Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

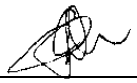
Кафедра Медіасистем та технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

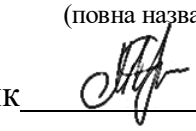
Дослідження процесу розробки веб-сайтів з використанням ШІ-інструментів
(тема)

Виконав:
здобувач 2 року навчання,
групи ТЕМВм-23-1



Сергєєв Д.С.
(прізвище, ініціали)

Спеціальність 186 Видавництво та поліграфія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма
Технології електронних мультимедійних видань
(повна назва освітньої програми)

Керівник  доц. Табакова І.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри МСТ _____ Дейнеко Ж.В.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Медіасистеми та технології
Рівень вищої освіти другий (магістерський)
Спеціальність 186 Видавництво та поліграфія
Тип програми Освітньо-професійна
Освітня програма Технології електронних мультимедійних видань
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри МСТ_____
(підпис)
« 18 » листопада 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві *Сергєєву Дмитру Сергійовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження процесу розробки веб-сайтів з використанням ШІ-інструментів*

Затверджена наказом по університету від 8 листопада 2024 р. 1191 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 22 січня 2025 р.

3. Вихідні дані до роботи *Штучний інтелект, інструменти для розробки програмного забезпечення, GitHub Copilot, OpenAI API, ChatGPT, Claude AI, асистенти на базі штучного інтелекту, промпт-інжиніринг, чат-інтерфейс, генерація програмного коду, Angular, ASP.NET Core, Microsoft SQL Server, автоматизація розробки, оптимізація коду, тестування програмного забезпечення, технічна документація, веб-технології, інтегровані середовища розробки, мови програмування, фреймворки, бази даних, клієнт-серверна архітектура, API, автентифікація, маршрутизація, користувацький інтерфейс*

4. Перелік питань, що потрібно опрацювати в роботі *Вступ; Огляд літератури за темою; Постановка задачі дослідження; Теоретична частина; Дослідження впливу ШІ-інтрументів на процес розробки; Експериментальна частина; Розробка рекомендацій по вибору; Економічна частина; Висновки*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) *Титульний слайд; Актуальність дослідження; Мета дослідження; Задачі досягнення мети; Об'єкт, предмет дослідження; Гіпотеза дослідження; Огляд сучасних ШІ-інструментів для розробки; Методика дослідження; Етапи дослідження; Архітектура розробленого веб-застосунку; Функціональність веб-застосунку; Спеціалізовані чат-інтерфейси; Інтеграція з OpenAI API; Система автентифікації; Демонстрація результату розробки застосунку; Результати тестування; Практична цінність результатів; Порівняння швидкості розробки; Економічна частина; Рекомендації щодо використання ШІ-інструментів; Акт впровадження; Висновки*

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	доц. Табакова І.С.		22.012025
Економічна частина	ас. Помогалова Н.В.		19.01.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Огляд літератури за темою та підготовка теоретичної	25.11.2024	виконано
2	Розробка методики оцінки впливу ШІ-інструментів	28.11.2024	виконано
3	Проєктування архітектури веб-застосунку	02.12.2024	виконано
4	Розробка клієнтської частини веб-застосунку	06.12.2024	виконано
5	Розробка серверної частини та бази даних	16.12.2024	виконано
6	Інтеграція з OpenAI API та тестування	24.12.2024	виконано
7	Розробка рекомендацій щодо впровадження	31.12.2024	виконано
8	Економічні розрахунки	09.01.2025	виконано
9	Оформлення пояснювальної записки	12.01.2025	виконано
10	Оформлення графічної частини	16.01.2025	виконано

Дата видачі завдання 18 листопада 2024 р.

Здобувач _____ Сергєєв Д.С.

(підпис)

Керівник роботи _____ доц. Табакова І.С.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 70 стор., 14 рис., 12 табл., 2 дод., 9 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, ВЕБ-РОЗРОБКА, ANGULAR, ASP.NET CORE, MICROSOFT SQL SERVER, OPENAI API, GITHUB COPILOT, АВТОДОПОВНЕННЯ КОДУ, ПРОМПТ-ІНЖИНІРИНГ, ГЕНЕРАТИВНИЙ ШІ, ЧАТ-ІНТЕРФЕЙС, АВТЕНТИФІКАЦІЯ, ЗБЕРІГАННЯ КОНТЕКСТУ, ПРОГРАМНИЙ КОД, ВЗАЄМОДІЯ КЛІЄНТ-СЕРВЕР.

Мета роботи полягає у дослідженні процесу розробки веб-сайтів з використанням інструментів на базі штучного інтелекту, а також проєктуванні та розробці веб-платформи для демонстрації можливостей застосування ШІ у веб-розробці.

У роботі проведено аналіз сучасного стану розвитку штучного інтелекту та його впливу на веб-розробку, досліджено популярні ШІ-інструменти для програмування, такі як GitHub Copilot, ChatGPT, Claude, та проаналізовано їх функціональність у порівнянні з традиційними засобами автодоповнення коду.

В рамках практичної частини розроблено веб-застосунок на базі Angular та ASP.NET Core для демонстрації можливостей ШІ у веб-розробці. Застосунок надає доступ до теоретичних матеріалів та спеціалізовані чат-інтерфейси на базі OpenAI API для допомоги з різними аспектами розробки: від написання коду до створення документації. Реалізована система автентифікації забезпечує розширені можливості взаємодії з ШІ, включаючи збереження контексту діалогів, що робить застосунок ефективним практичним інструментом для розробників.

ABSTRACT

The explanatory note of the qualification work contains: 70 p., 14 pic., 12 tab., 2 app., 9 sources.

ARTIFICIAL INTELLIGENCE, WEB DEVELOPMENT, ANGULAR, ASP.NET CORE, MICROSOFT SQL SERVER, OPENAI API, GITHUB COPILOT, CODE COMPLETION, PROMPT ENGINEERING, GENERATIVE CSS, CHAT INTERFACE, AUTHENTICATION, CONTEXT STORAGE, PROGRAM CODE, CLIENT-SERVER INTERACTION.

