найкращі варіанти, значення показників ефективності тощо.

- Сутність "Модуль Відображення Результатів" – ця сутність буде містити інформацію про модулі відображення результатів, які використовуються для візуалізації інформації про найкращі варіанти ІТ-технологій. Вона буде містити відомості про назву модуля, опис, автора тощо.

Тепер ми можемо визначити взаємозв'язки між цими сутностями:

- Сутність "ІТ-Технології" має взаємозв'язок багато-до-багатьох з сутністю "Джерело Даних", оскільки одна ІТ-технологія може мати декілька джерел, і одне джерело може мати багато ІТ-технологій. Ми можемо використати проміжну табличку для зберігання цього взаємозв'язку.

- Сутність "Модуль Збору Даних" має взаємозв'язок багато-до-багатьох з сутністю "Джерело Даних", оскільки один модуль може отримувати дані з багатьох джерел, і одне джерело може бути використане декількома модулями. Знову, ми можемо використати проміжну табличку для зберігання цього взаємозв'язку.

- Сутність "Модуль Збору Даних" також має взаємозв'язок один-до-багатьох з сутністю "ІТ-Технології", оскільки один модуль може збирати дані по одній або декільком ІТ-технологій.

- Сутність "Модуль Аналізу Даних" має взаємозв'язок один-до-багатьох з сутністю "Аналітичний Алгоритм", оскільки один модуль може використовувати один або декілька алгоритмів аналізу.

- Сутність "Результат Аналізу" має взаємозв'язок багато-до-одного з сутністю "IT-Технології", оскільки для кожного результату аналізу визначається IT-технологія.

- Сутність "Модуль Відображення Результатів" має взаємозв'язок один-до-багатьох з сутністю "Результат Аналізу", оскільки один модуль може відображати результати аналізу для багатьох IT-технологій.

Рис. 3.3. Діаграма зпроектованої БД системи

Це лише загальна структура бази даних для нашої системи підтримки прийняття рішень. Можна додати більше атрибутів та специфіку для кожної сутності відповідно до конкретної потреби. Також можуть знадобитися додаткові таблиці або сутності для зберігання вхідних та проміжних даних.

3.4.3. Розробка модуля збору даних

Якщо ми розглядаємо систему підтримки прийняття рішень для управління IT-проектом, модуль збору даних може включати отримання інформації про IT-технології з різних джерел.

Одним з можливих джерел може бути база даних, в якій вже зберігаються дані про IT-технології. В цьому випадку модуль збору даних може просто виконувати запити до бази даних для отримання необхідної інформації[19].

Наприклад, використовуючи мову програмування Java та фреймворк Spring, можна реалізувати модуль збору даних, який виконує запити до бази даних. Нижче наведено приклад коду для отримання інформації про IT-технології з бази даних за допомогою Spring Data JPA:

```
@Repository
public interface TechnologyRepository extends JpaRepository<Technology, Long>
{
    // Метод для отримання всіх IT-технологій з бази даних
    List<Technology> findAll();
}

@Service
public class TechnologyService {
    private final TechnologyRepository technologyRepository;

    public TechnologyService(TechnologyRepository technologyRepository) {
        this.technologyRepository = technologyRepository;
    }

    public List<Technology> getAllTechnologies() {
        return technologyRepository.findAll();
    }
}
```

Рис.3.4. Приклад TechnologyRepository

У цьому прикладі TechnologyRepository – це інтерфейс Spring Data JPA, який надає методи для взаємодії з базою даних. Клас TechnologyService є сервісним компонентом, який використовує TechnologyRepository для отримання інформації про IT-технології з бази даних.

Використання Spring Data JPA дозволяє спростити роботу з базою даних та виконання запитів до неї.

Крім бази даних, модуль збору даних може також використовувати інші джерела, такі як зовнішні API або веб-сторінки. Остаточний вибір

імплементації модуля збору даних залежить від конкретних потреб системи та джерел, з яких отримуєте дані.

Розглянемо приклад отримання даних про IT-технології з API Github. Спочатку нам знадобиться клас, який буде виконувати запити до API Github та отримувати дані. Ось приклад коду для цього:

```
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component
public class GithubApiService {
    private static final String GITHUB_API_URL = "https://api.github.com";

    public List<Technology> getTechnologiesFromGithub() {
        RestTemplate restTemplate = new RestTemplate();
        String url = GITHUB_API_URL +
"/repositories?language={language}&sort=stars&order=desc";

        Map<String, String> uriVariables = new HashMap<>();
        uriVariables.put("language", "java");

        ResponseEntity<List<GithubRepository>> response =
restTemplate.exchange(
                url, HttpMethod.GET, null,
                new ParameterizedTypeReference<List<GithubRepository>>() {},
                uriVariables
        );

        List<GithubRepository> repositories = response.getBody();

        // Перетворення даних з GithubRepository в Technology
        List<Technology> technologies = new ArrayList<>();
        for (GithubRepository repository : repositories) {
            Technology technology = new Technology();
            technology.setName(repository.getName());
            technology.setDescription(repository.getDescription());
            // Додатковий код для перетворення інформації з GithubRepository
y Technology
            // ...

            technologies.add(technology);
        }

        return technologies;
    }
}
```

Рис.3.5. Приклад RestTemplate

У цьому прикладі ми використовуємо RestTemplate, який є частиною Spring Framework, для виконання запитів до API Github. Ми виконуємо GET-запит до API, вказуючи мову "java" і сортуємо репозиторії за кількістю зірок.

Далі ми отримуємо відповідь в форматі ResponseEntity<List<GithubRepository>>, де GithubRepository – це модель, яка відповідає структурі даних, отриманих з API Github.

Потім перетворюємо дані з GithubRepository в нашу модель Technology та повертаємо список технологій.

3.4.4. Розробка модуля аналізу даних

Задачею модуля аналізу даних в системі підтримки прийняття рішень є виконання різних аналітичних алгоритмів для визначення найкращих варіантів IT-технологій. Ось приклад коду на Java, який демонструє реалізацію аналітичного алгоритму для оцінки технологій за їхніми характеристиками:

```
public class TechnologyAnalyzer {
    public List<Technology> analyze(List<Technology> technologies) {
        // Виконання аналітичних розрахунків із заданими їхніми
характеристиками
        for (Technology technology : technologies) {
            double score = calculateScore(technology);
            technology.setScore(score);
        }
        // Сортування технологій за оцінкою

technologies.sort(Comparator.comparing(Technology::getScore).reversed());
        return technologies;
    }
    private double calculateScore(Technology technology) {
        // Приклад аналітичного розрахунку оцінки технології
        double score = 0.0;
        // Додатковий код для розрахунку оцінки технології на основі її
характеристик
        return score;
    }
}
```

Рис.3.6. Приклад TechnologyAnalyzer

У цьому прикладі ми маємо клас TechnologyAnalyzer, який містить метод analyze, що відповідає за аналіз технологій. У цьому методі, для кожної технології, ми викликаємо метод calculateScore, який розраховує оцінку технології на основі її характеристик та інших параметрів. Оцінка зберігається в полі score об'єкту Technology.

Після отримання оцінок всіх технологій, ми сортуємо їх за спаданням оцінки. В результаті отримуємо список технологій відсортований за їхньою оцінкою.

