Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)
Кафедра обчислювальної техніки
(повна назва кафедри)

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

AI ВЕБ-СЕРВІС З НАДАННЯ ТЕХНІЧНОЇ ПІДТРИМКИ

Виконав: студент 5 курсу, групи 2КІ-23м
спеціальності 123 – «Комп'ютерна інженерія»
освітня програма – Комп'ютерна інженерія
Алгаш О.М.
(прізвище та ініціали)

Керівник: ст. викл. каф. ОТ
Обертюх М.Р.
(прізвище та ініціали)

Опонент: к.т.н., доц. кафедри ПЗ
Ракитянська Г.Б.
(прізвище та ініціали)

Допущено до захисту

Завідувач кафедри ОТ

д.т.н., проф. Азаров О. Д.

«20» 12 2024 р

Вінниця ВНТУ 2024

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій та комп'ютерної інженерії

Кафедра Обчислювальної техніки

Галузь знань 12 - Інформаційні технології

Освітній рівень - магістр

Спеціальність 123 - «Комп'ютерна інженерія»

Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н. проф Азаров О.Д.

« 17 » вересня 2024 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Алгашу Олегу Михайловичу

1 Тема проекту «AI веб-сервіс з надання технічної підтримки», керівник проекту Обертюх М.Р., ст. викл. каф. ОТ, затверджені наказом вищого навчального закладу від «17» вересня 2024 р. №310

2 Строк подання студентом проекту «25» грудня 2024 р

3 Вихідні дані до проекту: передавання даних http, максимальна кількість токенів 50, модель text-davinci-003, температура 0.5, інтерфейс користувача.

4 Зміст текстової частини (перелік питань, які потрібно розробити): вступ, огляд і аналіз актуальності штучного інтелекту, аналіз розробки програм на основі штучного інтелекту, вибір технологій та фреймворків, розробка веб-сервісу, розробка рекомендацій з введення в експлуатацію, економічна частина.

5 Перелік графічного матеріалу: інтерфейс реєстрації, інтерфейс чату, лістинг коду програми.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Обертюх М.Р., ст. викл. каф. ОТ		
5	Небава М.І., к.е.н., професор каф. ЕПВМ	Небава	Небава

7 Дата видачі завдання « 18 » вересня 2024 року

8 Календарний план виконання приведений в таблиці 2.

Таблиця 2 — Консультанти роботи

№ з/п	Назва етапів виконання бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	18.09-25.09	Виконано
2	Огляд інформаційних джерел	02.10-15.10	Виконано
3	Огляд і аналіз штучного інтелекту	15.10-03.11	Виконано
4	Розробка веб-сервісу з штучним інтелектом	03.11-06.11	Виконано
5	Підготовка матеріалів та опис розробки	06.11-08.11	Виконано
6	Оформлення пояснювальної записки та ілюстративного матеріалу	08.11-10.11	Виконано
7	Оцінка комерційного потенціалу розробки	10.11-17.11	Виконано
8	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	17.11-25.11	Виконано

Студент _____ Алгаш О.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Обертюх М.Р.
(підпис) (прізвище та ініціали)

Консультант з економічної частини Небава _____ Небава М.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004

Алгаш О.М. AI Веб-сервіс з надання технічної підтримки. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна інженерія, Вінниця: ВНТУ, 2024 — 100 с.

На укр. мові. Бібліогр. 9 назв; рис.: 19; табл. 7; ліст. коду 3.

У дипломній роботі розглянуто теоретичний та практичний підхід до розробки веб-сервісів на основі штучного інтелекту.

Проведено аналіз методів та інструментів створення веб-сервісів, досліджуються підходи до розробки та впровадження веб-сервісів з використанням технологій штучного інтелекту.

Результати дослідження показали, що розроблено веб-сервіс на основі штучного інтелекту дозволяє ефективно виконувати різноманітні завдання, забезпечуючи автоматизацію та покращення продуктивності. Він може бути використаний у різних сферах, таких як управління даними, прогнозування, рекомендації та багато інших.

Ключові слова: штучний інтелект, інтеграція штучного інтелекту, технології, веб-додатки.

ABSTRACT

Algash O.M. AI Web Service for Technical Support. Master's Qualification Thesis in Specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2024 — 100 p. In Ukrainian. Bibliography: 9 titles; figures: 19; code lists: 3; tables: 7.

The thesis examines theoretical and practical approaches to the development of web services based on artificial intelligence.

It analyzes the methods and tools for creating web services and explores approaches to the development and implementation of web services using artificial intelligence technologies.

The results of the study showed that the developed web application based on artificial intelligence allows to perform various tasks efficiently, providing automation and improving productivity. It can be used in various areas such as data management, forecasting, recommendations, and many others.

Keywords: artificial intelligence, AI integration, technologies, web applications.

ЗМІСТ

ВСТУП

Актуальність теми штучного інтелекту в системах технічної підтримки є надзвичайно високою. Це пояснюється зростаючими вимогами до швидкого та персоналізованого обслуговування клієнтів.

У сучасному світі штучний інтелект займає все більш значуще місце в різних сферах людського життя. Його потужність і можливості розширюються з кожним днем, а використання штучного інтелекту у веб-сервісах виявляється особливо перспективним і актуальним. Застосування штучного інтелекту системах технічної підтримки має забезпечити ефективніший, індивідуалізований та зручний підхід до клієнтів та користувачів.

Інтеграція мовних моделей дозволяє автоматизувати процеси підтримки, підвищуючи ефективність і знижуючи навантаження на людські ресурси.

Така технологія забезпечує не тільки цілодобову доступність послуг, але й можливість масштабування послуг без значних витрат, що робить її стратегічно важливою для сучасних бізнесів.

Об’єкт дослідження — інструменти для створення веб-сервісів та інтеграції до них штучного інтелекту.

Предметом дослідження є процеси застосування мовних моделей в розробці веб-сервісів та автоматизації взаємодії з користувачами у веб-середовищі, їх вплив на ефективність обслуговування, користувацький досвід і загальну продуктивність систем технічної підтримки.

Метою роботи є вдосконалення системи технічної підтримки веб-сервісів за рахунок оптимізації відповідей за допомогою штучного інтелекту.

Для досягнення поставленої мети у роботі розв’язані такі **задач**і:

— проведено аналіз сучасних технологій розробки веб-сервісів та моделей штучного інтелекту;

— вибрано фреймворк, модель штучного інтелекту та програмне забезпечення;

— розроблена система технічної підтримки з інтеграцією штучного

інтелекту.

Наукова новизна дослідження полягає у удосконаленні методів створення веб-сервісів за рахунок застосування сучасних мовних моделейдля автоматизації процесів технічної підтримки, що забезпечує можливість адаптації відповідей до специфічних потреб бізнесу, гнучкість та здатність до швидкої адаптації до потреб компаній.

Апробація результатів роботи здійснена у доповіді на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікація за темою роботи: Алгаш О.М.; Обертюх М.Р. Аналіз використання штучного інтелекту у веб-сервісах технічної підтримки. Факультет інформаційних технологій та комп'ютерної інженерії, Available at: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/22863/18861.

1 АНАЛІЗ СУЧАСНИХ ТЕНДЕНЦІЙ, АРХІТЕКТУР ТА ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Сучасні тенденції та підходи до розробки веб-сайтів

Сучасний світ веб-розробки є динамічним та швидкозмінним, пропонуючи нові технології та підходи кожного року. Ці інновації не лише розширюють можливості для розробників, але й вимагають від них адаптації до нових викликів. Наступні розділи розглядають ключові тенденції та підходи, які актуальні сьогодні у сфері веб-розробки.

Адаптивний дизайн у веб-розробці відіграє ключову роль у сучасному дизайні веб-сайтів. Цей підхід полягає у створенні веб-сайтів, що ефективно працюють та виглядають належним чином на різних пристроях, незалежно від їхнього розміру та роздільної здатності екрану. Головна ідея адаптивного дизайну - забезпечити оптимальний перегляд та взаємодію, мінімізуючи необхідність у зміні розміру, прокрутці та панорамуванні.

Адаптивний дизайн використовує різні техніки, включаючи гнучкі сітки, гнучкі зображення та медіа-запити CSS. Гнучкі сітки дозволяють елементам макету веб-сторінки розширюватися та стискатися відповідно до розміру екрану. Гнучкі зображення автоматично адаптуються до розміру екрану, забезпечуючи, щоб вони не виходили за рамки контейнера. Медіа-запити CSS дозволяють створювати різні правила стилів для різних умов перегляду, забезпечуючи, що веб-сайт виглядає ідеально на будь-якому пристрої.

Завдяки адаптивному дизайну, веб-сайти стають більш доступними та зручними для широкої аудиторії користувачів, включаючи тих, хто використовує мобільні пристрої. Це підхід, який підтримує сучасні стандарти веб-дизайну та гарантує, що веб-сайти будуть ефективно працювати в різноманітних сценаріях використання.

Односторінкові додатки (SPA) – це передовий підхід у веб-розробці, який полягає у створенні веб-сайтів або додатків, що складаються з однієї веб-сторінки. Головна особливість SPA – динамічне взаємодіяння з користувачем, при якому перевантажується лише необхідний контент, замість цілої сторінки. Це забезпечує

швидше завантаження та плавніше користувацьке взаємодіяння, а також ефективніше використання ресурсів.

Серцевиною SPA є AJAX (асинхронний JavaScript та XML), який дозволяє веб-додаткам асинхронно надсилати та отримувати дані з сервера. Це означає, що можливе оновлення частин сторінки без необхідності перезавантаження цілої сторінки. Такий підхід підвищує швидкість відгуку додатка та покращує загальний досвід користувача.

SPA часто використовують JavaScript фреймворки, такі як React, Angular чи Vue.js, для створення більш організованої та ефективної структури коду. Ці фреймворки сприяють створенню інтерактивних та динамічних веб-додатків, забезпечуючи високу продуктивність та гнучкість розробки.

Однак, SPA можуть мати певні недоліки, наприклад, складнощі з SEO оптимізацією та потенційні проблеми з продуктивністю при великих обсягах даних. Незважаючи на це, SPA залишаються популярним вибором для створення сучасних, швидких та зручних веб-додатків.

JavaScript фреймворки відіграють критичну роль у сучасній веб-розробці, пропонуючи структуровані та ефективні підходи до створення веб-додатків. Вони надають готові до використання шаблони та функції, що значно спрощують процес розробки, одночасно підвищуючи якість та надійність кінцевого продукту.

Серед найпопулярніших JavaScript фреймворків виділяються React, Angular та Vue.js. React, створений Facebook, відомий своїм гнучким підходом та ефективністю в управлінні інтерфейсами користувача. Він використовує компонентний підхід, дозволяючи розробникам створювати масштабовані та швидкі веб-додатки. Angular, розроблений Google, є повноцінним фреймворком, який пропонує суворішу архітектурну структуру і використовується для створення динамічних односторінкових застосунків. Vue.js, з іншого боку, пропонує більш легкий та гнучкий підхід, забезпечуючи легкість інтеграції та швидкість розвитку.

Використання цих фреймворків дозволяє розробникам ефективно управляти станом додатків, реалізовувати складні користувацькі інтерфейси та спрощувати процес тестування. Це також сприяє кращій організації коду, роблячи його більш

читабельним та легким для підтримки. Завдяки модульній природі, JavaScript фреймворки підвищують ефективність та швидкість розробки, що є важливим в умовах сучасного ритму роботи та постійно зростаючих вимог до функціональності веб-додатків.

Реактивне програмування в сучасній веб-розробці є парадигмою, яка зосереджена на створенні додатків з високим рівнем відгуку на події, швидким реагуванням на зміни, та ефективною обробкою асинхронних даних. Цей підхід базується на концепції потоків даних та поширенні змін у вигляді "реакцій", які автоматично викликаються відповідно до певних умов чи подій.

Реактивне програмування дозволяє розробникам ефективніше управляти складними потоками даних, такими як запити до сервера, взаємодія користувача, або різні фонові операції. Це досягається за допомогою утилізації патернів, таких як Observer, де спостерігачі підписуються на потоки даних та реагують на зміни в реальному часі.

Бібліотеки реактивного програмування, такі як RxJS у JavaScript, надають потужні інструменти для створення та управління асинхронними подіями та даними. Вони дозволяють легко створювати, комбінувати, фільтрувати, трансформувати, та управляти потоками даних у додатках.

Застосування реактивного програмування в веб-розробці сприяє створенню більш масштабованих, відгукових та інтерактивних додатків. Це значно покращує користувацький досвід, оскільки додатки стають більш чутливими до дій користувача та змін у даних, забезпечуючи плавну та неперервну взаємодію.

Мікросервісна архітектура в сучасній веб-розробці представляє собою підхід, який полягає у розбитті додатків на набір невеликих, незалежних служб, кожна з яких виконує конкретну функцію. Кожен мікросервіс працює як автономний процес і спілкується з іншими службами через легко визначені API. Така структура дозволяє розробникам та командам працювати над різними частинами додатку незалежно один від одного, що підвищує швидкість розробки та гнучкість управління проектом.

Однією з ключових переваг мікросервісної архітектури є її масштабованість. Завдяки тому, що служби є незалежними, їх можна легко масштабувати та оновлювати, не впливаючи на роботу інших компонентів системи. Це особливо корисно для великих, складних додатків, де потреба в нових функціях та швидкі оновлення є частими.

Мікросервісна архітектура також сприяє поліпшенню надійності додатків. У випадку виникнення проблем у одному мікросервісі, це не обов'язково впливає на роботу інших служб, що знижує ризик загальних збоїв системи. Крім того, оскільки мікросервіси можуть бути розроблені з використанням різних технологій та мов програмування, це дає розробникам свободу вибору найкращих інструментів для кожного конкретного завдання.

Однак, мікросервісна архітектура може принести й певні виклики, такі як складність управління зв'язками між службами та вищі вимоги до координації робіт між різними командами. Незважаючи на це, переваги, які вона пропонує, роблять її вкрай привабливою для багатьох сучасних веб-проектів, особливо тих, що вимагають високої масштабованості та гнучкості.

Прогресивні веб-додатки (PWA) представляють собою передовий підхід у створенні веб-додатків, що поєднують у собі переваги веб-сайтів та нативних мобільних додатків. Ця концепція дозволяє розробникам створювати веб-додатки, які можуть виглядати та функціонувати як нативні додатки на мобільних пристроях, пропонуючи користувачам більш зручний та інтерактивний досвід.

Основними характеристиками PWA є робота в офлайн-режимі, швидке завантаження, здатність надсилати push-сповіщення та доступ до апаратних можливостей пристрою, таких як камера та GPS. Це досягається завдяки використанню технологій, таких як Service Workers, Web App Manifests та кешування даних, що дозволяє додатку завантажуватися миттєво, навіть при поганому інтернет-з'єднанні або в його відсутності.

PWA також пропонують значні переваги з точки зору встановлення та обслуговування. Вони не вимагають завантаження через магазини додатків, такі як

Google Play або Apple App Store, і можуть бути легко оновлені розробниками без необхідності завантаження оновлень користувачами.

Однією з ключових переваг PWA є їхня висока ефективність та доступність. Вони підвищують залучення користувачів та покращують загальний досвід користувачів завдяки швидкому реагуванню та зручності використання. Крім того, PWA підвищують доступність веб-додатків для широкого кола користувачів, зокрема в регіонах із обмеженим доступом до високошвидкісного інтернету або передових смартфонів.

Враховуючи свою універсальність та ефективність, PWA відкривають нові горизонти для веб-розробки, пропонуючи підход, який ефективно відповідає на сучасні вимоги користувачів та ринку.

API-орієнтована розробка є ключовим елементом сучасної веб-розробки, акцентуючи на створенні міцних, гнучких і масштабованих веб-додатків. Цей підхід заснований на використанні програмних інтерфейсів додатків (API), які дозволяють різним системам взаємодіяти між собою, обмінюючись даними та функціональностями.

У серці API-орієнтованої розробки лежить ідея "розділу та володіння" — створення незалежних компонентів, які можуть взаємодіяти через добре визначені інтерфейси. Це дозволяє розробникам модулізувати додатки, відокремлюючи логіку бекенду від фронтенду та інших систем, таких як бази даних та служби третіх сторін.

API-орієнтована архітектура забезпечує високу гнучкість у розробці та розширенні функціоналу веб-додатків. Завдяки API, розробники можуть швидко інтегрувати нові функції, сервіси чи платформи, не змінюючи основний код додатку. Це також спрощує процеси тестування та впровадження, оскільки зміни можуть вноситися і перевірятися окремо для кожного компонента.

API-орієнтована розробка також сприяє кращій інтеграції з мобільними додатками та іншими зовнішніми системами. Це дозволяє веб-додаткам ефективно взаємодіяти з широким спектром сервісів, від соціальних мереж до платіжних

систем, розширюючи їхні можливості та забезпечуючи більш зручний досвід для кінцевих користувачів.

У сучасному цифровому світі, де швидкість, ефективність та масштабованість є ключовими факторами успіху, API-орієнтована розробка стає все більш важливою, надаючи розробникам гнучкі та потужні інструменти для створення високоякісних веб-додатків.

1.2 Огляд сучасних архітектур веб-сайтів

Монолітна архітектура є традиційним підходом у розробці веб-додатків, де всі компоненти системи – від інтерфейсу користувача до бізнес-логіки та бази даних — інтегровані в один застосунок. У контексті використання React, монолітна архітектура може бути реалізована з акцентом на створення односторінкових додатків (SPA), як показано на рисунках 1.1, 1.2, 1.3, 1.4 де React відповідає за управління інтерфейсом користувача.