The purpose of this work is to investigate the process of web application development using artificial intelligence-based tools, as well as to design and develop a web platform demonstrating the capabilities of AI application in web development.

The work presents an analysis of the current state of artificial intelligence development and its impact on web development. Popular AI programming tools such as GitHub Copilot, ChatGPT and Claude have been examined and their functionality has been analyzed in comparison with traditional code completion tools.

The practical component involves the development of a web application based on Angular and ASP.NET Core to demonstrate AI capabilities in web development. The application provides access to theoretical materials and specialized chat interfaces powered by OpenAI API to assist with various development aspects, from code writing to documentation creation. The implemented authentication system provides enhanced AI interaction capabilities, including dialogue context retention, making the application an effective practical tool for developers.

ЗМІСТ

С.

ВСТУП

У сучасному світі розробки програмного забезпечення відбувається стрімка інтеграція технологій штучного інтелекту (ШІ) в процеси створення програмних продуктів. Особливо помітним є вплив ШІ на веб-розробку, де інтелектуальні інструменти все частіше стають невід'ємною частиною робочого процесу розробників. Поява таких інструментів як GitHub Copilot, Amazon CodeWhisperer та інших асистентів на базі штучного інтелекту створює нову парадигму розробки, де машинний інтелект активно доповнює людський досвід та експертизу.

Актуальність дослідження зумовлена декількома ключовими факторами. По-перше, стрімке зростання складності веб-проєктів вимагає нових підходів до оптимізації процесу розробки. По-друге, постійно зростаючий попит на веб-розробку створює необхідність у прискоренні процесів написання та тестування коду без втрати якості. По-третє, інтеграція ШІ-інструментів у процес розробки є відносно новим явищем, яке потребує детального вивчення для розуміння їх реального впливу на продуктивність та якість розробки.

Об'єктом дослідження є процес розробки веб-застосунків з використанням сучасних інструментів на базі штучного інтелекту.

Предметом дослідження є методи та засоби інтеграції ШІ-інструментів у процес веб-розробки, їх вплив на швидкість та якість створення програмних продуктів.

Метою роботи є дослідження впливу різних ШІ-інструментів на процес веб-розробки та створення веб-платформи для демонстрації можливостей застосування штучного інтелекту в розробці програмного забезпечення.

Гіпотеза: «Інтеграція спеціалізованих ШІ-інструментів у процес веб-розробки може значно підвищити ефективність створення програмного забезпечення».

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– провести аналіз сучасного стану розвитку штучного інтелекту та його впливу на процес веб-розробки;

– дослідити та порівняти ефективність різних ШІ-асистентів (GitHub Copilot, Claude AI) у розробці програмних компонентів;

– розробити веб-застосунок для демонстрації можливостей ШІ у веб-розробці з використанням сучасного стеку технологій (Angular, ASP.NET Core, MS SQL Server);

– реалізувати інтеграцію з OpenAI API для створення спеціалізованих чат-інтерфейсів з різними аспектами веб-розробки;

– провести порівняльний аналіз швидкості та якості розробки при використанні різних підходів та інструментів.

Методи дослідження базуються на системному підході до аналізу процесів розробки та включають: порівняльний аналіз, експертне оцінювання, експериментальне дослідження, методи об'єктно-орієнтованого проєктування та розробки програмного забезпечення. Для забезпечення об'єктивності результатів було сформовано експертну групу з п'яти фахівців у сфері веб-розробки, які проводили оцінку ефективності різних підходів за встановленими критеріями.

Наукова новизна роботи має багатоаспектний характер та визначається кількома ключовими факторами. Перш за все, проведено комплексне експертне дослідження впливу сучасних ШІ-інструментів на процес веб-розробки. На відміну від попередніх досліджень, які часто фокусувались на окремих аспектах застосування ШІ в програмуванні, дана робота пропонує системний підхід до аналізу та порівняння різних ШІ-інструментів у контексті реальної розробки веб-застосунків, підкріплений кількісними показниками ефективності, отриманими в ході експертного оцінювання.

Особлива наукова цінність даної роботи полягає в комплексному дослідженні впливу сучасних ШІ-інструментів на процес веб-розробки та створенні методики їх ефективної інтеграції в робочий процес розробника. На

відміну від попередніх досліджень, які часто фокусувались на окремих аспектах застосування ШІ в програмуванні, дана робота пропонує системний підхід до аналізу та порівняння різних ШІ-інструментів у контексті реальної розробки веб-застосунків. Крім того, створена в рамках дослідження веб-платформа представляє собою унікальне рішення, що поєднує в собі як теоретичний базис для розуміння ШІ-інструментів, так і практичний інструментарій для їх безпосереднього застосування в розробці.

Практична цінність роботи полягає у створенні функціонального веб-застосунку та розробці науково обґрунтованих рекомендацій щодо впровадження ШІ-інструментів у процеси веб-розробки. Проведене експертне дослідження дозволило отримати конкретні метрики підвищення ефективності розробки при використанні різних ШІ-інструментів, що може бути безпосередньо використано при плануванні та організації процесів розробки програмного забезпечення.

На захист виносяться наступні положення:

– результати аналізу впливу ШІ-інструментів на ефективність веб-розробки;

– підходи до розробки з використанням ШІ;

– розробка веб-застосунок для демонстрації можливостей ШІ у веб-розробці з використанням сучасного стеку технологій (Angular, ASP.NET Core, MS SQL Server);

– архітектурні рішення та реалізація веб-застосунку з інтеграцією ШІ-компонентів;

– результати експериментального дослідження ефективності використання різних ШІ-інструментів у процесі розробки.

1 ОГЛЯД ЛІТЕРАТУРИ ЗА ТЕМОЮ

1.1 Аналіз досліджень інтеграції штучного інтелекту в розробку програмного забезпечення

Інтеграція штучного інтелекту в процеси розробки програмного забезпечення є однією з найбільш значущих тенденцій останніх років у сфері інформаційних технологій. Цей напрямок активно розвивається завдяки появі потужних мовних моделей та систем машинного навчання, які здатні розуміти та генерувати програмний код.