Зважаючи на тип системи підтримки прийняття рішень для управління IT-проектом, модуль аналізу даних може виконувати аналіз ризиків, які пов'язані з IT-технологіями. Ось приклад коду на Java, який демонструє реалізацію підхоплення і аналізу ризиків:

```
public class RiskAnalyzer {

    public List<Risk> analyze(List<Technology> technologies) {
        List<Risk> risks = new ArrayList<>();

        for (Technology technology : technologies) {
            if (isHighRisk(technology)) {
                risks.add(new Risk(technology.getName(), "High Risk"));
            } else if (isMediumRisk(technology)) {
                risks.add(new Risk(technology.getName(), "Medium Risk"));
            } else {
                risks.add(new Risk(technology.getName(), "Low Risk"));
            }
        }

        return risks;
    }

    private boolean isHighRisk(Technology technology) {
        // Логіка визначення високого ризику для технології
        // Приклад: технологія, яка ще не пройшла достатньо тестування
        return !technology.isTested();
    }

    private boolean isMediumRisk(Technology technology) {
        // Логіка визначення середнього ризику для технології
        // Приклад: технологія, яка має певні проблеми з безпекою
        return technology.hasSecurityIssues();
    }
}
```

Рис.3.7. Приклад RiskAnalyzer

У цьому прикладі ми маємо клас RiskAnalyzer, який містить метод analyze, що приймає список технологій і повертає список ризиків. Для кожної технології, ми перевіряємо, чи є вона високим ризиком, чи середнім ризиком, чи низьким ризиком, та створюємо відповідний об'єкт Risk.

У цьому прикладі логіка визначення ризиків досить загальна, і Ви можете розробити власні критерії для визначення ризиків, виходячи з особливостей вашого проекту та системи підтримки прийняття рішень.

Цей приклад демонструє спосіб аналізу ризиків на основі характеристик технологій, але в реальному проєкті можуть бути використані інші критерії і алгоритми для аналізу ризиків.

3.4.5. Розробка модуля аналізу даних

Для розробки бекенду модуля відображення результатів можна використовувати Spring Framework.

Основні компоненти, які можна розглядати в модулі відображення результатів:

Контролер – це клас, який відповідає за обробку запитів користувача та взаємодію з рештою компонентів модуля. Він створюється за допомогою анотації @Controller та може мати методи для отримання даних з моделі та передачі їх у представлення. Наприклад:

```
@Controller
public class ResultsController {
    @Autowired
    private ResultsService resultsService;
    @GetMapping("/results")
    public String showResults(Model model) {
        List<Result> results = resultsService.getResults();
        model.addAttribute("results", results);
        return "results";
    }
}
```

Рис.3.8. Приклад створення контролеру

Сервіс – це клас, в якому виконується бізнес-логіка модуля, така як отримання даних з бази даних або іншого джерела. Він створюється за допомогою анотації @Service та може містити методи, які повертають результати аналізу. Наприклад:

```
@Service
public class ResultsService {

    @Autowired
    private ResultsRepository resultsRepository;

    public List<Result> getResults() {
        return resultsRepository.findAll();
    }
}
```

Рис.3.9. Приклад створення сервісу

Репозиторій – це інтерфейс, який описує методи доступу до даних у базі даних або іншому джерелі. Він створюється за допомогою анотації @Repository та може використовувати JpaRepository або інші інтерфейси Spring Data JPA для спрощення роботи з базою даних. Наприклад:

```
@Repository
public interface ResultsRepository extends JpaRepository<Result, Long> {

    // Додаткові методи для роботи з базою даних

}
```

Рис.3.10. Приклад репозиторію(без імплементацій)

Для розробки фронтенду на Angular створимо компоненти, шаблони та служби Angular, які будуть відповідати за відображення даних та взаємодію з бекендом.

```
import { Component, OnInit } from '@angular/core';
import { ResultsService } from './results.service';

@Component({
        selector: 'app-results',
        templateUrl: './results.component.html',
        styleUrls: ['./results.component.css']
        })
export class ResultsComponent implements OnInit {
    projectId: string;
    results: any;

    constructor(private resultsService: ResultsService) { }

    ngOnInit() {
        // Отримання результатів при завантаженні компоненту
        this.getResults();
    }

    getResults() {
        this.resultsService.getResults(this.projectId).subscribe((data) => {
                this.results = data;
    });
    }

    saveResults() {
        this.resultsService.saveResults(this.projectId,
this.results).subscribe((data) => {
                console.log("Результати збережено");
    });
    }
}

import { Injectable } from '@angular/core';
        import { HttpClient } from '@angular/common/http';

@Injectable({
        providedIn: 'root'
        })
export class ResultsService {
    baseUrl = "http://localhost:8080/api/results";

    constructor(private http: HttpClient) { }

    getResults(projectId: string) {
        return this.http.get(`${this.baseUrl}/${projectId}`);
    }

    saveResults(projectId: string, results: any) {
        return this.http.post(`${this.baseUrl}/${projectId}`, results);
    }
}
```

Рис.3.11. Приклад створення Angular компоненти та сервісу

Також, можна використовувати стилі розбудови архітектури, такі як розподілений шаблон проектування (Distributed Design Pattern), щоб розділити

компоненти модуля на окремі мікросервіси або модулі для більшої гнучкості та масштабованості.

Загалом, архітектура модуля відображення результатів повинна бути пристосована до специфіки конкретного проекту та вимог користувачів, але з використанням MVC та інших добре відомих принципів проектування, можна створювати структуровану та ефективну архітектуру для даного модулю.

3.5. Технології розробки СППР в хмарному середовищі

В сучасних системах підтримки прийняття рішень (СППР) все більше застосовується архітектура мікросервісів, що дозволяє розгортати та масштабувати додатки в хмарному середовищі. Використання хмарних технологій дозволяє покращити ефективність, масштабованість та доступність системи, а також знизити вартість її розгортання та управління[19].

У цьому розділі ми розглянемо технології, що використовуються для розробки мікросервісів в системах підтримки прийняття рішень та їх розгортання в хмарному середовищі. Ми розглянемо такі технології, як контейнеризація, оркестрування, способи «спілкування» між мікросервісами, система контролю версій та інтеграція з CI/CD-платформами.

3.5.1. Способи комунікації мікросервісів між собою

Для комунікації між мікросервісами у хмарному середовищі можуть використовуватись різні способи залежно від потреб проекту. Основні способи комунікації між мікросервісами включають такі:

HTTP/REST API: Цей спосіб є найпоширенішим для комунікації між мікросервісами в хмарному середовищі. Використовуючи HTTP-протокол, мікросервіси можуть взаємодіяти один з одним, відправляючи запити та отримуючи відповіді через REST API (Representational State Transfer). REST API забезпечує масштабованість та гнучкість взаємодії між сервісами і може

бути реалізований за допомогою різних технологій, таких як Spring Boot або Express.js.

Message Queues: Іншим способом комунікації між мікросервісами може бути використання повідомленнь черги. В такому випадку, мікросервіси можуть відправляти та отримувати повідомлення через посередники повідомлень, такі як Kafka, RabbitMQ або AWS Simple Queue Service (SQS). Це дозволяє асинхронну комунікацію між сервісами та забезпечує здатність обробки повідомлень поза межами робочого процесу.