Рисунок 1.1 — Структура монолітної архітектури

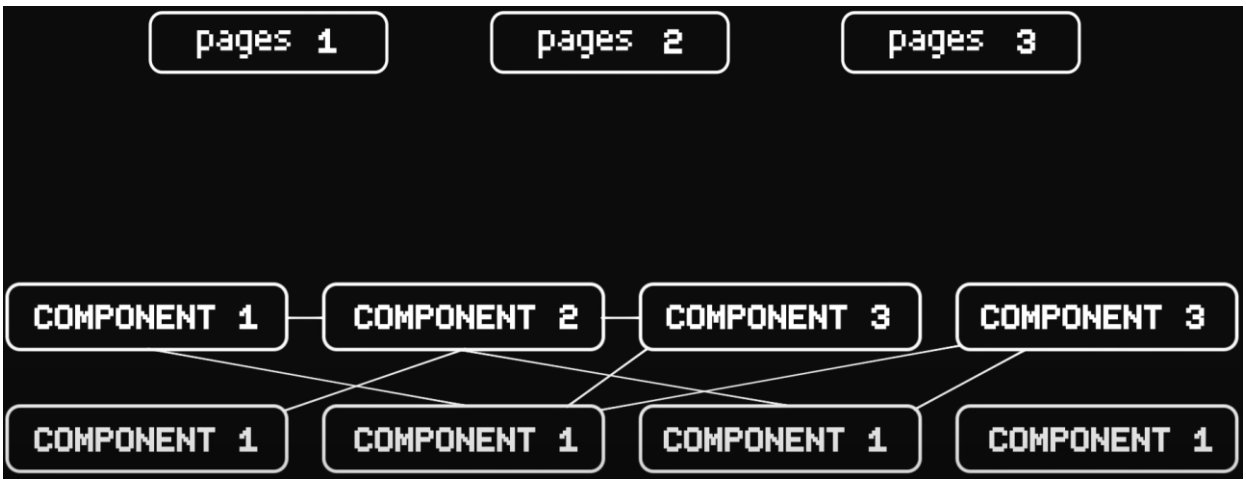

Рисунок 1.2 — Схема монолітної архітектури

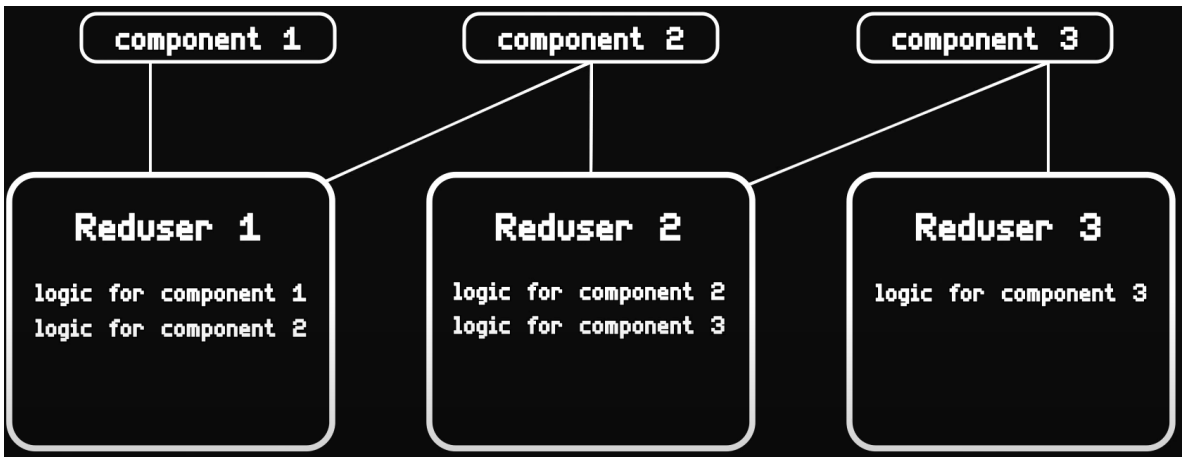

Рисунок 1.3 — Схема доступу до даних в монолітній архітектурі

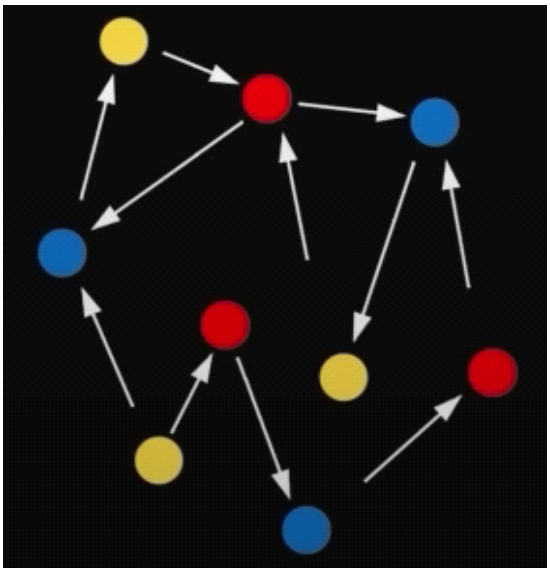

Рисунок 1.4 — Схема взаємозв’язків між компонентами в монолітній архітектурі

Основні характеристики монолітної архітектури:

— централізація — всі компоненти системи знаходяться в одному місці, що спрощує розробку, тестування, розгортання та управління інфраструктурою;

— цніфікація — монолітна архітектура дозволяє розробникам використовувати єдиний стек технологій, що може бути перевагою для команд з обмеженими ресурсами або досвідом у різноманітних технологія;

— спрощення розробки — оскільки всі компоненти тісно інтегровані, розробники можуть легше відстежувати взаємодії між різними частинами додатку.

У контексті React, монолітна архітектура може включати створення комплексного фронтенду, де всі користувацькі інтерфейси та фронтенд-логіка централізовано управляються за допомогою React. Це дає можливість ефективно використовувати станові компоненти React, контекст та інші функціональні можливості бібліотеки для створення динамічного та інтерактивного користувацького досвіду.

Однак, монолітна архітектура може зіткнутися з проблемами масштабування, особливо у великих проектах, де зміни в одній частині системи можуть вплинути на всю решту. Крім того, цей підхід може обмежувати гнучкість та ефективність внесення змін або впровадження нових технологій.

У підсумку, використання монолітної архітектури у контексті React може бути вигідним для дрібних або середніх проектів, де простота розробки та управління переважає потребу в масштабованості та гнучкості.

Модульна архітектура у веб-розробці є важливим підходом, що дозволяє створювати гнучкі, легко масштабовані та підтримувані веб-додатки. Цей підхід заснований на принципі розділення функціональності додатку на незалежні модулі, кожен з яких виконує певну роль чи бізнес-функцію. Структура модульної архітектури та схема взаємозв’язків між компонентами в модульної архітектури наведено на рисунках 1.5 та 1.6.

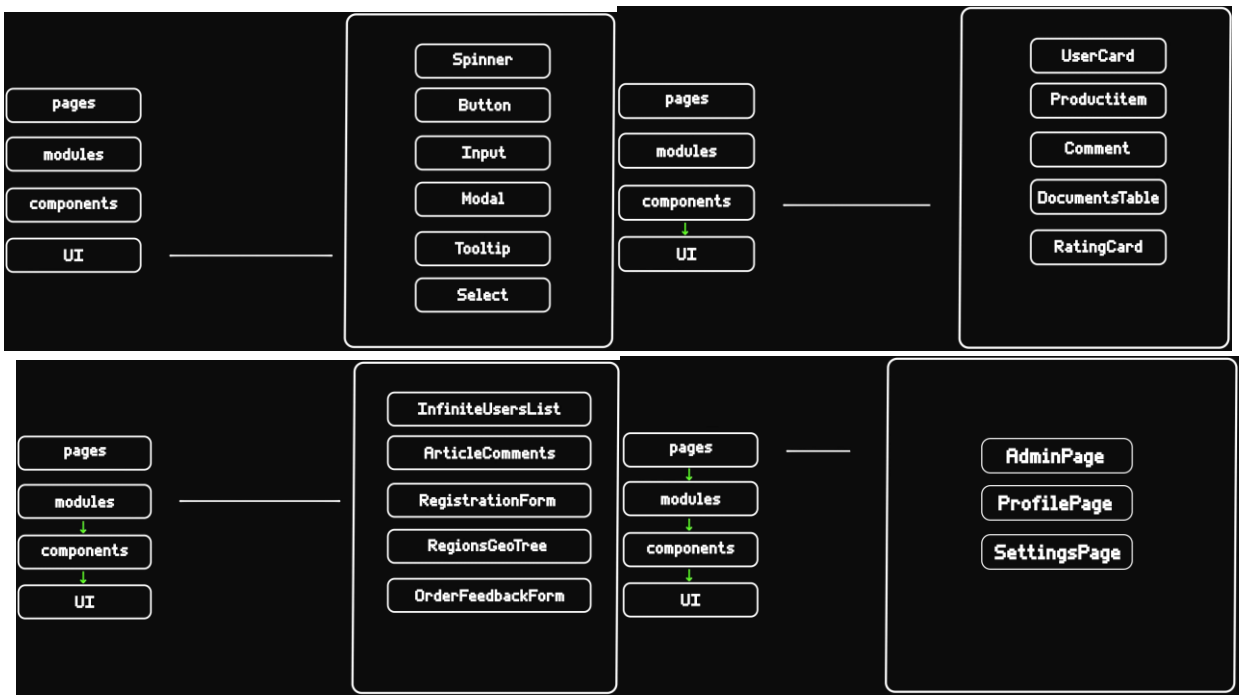

Рисунок 1.5 — Структура модульної архітектури

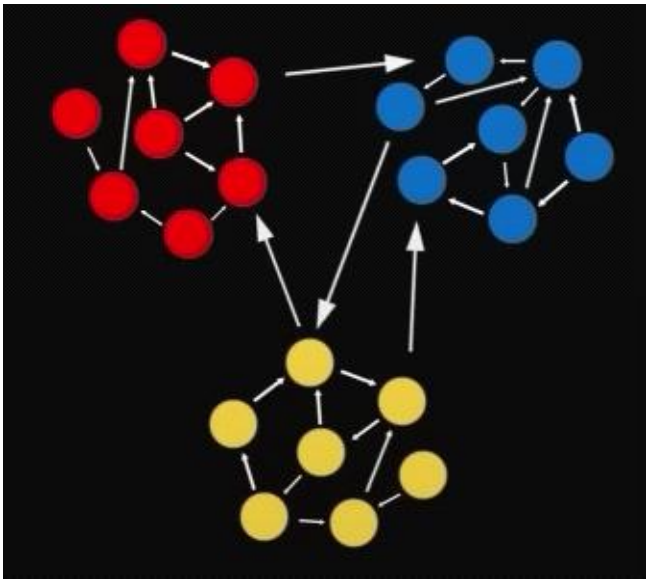

Рисунок 1.6 — Схема взаємозв’язків між компонентами в модульній архітектурі

Ключові аспекти модульної архітектури:

— розділення відповідальності — кожен модуль має чітко визначену відповідальність і містить всю необхідну логіку для реалізації певної функції, такий підхід сприяє зменшенню залежностей між різними частинами додатку, що полегшує розробку та тестування;

— легке масштабування та підтримка — у модульних системах внесення змін або оновлень у одну частину не обов'язково впливає на інші модулі, що робить процес розвитку та підтримки більш гнучким і ефективним;

— перевикористання коду — модулі можуть бути розроблені таким чином, щоб їх можна було легко перевикористовувати у різних частинах додатку або навіть у різних проектах.

У контексті веб-розробки, модульна архітектура часто реалізується за допомогою сучасних фреймворків і бібліотек, таких як React. Використання React дозволяє створювати додатки, організовані навколо компонентів, кожен з яких є самодостатнім модулем з власною логікою та інтерфейсом користувача. Це забезпечує високий рівень гнучкості та перевикористання коду, а також спрощує управління станами та данними в масштабних застосунках.

Однак, модульна архітектура також вимагає чіткого планування та управління залежностями між модулями, особливо у великих проектах. Важливо забезпечити, щоб інтерфейси між модулями були добре визначені та узгоджені, а залежності - мінімальними.

Завдяки своїй гнучкості та ефективності, модульна архітектура є відмінним вибором для розробки сучасних веб-додатків, які потребують швидкої адаптації до змінних вимог та технологічних трендів.

Atomic архітектура (див рис. 1.7), іноді відома як Atomic Design, є підходом у веб-дизайні та розробці, який організовує компоненти інтерфейсу в ієрархічну структуру, схожу на хімічні елементи, що формують складніші структури. Розроблено Бредом Фростом, цей підхід допомагає розробникам та дизайнерам створювати консистентні, масштабовані та перевикористовувані інтерфейси, розглядаючи їх як комбінацію базових елементів.

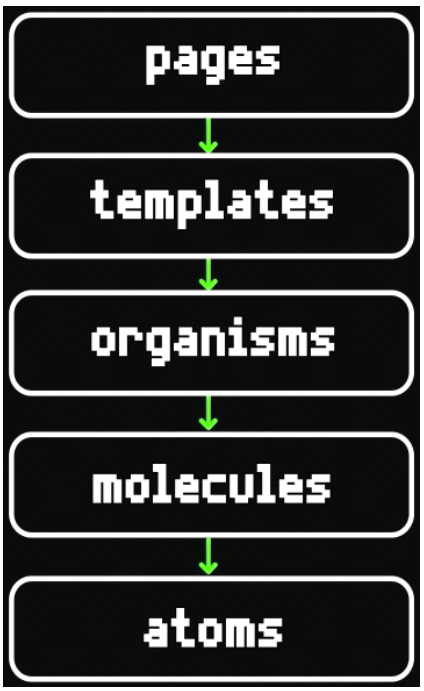

Рисунок 1.7 — Структура Atomic архітектури

Основні компоненти Atomic архітектури:

— атоми — це базові будівельні блоки інтерфейсу, такі як кнопки, інпути, лейбли та інші HTML-елементи, вони є найпростішими та не можуть бути розділені на дрібніші частини без втрати своєї функціональності.

— молекули — олекули створюються шляхом комбінування декількох атомів. Наприклад, форма пошуку може бути молекулою, що складається з інпуту (атома) та кнопки (атома);

— організми — організми є відносно складними компонентами, сформованими з комбінацій молекул та/або атомів, наприклад, заголовок веб-сторінки, що включає логотип, навігаційне меню (молекули) та пошукову форму (молекула);

— шаблони — шаблони є сторінками, де організми об'єднуються в макет, що показує загальний структурний вигляд інтерфейсу;

— сторінки — на цьому рівні шаблони наповнюються реальним контентом, що демонструє, як дизайн буде виглядати в реальному використанні.

Atomic архітектура ідеально підходить для використання з такими фреймворками, як React, де компонентний підхід лежить в основі розробки. Вона

дозволяє створювати перевикористовувані, легко підтримувані компоненти, забезпечуючи консистентність та спрощуючи процес розробки великих та складних інтерфейсів.

Завдяки своїй структурованості та систематизації, Atomic архітектура є важливим інструментом для створення ефективних та гнучких веб-дизайнів, які легко адаптуються до змінних вимог і трендів сучасного веб-простору.

Feature Sliced архітектура (див. рисунки 1.8, 1.9, 1.10) — це сучасний підхід до структурування та організації коду у веб-розробці, зосереджений на розділенні застосунку за функціональними особливостями (features) та відокремленні логіки від інтерфейсу. Цей підхід допомагає розробникам ефективно управляти великими та складними проектами, роблячи код більш модульним, зрозумілим та легким для тестування та підтримки.

Рисунок 1.8 — Структура Feature sliced архітектури

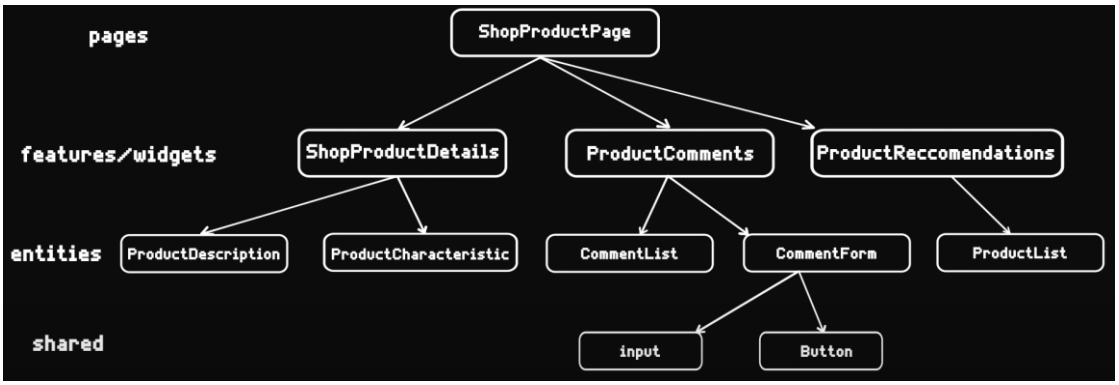

Рисунок 1.9 — Схема Feature sliced архітектури

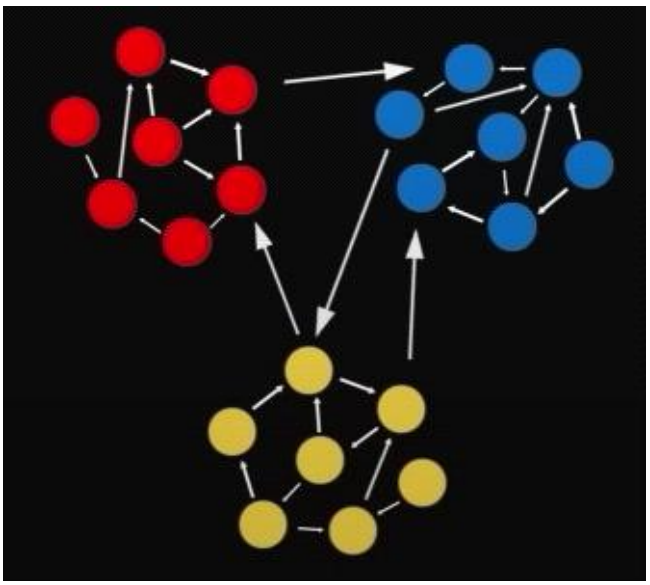

Рисунок 1.10 — Схема взаємозв’язків між компонентами в Схема взаємозв’язків між компонентами в Feature sliced архітектурі

Основні характеристики Feature Sliced архітектури:

— розділення за функціональними блоками — код розподіляється на окремі функціональні блоки або "сегменти", кожен з яких представляє певну особливість або бізнес-логіку додатку, може включати різні аспекти, такі як користувацькі інтерфейси, API взаємодії, стейт-менеджмент та інш.;

— модульність та перевикористання — структуруючи код у незалежні модулі, цей підхід сприяє перевикористанню коду, модулі можуть бути легко перенесені або інтегровані в інші частини додатку чи навіть інші проекти;

— гнучкість та легкість масштабування — Feature Sliced архітектура дозволяє легко додавати, видаляти або модифікувати функціональні блоки, що полегшує масштабування та адаптацію додатку до змінюваних вимог;

— покращення співпраці у команді — чітко визначені блоки та модулі спрощують роботу в команді, оскільки розробники можуть працювати над окремими функціональними частинами паралельно, зменшуючи конфлікти та залежності.

У контексті використання з такими бібліотеками як React, Feature Sliced архітектура дозволяє створювати чітко структуровані компоненти та хуки, розподіляючи їх за функціональними особливостями. Це забезпечує краще розуміння

та управління станом у масштабних застосунках, де різні особливості можуть взаємодіяти або використовуватися в різних контекстах.

Feature Sliced архітектура є важливим підходом у сучасній веб-розробці, особливо у великих проектах, де потрібна висока масштабованість, гнучкість та ефективність роботи команди розробників.

Мікрофронтенд — це архітектурний підхід у веб-розробці, який дозволяє розбити фронтенд великого веб-додатку на менші, незалежні та легко управлінні частини. Цей підхід дозволяє командам розробників працювати над окремими функціональними секціями веб-сайту або додатку, надаючи кожній команді свободу вибору технологій та відокремленого розгортання. Мікрофронтенди надають великі можливості для оптимізації продуктивності, гнучкості та швидкості розвитку. Структуру мікрофронтеду наведено на рисунку 1.11.