Еволюція використання штучного інтелекту в розробці програмного забезпечення пройшла декілька ключових етапів. Спочатку ШІ-системи використовувалися переважно для статичного аналізу коду та виявлення простих помилок. З розвитком технологій з'явилися більш складні системи, здатні надавати контекстні підказки та автодоповнення коду. Сучасний етап характеризується появою генеративних моделей, які можуть створювати функціональні фрагменти коду на основі природномовного опису.

Особливу увагу привертають системи автоматичного доповнення коду, які стали стандартним інструментом у сучасних середовищах розробки. Ці системи значно прискорюють процес написання коду, зменшують кількість помилок та допомагають дотримуватися єдиного стилю кодування. Проте їх ефективність часто обмежується контекстом конкретного проєкту та якістю навчальних даних. Важливим напрямком розвитку є створення спеціалізованих ШІ-асистентів для різних аспектів розробки. Наприклад, існують системи для автоматизації тестування, оптимізації продуктивності коду, генерації документації та рефакторингу. Кожен з цих інструментів має свої особливості та обмеження, що впливає на можливості їх практичного застосування.

У контексті веб-розробки інтеграція ШІ набуває особливого значення через складність та різноманітність використовуваних технологій. Сучасні

веб-застосунки часто включають складні клієнтські та серверні компоненти, роботу з різними API та базами даних, що створює широке поле для застосування ШІ-інструментів.

Аналіз доступних джерел показує, що існують певні обмеження та проблеми у використанні ШІ в розробці програмного забезпечення. Зокрема, це питання якості згенерованого коду, безпеки, конфіденційності даних та необхідності валідації результатів роботи ШІ-систем. Також важливим аспектом є навчання розробників ефективному використанню таких інструментів.

Незважаючи на значний прогрес, багато аспектів інтеграції ШІ в процеси розробки залишаються недостатньо дослідженими. Особливо це стосується питань комплексного використання різних ШІ-інструментів, їх впливу на якість кінцевого продукту та методології ефективної інтеграції в існуючі процеси розробки.

Окремого розгляду потребує питання впливу ШІ-інструментів на продуктивність розробників та якість створюваного програмного забезпечення. Хоча загальна тенденція свідчить про позитивний вплив, конкретні метрики та методології оцінки такого впливу ще потребують детального вивчення та систематизації.

Аналіз наявної інформації дозволяє стверджувати, що інтеграція ШІ в процеси розробки програмного забезпечення є перспективним напрямком, який активно розвивається. Проте існує потреба в більш глибокому дослідженні методів ефективного використання ШІ-інструментів у веб-розробці, особливо в контексті створення комплексних веб-застосунків з використанням сучасних технологій та фреймворків.

1.2 Критичний аналіз сучасних підходів до автоматизації веб-розробки

Сучасна веб-розробка характеризується високим рівнем складності та різноманітністю використовуваних технологій, що створює потребу в ефективних інструментах автоматизації. Традиційні підходи до

автоматизації, такі як використання шаблонів коду та готових компонентів, поступово доповнюються та замінюються більш складними рішеннями на базі штучного інтелекту. Традиційна автоматизація веб-розробки базується на використанні систем контролю версій, систем збірки проєктів та інструментів тестування. Ці інструменти добре зарекомендували себе для вирішення базових задач автоматизації, проте вони мають обмежені можливості щодо оптимізації самого процесу розробки. Вони не здатні адаптуватися до змін у вимогах проєкту та не можуть запропонувати інтелектуальні рішення для покращення коду.

Поява інструментів на базі штучного інтелекту відкрила нові можливості для автоматизації. Сучасні ШІ-асистенти здатні не лише генерувати код, але й аналізувати його якість, пропонувати оптимізації та виявляти потенційні проблеми. Проте використання таких інструментів також має свої обмеження та ризики. Зокрема, згенерований код може містити помилки або не відповідати специфічним вимогам проєкту.

Особливої уваги заслуговує автоматизація фронтенд-розробки. Сучасні фреймворки, такі як Angular, React та Vue.js, надають потужні інструменти для створення компонентів користувацького інтерфейсу. Інтеграція ШІ в ці процеси дозволяє автоматизувати створення типових компонентів, проте часто виникають проблеми з адаптацією згенерованого коду до специфічних вимог дизайну та функціональності.

У сфері бекенд-розробки автоматизація за допомогою ШІ дозволяє прискорити створення API та роботу з базами даних. Проте тут особливо гостро постають питання безпеки та оптимізації продуктивності. Згенерований код може містити вразливості або бути недостатньо оптимізованим для роботи з великими обсягами даних.

Важливим аспектом є інтеграція різних інструментів автоматизації в єдиний процес розробки. Сучасні IDE намагаються забезпечити безшовну інтеграцію різних ШІ-асистентів, проте часто виникають проблеми з сумісністю та узгодженістю їх роботи. Це створює додаткові складнощі для розробників та може знижувати ефективність процесу розробки.

Аналіз існуючих рішень показує, що найбільш перспективним є комбінований підхід, який поєднує традиційні методи автоматизації з сучасними ШІ-інструментами. Такий підхід дозволяє використовувати переваги обох типів інструментів, мінімізуючи їх недоліки. Проте реалізація такого підходу вимагає ретельного планування та розробки відповідних методологій.

Окремого розгляду потребує питання моніторингу та оцінки ефективності автоматизації. Існуючі метрики часто не враховують специфіку використання ШІ-інструментів та не дозволяють об'єктивно оцінити їх вплив на якість кінцевого продукту. Це створює складнощі при обґрунтуванні доцільності впровадження тих чи інших інструментів автоматизації.

Критичний аналіз сучасних підходів до автоматизації веб-розробки показує, що незважаючи на значний прогрес у цій сфері, залишається багато невирішених проблем. Особливо це стосується питань ефективної інтеграції різних інструментів, забезпечення якості згенерованого коду та розробки методологій оцінки ефективності автоматизації. Вирішення цих проблем є необхідною умовою для подальшого розвитку автоматизації веб-розробки з використанням штучного інтелекту.