Service Mesh: Service mesh є модерним підходом до комунікації між мікросервісами в хмарному середовищі. Він використовує проксі-сервери, які вбудовані в кожен мікросервіс, для обробки мережевого трафіку, моніторингу та контролю розподілу навантаження. Це дозволяє отримати більшу відмовостійкість та забезпечує багато функціональностей, такі як загальні механізми отримання метрик, логування та авторизації. Продукти, які пропонують сервіси мережі, включають Istio, Linkerd і AWS App Mesh.

Як інший підхід можуть використовуватись ***RPC*** (Remote Procedure Call) фреймворки, такі як gRPC або Apache Thrift. Вони надають інтерфейс клієнта-сервера для виклику методів між мікросервісами.

Обираючи спосіб комунікації між мікросервісами, слід враховувати такі фактори, як складність реалізації, масштабованість, надійність та виконавчі вимоги проекту[19].

Вибір Message Queues (Рис. 3.4) для комунікації між мікросервісами є розумним альтернативою для нашої системи підтримки прийняття рішень для управління IT-проектом з наступних причин:

Розв'язання асинхронності: Message Queues дозволяють нам відокремлювати відправлення повідомлень від їх отримання. Це означає, що ми можемо відправляти повідомлення без очікування результату, а одержувати їх у зручний момент. Це забезпечує нам актуальні дані та більшу пропускну здатність системи.

Розділення масштабування: Використання Message Queues дозволяє нам розміщувати наші мікросервіси на різних серверах та масштабувати їх окремо. Оскільки система підтримки прийняття рішень для управління IT-проектом має різні модулі, які можуть мати різне навантаження, ми можемо масштабувати частину системи, яка є найбільш вимогливою до ресурсів.

Tolerant to Failures: Message Queues дозволяють нам створювати резервні системи, які можуть приймати повідомлення, в разі відмови основної системи. Це забезпечує нам високу доступність та надійність системи.

Розділення модулів: Message Queues допомагають нам відокремити модулі нашої системи та забезпечити їх незалежність. Кожен модуль може мати свою власну чергу повідомлень, в яку він буде відправляти та з якої отримувати повідомлення. Це спрощує розгортання, тестування та підтримку окремих модулів системи.

Масштабованість: Message Queues дозволяють нам швидко масштабувати нашу систему в разі необхідності. Ми можемо додавати нові сервери та масштабувати черги повідомлень, щоб забезпечити надійну та продуктивну роботу системи.

Такі переваги дозволяють нам використовувати Message Queues як спосіб комунікації між мікросервісами для нашої системи підтримки прийняття рішень в управлінні IT-проектом.

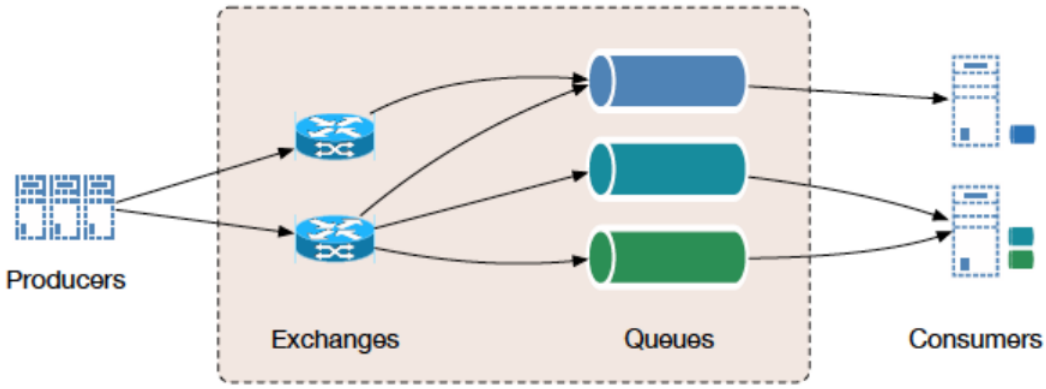


Рис. 3.12. Схема асинхронного обміну повідомленнями

Найбільш поширеними брокерами повідомлень є ActiveMQ, RabbitMQ та Apache Kafka.

Для розробки даної системи було обрано систему **Kafka** (Рис. 3.5), тому, що вона має певні переваги перед іншими.

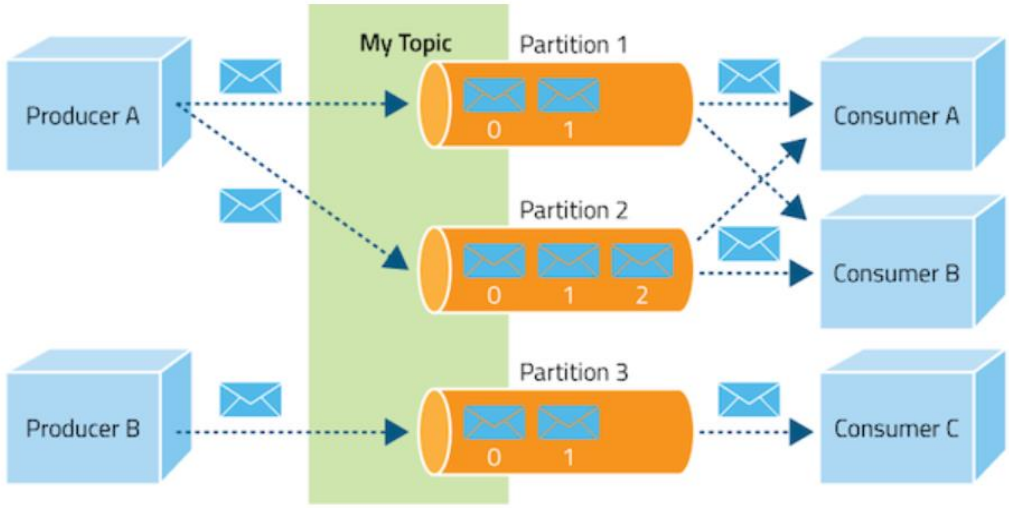


Рис. 3.13. Архітектура системи Apache Kafka

Kafka – це розподілена платформа обміну повідомленнями, яка дозволяє ефективно обробляти великі обсяги даних та забезпечує надійну та швидку передачу повідомлень між різними компонентами системи. Основні переваги Kafka включають:

Висока пропускна здатність: Kafka побудована для обробки великого обсягу даних, що дозволяє передавати велику кількість повідомлень за одиницю часу. Вона оптимізована для обробки десятків тисяч повідомлень за секунду, навіть на одному сервері.

Масштабованість: Kafka розроблена для масштабованості, що дозволяє легко розширювати та масштабувати систему в залежності від потреб. Вона може обробляти сотні та навіть тисячі брокерів і підтримує розподілену обробку повідомлень.

Надійність: Kafka забезпечує надійну передачу повідомлень, навіть у випадку відмови частини системи. Вона зберігає повідомлення на диску та реплікує їх на кількох серверах, щоб забезпечити надійність передачі.

Гарантія доставки повідомлень: Kafka забезпечує гарантію доставки повідомлень, що означає, що повідомлення будуть доставлені до отримувача або збережені, якщо отримувач недоступний. Це дозволяє гарантувати, що дані не будуть втрачені.

Гнучкість: Kafka може використовуватись для різних цілей, від зберігання журналів подій до обміну повідомленнями між різними компонентами системи. Вона надає різні API для вирішення різних задач та може бути інтегрована з різними технологіями.