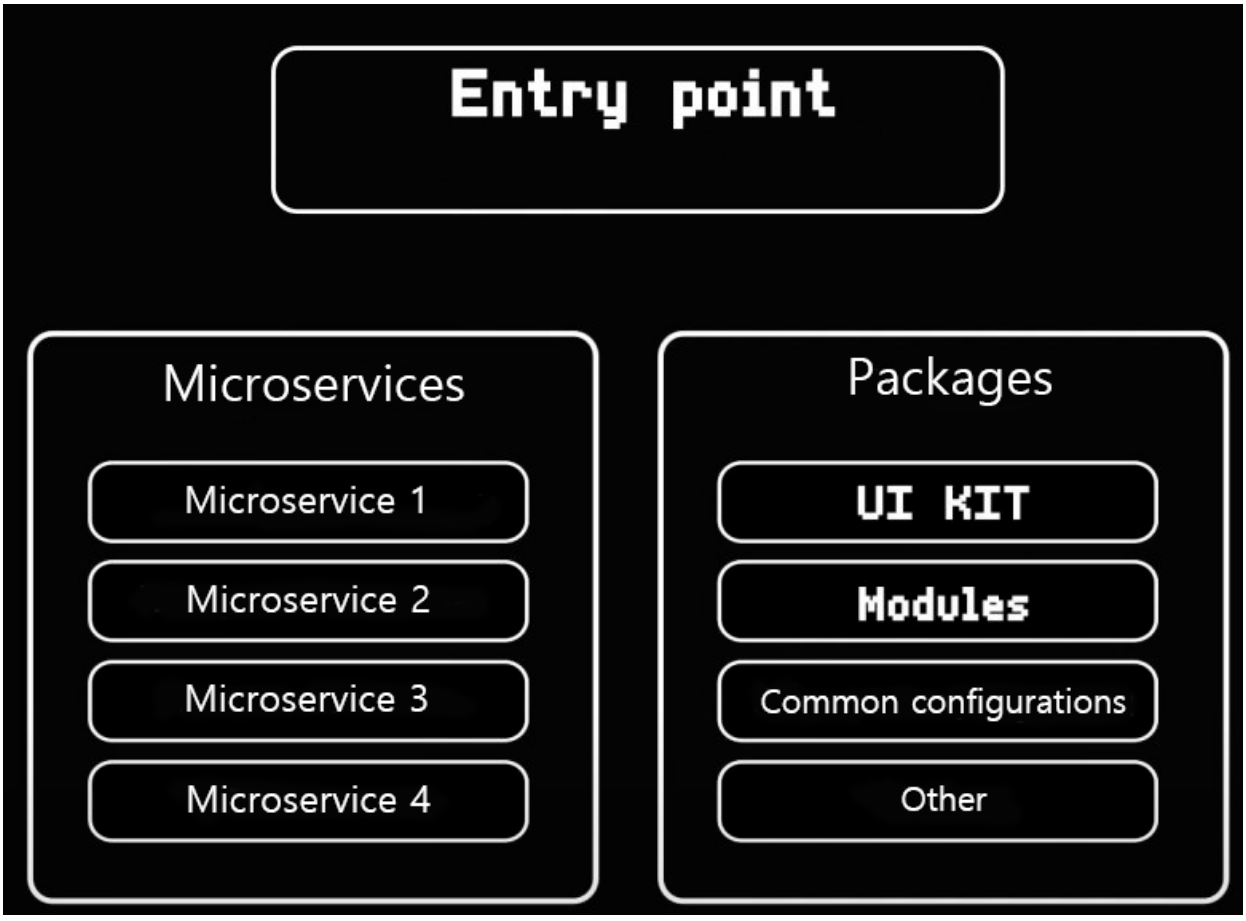

Рисунок 1.11 — Структура мікрофронтенду

Особливості мікрофронтенд архітектури:

— децентралізація та незалежність — кожен мікрофронтенд функціонує як незалежний блок, має власний репозиторій коду, набір залежностей та може бути розгорнутий окремо, також дозволяє різним командам розробників працювати паралельно, зменшуючи ризик конфліктів і залежностей.

— гнучкість у виборі технологій — мікрофронтенди надають можливість використовувати різні фреймворки та бібліотеки в одному додатку, наприклад, одна частина веб-сайту може бути побудована на React, а інша - на Vue.js;

— спрощення масштабування та підтримки — оскільки кожен мікрофронтенд є самодостатнім, додавання нових функцій або оновлення існуючих стає значно простішим і не вимагає втручання в інші частини додатку;

— оптимізація продуктивності — мікрофронтенди можуть бути оптимізовані окремо, дозволяючи завантажувати тільки необхідні ресурси, що покращує швидкість завантаження та загальну продуктивність додатку;

Мікрофронтендна архітектура також має свої виклики, особливо у відношенні управління залежностями та координації між командами. Це вимагає чіткої стратегії інтеграції та комунікації, щоб забезпечити консистентність та надійність загального додатку. Незважаючи на це, мікрофронтенди відкривають нові можливості для створення масштабованих, ефективних та гнучких веб-додатків.

1.3 Сучасні Фреймворки та бібліотеки для написання веб-сайтів

React був розроблений компанією Facebook і вперше був представлений громадськості в 2013 році. З тих пір він пройшов значний розвиток та став популярним серед розробників. Важливим моментом в історії React було введення концепції "віртуального DOM" - ця інноваційна ідея дозволяє React забезпечувати високу продуктивність та ефективну роботу зі змінами у даних.

React ґрунтується на кількох ключових концепціях:

— компоненти — React розбиває інтерфейс на невеликі, повторно використовувані компоненти. Це допомагає розробникам створювати інтерфейс

шляхом комбінування цих компонентів;

— віртуальний DOM — React використовує віртуальний DOM для ефективного оновлення інтерфейсу, замість безпосереднього маніпулювання реальним DOM, React спершу оновлює віртуальний DOM і порівнює його з реальним для знаходження змін, після чого вносить лише необхідні зміни в реальний DOM, це значно підвищує продуктивність додатків;

— JSX — для створення компонентів React використовує JSX, що дозволяє об'єднувати JavaScript і HTML-подібні теги для опису структури інтерфейсу, це полегшує читабельність та підтримку коду.

React став основою багатьох великих веб-проектів і платформ, включаючи Facebook, Instagram, WhatsApp, Airbnb та багато інших. Однією з основних причин його популярності є його висока продуктивність та здатність працювати з великими обсягами даних.

Крім того, багато сторонніх бібліотек та розширень були створені на основі React, що робить його ще більш потужним і гнучким фреймворком для розробників. Також, велика і активна спільнота розробників робить React ідеальним вибором для тих, хто шукає підтримку та відповіді на свої запитання.

Необхідно також відзначити, що React можна легко інтегрувати з іншими технологіями та бібліотеками, що дозволяє розробникам створювати різноманітні типи додатків, включаючи односторінкові додатки (SPA), прогресивні веб-додатки (PWA) та мікросервісну архітектуру.

У підсумку, React є надзвичайно важливим інструментом у сучасній веб-розробці. Він пропонує ефективний спосіб створення інтерфейсу, дозволяє розробникам писати чистий та організований код, та надає можливість швидко реагувати на зміни в даних. Таким чином, React завдяки своїм особливостям і функціональності продовжує залишатися лідером серед фреймворків для веб-розробки, і його значення важко переоцінити в сучасній індустрії веб-технологій.

Angular був розроблений компанією Google і був офіційно представлений як AngularJS у 2010 році. Однак, оригінальна версія AngularJS мала деякі обмеження

та недоліки. У зв'язку з цим було розроблено нову версію - Angular 2, яка була повністю переписана та перероблена.

Angular має кілька ключових концепцій, які визначають його структуру та функціональність:

— компоненти: У Angular, як і в React, інтерфейс розбивається на компоненти. Компоненти в Angular - це класи, які містять логіку та шаблони для відображення інтерфейсу. Вони дозволяють створювати реактивні та повторно використовувані елементи.

— сервіси: Angular рекомендує використовувати сервіси для обробки бізнес-логіки та обміну даними між компонентами. Сервіси допомагають розділити функціональність на окремі частини та забезпечують можливість роботи з асинхронними операціями.

— Модулі — Angular дозволяє структурувати додаток за допомогою модулів, модулі допомагають організовувати компоненти, сервіси та інші ресурси додатка у логічні групи;

— залежності та ін'єкція залежностей — Angular використовує систему ін'єкції залежностей для керування залежностями компонентів та сервісів, це допомагає зберігати код додатка легко читабельним і підтримуваним.

Angular відзначається своєю стабільністю та досконалістю у керуванні великими та складними веб-додатками. Він підходить для розробки різноманітних додатків, включаючи односторінкові додатки, адмін-панелі, магазини, соціальні мережі та інші.

Однією з основних переваг Angular є його повністю налаштована середовище, що включає в себе все необхідне для розробки та тестування додатків. Також, Angular має велику спільноту розробників і багато розширень, які допомагають спростити роботу з ним.

Angular також підтримує розробку для різних платформ, включаючи веб, мобільні пристрої та настільні додатки, що робить його універсальним інструментом для розробки кросплатформених додатків.

У підсумку, Angular є важливим фреймворком у світі веб-розробки і пропонує розробникам потужні інструменти для створення високоякісних інтерфейсів та додатків. Його структура та концепції дозволяють розробникам створювати організовані та ефективні додатки, і він залишається конкурентоспроможним у світі веб-розробки.

Vue.js, або просто Vue, був створений Єваном Ю (Evan You) у 2014 році. Початково це був проєкт-ідея, який Єван розробив для полегшення створення інтерактивних веб-інтерфейсів. З часом Vue набрав популярність завдяки своїй простоті та ефективності, і став одним із найбільш використовуваних фреймворків веб-розробки.

Vue має кілька ключових концепцій, які визначають його структуру та функціональність:

— директиви — Vue використовує директиви для опису взаємодії між елементами DOM та даними додатка, наприклад, директива v-bind дозволяє зв'язувати атрибути елементів з даними, а v-on - встановлювати обробники подій;

— компоненти — Vue також підтримує компонентний підхід до розробки, де інтерфейс розбивається на невеликі, самостійні компоненти, це допомагає створювати повторно використовувані та легко супроводжувані частини додатка;

— Vue Router— для створення односторінкових додатків (SPA) Vue використовує Vue Router - офіційний маршрутизатор для Vue, він дозволяє створювати сторінки та навігацію в додатках;

— Vuex — для управління станом додатку Vue використовує Vuex - це бібліотека для глобального управління станом, вона допомагає ефективно організувати та зберігати дані великих додатків.

Vue відомий своєю простотою та легкістю вивчення, що робить його відмінним вибором для початківців та досвідчених розробників. Він підходить для створення різних типів додатків, від невеликих веб-сайтів до складних корпоративних додатків.

Vue також відзначається гнучкістю та можливістю інтеграції з іншими бібліотеками та проєктами. Він дозволяє використовувати його як для великих односторінкових додатків, так і для часткової інтеграції в існуючі веб-сайти.

У підсумку, Vue є важливим гравцем у світі веб-розробки. Він пропонує простий та ефективний спосіб створення інтерфейсів та додатків, що приваблює розробників своєю легкістю використання та гнучкістю. Його активна спільнота розробників і ряд розширень дозволяють розробникам створювати різноманітні проекти і залишають Vue одним із конкурентоспроможних фреймворків у сучасній веб-розробці.

1.4 Аналіз штучного інтелекту OpenAI

OpenAI — один із лідерів у галузі штучного інтелекту, заснований з метою розробки технологій, що допомагають у безпечному та відповідальному впровадженні штучного інтелекту. Вагомий внесок OpenAI полягає в створенні та вдосконаленні генеративних моделей, які сьогодні застосовуються у багатьох сферах, від автоматизації бізнес-процесів до творчої роботи. Основні генеративні моделі OpenAI, зокрема серії GPT та Codex, продемонстрували здатність створювати тексти, коди та інші форми контенту з високою якістю та точністю.

У 2019 році OpenAI представила модель GPT-2, яка значно перевищувала можливості попередніх версій ШІ. Ця модель стала революційною завдяки своїм можливостям генерувати зв’язний, контекстуально обґрунтований текст, що робило її здатною до ведення діалогів, написання статей, ведення блогів та іншої роботи з текстом. Вона мала понад 1,5 мільярда параметрів, що дозволило значно підвищити точність у відповідях та адаптацію до запитів користувачів.

Модель GPT-2 сприяла зміні поглядів на можливості генеративного ШІ. Вона стала доступною для широкого загалу, що дозволило розробникам створювати нові програми на основі її API. Це також викликало багато дискусій щодо етичності використання таких моделей, оскільки GPT-2 могла використовуватися для створення недостовірного контенту, новин та маніпуляцій.

OpenAI продовжила розвиток генеративних моделей, випустивши у 2020 році GPT-3. Це був серйозний прорив, оскільки GPT-3 мала вже 175 мільярдів параметрів, що в десятки разів більше, ніж GPT-2. Така кількість параметрів дозволила досягти небаченої якості у генерації текстів та адаптації до різних типів запитів.

GPT-3 використовувалась не тільки для генерації текстів, але й для вирішення складних завдань у сфері науки, медицини, бізнесу та навіть для написання програмного коду. Наприклад, модель могла автоматично створювати статті, пояснювати наукові поняття, аналізувати великі обсяги даних і генерувати відповіді на специфічні запити користувачів. Її багатофункціональність дозволила застосовувати GPT-3 у таких проєктах, як віртуальні помічники, чат-боти та автоматизація документообігу.

GPT-3 стала основою для комерційного API OpenAI, який використовується багатьма компаніями для створення інноваційних продуктів і сервісів. Вона зробила революцію у використанні ШІ в бізнесі та науці.

GPT-4 — це наступне покоління мовної моделі від OpenAI, випущене у 2023 році, яке стало значним кроком уперед у порівнянні з GPT-3. Завдяки більшій кількості параметрів та вдосконаленій архітектурі, GPT-4 забезпечує ще вищу якість текстових відповідей, кращу узгодженість і здатність розуміти складніші запити, що робить його одним із найбільш досконалих генеративних моделей на сьогодні.

GPT-4 краще справляється зі складними запитами та забезпечує високу точність відповідей, зменшуючи кількість логічних помилок або неточностей, які могли траплятися в GPT-3. Наприклад, GPT-4 більш точно відповідає на запитання, що вимагають міждисциплінарного підходу, або запити, що містять багатоетапні інструкції.

GPT-4 також демонструє кращу здатність розуміти контекст, особливо у складних або неоднозначних запитах, що робить його ідеальним для підтримки високоякісних обговорень, аналітичних задач та генерації спеціалізованих текстів.

GPT-4 має мультимодальні можливості, тобто може обробляти не лише текст, а й зображення (у певних конфігураціях). Це означає, що користувачі можуть подавати на вхід зображення разом з текстом, що дає можливість розширити його застосування у багатьох нових сферах.

Наприклад, GPT-4 може аналізувати вміст зображення та відповідати на питання про те, що на ньому зображено, що корисно для застосувань у сфері обробки зображень, візуальної аналітики та створення інтерфейсів доступності.

OpenAI активно працює над тим, щоб GPT-4 став більш стійким до упереджень і неправомірних відповідей. Під час розробки використовувалася нова методика налаштування, яка дозволяє моделі уникати небажаних поведінок, таких як упереджені судження чи агресивні відповіді.

GPT-4 краще розпізнає запити, що стосуються етики, соціальних норм, і генерує відповіді, орієнтовані на підтримку безпечного та етичного використання штучного інтелекту.

GPT-4 можна налаштовувати під конкретні галузі знань, що дозволяє створювати спеціалізовані версії для різних завдань, як-от медичний діагноз, фінансовий аналіз або юридичні консультації. Це відкриває нові можливості для розширення застосувань GPT-4 в індустріях, які потребують високої точності та відповідальності.

Завдяки такій спеціалізації, GPT-4 може працювати як інтелектуальний помічник для професіоналів, підтримуючи їхню роботу у складних питаннях, що вимагають точності та аналітичних здібностей.

Одним із важливих покращень GPT-4 є здатність утримувати більш об'ємний контекст протягом розмови або роботи з користувачем. Модель може зберігати контекст більших обсягів тексту, що дозволяє вести довші розмови без втрати послідовності та розуміння попередніх повідомлень.

Це значно підвищує якість обслуговування клієнтів, допомагаючи автоматизувати складні консультації, підготовку документів, де кожна деталь має значення.

GPT-4 застосовується у багатьох сферах завдяки його гнучкості та розширеним можливостям:

— бізнес і фінанси — GPT-4 використовується для автоматизації бізнес-аналізу, підготовки звітів та надання порад з управління фінансами, це дозволяє скоротити час на аналітику, роблячи процес прийняття рішень швидшим та ефективнішим;

— медицина — GPT-4 застосовується для обробки медичних текстів, допомагає лікарям у клінічних дослідженнях та підготовці до діагностики, модель може синтезувати наукові дані, надавати рекомендації та полегшувати аналіз медичної інформації;

— юридична сфера— GPT-4 автоматизує частину юридичної роботи, допомагаючи у підготовці документів, юридичних консультаціях та аналізі законів, це значно полегшує роботу юристів, скорочуючи час на обробку великої кількості текстів.

OpenAI зробила значний внесок у розвиток генеративних моделей штучного інтелекту, створивши ряд інноваційних технологій, таких як GPT-2, GPT-3, Codex, DALL-E, Whisper та GPT-4, що стали революційними у своїй здатності до генерації тексту, програмного коду та навіть зображень. Всі ці моделі мають спільну мету — створювати ефективні, точні та універсальні інструменти для автоматизації складних завдань, полегшуючи роботу людей у різних сферах, від бізнесу до науки та творчості.

Особливо значущим досягненням стала модель GPT-4, яка стала наступним етапом розвитку технологій, продовживши успіх попередніх версій. GPT-4 не тільки демонструє вищу якість та точність, але й володіє мультимодальними можливостями, що дає змогу обробляти зображення та інші типи даних. Це відкриває нові горизонти для застосувань у таких галузях, як медіа, медицина, фінанси, освіта та юриспруденція, де потрібна точність, ефективність і здатність адаптуватися до специфічних завдань.

Завдяки своїй гнучкості та потужності, GPT-4 здатна обробляти великий обсяг інформації, підтримувати складні діалоги та генерувати високоякісні

результати, що значно полегшує роботу в різних професійних сферах. Однак, незважаючи на всі досягнення, важливо зазначити, що разом з розвитком таких технологій виникають нові виклики. Серед них — етичні питання, ризики щодо безпеки та захисту даних, а також необхідність контролю за можливим використанням таких моделей для маніпуляцій чи створення шкідливого контенту.