1.3 Аналіз сучасних ШІ-інструментів для веб-розробки

Сучасний ринок інструментів розробки програмного забезпечення пропонує широкий спектр рішень на базі штучного інтелекту. Особливої уваги заслуговують інтелектуальні асистенти, інтегровані в середовища розробки, та спеціалізовані сервіси для генерації та аналізу коду. Кожен з цих інструментів має свої особливості та специфічні сфери застосування.

GitHub Copilot [3], який використовує технологію OpenAI Codex, представляє собою один з найбільш розвинених інструментів для автоматизації програмування. Його основною перевагою є здатність розуміти контекст розробки та генерувати релевантний код на основі коментарів та існуючого коду проєкту. Проте практика показує, що цей інструмент має певні

обмеження, особливо при роботі зі складними архітектурними рішеннями та специфічними вимогами до безпеки коду.

Amazon CodeWhisperer пропонує альтернативний підхід, зосереджуючись на безпеці та відповідності найкращим практикам програмування. Цей інструмент особливо ефективний при розробці застосунків, що використовують сервіси AWS, але може мати обмежену ефективність у проєктах з іншими технологічними стеками. Важливою особливістю є вбудована система перевірки безпеки згенерованого коду.

Сервіси на базі великих мовних моделей, такі як Claude AI [7] та ChatGPT, демонструють значний потенціал у контексті програмування. Ці системи здатні не лише генерувати код, але й надавати розгорнуті пояснення, проводити аналіз архітектурних рішень та допомагати у вирішенні складних технічних проблем. Однак їх використання вимагає критичного підходу до отриманих результатів та розуміння обмежень моделей.

Інтегровані в IDE системи автодоповнення коду, такі як IntelliCode та Tabnine, пропонують більш спеціалізований підхід, зосереджуючись на підвищенні продуктивності при написанні коду. Ці інструменти ефективно працюють з типовими паттернами програмування, але можуть бути менш корисними при розробці нестандартних рішень.

Особливу категорію становлять інструменти для аналізу та оптимізації коду. Сучасні системи статичного аналізу, доповнені можливостями ШІ, здатні виявляти потенційні проблеми та пропонувати оптимізації на рівні архітектури застосунку. Проте їх ефективність значною мірою залежить від якості навчальних даних та специфіки конкретного проєкту.

У контексті веб-розробки важливо відзначити інструменти для автоматизації фронтенд-розробки. Існуючі рішення дозволяють генерувати компоненти користувацького інтерфейсу та стилі, але часто стикаються з проблемами адаптації до специфічних вимог дизайну та забезпечення консистентності інтерфейсу.

При розробці серверної частини веб-застосунків ШІ-інструменти демонструють високу ефективність у генерації типового коду для роботи з базами даних та створення API. Проте їх використання для розробки складної бізнес-логіки вимагає ретельної валідації та часто потребує суттєвого доопрацювання згенерованого коду.

Аналіз практичного досвіду використання ШІ-інструментів виявляє ряд спільних проблем. Зокрема, це складність інтеграції різних інструментів у єдиний процес розробки, необхідність постійної валідації згенерованого коду та потреба в адаптації існуючих методологій розробки. Також важливим аспектом є необхідність навчання розробників ефективному використанню цих інструментів.

Особливої уваги заслуговує питання впливу ШІ-інструментів на якість кінцевого продукту. Практика показує, що ефективність їх використання значною мірою залежить від кваліфікації розробників та їх здатності критично оцінювати та адаптувати запропоновані рішення. При цьому важливо забезпечити баланс між автоматизацією та збереженням контролю над процесом розробки.

1.4 Виявлення проблем та обмежень у сучасних ШІ-рішеннях для веб-розробки

Аналіз сучасного стану інтеграції штучного інтелекту у веб-розробку виявляє ряд суттєвих проблем та обмежень, які потребують детального розгляду та пошуку шляхів їх вирішення. Ці проблеми охоплюють різні аспекти розробки програмного забезпечення та мають різний ступінь впливу на ефективність процесу розробки.

Одним з найбільш критичних обмежень сучасних ШІ-інструментів є їх недостатня здатність розуміти загальний контекст проєкту. Незважаючи на значний прогрес у розвитку мовних моделей, існуючі рішення часто генерують код, який технічно коректний, але не враховує специфічні вимоги

проєкту, архітектурні обмеження та встановлені стандарти кодування. Це створює додаткове навантаження на розробників, які змушені витрачати час на адаптацію та доопрацювання згенерованого коду.

Суттєвою проблемою залишається забезпечення якості та надійності згенерованого коду. Сучасні ШІ-системи можуть створювати код, який містить приховані помилки, неоптимальні рішення або потенційні вразливості безпеки. Відсутність ефективних механізмів автоматичної валідації згенерованого коду змушує розробників проводити ретельний код-ревю, що частково нівелює переваги автоматизації.

У контексті веб-розробки особливої гостроти набуває проблема інтеграції різних ШІ-інструментів. Існуючі рішення часто працюють як ізольовані системи, що ускладнює створення єдиного ефективного процесу розробки. Відсутність стандартизованих підходів до інтеграції різних інструментів призводить до фрагментації процесу розробки та знижує загальну ефективність використання ШІ-асистентів.

Важливим обмеженням є недостатня адаптивність ШІ-систем до специфічних потреб конкретних проєктів. Більшість існуючих інструментів базується на загальних моделях та патернах, що не завжди відповідає унікальним вимогам різних проєктів. Особливо це помітно при розробці нестандартних рішень або при роботі з специфічними технологічними стеками.

Проблема масштабування та продуктивності також заслуговує окремої уваги. При роботі з великими проєктами ШІ-інструменти можуть демонструвати зниження ефективності, збільшення часу відгуку та підвищення споживання ресурсів. Це створює обмеження для їх використання в великих командах та при розробці складних веб-застосунків.