Система розподіленого потоку даних: Kafka може використовуватись як система розподіленого потоку даних, що дозволяє обробляти потоки даних у реальному часі та аналізувати їх. Вона підтримує механізми для обробки даних в реальному часі та зберігання їх у тривалості для аналітики.

3.5.2. Контейнеризації системи за допомогою Docker

Одним з ключових завдань у розробці програмних застосунків за використання мікросервісної архітектури є розгортання та управління великою кількістю серверних додатків у єдиному середовищі, налаштування масштабування, реплікації сервісів та виявлення сервісів у єдній мережі (service discovery). Автоматизовані завдання розгортання, координації та управління розгортанням великих комп'ютерних систем описуються терміном "оркестрація". Оркестрація визначає, як сервіси повинні взаємодіяти, обмінюючись повідомленнями, включаючи бізнес-логіку та послідовність дій[20].

В основі будь-якої технології автоматизованої оркестрації лежить віртуалізація. Однак відомо, що будь-яка технологія віртуалізації, така як

VirtualBox, має суттєвий вплив на продуктивність системи. Тому, враховуючи проблеми віртуалізації, виникла область контейнеризації.

Контейнери з'явилися на корпоративному ринку недавно, і багато цього досягнення пов'язане з технологією **Docker**. Ця технологія дозволяє запускати додатки в контейнерах, отримуючи результат, схожий на звичайну віртуальну машину, але більш ефективний. Легкість, зменшена витрати ресурсів і практична незалежність від інфраструктури практично повністю сприяли переходу від звичайних віртуальних машин до контейнерів.

Розглянемо архітектуру системи Docker (Рис. 3.6). Docker використовує клієнт-серверну архітектуру. Docker клієнт взаємодіє з так званим процесом Docker (демоном), чиєю відповідальністю є виконання завдань, пов'язаних з створенням, запуском та розподілом контейнерів. Важливо відзначити, що як клієнт, так і сервер можуть працювати на одній системі; також можна підключити клієнта до віддаленого процесу Docker. Клієнт та сервер взаємодіють через сокет або RESTful API.

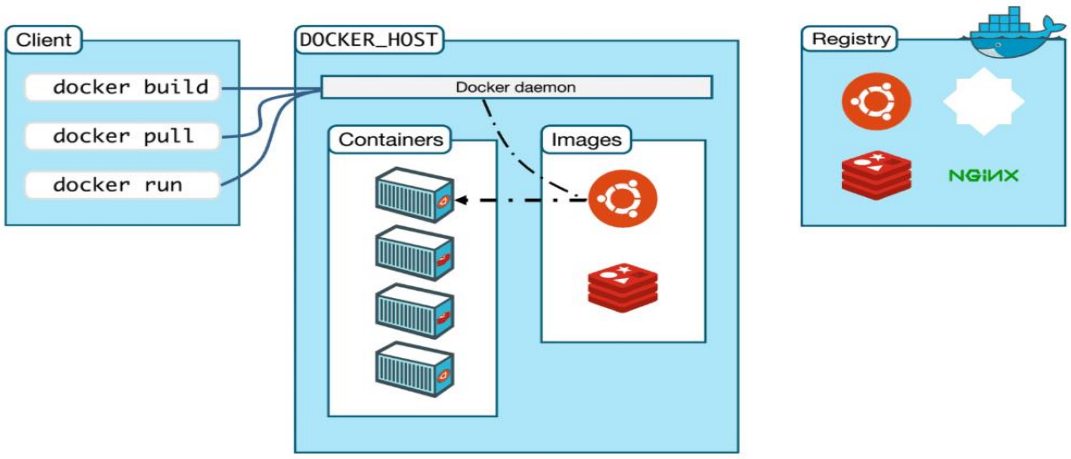


Рис. 3.14. Архітектура системи Docker

Контейнеризація системи за допомогою Docker дозволяє нам зберігати та виконувати кожну компоненту нашої системи у власному ізольованому середовищі, відокремленим від інших компонентів та залежностей. Це забезпечує портативність, стабільність та простоту установки та розгортання.

Ось кроки, які ми можемо виконати, щоб сконтейнеризувати нашу систему за допомогою Docker:

Створення Dockerfile: Dockerfile є текстовим файлом, в якому ми описуємо всі кроки необхідні для створення контейнера. У нашому Dockerfile ми вказуємо базовий образ, установлюємо всі залежності та наше програмне забезпечення.

Приклад Dockerfile для модуля збору даних може виглядати так:

```
FROM openjdk:8-jdk-alpine
COPY target/datagathering.jar /app/datagathering.jar
CMD ["java", "-jar", "/app/datagathering.jar"]
```

Побудова контейнера: За допомогою команди docker build ми побудуємо контейнер на основі Dockerfile. Команда шукає Dockerfile в поточній директорії та виконує всі необхідні кроки для створення образу контейнера.

Приклад команди для побудови контейнера модуля збору даних:

```
docker build -t datagathering .
```

Запуск контейнера: Контейнер можна запустити за допомогою команди docker run. Ми також можемо вказати необхідні параметри, такі як налаштування з'єднання з базою даних або встановлення змінних середовища.

Приклад команди для запуску контейнера модуля збору даних:

```
docker run -d --name datagathering-container -p 8080:8080 datagathering
```

Продовження для інших компонентів: Аналогічні кроки будуть виконані для контейнерів інших компонентів нашої системи, таких як модуль аналізу даних та модуль відображення результатів. Кожен контейнер може мати свій власний Dockerfile та конфігураційні параметри.

Охорона контейнерів: Docker дає нам можливість охороняти контейнери за допомогою мережевих політик та доступу до ресурсів. Ми можемо налаштувати файрвол та використовувати мережеві контейнери для обмеження доступу до наших компонентів для інших контейнерів або зовнішнього середовища.

Управління контейнерами: Docker також надає набір команд для управління контейнерами, таких як зупинка, перезапуск або видалення. Ми можемо використовувати команди ***docker stop***, ***docker restart*** та ***docker rm*** для керування нашими контейнерами.

Таким чином, ми зберігаємо та запускаємо кожен компонент нашої системи у власному контейнері, що дозволяє нам легко керувати та масштабувати систему, а також забезпечує портативність і відокремленість компонентів.

3.5.3. Оркестрація контейнерів системи за допомогою Kubernetes

Kubernetes, який часто називають "K8s" (Рис. 3.7), є потужною та високоефективною системою управління контейнерами, розробленою для автоматизації деплою, масштабування та керування контейнеризованими додатками. Розроблений компанією Google і випущений як відкрите програмне забезпечення, Kubernetes надає рішення для вирішення складних завдань оркестрації, які виникають під час експлуатації розподілених систем.

Ключовим принципом Kubernetes є управління контейнерами — уніфікованим підходом до упакування, доставки та виконання програм в контейнерах. Контейнери забезпечують стандартизацію середовища виконання, що дозволяє легко переносити додатки між різними областями розробки, тестування та продуктивного використання.

Як система оркестрації, Kubernetes дозволяє автоматизувати процеси розгортання, масштабування та керування контейнеризованими додатками. Це досягається за допомогою концепції "підключення до декларативного конфігураційного файлу", де розробники можуть визначити потрібний стан системи, і Kubernetes самостійно працює над досягненням цього стану, розподіляючи ресурси та управляючи життєвим циклом контейнерів[20].