Незважаючи на ці виклики, досягнення OpenAI в сфері генеративних моделей значно змінюють не лише технічну, але й соціальну картину, відкриваючи нові можливості для розвитку та застосування штучного інтелекту. В подальшому, ці технології, ймовірно, будуть ставати дедалі важливішими у створенні нових інноваційних продуктів та послуг, що зроблять наше життя ще зручнішим та ефективнішим.

2 ОБГРУНТУВАННЯ ВИБОРУ ФРЕЙМОРКІВ ТА ПІДХІД ДО РОЗРОБКИ

2.1 Теорія фреймворків

Фреймворки для розробки веб-сервісів відіграють ключову роль у сучасній веб-розробці завдяки своїй здатності значно спростити і стандартизувати процес створення складних веб-додатків. Вони об’єднують набір інструментів, бібліотек і шаблонів, що дозволяють автоматизувати більшість рутинних завдань, залишаючи розробнику більше часу для реалізації специфічної бізнес-логіки. Завдяки цьому зменшується кількість помилок, зростає ефективність процесу розробки та спрощується подальше обслуговування програмного продукту.

Одним із найважливіших аспектів використання фреймворків є абстракція процесів розробки. Завдяки фреймворкам розробник може не витрачати час на створення базових функцій, таких як обробка запитів, маршрутизація або взаємодія з базою даних. Наприклад, фреймворк Django на Python надає потужний інструмент для роботи з базами даних, включаючи ORM (Object-Relational Mapping), що дозволяє працювати з базою даних за допомогою об’єктів замість написання SQL-запитів вручну. Це значно знижує ймовірність помилок і спрощує процес розробки.

Масштабованість є важливою вимогою для сучасних веб-сервісів, які часто повинні обробляти великий обсяг даних або запитів від користувачів. Більшість фреймворків, таких як Spring для Java або Express для Node.js, дозволяють розробникам створювати додатки, які можуть масштабуватися як горизонтально (через додавання нових серверів), так і вертикально (через підвищення потужностей існуючих серверів). Це забезпечує зростання додатків без потреби переписувати їх код або повністю змінювати архітектуру. Фреймворки також пропонують вбудовані рішення для балансування навантаження, що дозволяє рівномірно розподіляти запити між кількома серверами.

Безпека є ще однією важливою перевагою використання фреймворків. Веб-додатки часто працюють з конфіденційною інформацією, тому забезпечення їх безпеки є критичним аспектом. Фреймворки надають вбудовані інструменти для захисту від найпоширеніших атак, таких як SQL-ін’єкції, Cross-Site Scripting (XSS)

і Cross-Site Request Forgery (CSRF). Наприклад, у фреймворках, як Rails (Ruby) або Laravel (PHP), вбудовані механізми для захисту сесій користувачів та шифрування паролів, що значно знижує ризик виникнення уразливостей. Такі інструменти дозволяють розробникам зосередитись на функціональності, а не на реалізації стандартних заходів безпеки.

Однією з основних переваг фреймворків є зручність у розробці. Вони зазвичай пропонують чітко визначену структуру проекту, що дозволяє стандартизувати підхід до створення веб-додатків. Розробникам не потрібно створювати код з нуля для обробки запитів, взаємодії з базами даних або керування сесіями користувачів. Наприклад, у фреймворку Angular є вбудовані інструменти для роботи з формами, маршрутизацією, а також інтерсепторами HTTP-запитів. Це дозволяє розробникам швидко створювати потужні та масштабовані додатки.

Гнучкість фреймворків також є важливим аспектом. Багато сучасних фреймворків дозволяють додавати або вилучати модулі в залежності від вимог проекту. Наприклад, у Node.js можна вибирати різні бібліотеки для роботи з базами даних (MongoDB, PostgreSQL), для обробки запитів (Express.js) або для розширення функціональності додатка (наприклад, для автентифікації через JWT). Це дозволяє створювати як прості додатки, так і більш складні, з урахуванням специфічних потреб бізнесу.

Завдяки таким характеристикам, як абстракція, масштабованість, безпека, зручність і гнучкість, фреймворки є необхідним інструментом для сучасних веб-сервісів. Вони дозволяють значно зменшити витрати часу на розробку, підвищити якість коду та полегшити підтримку додатків. Вибір фреймворку залежить від специфіки проекту, але всі вони надають потужні можливості для створення ефективних і безпечних веб-додатків.

2.2 Обгрунтування вибору фреймворку Vue

Vue.js є одним з найпопулярніших JavaScript-фреймворків для розробки веб-додатків завдяки своїй простоті, гнучкості та потужним можливостям. Він був розроблений для того, щоб спростити створення інтерактивних інтерфейсів

користувача та односторінкових додатків, і вже здобув велику популярність серед розробників по всьому світу. Основна ідея Vue полягає в створенні реактивних компонентів, що дозволяє веб-додаткам бути динамічними та зручними для користувачів, а також значно спрощує процес розробки.

Однією з основних особливостей Vue є його реактивність. Це означає, що дані, з якими працює додаток, автоматично синхронізуються з інтерфейсом користувача, що дозволяє значно зменшити кількість коду, необхідного для оновлення веб-сторінок. Відразу після зміни значень у даних компонента Vue оновлює DOM, і це відображається на екрані без необхідності в явному виклику оновлення або змінюванні елементів вручну. Цей двосторонній зв'язок даних є важливою перевагою Vue (див. рисунок 2.1), оскільки значно полегшує створення інтерактивних та динамічних інтерфейсів, таких як форми, таблиці, фільтри та інші елементи, що залежать від введених користувачем даних.

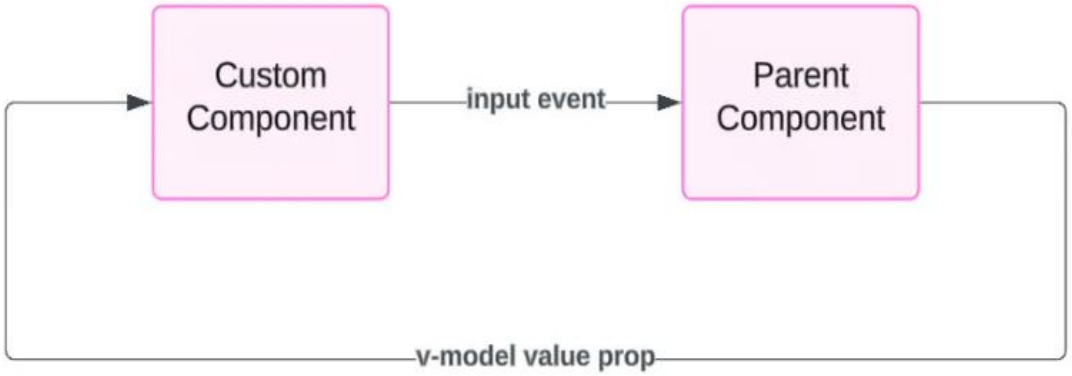

Рисунок 2.1 — Схема двостороннього зв'язку у Vue.js

Фреймворк побудований на компонентному підході, що дозволяє структурувати веб-додатки в модульні одиниці — компоненти, кожен з яких має свій шаблон, стилі та логіку. Така організація коду допомагає значно спростити розробку великих додатків, оскільки розробники можуть працювати з окремими компонентами без необхідності розбиратися в усій структурі додатка. Кожен компонент може бути легко повторно використаний в інших частинах додатку, що підвищує ефективність розробки та підтримки проекту.

Ще однією значущою перевагою Vue є його легкість у вивченні та інтеграції. Фреймворк має низький поріг входження, що дозволяє навіть новачкам швидко освоїти основи роботи з ним. Завдяки чіткій документації та зрозумілому синтаксису, розробники можуть легко освоїти Vue, навіть якщо вони мають обмежений досвід роботи з JavaScript. Крім того, Vue можна інтегрувати в існуючі проекти, що робить його відмінним вибором для поетапного переходу до сучасного фреймворку без потреби переписувати великий обсяг існуючого коду.

Vue також відрізняється високою продуктивністю, що дозволяє використовувати його для створення масштабованих додатків. Фреймворк має легку та оптимізовану архітектуру, що забезпечує швидке завантаження сторінок та ефективне управління станом. Завдяки використанню віртуального DOM, Vue мінімізує кількість змін у реальному DOM, що допомагає зменшити час на рендеринг і зробити додатки більш швидкими та відгукними.

Ще однією важливою характеристикою є гнучкість Vue. Він надає розробникам можливість використовувати лише необхідні йому частини фреймворку, без обов'язкової прив'язки до конкретних рішень. Наприклад, розробники можуть обрати лише систему компонентів і реактивність, не використовуючи повну екосистему інструментів, таких як Vue Router або Vuex. Така гнучкість дає можливість будувати проекти різного рівня складності, від простих до більш комплексних, що потребують додаткових функцій.

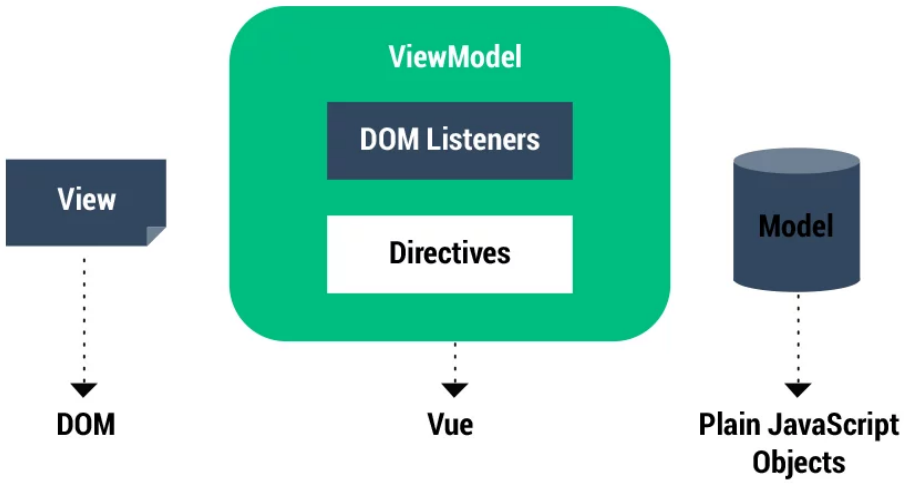

Рисунок 2.2 — Схема архітектури Vue.js

Попри всі ці переваги, Vue має і свої обмеження. Оскільки це досить новий фреймворк порівняно з іншими популярними рішеннями, такими як React чи Angular, у Vue може бути обмежене ком'юніті в певних аспектах, наприклад, у кількості готових рішень або підтримки бібліотек. Також, через відсутність великих корпорацій, які активно підтримують фреймворк, деякі підприємства можуть бути більш скептичними щодо його використання на підприємницькому рівні, порівняно з більш перевіреними технологіями.

Загалом, Vue є потужним і гнучким фреймворком, який ідеально підходить для розробки динамічних веб-додатків. Його простота в освоєнні, реактивність, компонентний підхід і висока продуктивність роблять його відмінним вибором для створення як невеликих, так і великих проектів. Він дозволяє розробникам зосередитися на розвитку бізнес-логіки та користувацького досвіду, автоматизуючи багато аспектів взаємодії з даними та інтерфейсом.

2.3 Переваги вибору Vue

Вибір Vue.js для розробки веб-додатків на основі моделі Davinci є обґрунтованим та вигідним з кількох причин, які забезпечують ефективну і гнучку інтеграцію штучного інтелекту з веб-інтерфейсом. Vue.js, з його реактивною системою та компонентним підходом, ідеально підходить для створення інтерактивних веб-додатків, що використовують потужні можливості Davinci для генерації та обробки даних.

Перше, що варто відзначити, це легкість інтеграції Vue з різними зовнішніми технологіями та бібліотеками. Davinci API дає змогу додавати розширені можливості штучного інтелекту в веб-додатки, а Vue дозволяє просто та ефективно вбудовувати ці функції в інтерфейс. Завдяки чітко структурованому компонентному підходу Vue, розробники можуть швидко та зручно додавати нові елементи або оновлювати інтерфейси, що активно взаємодіють з API Davinci, наприклад, для створення динамічних чат-ботів, персоналізованих рекомендацій або інтерактивних веб-сайтів, що генерують контент на основі запитів користувачів.

Іншою важливою перевагою є реактивність Vue, що дозволяє інтегрувати обробку даних від Davinci в реальному часі. Якщо Davinci генерує новий контент або здійснює аналітичну обробку, реактивна система Vue автоматично оновлює інтерфейс без необхідності вручну маніпулювати DOM. Коли веб-додаток потребує швидкого оновлення даних на основі змін у моделі Davinci. Наприклад, коли AI генерує відповіді на запити користувача, Vue дозволяє миттєво оновлювати UI з новим текстом або даними, надаючи безперервний та інтерактивний досвід. Реактивність Vue.js наведено на рис. 2.3.

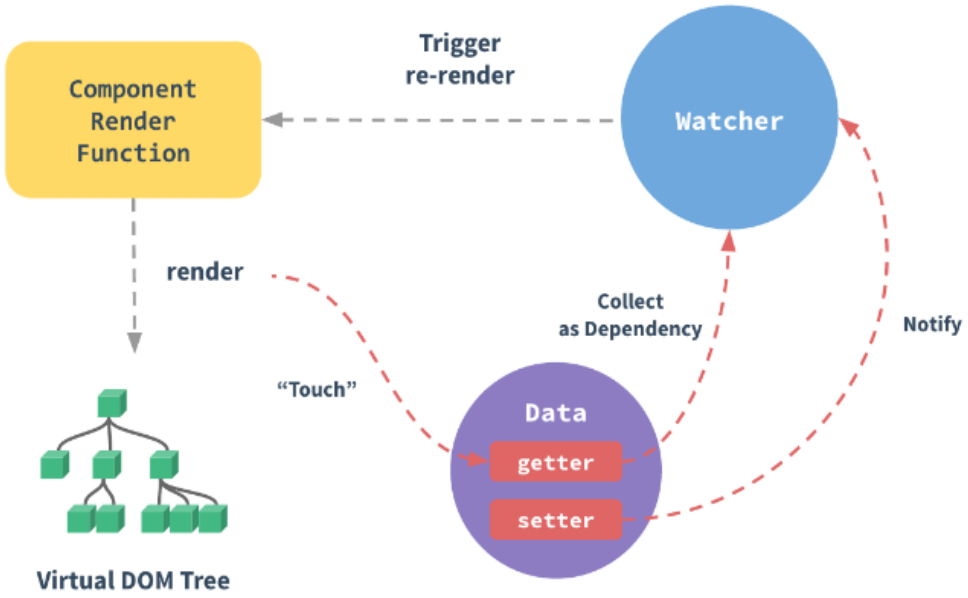

Рисунок 2.3 — Реактивність Vue.js

Vue також забезпечує високу гнучкість в створенні інтерфейсів, що взаємодіють з Davinci. Завдяки компонентній структурі можна легко розділяти код на окремі частини, що відповідають за взаємодію з конкретними аспектами Davinci. Наприклад, один компонент може бути відповідальний за запит до API для генерації тексту, інший — за відображення результатів, а третій — за обробку взаємодії з користувачем. Така організація коду допомагає спростити процес розробки і забезпечує високу модульність, що важливо при роботі з складними веб-додатками, які інтегрують зовнішні технології, такі як Davinci.

Також варто зазначити, що Vue.js надає великий вибір готових інструментів і плагінів, які можна використовувати для полегшення взаємодії з різними API, зокрема Davinci. Завдяки екосистемі Vue, розробники можуть використовувати готові рішення для роботи з API, керування станом додатка або реалізації роутінгу, що дозволяє зосередитися на більш важливих аспектах проекту, таких як створення корисних і ефективних функцій, що використовують штучний інтелект.

Крім того, Vue.js дозволяє оптимізувати продуктивність веб-додатків, що є важливою перевагою при роботі з такими складними інструментами, як Davinci. Зважаючи на обсяг даних і складність обчислень, які можуть бути необхідні для генерування контенту або виконання запитів до API, Vue дозволяє ефективно організувати управління станом і оновлення інтерфейсу, забезпечуючи стабільну і швидку роботу додатка навіть при високих навантаженнях.

І, звісно, важливою перевагою є швидкість розробки за допомогою Vue. Завдяки його простоті в освоєнні, легкості в інтеграції з іншими технологіями та можливості швидко створювати інтерактивні інтерфейси, Vue дозволяє значно скоротити час розробки. Це особливо корисно, коли потрібно швидко впроваджувати нові функції на основі Davinci, адже це дозволяє фокусуватися на реалізації специфічних задач, пов'язаних із штучним інтелектом, без витрачання часу на налагодження основної інфраструктури веб-додатка.

Вибір Vue для розробки на базі Davinci також обґрунтований високою підтримкою ком'юніті та широким набором ресурсів. З огляду на активну спільноту розробників Vue, що постійно ділиться своїм досвідом і знаннями, а також на велику кількість бібліотек і плагінів, розробники можуть швидко знайти рішення для специфічних задач, що виникають при інтеграції з Davinci. Це зменшує кількість можливих технічних проблем та дозволяє більш ефективно вирішувати завдання в процесі розробки.

Таким чином, вибір Vue.js для розробки веб-додатків на основі Davinci є дуже вигідним, оскільки цей фреймворк дозволяє створювати інтерактивні, ефективні та масштабовані інтерфейси, що безперешкодно взаємодіють з потужними можливостями штучного інтелекту. Завдяки своїй простоті, гнучкості,

високій продуктивності та чудовій підтримці спільноти, Vue є ідеальним вибором для створення сучасних веб-сервісів, які інтегрують Davinci для вирішення складних задач.

2.4 Розробка веб-сервісу

Розробка веб-сервісу на базі Vue.js має низку важливих аспектів, які забезпечують ефективне та зручне створення інтерактивних додатків. Основним моментом є використання компонентного підходу, де кожен компонент є окремою частиною інтерфейсу з власними стилями та логікою. Це дозволяє розділяти код на дрібні, керовані частини, що спрощує процес розробки, підтримки та масштабування додатка.

Vue.js дозволяє створювати веб-сервіси, що взаємодіють із користувачем через інтуїтивно зрозумілий інтерфейс, і має дуже сильні можливості для інтеграції з різноманітними сторонніми сервісами та технологіями. Одним з таких інструментів є Vue Router, який дозволяє організувати маршрутизацію в односторінкових додатках (SPA), що є основною вимогою для багатьох сучасних веб-сервісів. Завдяки цій технології можна створювати динамічні переходи між різними частинами додатка без перезавантаження сторінки, що значно покращує користувацький досвід. Архітектурна схема SPA Vue.js наведена на рис. 2.4.