Окремо слід відзначити проблеми, пов'язані з навчанням та адаптацією розробників до роботи з ШІ-інструментами. Відсутність систематизованих підходів до навчання та недостатня документація щодо ефективного використання цих інструментів створюють бар'єри для їх широкого впровадження в процеси розробки.

Питання безпеки та конфіденційності даних також залишається критичним. Використання хмарних ШІ-сервісів може викликати занепокоєння щодо захисту інтелектуальної власності та конфіденційної інформації проєктів. Існуючі рішення не завжди надають достатні гарантії щодо збереження конфіденційності коду та даних, що обробляються.

Суттєвою проблемою є відсутність об'єктивних метрик та методологій оцінки ефективності використання ШІ-інструментів. Це ускладнює процес прийняття рішень щодо впровадження конкретних рішень та оцінки їх впливу на продуктивність розробки та якість кінцевого продукту.

Виявлені проблеми та обмеження вказують на необхідність розробки нових підходів до інтеграції штучного інтелекту в процеси веб-розробки. Зокрема, потрібні рішення, які б забезпечували більш глибоке розуміння контексту проєктів, кращу інтеграцію різних інструментів та більш ефективні механізми контролю якості згенерованого коду.

1.5 Обґрунтування необхідності нового дослідження

Проведений аналіз сучасного стану інтеграції штучного інтелекту у веб-розробку демонструє наявність суттєвих можливостей для подальшого розвитку та вдосконалення існуючих підходів. Актуальність нового дослідження обумовлена необхідністю вирішення виявлених проблем та розробки більш ефективних методів використання ШІ-інструментів у процесі створення веб-застосунків.

Існуючі дослідження в області інтеграції ШІ у веб-розробку переважно зосереджені на окремих аспектах автоматизації та не пропонують комплексного підходу до вирішення виявлених проблем. Зокрема, недостатньо вивченими залишаються питання ефективної інтеграції різних ШІ-інструментів у єдиний процес розробки, що створює передумови для проведення нових досліджень у цьому напрямку.

Особливої актуальності набуває дослідження методів підвищення якості згенерованого коду та забезпечення його відповідності специфічним вимогам проєктів. Існуючі рішення не надають достатніх механізмів для автоматичної валідації та оптимізації згенерованого коду, що відкриває простір для розробки нових підходів та методологій.

Важливим аспектом, який потребує детального дослідження, є розробка методів ефективної інтеграції ШІ-інструментів у процеси веб-розробки з урахуванням специфіки різних технологічних стеків. Сучасні веб-застосунки часто використовують складні комбінації технологій, що створює потребу в розробці спеціалізованих підходів до їх автоматизації.

У контексті веб-розробки особливого значення набуває дослідження можливостей використання ШІ для оптимізації архітектури застосунків та покращення їх продуктивності. Існуючі інструменти мають обмежені можливості в цьому напрямку, що створює потребу в розробці нових методів та підходів до архітектурної оптимізації з використанням штучного інтелекту.

Практична значимість нового дослідження полягає в можливості розробки конкретних рекомендацій та методологій для ефективного використання ШІ-інструментів у веб-розробці. Це включає створення підходів до вибору та комбінування різних інструментів, розробку методів оцінки їх ефективності та формування практичних рекомендацій щодо їх впровадження в робочі процеси.

Важливим аспектом дослідження методів навчання розробників ефективному використанню ШІ-інструментів. Існуючі підходи до навчання часто не враховують специфіку роботи з такими інструментами, що створює потребу в розробці спеціалізованих навчальних методик та матеріалів.

Результати нового дослідження можуть мати значний вплив на розвиток методів веб-розробки та сприяти підвищенню ефективності створення програмного забезпечення. Розробка комплексного підходу до інтеграції ШІ-інструментів дозволить вирішити ряд існуючих проблем та створити основу для подальшого розвитку цього напрямку.

Таким чином, необхідність нового дослідження обумовлена як теоретичними, так і практичними потребами галузі веб-розробки. Його результати можуть стати важливим кроком у розвитку методів створення програмного забезпечення з використанням штучного інтелекту та сприяти підвищенню ефективності розробки веб-застосунків.

1.6 Формулювання задач дослідження

На основі проведеного аналізу літературних джерел та виявлених проблем у сфері інтеграції штучного інтелекту в процеси веб-розробки можна визначити основні напрямки та конкретні задачі дослідження, які потребують вирішення для досягнення поставленої мети.

Першочерговою задачею дослідження є розробка методики комплексної інтеграції різних ШІ-інструментів у процес веб-розробки. Ця задача включає визначення критеріїв вибору інструментів для різних етапів розробки, розробку підходів до їх ефективного комбінування та створення механізмів забезпечення їх взаємодії. Важливим аспектом є врахування специфіки різних технологічних стеків та особливостей конкретних проєктів.

Друга важлива задача полягає у дослідженні впливу різних ШІ-інструментів на якість та швидкість розробки веб-застосунків. Це передбачає проведення порівняльного аналізу ефективності різних підходів до використання ШІ в процесі розробки, включаючи оцінку якості згенерованого коду, швидкості розробки та зручності використання різних інструментів.

Третя задача спрямована на розробку методів оцінки та валідації коду, створеного за допомогою ШІ-інструментів. Це включає створення критеріїв оцінки якості згенерованого коду, розробку методик його тестування та валідації, а також формування рекомендацій щодо його оптимізації та адаптації до специфічних вимог проєкту.

Четверта задача передбачає практичну реалізацію веб-застосунку для демонстрації можливостей використання ШІ у веб-розробці. Цей застосунок

повинен не лише демонструвати теоретичні концепції, але й надавати практичні інструменти для роботи з різними аспектами веб-розробки за допомогою ШІ. Важливим аспектом є забезпечення можливості порівняння та оцінки ефективності різних підходів до використання ШІ-інструментів.

П'ята задача полягає у розробці рекомендацій щодо ефективного впровадження ШІ-інструментів у процеси веб-розробки. Це включає створення методик навчання розробників, формування практичних рекомендацій щодо вибору та використання різних інструментів, а також розробку підходів до оцінки ефективності їх впровадження.