Використовуючи розподілені ресурси, Kubernetes може ефективно масштабувати додатки, забезпечуючи стійку та високодоступну

інфраструктуру. Завдяки системі плагінів та великої спільноті розробників, Kubernetes став популярним інструментом для розгортання та управління мікросервісними архітектурами, забезпечуючи простоту та надійність вирішення ряду завдань у сфері хмарних технологій.

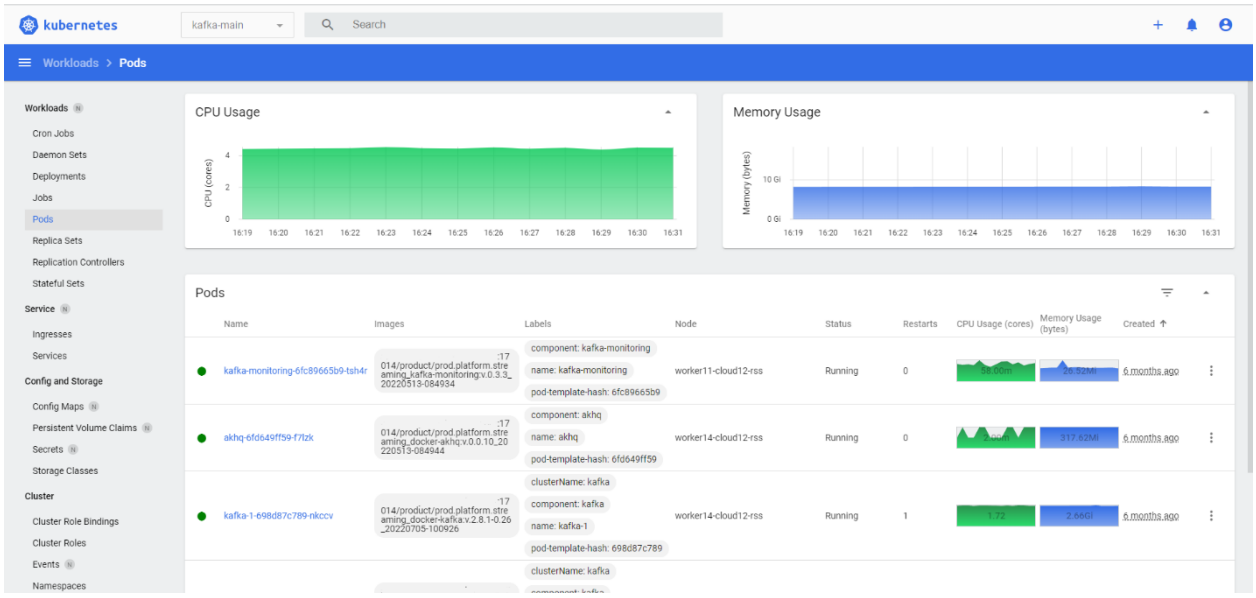


Рис. 3.15. Головний екран Kubernetes

Оркестрація контейнерів за допомогою Kubernetes дозволяє автоматизувати розгортання, масштабування та керування контейнерами в розподіленому середовищі. Для оркестрації контейнерів нашої системи підтримки прийняття рішень для управління IT-проектом, ми можемо виконати наступні дії:

- Налаштування кластера Kubernetes: Спочатку нам потрібно налаштувати кластер Kubernetes, що міститиме наші контейнери. Це може бути локальний кластер, який використовує віртуальні машини, або публічний веб-хмарний провайдер, такий як Amazon Web Services (AWS) або Google Cloud Platform (GCP).
- Підготовка конфігураційних файлів Kubernetes: Ми повинні створити конфігураційні файли для наших контейнерів і визначити ресурси,

такі як CPU, пам'ять та мережу, які вони використовують. Крім того, ми можемо налаштувати політики автоматичного масштабування і навантаження рівня.

- Створення маніфестів Kubernetes: Кубічна інфраструктура використовує маніфести, які описують потрібні ресурси та конфігурацію для розгортання і керування контейнерами. Ми повинні створити маніфести для кожного з наших контейнерів і встановити їх в кластері Kubernetes.
- Розгортання контейнерів: Після створення маніфестів ми можемо розгорнути наші контейнери, виконавши команду kubectl apply для кожного з маніфестів. Kubernetes автоматично розмістить контейнери на вільних вузлах кластера та надасть їм потрібні ресурси.
- Моніторинг та логування: Kubernetes надає механізми моніторингу та логування, такі як Prometheus та Grafana. Ми можемо налаштувати моніторинг та логування наших контейнерів для виявлення проблем та аналізу продуктивності системи.
- Масштабування контейнерів: За допомогою Kubernetes ми можемо масштабувати наші контейнери по вертикалі та горизонталі. Масштабування по вертикалі означає зміну розміру ресурсів, таких як CPU та пам'ять, які виділені для кожного контейнера. Масштабування по горизонталі означає додавання або видалення кількості реплік контейнера для розподілення навантаження.
- Керування та оновлення контейнерів: Kubernetes надає механізми для керування та оновлення контейнерів. Ми можемо вказати вмикання, вимкнення або перезапуск контейнерів, а також автоматичні оновлення контейнерів до нових версій.

Оркестрування контейнерів за допомогою Kubernetes надає нам багато переваг, таких як автоматичне розподілення навантаження, високу доступність, масштабованість та легке керування контейнерами. Це дозволяє нам створити стабільну та ефективну систему підтримки прийняття рішень для управління IT-проектом.

ВИСНОВОК ДО РОЗДІЛУ 3

Мікросервісна архітектура є ефективним варіантом для розробки систем підтримки прийняття рішень, такої як система управління IT-проектом. Вона дозволяє гнучко розподілити функціональність системи на окремі компоненти, які можуть бути розроблені, тестовані та розгортані незалежно один від одного. Це забезпечує швидке впровадження та масштабованість системи.

У нашому випадку, ми розглянули розробку трьох основних модулів для системи підтримки прийняття рішень управління IT-проектом: модуль збору даних, модуль аналізу даних і модуль відображення результатів. Кожен з цих модулів був реалізований як окремий мікросервіс, що забезпечує їх незалежність і перевикористання в інших системах.

Backend фреймворк Java з використанням Spring Framework був обраний для розробки модулів збору даних та аналізу даних. Java є потужною та універсальною мовою програмування, а Spring Framework надає широкий функціонал для створення високопродуктивних та масштабованих додатків.

Frontend фреймворк Angular був використаний для реалізації модуля відображення результатів. Angular є популярним фреймворком JavaScript для розробки користувацького інтерфейсу і надає потужні функції для створення динамічних та інтерактивних веб-додатків.

Для контейнеризації нашої системи ми використали Docker, що дозволяє упаковувати додатки та їх залежності в контейнери для легкої та консистентної поставки програмного забезпечення.

Керування контейнеризованими додатками було забезпечено за допомогою Kubernetes, що дозволяє автоматизувати розгортання, масштабування та керування додатками у контейнерах.

Система контролю версій Git була використана для управління та спільної розробки кодової бази проекту, що спрощує спільну роботу розробників та дозволяє ефективно відстежувати зміни в коді.

Для автоматизації процесу збирання, тестування та розгортання додатків ми можемо використовувати інструменти, такі як Jenkins або GitLab CI/CD, які надають широкий функціонал для створення та керування CI/CD-пайплайнами.