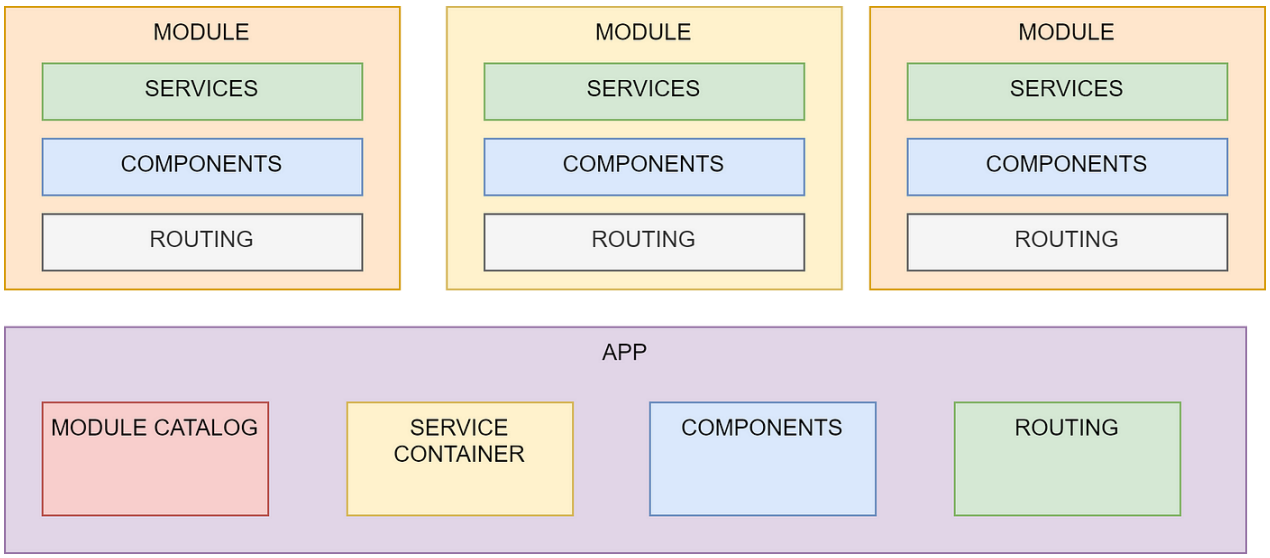

Рисунок 2.4 — Архітектурна схема SPA Vue.js

Для більш складних сценаріїв використовується Vuex, що дає можливість централізовано керувати станом додатка, особливо коли мова йде про великі веб-сервіси, які потребують обміну даними між різними компонентами. Веб-сервіс, що має безліч компонентів або підсистем, може отримати значні переваги від такого централізованого управління станом, що полегшує синхронізацію даних між різними частинами додатка.

Окрім цього, Vue.js має вбудовану систему для роботи з асинхронними запитами, що є важливим при взаємодії з зовнішніми API або базами даних. Інтеграція з такими інструментами, як Axios, дозволяє ефективно здійснювати запити до серверів для отримання даних або обробки запитів без переривання роботи додатка. Це дає можливість легко додавати функціональність, що взаємодіє з бекенд-сервісами, наприклад, для отримання користувацьких даних або відправлення запитів до штучного інтелекту через API.

Vue.js також підтримує гарну інтеграцію з системами побудови та пакування додатків, такими як Webpack чи Vite, що дозволяє оптимізувати процес розробки та зменшити час завантаження сторінок. Ці інструменти дозволяють мінімізувати розміри файлів, використовувати модульний підхід до коду, а також налаштовувати обробку стилів, зображень і скриптів для максимальної ефективності роботи веб-сервісу.

Що стосується безпеки, то хоча Vue.js сам по собі не надає вбудованих функцій для безпеки, його можна легко інтегрувати з різними рішеннями для захисту даних, а також реалізувати безпечну аутентифікацію і авторизацію користувачів. Використання сторонніх бібліотек для захисту запитів та даних, таких як OAuth або JWT (JSON Web Token), дозволяє забезпечити безпечну передачу даних між клієнтом і сервером, що особливо важливо для веб-сервісів, які працюють з чутливими даними користувачів.

Важливою перевагою Vue є його легкість у тестуванні. Завдяки наявності спеціальних інструментів, таких як Vue Test Utils, а також інтеграції з популярними тестовими фреймворками, такими як Jest або Mocha, можна легко проводити юніт-тестування окремих компонентів додатка. Це дозволяє виявляти проблеми на

ранніх етапах розробки, забезпечуючи високу якість коду та стабільність веб-сервісу на всіх етапах його життєвого циклу.

Фреймворки для розробки додатків, такі як Vue.js, є важливими інструментами для створення сучасного, масштабованого і зручного у підтримці програмного забезпечення. Вони дозволяють стандартизувати структуру проєкту, роблячи її зрозумілою та логічною для розробників, що працюють у команді, та сприяють реалізації принципів модульності

Завдяки фреймворкам розробники можуть зосередитися на вирішенні бізнес-завдань, залишаючи реалізацію складних технічних аспектів, таких як маршрутизація, управління станом, обробка запитів і оптимізація продуктивності, під контролем цих інструментів. Фреймворки також підтримують активні спільноти, які забезпечують надійну документацію, приклади використання та готові рішення для поширених завдань.

Таким чином, фреймворки є фундаментом для створення високоякісного програмного забезпечення.

3 ТЕОРІЯ ТА ВИБІР МОДЕЛІ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Теорія штучного інтелекту

Теорія штучного інтелекту є галуззю науки, яка досліджує принципи, методи й алгоритми, що лежать в основі створення та використання інтелектуальних систем. Вона зосереджена на розумінні природи інтелекту та створенні комп'ютерних моделей, здатних виконувати завдання, які традиційно вважалися суто людськими.

Ця наука базується на різноманітних підходах, таких як символьне представлення знань, машинне навчання, нейронні мережі, обчислення з нечіткими множинами та еволюційні алгоритми. Вона охоплює широкий спектр завдань: розпізнавання образів, обробку природної мови, прийняття рішень, планування, робототехніку тощо. Основною метою є розробка моделей та алгоритмів, які дозволяють комп'ютерним системам аналізувати, розуміти та використовувати знання для вирішення практичних проблем.

Одним із ключових питань теорії є моделювання процесу мислення. Вивчаються методи представлення знань і логічного виведення, такі як байєсівські мережі, логічне програмування та інші. Це допомагає створювати системи, здатні до самостійного навчання, адаптації та розв'язання складних завдань.

Особливу увагу приділяють ефективності та оптимізації алгоритмів, що сприяє підвищенню продуктивності інтелектуальних систем. Також розглядаються етичні й соціальні аспекти впровадження ШІ, зокрема питання прозорості, автономності, захисту приватності й розробки стандартів використання. Іншим важливим напрямом є дослідження в робототехніці, зокрема інтеграція роботів у різні сфери діяльності та взаємодія людини з інтелектуальними системами.

Теорія штучного інтелекту також активно досліджує багатоагентні системи, де велика система складається з автономних агентів, що взаємодіють між собою та з довкіллям. Ще одним значущим напрямом є обробка природної мови, яка спрямована на розробку алгоритмів для розуміння та генерації людської мови, що використовується, наприклад, у перекладі, аналізі тексту чи пошуку інформації.

3.2 Різновидність моделей штучного інтелекту

Моделі штучного інтелекту є фундаментальним елементом сучасних інтелектуальних систем і можуть бути класифіковані за їх здатністю до вирішення завдань, навчання, аналізу даних та імітації людського інтелекту. Загалом вони поділяються на вузькі (спеціалізовані) та універсальні, кожна з яких має свої унікальні риси і застосування.

Вузькі моделі спрямовані на виконання конкретних завдань. Вони створюються для спеціалізованих функцій, таких як розпізнавання мовлення, аналіз зображень чи прогнозування тенденцій. Ці моделі відзначаються високою точністю в обмежених областях, але не можуть виконувати завдання за межами своєї спеціалізації.

Універсальні моделі мають набагато ширший спектр застосування. Вони навчаються на великих масивах даних і здатні виконувати різноманітні задачі, включаючи розуміння тексту, генерування мовлення, створення зображень і навіть розв'язання складних проблем у різних доменах. Такі моделі, як GPT (зокрема, GPT-4), є прикладом універсальних систем, що здатні адаптуватися до багатьох контекстів завдяки своїй масштабованій архітектурі та здатності працювати з різними форматами даних.

Важливою рисою сучасних моделей є їх здатність до самонавчання та адаптації. Завдяки інтеграції принципів глибокого навчання та роботи з великими обсягами інформації, вони демонструють виняткову гнучкість і здатність покращувати свої результати на основі нових даних.

Також ці моделі відіграють ключову роль у впровадженні штучного інтелекту у повсякденне життя, забезпечуючи нові можливості для автоматизації, підвищення продуктивності та створення інноваційних продуктів. Їхня розробка підкріплюється значними досягненнями в обчислювальних ресурсах, що дозволяє створювати масштабовані системи, які дедалі більше наближаються до розуміння та імітації людської інтелектуальної діяльності.

3.3 Підхід розробки на базі Davinchi

Використання моделі Davinci від OpenAI для розробки веб-сайтів і веб-сервісів дозволяє значно покращити ефективність і продуктивність процесу створення та підтримки веб-проектів. Модель, яка є частиною серії GPT-3, демонструє надзвичайні можливості в автоматизації багатьох аспектів розробки веб-сайтів, починаючи від генерування контенту та коду, до покращення взаємодії з користувачем і підтримки високої якості продукту. Аналіз цієї технології дає змогу зрозуміти, як Davinci може змінити традиційні підходи до розробки веб-сервісів.

Архітектура GPT (див. рисунок 3.1) складається з кількох важливих компонентів, що забезпечують її роботу. На початковому етапі текст перетворюється у вигляді токенів, які отримують векторні представлення за допомогою спеціального шару перетворень. До цих векторів додається інформація про позицію кожного токена в послідовності, що дозволяє моделі враховувати порядок слів у тексті.

Далі дані обробляються через серію трансформерних блоків. У кожному з них реалізовано механізм багатоголової самоповної уваги, який аналізує взаємозв’язки між різними словами тексту. Крім того, блоки містять повнозв’язну нейронну мережу, яка забезпечує додаткове обчислювальне перетворення. Для підтримки стабільності навчання використовуються нормалізація та спеціальні залишкові з’єднання.

Завершальним етапом є обчислення ймовірності наступного слова за допомогою шару Softmax. Цей процес дозволяє моделі будувати текст на основі попередніх слів, навчаючись на великих обсягах текстової інформації.

З одного боку, використання Davinci дозволяє автоматизувати багато рутинних завдань, що зазвичай потребують значного часу і ресурсів. З іншого боку, вона дає можливість розробникам і дизайнерам зосередитися на більш креативних і складних завданнях, залишаючи частину технічної роботи, наприклад, генерування коду або контенту, на плечах штучного інтелекту.

Одним із основних напрямів, де Davinci може бути використана, є автоматизація створення контенту для веб-сайтів. Це особливо актуально для

бізнесів, що потребують регулярного оновлення контенту, наприклад, для ведення блогів, створення рекламних матеріалів або для інтернет-магазинів, де потрібно генерувати численні описи товарів.

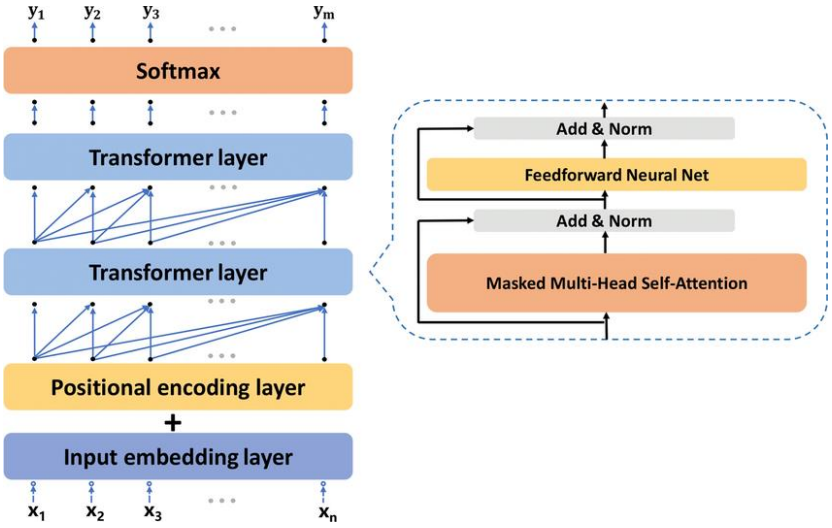

Рисунок 3.1 — Архітектурна схема GPT

Модель може не лише допомогти швидко створювати контент, але й робити це в рамках певного стилістичного та тематичного напрямку, що дуже важливо для забезпечення унікальності і відповідності вимогам SEO. Важливо також, що Davinci здатна генерувати тексти, які є логічними та граматично коректними, що значно знижує ризики помилок і необхідність подальшого редагування.

Не менш важливим є те, що модель Davinci може бути використана для оптимізації взаємодії з користувачем. Розробка веб-сайтів, орієнтованих на користувача, вимагає глибокого розуміння його потреб і поведінки. Здатність моделі генерувати природну мову і підтримувати діалог дозволяє створювати ефективні чат-боти та автоматизовані системи підтримки користувачів. Такий асистент може не тільки відповідати на запитання, але й допомагати користувачам у вирішенні конкретних проблем, надаючи корисну інформацію або рекомендації. Крім того, модель може бути використана для персоналізації контенту: на основі попередніх взаємодій з користувачем, вона може пропонувати продукти чи послуги, що відповідають його інтересам, а також адаптувати інформацію, що відображається на сайті, до індивідуальних переваг.

Особливо цікавою є можливість використання моделі для автоматизації тестування веб-сайтів. Тестування — це невід'ємна частина будь-якої розробки, що гарантує якість продукту та його функціональність. Davinci може допомогти не тільки в написанні тестових сценаріїв, а й у проведенні перевірок на наявність помилок у коді. Вона здатна розпізнавати типові помилки, пропонувати способи їх виправлення або навіть знаходити оптимізації в існуючому коді. Це дозволяє розробникам швидше знаходити й усувати помилки, знижуючи ризик виникнення багів на етапі тестування.

Що стосується дизайну веб-сайтів, то хоча Davinci не генерує графіку, вона може бути дуже корисною в процесі створення текстових елементів дизайну, таких як заголовки, кнопки, підписи до зображень та інші текстові блоки. Вона може також допомогти в покращенні навігаційної структури сайту, надаючи рекомендації щодо того, як краще організувати контент на сторінках для зручності користувача. Крім того, вона може адаптувати сайт для мобільних пристроїв, генеруючи відповідні CSS-стилі та структуру, що відповідає вимогам мобільної версії.

Потрібно також зазначити, що хоча використання моделі Davinci в розробці веб-сайтів може значно зменшити час, витрачений на виконання рутинних завдань, існують деякі обмеження. Одним із таких обмежень є необхідність ретельного налаштування й моніторингу роботи моделі. Не кожен запит може бути коректно зрозумілий або виконаний без подальшого втручання людини. Тому важливо мати досвідчених фахівців, які зможуть перевіряти й коригувати роботу моделі.

Загалом, використання Davinci для розробки веб-сайтів і веб-сервісів дозволяє значно автоматизувати процеси, підвищити ефективність і знизити ймовірність помилок. Це створює нові можливості для розробників і дизайнерів, оскільки дозволяє зосередитися на більш креативних завданнях і вирішенні складних технічних проблем, залишаючи рутинну роботу на штучному інтелекті. Але для досягнення найкращих результатів, важливо використовувати цей інструмент у поєднанні з досвідом і професіоналізмом команди розробників, щоб досягти балансу між автоматизацією і якістю кінцевого продукту.

3.4 Обгрунтування вибору моделі Davinchi

Вибір моделі Davinci для розробки веб-сайтів і веб-сервісів обґрунтований її здатністю значно покращити ефективність різних процесів, пов'язаних із створенням і підтримкою веб-продуктів. Однією з основних переваг є її високий рівень адаптивності та можливість інтеграції в різні аспекти розробки. Модель Davinci здатна не лише автоматизувати багато задач, а й покращити взаємодію з користувачами, що має критичне значення для успіху будь-якого веб-сервісу.

Davinci є надзвичайно потужним інструментом для оптимізації взаємодії з користувачами. Завдяки її здатності до глибокого розуміння контексту та природної мови, вона може бути використана для автоматизованих систем підтримки, чат-ботів або інтерактивних елементів на сайті. Це дозволяє значно знизити навантаження на персонал, а також забезпечити високий рівень обслуговування, зменшуючи час очікування для користувачів і підвищуючи їхній досвід взаємодії з веб-сайтом.

Ще однією важливою характеристикою є здатність моделі адаптувати веб-сайт до потреб конкретного користувача. Це дозволяє створювати персоналізовані сторінки, які відповідають інтересам і вподобанням кожного відвідувача. Персоналізація на основі аналізу поведінки користувача, її переваг або попередніх взаємодій з сайтом дозволяє зробити досвід більш інтуїтивно зрозумілим і зручним. Це, в свою чергу, підвищує конверсію та сприяє залученню постійних користувачів.

Іншою важливою перевагою є здатність Davinci до оптимізації внутрішніх процесів розробки та тестування. Веб-розробка зазвичай пов'язана з великим обсягом тестування функціональності, стабільності та безпеки продукту. Модель може допомогти в автоматизації перевірок на наявність помилок, що дозволяє прискорити цикл тестування і зменшити кількість помилок, що можуть виникнути під час запуску сайту. Це допомагає знизити витрати на підтримку та зберігати високий рівень якості веб-продукту в умовах постійних змін і оновлень.

Також, важливою перевагою вибору Davinci є її здатність до інтеграції з різними технологіями та платформами. Веб-сайти часто працюють з різноманітними сторонніми сервісами, такими як CRM-системи, бази даних,

платіжні системи або аналітичні інструменти. Davinci може бути ефективно використана для автоматизації взаємодії між цими системами, що дозволяє знизити витрати на інтеграцію та підтримку. Її гнучкість дозволяє адаптувати модель до різних потреб і технічних вимог проекту, що робить її універсальним інструментом для багатьох типів веб-сайтів і сервісів.

Крім того, використання Davinci дозволяє значно знижувати ризики, пов'язані з людським фактором. Веб-розробка часто передбачає виконання рутинних завдань, що може призвести до помилок або несумісностей між компонентами системи. Модель може автоматично перевіряти і коригувати ці процеси, забезпечуючи більшу точність і узгодженість роботи веб-сервісу.

Нарешті, Davinci дозволяє скоротити час, необхідний для розробки та впровадження нових функцій на веб-сайті. Завдяки її здатності швидко виконувати завдання і автоматизувати багато процесів, розробники можуть зосередитися на складніших аспектах, що вимагають креативного підходу. Це не лише покращує продуктивність команди, але й дозволяє швидше адаптуватися до змін у ринку чи вимогах користувачів.