Вирішення поставлених задач дозволить створити комплексний підхід до інтеграції штучного інтелекту в процеси веб-розробки та забезпечити підвищення ефективності створення програмного забезпечення. Результати дослідження можуть бути використані як основа для подальшого розвитку методів та інструментів автоматизації веб-розробки з використанням штучного інтелекту.

2 АНАЛІЗ МЕТОДІВ ІНТЕГРАЦІЇ ТА ВИКОРИСТАННЯ ШІ-ІНСТРУМЕНТІВ У ВЕБ-РОЗРОБЦІ

2.1 Дослідження існуючих методів інтеграції ШІ-інструментів

Інтеграція ШІ-інструментів у процес веб-розробки може здійснюватися різними способами, кожен з яких має свої особливості та сфери застосування. На основі аналізу існуючих підходів можна виділити три основні методи інтеграції: локальна інтеграція через IDE, використання хмарних сервісів та гібридний підхід.

Локальна інтеграція через IDE представляє собою вбудовування ШІ-функціональності безпосередньо в середовище розробки. Цей метод реалізується через встановлення спеціальних плагінів або розширень, які забезпечують доступ до ШІ-можливостей. Прикладом такого підходу є GitHub Copilot [3], який інтегрується в популярні IDE через відповідні розширення. Основними перевагами цього методу є:

– висока швидкість відгуку за рахунок локальної обробки частини запитів;

– можливість роботи з кодом проєкту в контексті IDE;

– зручність використання завдяки інтеграції з існуючими інструментами розробки.

Використання хмарних сервісів передбачає взаємодію з ШІ-інструментами через веб-інтерфейси або API. Цей метод не потребує встановлення додаткового програмного забезпечення та забезпечує доступ до потужних обчислювальних ресурсів. До переваг цього підходу належать:

– доступ до найновіших версій моделей та алгоритмів;

– відсутність навантаження на локальні ресурси;

– можливість масштабування та адаптації до потреб проєкту.

Гібридний підхід поєднує елементи обох попередніх методів, дозволяючи використовувати як локальні, так і хмарні ресурси. Цей метод

забезпечує найбільшу гнучкість у використанні ШІ-інструментів та може адаптуватися до різних сценаріїв розробки. Основними характеристиками цього підходу є:

– можливість вибору оптимального способу обробки для різних типів задач;

– балансування навантаження між локальними та хмарними ресурсами;

– підвищена надійність за рахунок резервування.

При виборі методу інтеграції важливо враховувати специфічні вимоги проєкту та обмеження, що накладаються на процес розробки. До ключових факторів, які впливають на вибір методу, належать:

– вимоги до безпеки та конфіденційності коду;

– доступні обчислювальні ресурси;

– бюджетні обмеження;

– специфіка технологічного стеку;

– розмір команди розробників.

Особливої уваги заслуговує питання безпеки при інтеграції ШІ-інструментів. Для кожного методу існують свої підходи до забезпечення безпеки:

– для локальної інтеграції – використання локальних моделей та обмеження доступу до мережі;

– для хмарних сервісів – шифрування даних та використання захищених каналів зв'язку;

– для гібридного підходу – комбінація різних методів захисту та розподіл критичних даних.

При реалізації інтеграції важливо забезпечити моніторинг та оцінку ефективності використання ШІ-інструментів. Для цього можуть використовуватися різні метрики:

– час, витрачений на написання коду;

– кількість помилок та їх складність;

– обсяг згенерованого коду;

– якість згенерованих рішень.

Практика показує, що успішна інтеграція ШІ-інструментів вимагає системного підходу та врахування всіх аспектів розробки. Важливо забезпечити:

– навчання розробників ефективному використанню інструментів;
– створення процедур валідації та тестування згенерованого коду;
– розробку механізмів зворотного зв'язку для покращення якості роботи ШІ;
– регулярний аналіз та оптимізацію процесів інтеграції.

2.2 Комплексне використання різних ШІ-інструментів

Ефективне використання ШІ-інструментів у веб-розробці вимагає системного підходу до їх комбінування та застосування на різних етапах розробки. Розроблена методика базується на принципі максимального використання сильних сторін кожного інструменту та їх оптимального поєднання для досягнення найкращих результатів.

У процесі веб-розробки можна виділити декілька ключових етапів, на кожному з яких доцільно застосовувати специфічні ШІ-інструменти. На етапі планування та проєктування особливо ефективним є використання потужних мовних моделей, таких як ChatGPT та Claude. Ці інструменти дозволяють проводити глибокий аналіз вимог та генерувати попередні архітектурні рішення. Крім того, вони допомагають у створенні початкової структури проєкту та оцінці його складності.

При розробці клієнтської та серверної частини веб-застосунку найбільш ефективним виявляється використання GitHub Copilot. Цей інструмент особливо корисний при написанні компонентів та реалізації бізнес-логіки. Додатково можуть застосовуватися спеціалізовані інструменти для генерації стилів та оптимізації користувацького інтерфейсу. Значну користь приносить також автоматизація генерації типових серверних компонентів, що дозволяє суттєво прискорити процес розробки.

Для забезпечення ефективної інтеграції різних інструментів важливо дотримуватися чіткого алгоритму дій. Спочатку проводиться детальний аналіз

поставленої задачі, визначається її тип та складність. На основі цього аналізу обирається оптимальна комбінація ШІ-інструментів. Наступним кроком є підготовка контексту, що включає збір необхідної інформації про проєкт та формування чітких вимог і обмежень.

У процесі виконання роботи важливо забезпечити послідовне застосування вибраних інструментів та їх ефективну взаємодію. Особлива увага приділяється координації між різними компонентами та контролю якості отриманих результатів. Завершальним етапом є валідація результатів, яка включає перевірку відповідності вимогам, тестування функціональності та оптимізацію згенерованого коду.