Загальний висновок полягає в тому, що розробка системи підтримки прийняття рішень для управління IT-проектом на основі мікросервісної архітектури забезпечує гнучкість, швидкість впровадження та масштабованість системи. Ця архітектура дозволяє незалежно розробляти, тестувати та розгортати окремі компоненти системи, що спрощує розробку та підтримку системи.

ВИСНОВКИ

У даній дипломній роботі було проведено дослідження методів, алгоритмів та моделей для побудови та оптимізації дерев рішень, а також їх використання в системах підтримки прийняття рішень.

У першому розділі розглянуто загальні цілі та методологію дослідження. Була проведена детальна аналітична робота з огляду літератури та існуючих методів для побудови дерев рішень. Вивчені різні структури даних та моделі для представлення і управління деревами рішень. Також було проаналізовано інструменти та технології для візуалізації дерев рішень. На основі проведеного дослідження був розроблений прототип системи підтримки прийняття рішень, який дозволяє створювати, оптимізовувати та візуалізувати дерева рішень.

У другому розділі досліджено використання інтелектуальних методів в системах управління проектами. Були розглянуті такі методи, як інтелектуальний аналіз даних, нейронні мережі, машинне навчання та обробка природних мов. Вивчено, як ці методи можуть бути застосовані для аналізу та прийняття рішень в управлінні проектами. Розглянуто різні сценарії використання цих методів, такі як прогнозування ризиків та можливостей проекту, класифікація даних проекту, автоматичний моніторинг ходу проекту та аналіз текстової інформації.

У третьому розділі розроблена архітектура та реалізована система підтримки прийняття рішень в управлінні IT-проектом. Проведено аналіз вимог до системи та визначено архітектурні принципи. Також розглянуті питання масштабованості та продуктивності системи в умовах управління IT-проектами, проаналізовано існуючі рішення для розміщення мікросервісного додатку в хмарному середовищі. Була розроблена модель процесів управління IT-проектом з використанням BPMN та оцінено можливості їх інтеграції з існуючими інструментами.

Загальний висновок до дипломної роботи полягає в тому, що побудова та оптимізація дерев рішень, а також використання інтелектуальних методів в

системах прийняття рішень, можуть відігравати важливу роль в управлінні проектами. Досліджений прототип системи підтримки прийняття рішень в управлінні ІТ-проектом дозволяє ефективно використовувати дерева рішень та інтелектуальні методи для аналізу та оптимізації управлінських процесів. Результати дослідження можуть бути використані як основа для подальшого вдосконалення систем управління проектами та розробки нових інтелектуальних методів для прийняття рішень.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Жук, О. (2014). Методи прийняття рішень в умовах невизначеності. Київ: КНЕУ. [Електронний ресурс]. – Режим доступу: https://core.ac.uk/download/pdf/86628377.pdf

2. Мельник, Л. (2010). Системний аналіз у прийнятті рішень. Київ: Центр учбової літератури. [Електронний ресурс]. – Режим доступу: http://www.kdu.edu.ua/new/PHD/RP_SAPUR.pdf

3. Шевченко, Т. (2015). Візуалізація даних у наукових дослідженнях. Київ: Видавництво Київського університету. [Електронний ресурс]. – Режим доступу: https://issuu.com/victoryshe/docs/3-0-1visual_2022

4. Бекман, Г. (2005). Моделювання процесів прийняття рішень. Київ: КНЕУ. [Електронний ресурс]. – Режим доступу: https://pidru4niki.com/2015101166606/menedzhment/modelyuvannya_protsesu_priynyattya_upravlinskih_rishen

5. Бабенко, Ю. (2012). Вступ до машинного навчання та розпізнавання образів. Київ: НТУУ "КПІ". [Електронний ресурс]. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/41525/1/Mashynne_navchania_Konspekt.pdf

6. Грабовенко, І. (2013). Обробка природних мов в системах управління. Київ: Видавництво КНУ. [Електронний ресурс]. – Режим доступу: https://www.setlab.net/?view=Philosophy_Knowledge

7. Король, О. (2018). Хмарні технології в розробці програмного забезпечення. Київ: Видавництво "Техніка". [Електронний ресурс]. – Режим доступу: http://www.economy.nayka.com.ua/pdf/8_2021/88.pdf

8. Quinlan, J. R. (1986). Induction of decision trees. Machine learning. [Електронний ресурс]. – Режим доступу: https://hunch.net/~coms-4771/quinlan.pdf

9. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning. [Електронний ресурс]. – Режим доступу: https://hastie.su.domains/Papers/ESLII.pdf

10. Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. [Електронний ресурс]. – Режим доступу: https://www.jstor.org/stable/2699986

11. Project Management Institute (PMI). A guide to the project management. [Електронний ресурс]. – Режим доступу: https://www.pmi.org/pmbok-guide-standards/foundational/pmbok

12. Kerzner, H. (2017). Project management: a systems approach to planning, scheduling, and controlling. [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books/about/Project_Management.html?id=DvInzgEACAAJ&redir_esc=y

13. Breiman, L. (2001). Random forests. Machine learning. [Електронний ресурс]. – Режим доступу: https://link.springer.com/article/10.1023/A:1010933404324

14. Bishop, C. M. (2006). Pattern recognition and machine learning. [Електронний ресурс]. – Режим доступу: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

15. Jurafsky, D., & Martin, J. H. (2019). Speech and language processing. [Електронний ресурс]. – Режим доступу: https://web.stanford.edu/~jurafsky/slp3/

16. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval. [Електронний ресурс]. – Режим доступу: https://nlp.stanford.edu/IR-book/information-retrieval-book.html

17. Sommerville, I. (2011). Software engineering (9th ed.). [Електронний ресурс]. – Режим доступу: https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf

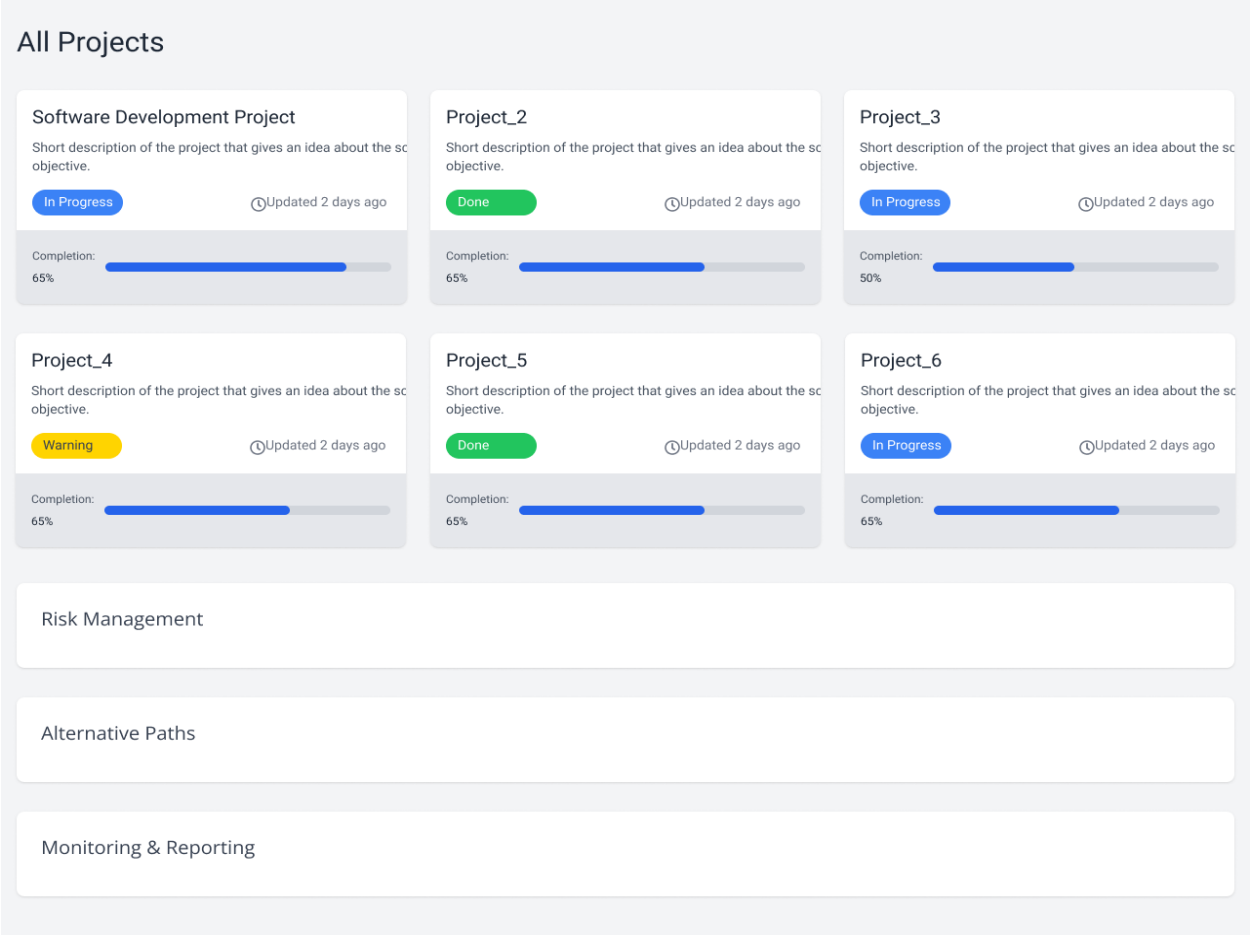
18. Pressman, R. S. (2014). Software engineering: a practitioner's approach. [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books/about/Software_Engineering_A_Practitioner_s_Ap.html?id=i8NmnAEACAAJ&redir_esc=y

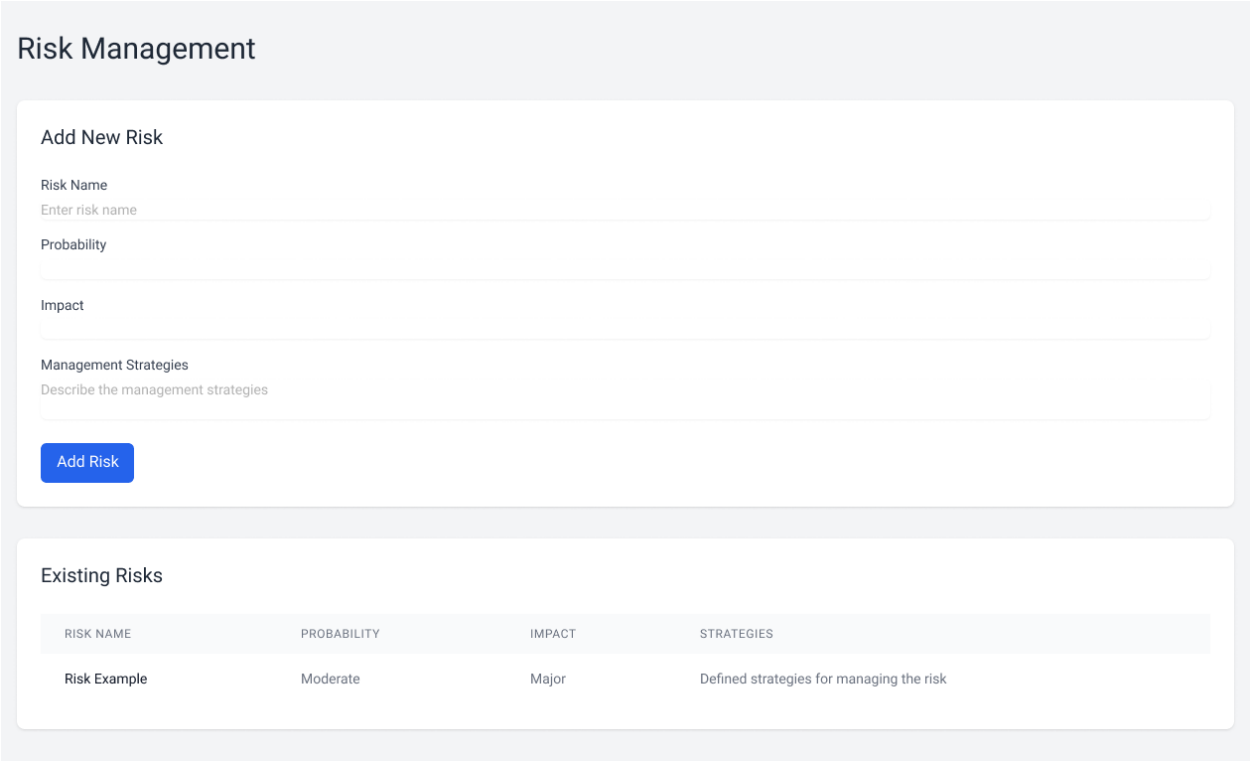
19. Chong, F., & Carraro, G. (2006). Architecture strategies for catching the long tail. [Електронний ресурс]. – Режим доступу: http://msdn. microsoft.com/en-us/library/aa479069.aspx

20. Pahl, C., & Jamshidi, P. (2015). Microservices: Architecture, design principles, and deployment. [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/369039197_Microservices_Architecture