Загалом, обґрунтованість вибору Davinci для розробки веб-сайтів і веб-сервісів можна пояснити її здатністю значно підвищити ефективність та гнучкість процесів розробки. Вона дозволяє автоматизувати рутинні завдання, покращити взаємодію з користувачами, забезпечити персоналізацію контенту та оптимізувати внутрішні процеси. Це робить модель потужним інструментом для досягнення високої якості веб-сервісів при зменшенні витрат і часу на розробку та підтримку.

3.5 Вбудова моделі штучного інтелекту

Вбудова Davinci в веб-сервіс є процесом інтеграції передової мовної моделі від OpenAI в веб-додаток, що дає змогу скористатися потужностями штучного інтелекту для виконання складних задач, таких як генерація тексту, відповіді на запити користувачів, створення рекомендацій або аналіз великих обсягів інформації. інтеграція Davinci в веб-сервіс забезпечує можливість створення інтелектуальних, інтерактивних функцій, що можуть автоматизувати процеси,

надавати користувачам персоналізований досвід або допомагати в рішенні складних завдань. Основними кроками для успішної вбудови є налаштування API, обробка запитів і відповіді, а також забезпечення безпеки і продуктивності.

Першим етапом інтеграції є налаштування API Davinci, яке включає отримання ключа API від OpenAI. Це дозволяє вашому веб-сервісу здійснювати запити до сервера Davinci для отримання результатів на основі вхідних даних. Для цього зазвичай використовується бібліотека для роботи з HTTP-запитами, така як Axios в JavaScript, яка дозволяє передавати запити на сервер OpenAI. Запити можуть включати текстовий запит, що потребує аналізу або генерації відповіді, а також додаткові параметри для налаштування результатів, наприклад, максимальна кількість символів у відповіді або рівень креативності.

Коли запит до Davinci надсилається, модель обробляє його на сервері OpenAI, виконуючи необхідні обчислення для генерації відповідного результату. Після цього модель повертає відповідь у вигляді тексту, який можна використовувати для відображення на веб-сторінці або для подальшої обробки в додатку. Завдяки такій інтеграції, веб-сервіс може автоматично генерувати текст, пропонувати користувачам персоналізовані рекомендації, або навіть допомагати в обробці запитів за допомогою чат-ботів.

Одним із важливих аспектів вбудови Davinci є правильне оброблення запитів і відповіді від API. Запити повинні бути оптимізованими для досягнення максимальної ефективності та коректності результатів. Наприклад, можна використовувати різні моделі для різних завдань — для простих запитів використовувати швидші моделі, а для складніших завдань, які потребують глибокої генерації тексту, застосовувати більш потужні версії Davinci. Потрібно також налаштувати обробку помилок, щоб у разі невдачі API, наприклад через перевантаження чи неправильний запит, система коректно повідомляла користувача про проблему і дозволяла йому продовжити взаємодію з веб-сервісом.

Важливим аспектом є і керування даними користувачів. Оскільки моделі штучного інтелекту можуть обробляти чутливу інформацію, необхідно забезпечити відповідний рівень безпеки та конфіденційності даних. Для цього слід реалізувати

безпечне зберігання та передачу даних, а також дотримуватися вимог GDPR та інших регуляцій, що стосуються обробки персональних даних.

Також важливо продумати, як результати, що генерує Davinci, будуть інтегровані в інтерфейс веб-сервісу. Це може бути текст, що динамічно змінюється на веб-сторінці, або дані, що використовуються для подальших дій. Наприклад, якщо веб-сервіс передбачає генерацію персоналізованих рекомендацій, результати, отримані від Davinci, можна використати для оновлення списку товарів або послуг на основі запитів користувача. Завдяки реактивній системі Vue.js, можна без затримок оновлювати UI, коли отримано нові дані від API Davinci, таким чином забезпечуючи інтерактивність і швидкість відгуку на запити користувача.

Крім того, для оптимізації використання Davinci в веб-сервісі важливо ретельно планувати обмеження на кількість запитів до API, оскільки часті запити можуть призвести до високих витрат і перевантаження серверу. Для цього можна використовувати кешування результатів або обмеження частоти запитів від користувачів, наприклад, через реалізацію функцій дебаунсу або затримки перед наступним запитом. Це дозволить зменшити навантаження на API та забезпечити економічне використання ресурсів.

Продуктивність також може бути важливою проблемою, коли мова йде про інтеграцію Davinci в складні веб-сервіси, що обробляють великі обсяги запитів або працюють в реальному часі. Для покращення роботи можна використовувати технології кешування на сервері або фронтенді, щоб мінімізувати кількість запитів до Davinci, а також забезпечити швидший відгук на запити користувачів.

У випадку, коли Davinci використовується для чат-ботів або інших інтерактивних функцій, важливим є налаштування логіки взаємодії, щоб штучний інтелект розумів контекст і міг правильно відповідати на запити. Це може включати зберігання попередніх запитів та відповідей у сесії або використання більш складних алгоритмів для аналізу тексту та визначення намірів користувача.

В загальному, вбудова Davinci в веб-сервіс відкриває широкі можливості для автоматизації, персоналізації та створення інтелектуальних функцій, що можуть значно покращити користувацький досвід і додати веб-сервісу значну додану

вартість. Однак для цього потрібно враховувати всі аспекти інтеграції, від налаштування API до забезпечення безпеки і продуктивності.

Розробка веб-сервісу на базі Davinci надає можливість створити потужний та інтерактивний веб-додаток, який поєднує сучасні підходи до розробки фронтенду з передовими можливостями штучного інтелекту для обробки даних та взаємодії з користувачем.

Інтеграція Davinci в цей процес відкриває можливості для використання штучного інтелекту на етапі взаємодії з користувачем. Веб-сервіс може отримувати та обробляти складні запити, генерувати текст, аналізувати інформацію та давати персоналізовані відповіді. Це дозволяє не лише автоматизувати рутинні завдання, але й додавати інтелектуальні функції, такі як чат-боти, рекомендаційні системи та інші складні механізми, що вимагають розуміння контексту та взаємодії в реальному часі.

Як висновок, розробка веб-сервісу на базі Davinci є перспективним напрямком для створення інтелектуальних додатків з інтерактивними та адаптивними інтерфейсами, здатними надавати високоякісні сервіси користувачам у реальному часі. Така інтеграція дозволяє реалізувати найсучасніші можливості в області фронтенд-розробки та штучного інтелекту, забезпечуючи зручність, ефективність і масштабованість веб-сервісів.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Вибір програмного забезпечення

При створенні веб-сервісу важливим етапом є вибір програмного забезпечення, яке забезпечить зручне середовище для розробки, тестування та підтримки коду. У межах даного проекту було обрано Visual Studio Code як основний інструмент для роботи з кодом.

Visual Studio Code – це сучасний текстовий редактор, який поєднує в собі функціональність інтегрованого середовища розробки (IDE) та легкість використання.

Серед основних переваг можна виділити:

— кросплатформеність — підтримка Windows, macOS і Linux дозволяє використовувати VS Code незалежно від операційної системи розробника;

— гнучкість і налаштування — велика кількість доступних розширень дозволяє адаптувати середовище до потреб конкретного проекту;

— швидкість і легкість — відносно невеликий розмір і швидка робота забезпечують комфорт у роботі навіть на комп’ютерах із середніми характеристиками;

— інтеграція з Git: VS Code забезпечує зручну підтримку системи контролю версій прямо в редакторі;

— підтримка сучасних мов програмування — будована підтримка JavaScript, TypeScript, HTML, CSS та інших технологій, а також можливість додавання мов через розширення.

Для підвищення ефективності розробки веб-сервісу на основі Vue.js у цьому проєкті було використано такі розширення:

— Vetur — основний інструмент для роботи з Vue.js, що забезпечує підсвічування синтаксису, автозавершення та перевірку коду у файлах .vue, він також пропонує інтеграцію функцій, які допомагають структурувати код у багатосекційних файлах.

— Prettier – Code Formatter — цей плагін автоматично форматує код відповідно до заданих стандартів, зберігаючи єдиний стиль написання, наприклад, Prettier вирівнював відступи, розміщував лапки та форматував багаторядкові вирази, що робило код більш читабельним і легким для підтримки.

— ESLint: ESLint забезпечує перевірку якості JavaScript і TypeScript-коду, виявляючи синтаксичні помилки, порушення стилістичних правил та інші недоліки, у випадку Vue.js він працював разом із Vetur, перевіряючи всі частини файлів .vue.

— Live Server забезпечує миттєвий перегляд змін у реальному часі, полегшуючи тестування та налагодження Vue-компонентів, наприклад, розробники могли одразу перевіряти роботу стилів, JavaScript-логіки та поведінку елементів сторінки.

— Vue VSCode Snippets допомагав швидко створювати типові структури коду Vue, такі як компоненти з директивами v-model, v-for чи шаблони для методів, це дозволяло зменшити час на написання повторюваних частин коду.

— Path IntelliSense полегшує навігацію в проєкті, забезпечуючи автоматичне підказування шляхів до файлів і ресурсів, наприклад, при імпорті компонентів у Vue-файли Path IntelliSense знижував ризик допущення помилок у шляхах.

4.2 Розробка веб-сервісу

Інтерфейс системи авторизації веб-сервісу реалізує форму входу, яка включає поля для введення електронної пошти та пароля, опцію запам’ятовування користувача та посилання на сторінку реєстрації. Здійснено валідацію даних форми для перевірки заповнення обов’язкових полів. У разі помилок система надає зворотний зв’язок. Кнопка входу динамічно змінює свій стан залежно від виконання запиту, що забезпечує інтуїтивну взаємодію.

Інтерфейс реєстрації містить поля для введення імені, електронної пошти, пароля та підтвердження пароля. Передбачено перевірку відповідності пароля його підтвердженню. Після успішного заповнення форми надсилається запит до сервера, який створює новий обліковий запис. Користувач перенаправляється на сторінку

верифікації електронної пошти, а також отримує лист для підтвердження облікового запису.

Функціонал аутентифікації обробляється через сервіс, який взаємодіє з API сервера для виконання операцій входу, реєстрації, виходу, верифікації електронної пошти та скидання пароля. Здійснено інтеграцію з Vuex для управління станами, зокрема збереженням токенів, даних користувача та повідомлень про статус виконання операцій.

Таблиця 4.1 — Опис етапів алгоритму роботи реєстрації та авторизації веб-сервісу

Етапи	Опис
Створення форми авторизації	Створення форми авторизації з полями для електронної пошти та пароля, які зв'язуються з Vue компонентами через v-model. Реалізовано перевірку полів перед відправкою форми.
Перевірка даних користувача	Перед відправкою форми перевіряється заповненість полів, якщо поле не заповнене, виводиться повідомлення про помилку
Створення форми реєстрації	Форма реєстрації містить поля для введення і підтвердження пароля, а також перевірку співпадіння паролів перед відправкою.
Взаємодія з API для реєстрації	Після успішної реєстрації на сервер відправляється запит для створення нового користувача.

Продовження таблиці 4.1 — Опис етапів алгоритму роботи реєстрації та авторизації веб-сервісу

Перевірка електронної пошти	Після реєстрації відправляється запит для перевірки електронної пошти. Якщо підтверджено, користувач перенаправляється на головну панель.
Взаємодія з API для авторизації	Після успішної перевірки даних формується запит до сервера для авторизації користувача через API. Якщо дані правильні, користувач перенаправляється на панель інструментів.

У коді компонента Login та Register, використовуються форми для введення даних користувача, таких як email, пароль та інші атрибути. У компоненті для реєстрації створюється об'єкт formData, який зберігає значення полів форми. Ці дані передаються на сервер для обробки запиту реєстрації чи авторизації.

Після того, як користувач вводить свої дані, наступним етапом є валідація цих даних перед їх відправленням на сервер. Це здійснюється в методах login та register, де перевіряється, чи заповнені всі необхідні поля. Якщо дані некоректні, система генерує відповідні помилки, які потім відображаються в інтерфейсі для коригування користувачем.

Далі відбувається взаємодія з API, що включає надсилання POST-запитів для реєстрації та авторизації користувача. Наприклад, у методі register дані з форми передаються в API для створення нового користувача. Сервер обробляє ці запити, а у разі успіху, система зберігає токен доступу та інформацію про користувача. У разі успішної реєстрації система також ініціює процес відправки електронного листа для верифікації email.

4.3 Розробка асистента технічної підтримки

Інтерфейс асистента є важливою складовою системи технічнлї підтримки, адже він безпосередньо впливає на ефективність взаємодії між користувачем і системою. Важливо, що розроблений інтерфейс адаптований для різних типів

користувачів, включаючи як фахівців, які мають досвід у технічному обслуговуванні пристроїв, так і звичайних користувачів без спеціальних знань у галузі. Це дозволяє кожному користувачу отримувати необхідну інформацію та рекомендації для вирішення проблем.

Особливістю інтерфейсу є те, що він надає можливість чітко бачити процес діагностики пристрою в реальному часі. Користувач може спостерігати за перевіркою окремих компонентів, виявленими симптомами та проміжними результатами. Така структура інтерфейсу забезпечує користувачу зрозумілість та зручність у процесі діагностики, що сприяє ефективнішому вирішенню проблем. Крім того, можливість перегляду історії діагностики дає користувачеві змогу порівнювати результати та спостерігати зміни у стані пристрою протягом часу.

Важливо зазначити, що інтерфейс також інтегрується з системою на основі OpenAI для надання автоматизованих відповідей та рекомендацій. Після того, як користувач вводить своє питання або опис проблеми, система надсилає запит до API OpenAI для отримання відповіді. Отримана відповідь відображається в інтерфейсі, що дозволяє користувачеві отримувати точні рекомендації та рішення щодо усунення несправностей. Архітектура асистанта наведена на рис. 4.1.

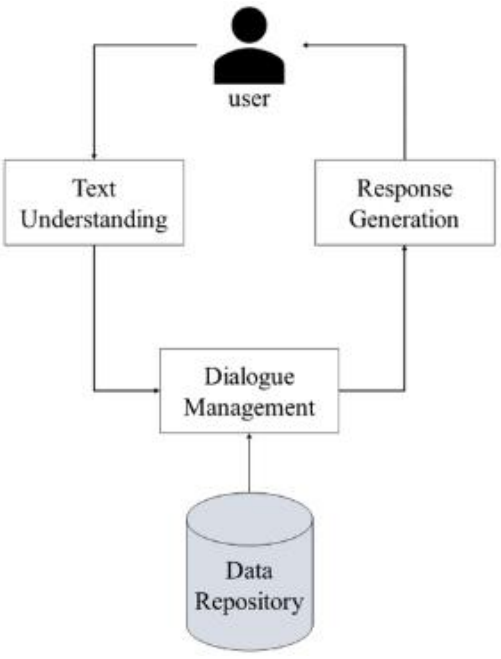

Рисунок 4.1 — Архітектура асистента

Наступним етапом є налаштування інтерфейсу користувача. Для цього використовується бібліотека @chatscopeic, яка містить готові компоненти для створення чатів, такі як MainContainer, ChatContainer, MessageList, Message, MessageInput та TypingIndicator. Ці компоненти дозволяють побудувати основу інтерфейсу для відображення повідомлень, введення тексту користувачем і відображення індикаторів набора тексту.

Лістинг коду 4.1 — Налаштування інтерфейсу

```
<MainContainer>
 <ChatContainer>
  <MessageList
   scrollBehavior="smooth"
   typingIndicator={isTyping ? <TypingIndicator content="Асистент відповідає.." /> :
null}
  >
   {messages.map((message, i) => {
    return <Message key={i} model={message} />
   })}
  </MessageList>
  <MessageInput placeholder="Type message here" onSend={handleSend} />
 </ChatContainer>
</MainContainer>
```

Для запитів користувачів створюється функція handleSend, яка реагує на надсилання повідомлень користувачем. Вона додає нове повідомлення до стану додатку, а потім викликає функцію processMessageToChatGPT, яка перетворює повідомлення користувача в формат, зручний для API OpenAI, і відправляє їх через HTTP запит до сервера OpenAI.

Лістинг коду 4.2 — Виклики запиту до асистента

```
const handleSend = async (message) => {
  const newMessage = {
    message,
    direction: 'outgoing',
    sender: "user"
  };

  const newMessages = [...messages, newMessage];
  setMessages(newMessages);

  setIsTyping(true);
  await processMessageToChatGPT(newMessages);
};
```

У процесі розробки асистента для взаємодії з користувачем, важливим аспектом є правильна обробка введених користувачем повідомлень та їх відправка до сервера, який надає відповіді на запити. У цьому випадку застосовується методика асинхронної обробки запитів, яка дозволяє ефективно взаємодіяти з API OpenAI.

Коли користувач вводить повідомлення в інтерфейс чату, цей процес ініціюється функцією handleSend. Вона отримує текстове повідомлення від користувача та створює об'єкт повідомлення, що містить важливі атрибути, зокрема сам текст повідомлення, напрямок (outgoing для користувача) та відправника. Це повідомлення додається до поточного стану додатку, що забезпечує його відображення в інтерфейсі користувача. Завдяки використанню React hook'у useState, оновлення стану відбувається швидко та зручним способом, що дозволяє динамічно оновлювати інтерфейс.

Після того як повідомлення додано до списку, викликається функція processMessageToChatGPT, яка відповідає за перетворення отриманих повідомлень у формат, який розуміє API OpenAI. Зокрема, функція перетворює кожне повідомлення в об'єкт, що містить два важливі поля: роль відправника та його контент. Якщо відправник — це система або асистент, його роль визначається як assistant, а якщо це користувач — роль стає user. Така структура повідомлень

дозволяє чітко визначати, хто є автором кожного з повідомлень, що важливо для належної взаємодії з моделлю.

Для взаємодії з API OpenAI, запит форматується у вигляді об'єкта apiRequestBody, який включає модель, та список повідомлень, включаючи попередньо задане системне повідомлення, яке формує контекст для подальшої взаємодії. Це системне повідомлення може містити інструкції, які допомагають моделі правильно реагувати на запити, наприклад, пояснення, як відповідати користувачам.

Після цього, здійснюється HTTP POST-запит до API OpenAI. Завдяки асинхронному виконанню запиту через використання async/await, система не блокує інтерфейс під час обробки запиту, що дозволяє забезпечити безперервну взаємодію з користувачем. У разі успішного виконання запиту, отримана відповідь з сервера додається до списку повідомлень, що виводиться у чаті. Якщо ж запит не вдався або виникла помилка, користувач отримує відповідь з повідомленням про помилку, що дозволяє йому зрозуміти причину несправності.

Така архітектура взаємодії, заснована на асинхронних запитах та чіткому формуванні контексту для кожної взаємодії, дозволяє створювати ефективний чат-інтерфейс, який забезпечує швидку та інтуїтивно зрозумілу взаємодію з користувачем.