Підхід базується на трьох ключових принципах: спеціалізації, послідовності та контролю. Принцип спеціалізації передбачає використання кожного інструменту для тих задач, де він демонструє найвищу ефективність. Принцип послідовності забезпечує оптимальний порядок застосування інструментів та передачу контексту між етапами розробки. Принцип контролю гарантує постійний моніторинг якості результатів та своєчасне виявлення можливих проблем.

Представлений теоретичний аналіз методів інтеграції та використання ШІ-інструментів у веб-розробці створює підґрунтя для проведення практичного дослідження їх ефективності. Для перевірки теоретичних висновків та отримання кількісних показників ефективності різних підходів було проведено експертне дослідження, результати якого представлені у наступному розділі. Особлива увага в дослідженні приділяється оцінці впливу ШІ-інструментів на ключові показники процесу розробки: час розробки компонентів, кількість рядків коду, кількість помилок та час їх виправлення.

Практичне застосування розробленої методики показує значне підвищення ефективності процесу розробки. Спостерігається помітне скорочення часу виконання типових задач, зменшення кількості помилок та обсягу необхідних доопрацювань. Важливим показником є також загальне підвищення продуктивності розробки та оптимізація використання ресурсів команди.

В цілому, комплексне використання різних ШІ-інструментів за розробленою методикою дозволяє досягти суттєвого покращення як швидкості розробки, так і якості кінцевого продукту. При цьому важливо забезпечувати постійний моніторинг та адаптацію методики відповідно до специфіки конкретних проєктів та потреб команди розробників.

Проведений у даному розділі аналіз методів інтеграції та використання ШІ-інструментів у веб-розробці дозволив сформувати цілісне розуміння сучасних підходів до автоматизації процесів розробки програмного забезпечення.

Дослідження існуючих методів інтеграції ШІ-інструментів показало наявність трьох основних підходів: локальна інтеграція через IDE, використання хмарних сервісів та гібридний підхід. Кожен з цих методів має свої переваги та обмеження, що впливає на їх ефективність у різних умовах розробки. Важливим результатом аналізу стало виявлення критичних факторів, які необхідно враховувати при виборі методу інтеграції, зокрема:

– вимоги до безпеки та конфіденційності даних;
– наявні технічні ресурси та обмеження;
– специфіка проєкту та потреби команди розробників.

Розроблена методика комплексного використання різних ШІ-інструментів представляє собою систематизований підхід до організації процесу веб-розробки з використанням штучного інтелекту. Ключовою особливістю запропонованої методики є її гнучкість та адаптивність до різних умов розробки. Методика враховує специфіку різних етапів розробки та пропонує оптимальні комбінації інструментів для кожного з них.

Представлений підхід потребує експериментальної перевірки та підтвердження їх ефективності в реальних умовах розробки. Для цього було проведено експертне дослідження, що дозволило отримати конкретні кількісні показники ефективності використання різних ШІ-інструментів у процесі веб-розробки. Результати цього дослідження дозволяють не лише підтвердити теоретичні висновки, але й надати конкретні рекомендації щодо оптимального використання ШІ-інструментів у різних сценаріях розробки.

Практична цінність отриманих результатів полягає в можливості їх безпосереднього застосування при організації процесів веб-розробки. Запропоновані підходи дозволяють суттєво підвищити ефективність розробки за рахунок оптимального використання можливостей різних ШІ-інструментів, зменшення часу на рутинні операції, покращення якості генерованого коду, підвищення загальної продуктивності команди розробників.

Важливим результатом дослідження стало визначення основних принципів ефективної інтеграції ШІ-інструментів, що включають спеціалізацію, послідовність та контроль. Дотримання цих принципів дозволяє забезпечити системний підхід до використання штучного інтелекту в процесі розробки та мінімізувати можливі ризики.

Проведений аналіз також виявив ряд потенційних напрямків для подальшого дослідження, зокрема:

– розробка методів автоматизованої оцінки ефективності використання ШІ-інструментів;

– вдосконалення механізмів інтеграції різних інструментів;

– створення спеціалізованих методик навчання розробників роботі з ШІ-асистентами.

Результати цього розділу створюють теоретичну основу для подальшої практичної реалізації веб-застосунку з використанням ШІ-інструментів. Розроблені підходи та методики будуть використані при створенні конкретних програмних рішень, що дозволить перевірити їх ефективність на практиці та визначити напрямки для подальшого вдосконалення.

3 ДОСЛІДЖЕННЯ ВПЛИВУ ШІ-ІНСТРУМЕНТІВ НА ПРОЦЕС РОЗРОБКИ

3.1 Оцінка впливу ШІ-інструментів на якість та швидкість розробки

Для проведення об'єктивного дослідження впливу ШІ-інструментів на процес веб-розробки розроблено комплексну методику оцінки, яка дозволяє отримати кількісні та якісні показники ефективності їх використання. Дослідження проводиться на основі розробки однакових компонентів веб-застосунку різними методами, що дозволяє провести порівняльний аналіз результатів. У дослідженні бере участь група з п'яти фахівців у сфері веб-розробки. Фахівці виконують розробку компонентів трьома різними способами: традиційна розробка без використання ШІ-інструментів, розробка з використанням GitHub Copilot, та розробка з використанням веб-сервісу Claude AI. Для кожного компонента веб-застосунку фіксуються наступні кількісні показники:

– час, витрачений на розробку компонента, включаючи створення базової структури, реалізацію логіки, наповнення функціоналом та документування;

– кількість рядків написаного коду, враховуючи як функціональний код, так і коментарі;

– кількість помилок, виявлених під час тестування, включаючи синтаксичні помилки, логічні помилки та проблеми з продуктивністю;

– час, витрачений на виправлення виявлених помилок;

– кількість ітерацій розробки до досягнення необхідного результату.

Для забезпечення об'єктивності результатів усі експерименти проводяться в однакових умовах. Фахівці працюють з однаковим набором завдань, використовують однакове середовище розробки та мають доступ до необхідної документації. При розробці компонентів застосовуються єдині стандарти кодування та вимоги до функціональності.