ДОДАТОК А

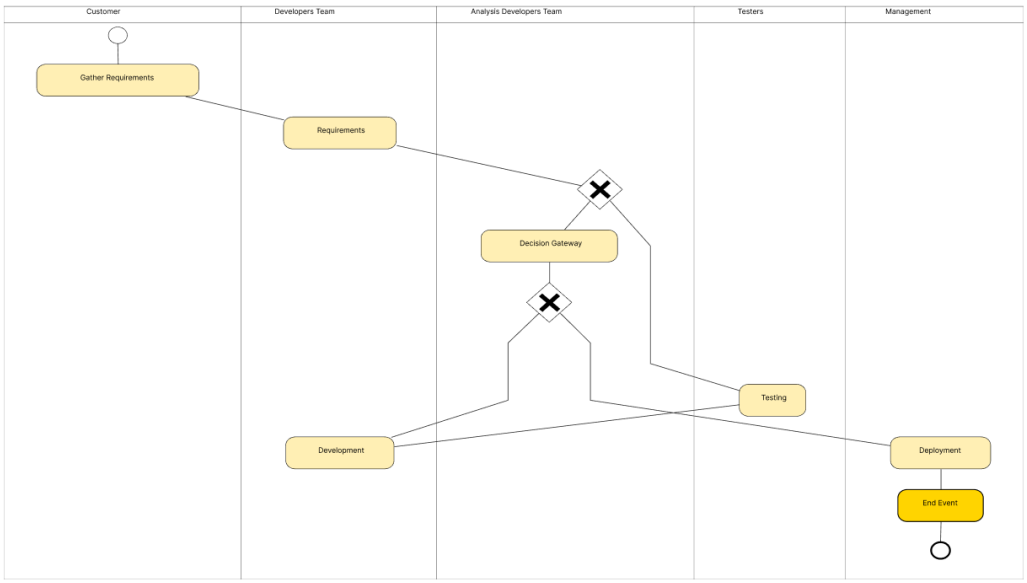
Скріншоти екранів розробленої СППР





Software Development Project

BPMN Diagram



Project Status

Status: On Track
Active Projects: 5
Upcoming Deadlines: 3

Planning

Next Milestone: Design Phase Completion
Resource Allocation: 75%

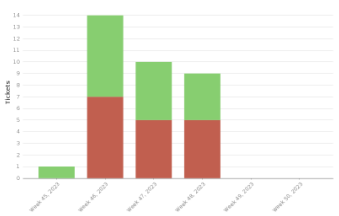
Risks

High Priority: 2
Medium Priority: 4
Low Priority: 1

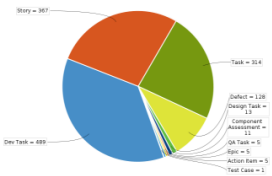
Communication

Last Meeting: April 10, 2023
Next Meeting: April 17, 2023

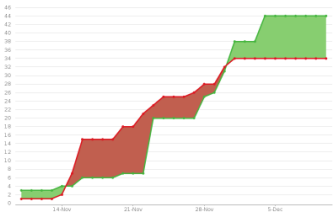
Task Completion Bar Chart



Resource Allocation Pie Chart



Project Timeline Line Chart



Project Statistics

75%	45	5	10
Completion Rate	Tasks Completed	Overdue Tasks	Pending Approvals

ДОДАТОК Б

Код програми модуля збору даних

```
import org.eclipse.egit.github.core.Repository;
import org.eclipse.egit.github.core.client.GitHubClient;
import org.eclipse.egit.github.core.service.RepositoryService;
import <your-database-driver>;
import <your-http-library>;

public class DataCollector {
    public static void main(String[] args) {
        // Збір даних з GitHub
        GitHubClient client = new GitHubClient();
        RepositoryService service = new RepositoryService(client);

        try {
            // Аутентифікація на GitHub
            client.setCredentials("<your-username>", "<your-password>");

            // Отримання інформації про репозиторії з GitHub API
            Repository repository = service.getRepository("<owner>", "<repo-
name>");

            // Запис даних в базу даних
            <your-database-driver>.getConnection("<db-url>", "<db-username>",
"<db-password>");
            <your-database-driver>.executeQuery("INSERT INTO repositories
(name, owner) VALUES ('" + repository.getName() + "', '" +
repository.getOwner().getLogin() + "')");

            // Збір даних за допомогою REST API
            <your-http-library>.get("<api-url>");
            // Обробка отриманих даних та запис у базу даних

            // Закриття з'єднання з базою даних
            <your-database-driver>.closeConnection();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ДОДАТОК В

Код програми модуля аналізу даних

```
import org.deeplearning4j.datasets.iterator.impl.MnistDataSetIterator;
import org.deeplearning4j.nn.api.Model;
import org.deeplearning4j.nn.api.OptimizationAlgorithm;
import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.layers.DenseLayer;
import org.deeplearning4j.nn.conf.layers.OutputLayer;
import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
import org.deeplearning4j.nn.weights.WeightInit;
import org.deeplearning4j.optimize.listeners.ScoreIterationListener;
import org.nd4j.evaluation.classification.Evaluation;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.dataset.DataSet;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.lossfunctions.LossFunctions;
public class DataAnalyzer {
    public static void main(String[] args) throws Exception {
        int numInput = 784;
        int numOutput = 10;
        int batchSize = 64;
        int numEpochs = 10;
        // Завантаження та підготовка даних
        DataSetIterator mnistTrain = new MnistDataSetIterator(batchSize,
true, 12345);
        DataSetIterator mnistTest = new MnistDataSetIterator(batchSize,
false, 12345);
        // Конфігурація нейронної мережі
        MultiLayerConfiguration configuration = new
NeuralNetConfiguration.Builder()
                .seed(12345)

.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
                .weightInit(WeightInit.XAVIER)
                .list()
                .layer(0, new DenseLayer.Builder()
                        .nIn(numInput)
                        .nOut(256)
                        .activation(Activation.RELU)
                        .build())
                .layer(1, new
OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                        .nIn(256)
                        .nOut(numOutput)
                        .activation(Activation.SOFTMAX)
                        .build())
                .build();
        // Ініціалізація та навчання нейронної мережі
        MultiLayerNetwork network = new MultiLayerNetwork(configuration);
        network.init();
        network.setListeners(new ScoreIterationListener(10));
        for (int i = 0; i < numEpochs; i++) {
            network.fit(mnistTrain);
        }
        // Оцінка точності моделі
        Evaluation evaluation = network.evaluate(mnistTest);
        System.out.println("Accuracy: " + evaluation.accuracy());
    }
}
```

ДОДАТОК Г

Код програми модуля моніторингу та статистики

```
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.KafkaConsumer;

public class MonitoringAndStatisticsModule {

    private Map<String, Integer> requestsCountByEndpoint;
    private Map<String, Double> responseTimeByEndpoint;
    String topic = "requests-topic";
    String bootstrapServers = "localhost:9092";
    String groupId = "monitoring-group";

    public static void main(String[] args) {
        MonitoringAndStatisticsModule module = new
MonitoringAndStatisticsModule();

        // Приклад обробки запитів
        module.processRequest("/api/users", 1.5);
        module.processRequest("/api/users", 2.0);
        module.processRequest("/api/posts", 0.8);
        module.processRequest("/api/posts", 1.2);

        Properties props = new Properties();
        props.put("bootstrap.servers", bootstrapServers);
        props.put("group.id", groupId);
        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");

        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
        consumer.subscribe(Arrays.asList(topic));

        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(100);

            for (ConsumerRecord<String, String> record : records) {
                String endpoint = record.key();
                double responseTime = Double.parseDouble(record.value());

                // Обробити запит
                processRequest(endpoint, responseTime);
            }
        }
    }

    public int getRequestsCount(String endpoint) {
        return requestsCountByEndpoint.getOrDefault(endpoint, 0);
    }

    public double getAverageResponseTime(String endpoint) {
        int requestsCount = requestsCountByEndpoint.getOrDefault(endpoint,
0);
        double totalResponseTime =
responseTimeByEndpoint.getOrDefault(endpoint, 0.0) * requestsCount;
        return totalResponseTime / Math.max(requestsCount, 1);
    }
```

```
    public static void processRequest(String endpoint, double responseTime) {
        // Збільшити кількість запитів для даного ендпоінту
        if (requestsCountByEndpoint.containsKey(endpoint)) {
            int count = requestsCountByEndpoint.get(endpoint);
            requestsCountByEndpoint.put(endpoint, count + 1);
        } else {
            requestsCountByEndpoint.put(endpoint, 1);
        }

        // Оновити час відповіді для даного ендпоінту
        responseTimeByEndpoint.put(endpoint, responseTime);
    }
}
```