Для обробки запиту створюємо функцію processMessageToChatGPT, завдяки неї створюється масив повідомлень, що передаються в GPT API. Всі повідомлення формуються з урахуванням ролі (якщо повідомлення від асистента, воно має роль "assistant", якщо від користувача — роль "user"). Ці дані відправляються на сервер через метод POST, де вказується API ключ у заголовках для авторизації.

Лістинг коду 4.3 — Відправка даних асистенту

```
function App() {
  const [messages, setMessages] = useState([
    {
      message: "Привіт, я асистент. Запитуй!",
      sentTime: "just now",
      sender: "Провайдер Апекс"
    }
```

```
]);
const [isTyping, setIsTyping] = useState(false);

const handleSend = async (message) => {
  const newMessage = {
    message,
    direction: 'outgoing',
    sender: "user"
  };

  const newMessages = [...messages, newMessage];
  setMessages(newMessages);

  setIsTyping(true);
  await processMessageToChatGPT(newMessages);
};

async function processMessageToChatGPT(chatMessages) {
  let apiMessages = chatMessages.map((messageObject) => {
    let role = "";
    if (messageObject.sender === "Провайдер Апекс") {
      role = "assistant";
    } else {
      role = "user";
    }
    return { role: role, content: messageObject.message };
  });

  const apiRequestBody = {
    "model": "gpt-4",
    "messages": [
      systemMessage,
      ...apiMessages
    ]
  };

  try {
    const response = await fetch("https://api.openai.com/v1/chat/completions", {
      method: "POST",
      headers: {
        "Authorization": "Bearer " + API_KEY,
        "Content-Type": "application/json"
      },
      body: JSON.stringify(apiRequestBody)
    });

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const data = await response.json();
    console.log(data);
```

```
      setMessages([...chatMessages, {
        message: data.choices[0].message.content,
        sender: "ChatGPT"
      }]);
    } catch (error) {
      console.error('Error:', error);
      setMessages([...chatMessages, {
        message: 'Помилка під час зв*язку з сервером. Спробуйте пізніше.',
        sender: "ChatGPT"
      }]);
    } finally {
      setIsTyping(false);
    }
  }
```

Кожен раз, коли користувач надсилає повідомлення через інтерфейс, його повідомлення додається до масиву поточних повідомлень. На цьому етапі важливо розрізняти, хто є відправником повідомлення: користувач або асистент. Для кожного повідомлення визначається роль, що дозволяє коректно обробляти дані на сервері.

Після того як повідомлення сформовано, вони перетворюються в формат, який розуміє сервер OpenAI. Це включає додавання ролей до кожного повідомлення та підготовку структури запиту для відправки до API. Відправлене запитання має включати модель, а також ряд повідомлень, що складають діалогову історію. Важливим аспектом є також додавання системного повідомлення, яке вказує модельним алгоритмам, як повинна поводитись система.

Далі сформований запит передається на сервер OpenAI. Для цього використовують HTTP POST запит, в якому включаються усі необхідні заголовки для авторизації, зокрема API-ключ, що підтверджує право доступу до сервісу. Це дозволяє забезпечити безпеку обміну даними між сервером і клієнтом.

Якщо сервер успішно обробив запит, він надсилає відповідь, яка містить текстову інформацію або рекомендації. Цей текст передається назад у чат, де він відображається користувачу. Важливо також, що, якщо виникає помилка наприклад, через неправильну авторизацію або збої в роботі сервера, система інформує користувача про проблему, надаючи відповідне повідомлення. Схема циклу запиту та відповіді GPT наведено на рис. 4.5.

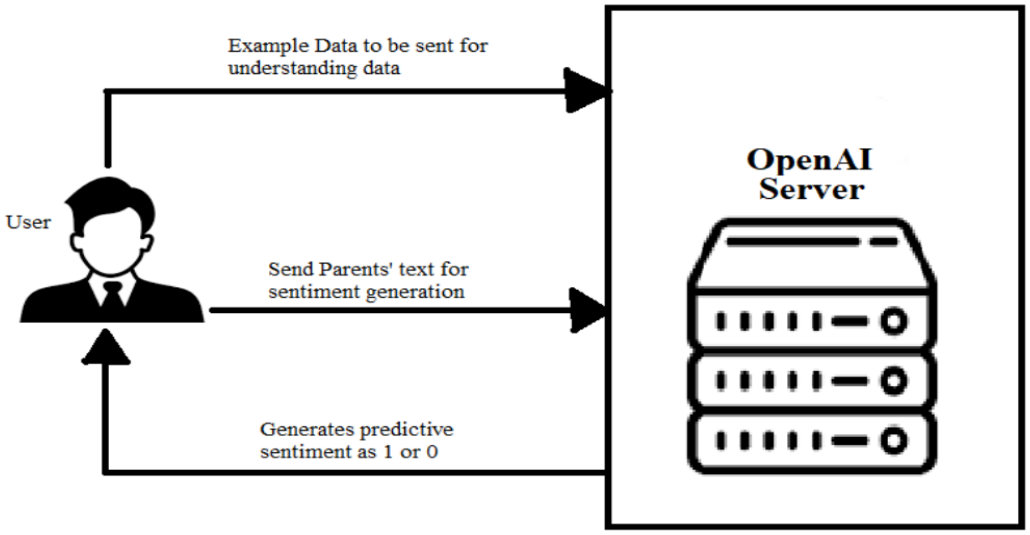

Рисунок 4.2 — Схема циклу запиту та відповіді GPT

Усі ці етапи взаємодії з асистентом супроводжуються візуальними індикаторами для покращення взаємодії. Наприклад, індикатор набору тексту відображає, що асистент працює над відповіддю, і вимикається після отримання результату від сервера.

Цей процес є важливим для забезпечення інтерактивної і плавної комунікації між користувачем і системою, гарантуючи, що відповіді будуть надані вчасно та у відповідному контексті.

Після отримання відповіді від API, дані обробляються, і відповідь додається до стану додатку як нове повідомлення від асистента. Якщо виникає помилка при підключенні до сервера або інша проблема, система відображає повідомлення про помилку.

4.4 Тестування системи технічної підтримки

Для тестування програми на локальному сервері потрібно відкрити проект у середовищі розробки, у Visual Studio Code. Далі в терміналі VS Code або командному рядку потрібно перейти в директорію веб-сервісу та виконати команду npm run dev. Ця команда запускає сервер для вашої програми, і в консолі має з'явитися повідомлення, що сервер запущено, і він доступний, наприклад, за адресою http://localhost:3000.

Тепер ми можемол відкрити браузер і перейти на зазначену адресу, де буде відображатися інтерфейс вашого чату. Користувач може взаємодіяти з програмою, вводячи повідомлення, які відправляються на сервер, і отримувати відповіді через API, зокрема від GPT-4, яке обробляє запит користувача.

Це дозволить тестувати асистента, відправляти запити та перевіряти, як програма реагує на різні введення, а також перевірити коректність інтеграції з OpenAI API. Тестування асистента наведено в рис. 4.6.

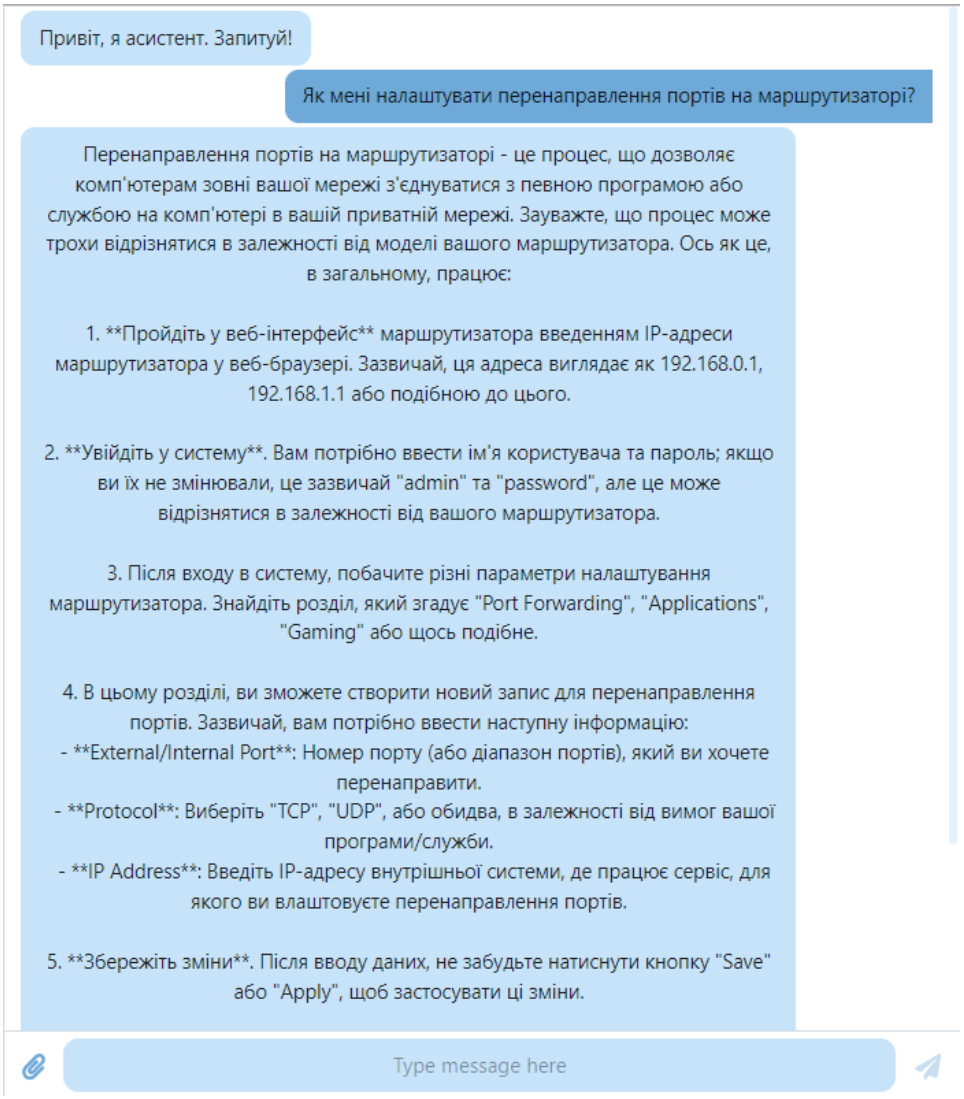

Рисунок 4.3 — Тестування роботи асистента

Тестування показало, що штучний інтелект стабільно обробляє запити, а інтеграція з OpenAI API працює коректно. Всі функції, включаючи відправку

повідомлень, отримання відповідей та відображення їх у чаті, функціонують без збоїв.

Процес інтеграції з OpenAI API став важливою частиною розробки, оскільки він забезпечує функціонування асистента на основі GPT-4, який може надавати автоматизовані відповіді на запитання користувача. Зібрані дані з повідомлень користувача передаються до API, де GPT-4 формує відповідь, яка потім надсилається назад до інтерфейсу користувача. Це дозволяє забезпечити високий рівень інтерактивності та точності відповідей.

Тестування програми підтвердило ефективність роботи всіх компонентів. Програма працює стабільно, відповіді надходять швидко, і чат-бот ефективно справляється із завданнями, надаючи корисні та точні рекомендації для вирішення технічних проблем користувачів. У результаті, веб-сервіс показав свою працездатність, і він може бути використаний для надання технічної підтримки чи допомоги в інших сферах, де необхідна інтерактивна взаємодія з користувачем.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту системи технічної підтримки з використанням штучного інтелекту, яка здатна надавати інтерактивні рекомендації та вирішення технічних проблем користувачів.

Така система може бути застосована в різних галузях, зокрема в технічній підтримці, електронній комерції, освітніх платформах, а також для автоматизації бізнес-процесів.

Аналогами такої розробки можуть бути системи автоматизованої технічної підтримки, такі як Zendesk AI чи IBM Watson Assistant – 45000-55000 тис. грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

	Бали (за 5-ти бальною шкалою)				
Кри-те-рій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продук-кту в реаль-них умовах
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 5.1

Ринкові переваги					
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження табл. 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	3	4
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	4
Експлуатаційні витрати	4	4	3
Ринок збуту	4	3	3
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	3	4	4
Супровідна документація	4	3	4
Сума	43	42	43
Середньоарифметична сума балів	(43+42+43) / 3 = 42,66		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з аналізу, рівень комерційного потенціалу розроблюваного веб-сервісу з інтеграцією штучного інтелекту є високим. Це досягається завдяки використанню сучасної мовної моделі GPT-4, яка забезпечує високу точність і релевантність відповідей на запити користувачів.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t, \quad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ — число робочих днів за місяць, 21 днів;

t – число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

Таблиця 5.4 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	40000	1904,76	35	66666,667
Програміст	36000	1714,29	35	60000,000
Всього				126666,67

5.2.2 Додаткова заробітна плата розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 12 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 12\ \% / 100\ \% \quad (5.2)$$

$$З_д = (126666,67 \cdot 12\ \% / 100\ \%) = 15200,00\ \text{(грн.)}$$

5.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \quad (5.3)$$

$$Н_з = (126666,67 + 15200,00) \cdot 22\ \% / 100\ \% = 31210,67\ \text{(грн.)}$$

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \quad (5.4)$$

де Ц – балансова вартість обладнання, грн.;

Т – термін корисного використання обладнання згідно податкового законодавства, років

$t_{вик}$ – термін використання під час розробки, місяців

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 120000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 1,67 міс.

$$A_{обл} = \frac{20000}{2} \times \frac{1{,}67}{12} = 8333{,}33 \; грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.5. Так як вартість ліцензійної ОС менше 20000 грн, то вона включається в розробку повністю, $B_{нем.ак.}$ = 8700 грн.

Таблиця 5.5 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	120000	2	1,67	8333,333
Офісне обладнання (меблі)	35000	4	1,67	1215,278
Приміщення	1300000	20	1,67	9027,778
Всього				18576,39

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників

(енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,3 кВт;

Ф — фактична кількість годин роботи обладнання, годин.

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,3 \cdot 8 \cdot 35 \cdot 10,42 = 787,752 \text{ (грн.)}$$

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$I_в = 126666,67 * 80\% / 100\% = 101333,3 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з

освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$H_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \quad (5.7)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$H_{нзв}$ = 126666,67 * 130 % / 100 % = 164667 (грн.)

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$B_{заг} = 126666,67+15200,00+31210,67+18576,39+8700+787,75+101333,3+$$
$$+164667 = 467141,47 \text{ грн.}$$

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{B_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η – коефіцієнт, який характеризує етап виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 467141,47 / 0,5 = 934283 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.9)$$

де $\pm\Delta Ц_o$ – зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ – вартість програмного продукту у році до впровадження результатів розробки;

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$.

p – коефіцієнт, який враховує рентабельність продукту;

ϑ – ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 85000 грн. за одиницю, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *1000* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 1000 шт., протягом другого року – на 900 шт., протягом третього року на 800 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*1000 + (85000 + 1000)*1000)* 0{,}8333* 0{,}18) * (1 - 0{,}18) = 10454999{,}582 \text{ грн.}$$

$$\Delta П_2 = (0*1000 + (85000 + 1000)*(1000+900)* 0{,}8333* 0{,}18) * (1 - 0{,}18) = 20098199{,}196 \text{ грн.}$$

$$\Delta П_3 = (0*1000 + (85000 + 1000)*(1000+900+800)* 0{,}8333* 0{,}18) * (1 - 0{,}18) = 28560598{,}858 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 59113797,64 грн.

5.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т – період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,05…0,15;

t – період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (10454999,582/(1+0,1)^1)+(20098199,196/(1+0,1)^2)+(28560598,858/$$
$$/(1+0,1)^3) = 9504545,07 + 16610081,98 + 21458000,64 = 47572627,7 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 934283 = 1868565,90 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП – PV$$

$$E_{абс} = 47572627,7 – 1868565,90 = 45704061,80 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження

результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розра-хувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \qquad (5.13)$$

де $T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{(1 + 45704061{,}80/1868565{,}90} - 1 = 1{,}942$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \qquad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, (5.15)$$

$$T_{ок} = 1 / 1{,}942 = 0{,}51 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,51 роки, то фінансування даної наукової розробки є доцільним.

Економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 934283 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,51 роки.

ВИСНОВКИ

Застосування мовних моделей у якості асистентів технічної підтримки у веб-сервісах суттєво змінює підхід до взаємодії з клієнтами. GPT-помічники здатні обробляти запити в реальному часі, надаючи детальні та персоналізовані відповіді, що підвищує загальну ефективність підтримки користувачів.

GPT-асистенти можуть інтегруватися з іншими системами компанії, такими як бази знань, CRM або системи управління замовленнями, що дозволяє їм надавати більш комплексну та контекстну підтримку. Це сприяє швидшому доступу до необхідної інформації та оперативнішому вирішенню запитів клієнтів. Вони також здатні обробляти великий обсяг запитів одночасно, що особливо важливо під час пікових навантажень.

Інтеграція таких асистентів, які використовують OpenAI API, дозволяє компаніям автоматизувати процеси підтримки, знижувати навантаження на операторів і підвищувати задоволеність клієнтів. Завдяки можливості адаптувати тон і стиль відповіді через системні повідомлення, такі рішення можуть бути налаштовані відповідно до специфічних потреб бізнесу, забезпечуючи більш теплу та професійну взаємодію. Крім того, GPT-асистенти можуть навчатися на прикладах і швидко адаптуватися до змін у запитах, що робить їх гнучкими і здатними до подальшого вдосконалення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методи та системи штучного інтелекту [Електронний ресурс]. – Режим доступу: http://elar.tsatu.edu.ua/bitstream/123456789/15462/1/5_lubko_metody_2019.pdf

2. Системи навчання з елементами штучного інтелекту [Електронний ресурс]. Режим доступу: http://repository.hneu.edu.ua/bitstream/123456789/6219/1/Mono_2009_ukr.pdf

3. Логічне і функціональне програмування. [Електронний ресурс]. – Режим доступу: https://ep3.nuwm.edu.ua/15918/1/Логічне%20і%20функціональне%20програмування.pdf

4. Сутність і проблематика штучного інтелекту [Електронний ресурс]. – Режим доступу: http://dspace.onua.edu.ua/bitstream/handle/11300/15083/Порохова%20О.%20Є.%20Сутність%20і%20проблематика%20штучного%20інтелекту.pdf?sequence=1&isAllowed=y

5. Douglas Crockford: How JavaScript Works [Електронний ресурс]. – Режим доступу: https://balka-book.com/ua/javascript_jquery_dojo313/how_javascript_works-89524

6. Ethan Brown: Learning JavaScript: JavaScript Essentials for Modern Application Development 3rd Edition [Електронний ресурс]. – Режим доступу: https://balka-book.com/ua/javascript_jquery_dojo-313/learning_javascript_javascript_essentials_for_modern_application_development_3rd_edition-67168

7. Callum Macrae: Vue.js: Up and Running: Building Accessible and Performant Web Apps 1st Edition [Електронний ресурс]. – Режим доступу: https://balka-book.com/ua/web-programmirovanie-14/vue_js_up_and_running_building_accessible_and_performant_web_apps_1st_edition-

66978?utm_source=google&utm_medium=cpc&utm_campaign=18680074526&utm_content=&utm_term=&gad_source=1&gclid=Cj0KCQiA6Ou5BhCrARIsAPoTxrB5S8hyTwXJNxUz-AGYF6F1KLq-Ma1MB5iigmYNGozTXvh7jWiF7ZYaAtpIEALw_wcB

7. Damiano Fusco: Large Scale Apps with Vue, Vite and TypeScript [Електронний ресурс]. – Режим доступу: https://www.amazon.com/Large-Scale-Apps-Vite-TypeScript-ebook/dp/B0CJDRYF5Y

8. Tom Taulli: Ai-assisted Programming: Better Planning, Coding, Testing, and Deployment 1st Edition [Електронний ресурс]. – Режим доступу: https://balka-book.com/iskusstvennyiy_intelekt_neyronnyie_seti-19/aiassistedprogrammingbetterplanningcodingtestinganddeployment1stedition-275376?utm_source=google&utm_medium=cpc&utm_campaign=18680074526&utm_content=&utm_term=&gad_source=1&gclid=Cj0KCQiA6Ou5BhCrARIsAPoTxrBfOuzNKMI1ZlDdSEnwwkOrfTQFE6ggetXc9N9com0rJAhUo3b23CcaAotxEALw_wcB&lang=ua

9. Lewis Tanstall: Natural Language Processing with Transformers [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/natural-language-processing/9781098136789/

10. Aston Zhang: Dive into Deep Learning [Електронний ресурс]. – Режим доступу: https://www.amazon.com/Dive-into-Learning-Aston-Zhang/dp/1009389432

11. Chenguang Zhu: Machine Reading Comprehension: Algorithms and Practice [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/book/9780323901185/machine-reading-comprehension

12. Nilson Jacques: Jump Start Vue.js [Електронний ресурс]. – Режим доступу: https://www.amazon.com/Jump-Start-Vue-js-Nilson-Jacques/dp/1925836096

13. NcuSun Yang: Vue. JS Framework Design and Implementation [Електронний ресурс]. – Режим доступу: https://link.springer.com/book/10.1007/978-981-99-4947-2

14. Olga Filipova: Learning Vue.js 2 [Електронний ресурс]. – Режим доступу: https://www.amazon.com/Learning-Vue-js-amazingreactiveapplications/dp/1786469944

15. Mike Street: Complete Vue. js 2 Web Development [Електронний ресурс]. – Режим доступу: https://ridit.com.ua/uk/p/1597159622-complete-vue-js-2-web-development-practical-guide-to-building-end-to-end-web-development-solutions-with-vue-js-2/

16. Melanie Mitchell: Artificial Intelligence: A Guide for Thinking Humans [Електронний ресурс]. – Режим доступу: https://www.amazon.com/Artificial-IntelligenceGuideThinkingHumans/dp/0374257833

17. Clifford Clipover: Artificial Intelligence: An Illustrated History: From Medieval Robots to Neural Networks [Електронний ресурс]. – Режим доступу: https://www.amazon.com/ArtificialIntelligenceIllustratedMedievalHistories/dp/1454933593

18. Valentina Alto: Modern Generative AI with ChatGPT and OpenAI Models: [Електронний ресурс]. – Режим доступу: https://www.amazon.com/Modern-Generative-ChatGPTOpenAIModels/dp/1805123335

19. Oswald Campesato: GPT-4 For Developers [Електронний ресурс]. – Режим доступу: https://www.amazon.com/GPT-4-Developers-MLI-Generative-AI/dp/1501522485

20. Douglas Barry: Web Services, Service-Oriented Architectures, and Cloud Computing [Електронний ресурс]. – Режим доступу: https://www.amazon.com/ServicesServiceOrientedArchitecturesCloudComputing/dp/0123983576

21. Алгаш О.М.; Обертюх М.Р.: Аналіз використання штучного інтелекту у веб-сервісах технічної підтримки. Факультет інформаційних технологій та комп'ютерної інженерії [Електронний ресурс]. – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/22863/18861

ДОДАТОК А

Технічне завдання Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д.

_____

" ____" _____2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

АІ ВЕБ-СЕРВІС З НАДАННЯ ТЕХНІЧНОЇ ПІДТРИМКИ

08-54.КМКР.025.00.000 ПЗ

Керівник роботи ст. викл. каф. ОТ
_____Обертюх М.Р.

Студент групи 2КІ–23м
_____Алгаш О.М.

ВНТУ 2024

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність роботи обумовлена необхідністю покращення роботи систем технічної підтримки за допомогою інтеграції штучного інтелекту.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Мета роботи — покращення автономності та швидкості надання послуг у веб-сервісах за рахунок інтеграції штучного інтелекту.

2.2 Призначення розробки — створення веб-сервісу з вбудованим асистентом на базі моделі штучного інтелекту.

3 Вихідні дані для виконання МКР

3.1 Призначення системи — покращення надання послуг за допомогою інтеграції моделі штучного інтелекту.

3.2 Організація — інтеграція з хмарними сервісами.

3.3 Архітектура — розподілена з можливістю легкого масштабування.

4 Вимоги до виконання МКР

4.1 Провести сучасних технологій розробки веб-сервісів та інтеграції штучного інтелекту;

4.2 Визначити архітектуру взаємодії користувача з штучним інтелектом веб-сервісу.

4.3 Розробити веб-сервіс на базі штучного інтелекту.

4.4 Оцінити комерційний потенціал розробки.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз сучасних тенденцій, архітектур та технологій штучного інтелекту			Вступ, Розділ 1
2	Обгрунтування вибору фреймворків та підхід до розробки			Розділ 2
3	Теорія та вибір штучного інтелекту			Розділ 3
4	Практична реалізація та тестування			Розділ 4
5	Економічна частина			Розділ 5
6	Оформлення пояснювальної записки, графічного матеріалу і презентації			ПЗ, графічний матеріал і презентація
7	Підготовка підпис супроводжуючих документів, нормоконтроль та тест на плагіат			Оформленні документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

Вимоги до оформлювання та порядок виконання МКР

7.1 При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104–2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп'ютерна інженерія»;

— документи на які посилаються у вище вказаних.

7.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21

ДОДАТОК Б

Схема циклу запиту та відповіді GPT

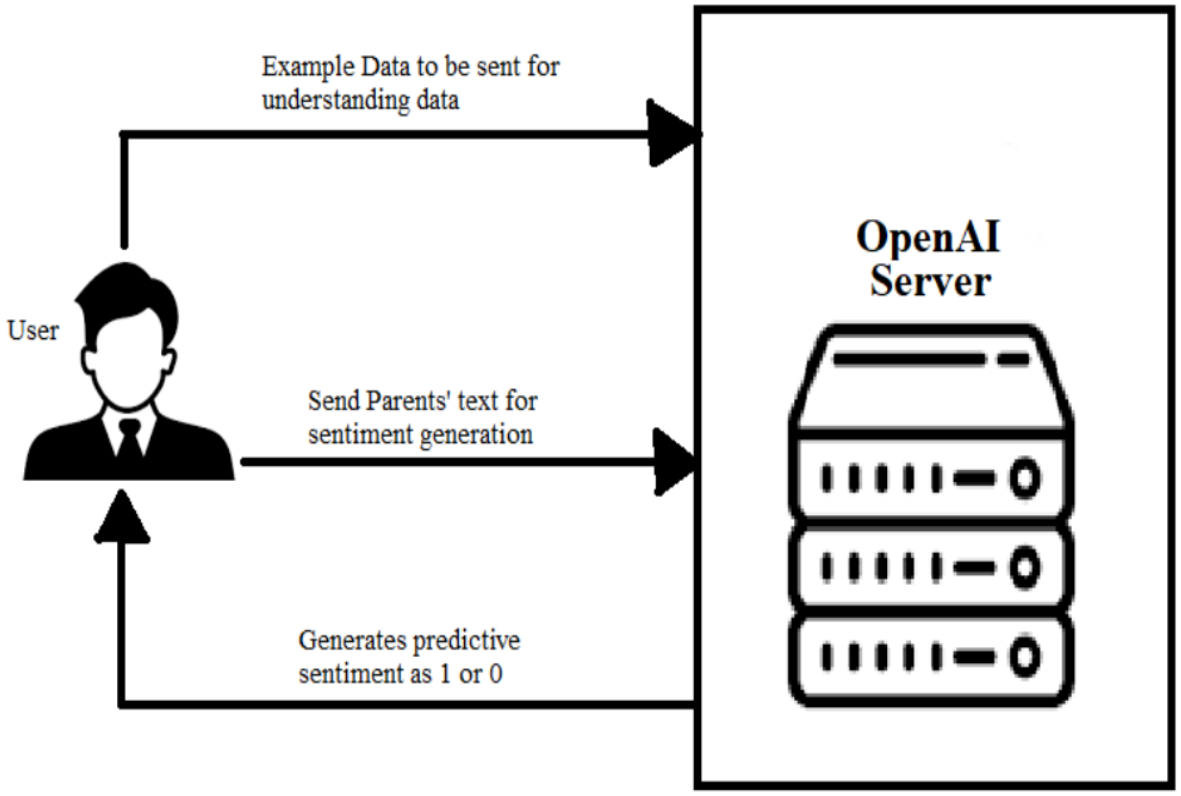

Рисунок Б.1 — Схема циклу запиту та відповіді GPT

ДОДАТОК В

Тестування роботи асистента

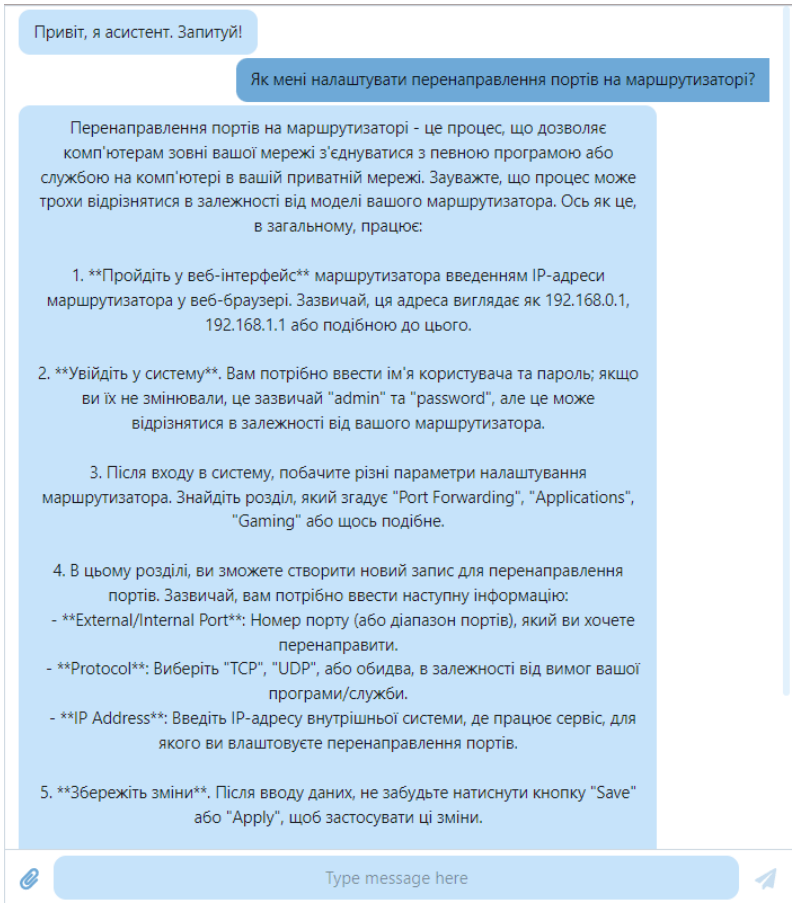

Рисунок В.1 — Тестування роботи асистента

ДОДАТОК Г

Лістинг файлу loginForm.vue

```
<template>
  <div class="flex min-h-full items-center justify-center py-12 px-4 sm:px-6 lg:px-8">
    <div class="w-full max-w-md space-y-8">
      <div>
        <h2 class="mt-6 text-center text-3xl font-bold tracking-tight text-gray-900">Sign in to your account</h2>
        <p class="mt-2 text-center text-sm text-gray-600">
          Or
          {{ ' ' }}
          <router-link
            :to="{name:'Register'}"
            class="font-medium text-indigo-600 hover:text-indigo-500">
            Register for an account
          </router-link>
        </p>
      </div>
      <form class="mt-8 space-y-6" @submit.prevent="login">
        <div class="-space-y-px rounded-md shadow-sm">
          <div>
            <label for="email-address" class="sr-only">Email address</label>
            <input
              id="email-address"
              type="email"
              v-model="formData.email"
              autocomplete="email"
              required="true"
              class="relative block w-full appearance-none rounded-none rounded-t-md border border-gray-300 px-3 py-2 text-gray-900 placeholder-gray-500 focus:z-10 focus:border-indigo-500 focus:outline-none focus:ring-indigo-500 sm:text-sm"
```

```
            :class="[errors.length>0 && !formData.email ? 'empty-field' : '']"
            placeholder="Email address"
          />
        </div>
        <div>
          <label for="password" class="sr-only">Password</label>
          <input
            id="password"
            type="password"
            autocomplete="current-password"
            v-model="formData.password"
            required="true"
            class="relative block w-full appearance-none rounded-none rounded-b-md border border-gray-300 px-3 py-2 text-gray-900 placeholder-gray-500 focus:z-10 focus:border-indigo-500 focus:outline-none focus:ring-indigo-500 sm:text-sm"
            :class="[errors.length>0 && !formData.password ? 'empty-field' : '']"
            placeholder="Password"
          />
        </div>
      </div>
      <div class="flex items-center justify-between">
        <div class="flex items-center">
          <input
            id="remember"
            v-model="formData.remember"
            type="checkbox"
            class="h-4 w-4 rounded border-gray-300 text-indigo-600 focus:ring-indigo-500"
          />
          <label for="remember" class="ml-2 block text-sm text-gray-900">Remember me</label>
```

```
        </div>
        <div class="text-sm">
          <router-link
            :to="{name: 'ForgotPassword'}"
            class="font-medium text-indigo-600 hover:text-indigo-500"
          >
            Forgot your password?
          </router-link>
        </div>
      </div>
      <div>
        <button
          v-if="!attempt"
          type="submit"
          class="group relative flex w-full justify-center rounded-md border border-transparent bg-indigo-600 py-2 px-4 text-sm font-medium text-white hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:ring-offset-2"
        >
          Sign in
        </button>
        <button
          v-else
          disabled="true"
          type="button"
          class="group relative flex w-full justify-center rounded-md border border-transparent bg-indigo-300 py-2 px-4 text-sm font-medium text-white"
        >
          <span class="absolute inset-y-0 left-0 flex items-center pl-3">
            <LockClosedIcon class="h-5 w-5 text-indigo-400" aria-hidden="true" />
          </span>
          Sign in
```

```
          </button>
        </div>
      </form>
    </div>
  </div>
</template>
<script setup>
  import { ref, computed } from 'vue'
  import store from '@/store'
  import Auth from '@/services/Auth'
  import { LockClosedIcon } from '@iconsolin
  const formData = ref( {
    email: null,
    password: null,
    remember: false
  } )
  const attempt = computed( () => store.getters['system/getAttempt'] )
  const errors = ref( [] )
  const login = () => {
    errors.value = []
    if( ! formData.value.email ) errors.value.push( 'Fill in email' )
    if( ! formData.value.password ) errors.value.push( 'Fill in password' )
    if( ! formData.value.email || ! formData.value.password ) return
    store.commit( {
      type: 'system/SET_ATTEMPT',
      attempt: true
    } )
    Auth.login( formData.value )
  }
</script>
```

ДОДАТОК Д

Лістинг файлу App.js

```
const API_KEY = “”
const systemMessage = {
  "role": "system", "content": "Explain things like you're talking to a client of Provider, etc."
};
function App() {
  const [messages, setMessages] = useState([
    {
      message: "Привіт, я асистент. Запитуй!",
      sentTime: "just now",
      sender: "Провайдер Апекс"
    }
  ]);
  const [isTyping, setIsTyping] = useState(false);
  const handleSend = async (message) => {
    const newMessage = {
      message,
      direction: 'outgoing',
      sender: "user"
    };
    const newMessages = [...messages, newMessage];
    setMessages(newMessages);
    setIsTyping(true);
    await processMessageToChatGPT(newMessages);
  };
  async function processMessageToChatGPT(chatMessages) {
    let apiMessages = chatMessages.map((messageObject) => {
      let role = "";
```

```
  if (messageObject.sender === "Провайдер Апекс") {
    role = "assistant";
  } else {
    role = "user";
  }
  return { role: role, content: messageObject.message };
});
const apiRequestBody = {
  "model": "gpt-4",
  "messages": [
    systemMessage,
    ...apiMessages
  ]
};
try {
  const response = await fetch("https://api.openai.com/v1/chat/completions", {
    method: "POST",
    headers: {
      "Authorization": "Bearer " + API_KEY,
      "Content-Type": "application/json"
    },
    body: JSON.stringify(apiRequestBody)
  });
  if (!response.ok) {
    throw new Error(`HTTP error! Status: ${response.status}`);
  }
  const data = await response.json();
  console.log(data);
  setMessages([...chatMessages, {
    message: data.choices[0].message.content,
```

```
        sender: "ChatGPT"
      }]);
    } catch (error) {
      console.error('Error:', error);
      setMessages([...chatMessages, {
        message: 'Помилка під час зв*язку з сервером. Спробуйте пізніше.',
        sender: "ChatGPT"
      }]);
    } finally {
      setIsTyping(false);
    }
  }
  return (
    <div className="App">
      <div style={{ position: "relative", height: "800px", width: "700px" }}>
        <MainContainer>
          <ChatContainer>
            <MessageList
              scrollBehavior="smooth"
              typingIndicator={isTyping ? <TypingIndicator content="Асистент відповідає.." /> : null}
            >
              {messages.map((message, i) => {
                return <Message key={i} model={message} />
              })}
            </MessageList>
            <MessageInput placeholder="Type message here" onSend={handleSend} />
          </ChatContainer>
        </MainContainer>
      </div>
```

```
    </div>
  );
}
export default App;
```

ДОДАТОК Е

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:_____ AI Веб-сервіс з надання технічної підтримки _____
Тип роботи:_____Магістерська кваліфікаційна робота_____
(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))
Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет (інститут), навчальна група)
Керівник_____ Обертюх М.Р., phd., ст.викл.каф. ОТ_____
(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	99
Загальнасхожість	1

Аналіз звіту подібності (відмітитипотрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор_____ Алгаш_О.М._____
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____
_____
_____
_____

Особа, відповідальна за перевірку_____ _Захарченко С.М__
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)