Фіксація результатів відбувається безпосередньо в процесі розробки. Кожен експерт веде детальний журнал, де записує всі кількісні показники для кожного етапу розробки. Ці дані потім агрегуються для проведення порівняльного аналізу різних підходів до розробки. Для аналізу отриманих даних використовуються методи статистичної обробки, що дозволяють визначити середні показники для кожного методу розробки та оцінити їх ефективність. Особлива увага приділяється порівнянню часових витрат та якості отриманого коду при використанні різних підходів.

Експеримент передбачає поетапну оцінку кожного з досліджуваних інструментів, що дозволяє виявити їх сильні та слабкі сторони в контексті різних аспектів веб-розробки. Такий системний підхід забезпечує отримання комплексної картини впливу ШІ-інструментів на процес розробки та дозволяє сформувати обґрунтовані рекомендації щодо їх ефективного використання.

3.2 Порівняльний аналіз ефективності різних підходів

3.2.1 Аналіз традиційного підходу до розробки

При традиційному підході фахівці розробляли компоненти без використання ШІ-інструментів, спираючись на стандартні можливості середовища розробки та власний досвід.

За результатами експериментів було отримано наступні показники.

Часові витрати на один компонент склали 15-20 хвилин для базової структури, 60-120 хвилин для складної логіки, 40-60 хвилин для наповнення компонента функціоналом, та 30-45 хвилин для документації. Середня кількість рядків коду для типового компонента становила 180-220 рядків, включаючи коментарі та документацію. При тестуванні в середньому виявлялось 8-12 помилок різного типу на компонент, при цьому на їх виправлення витрачалось 45-60 хвилин. Кількість ітерацій розробки до досягнення необхідного результату в середньому становила 4-5.

3.2.2 Аналіз розробки з використанням GitHub Copilot

При використанні GitHub Copilot [3] (рис. 3.1) спостерігалось значне покращення показників. Часові витрати скоротились до 5-8 хвилин для базової структури, 30-65 хвилин для складної логіки, 10-12 хвилин для наповнення функціоналом та 8-10 хвилин для документації.

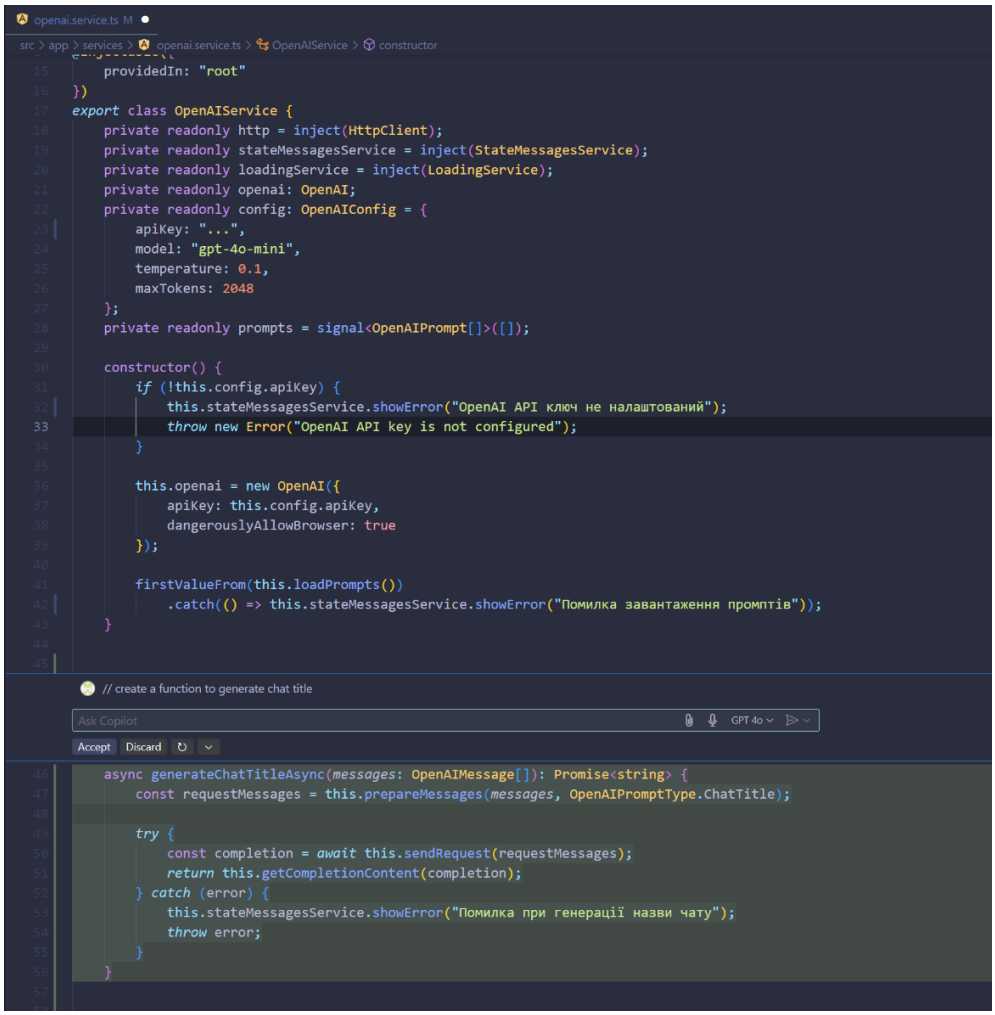


Рисунок 3.1 – Генерація коду з використанням GitHub Copilot

Середня кількість рядків коду зменшилась до 150-180 рядків при збереженні повної функціональності. Кількість виявлених помилок знизилась до 5-7 на компонент, а час на їх виправлення становив 20-30 хвилин. Кількість необхідних ітерацій розробки скоротилась до 2-3.

3.2.3 Аналіз розробки з використанням Claude AI

Claude AI [7] (рис. 3.2) продемонстрував схожі показники ефективності. Часові витрати становили 7-10 хвилин для базової структури, 20-45 хвилин для складної логіки, 12-15 хвилин для функціонального наповнення та 7-10 хвилин для документації.

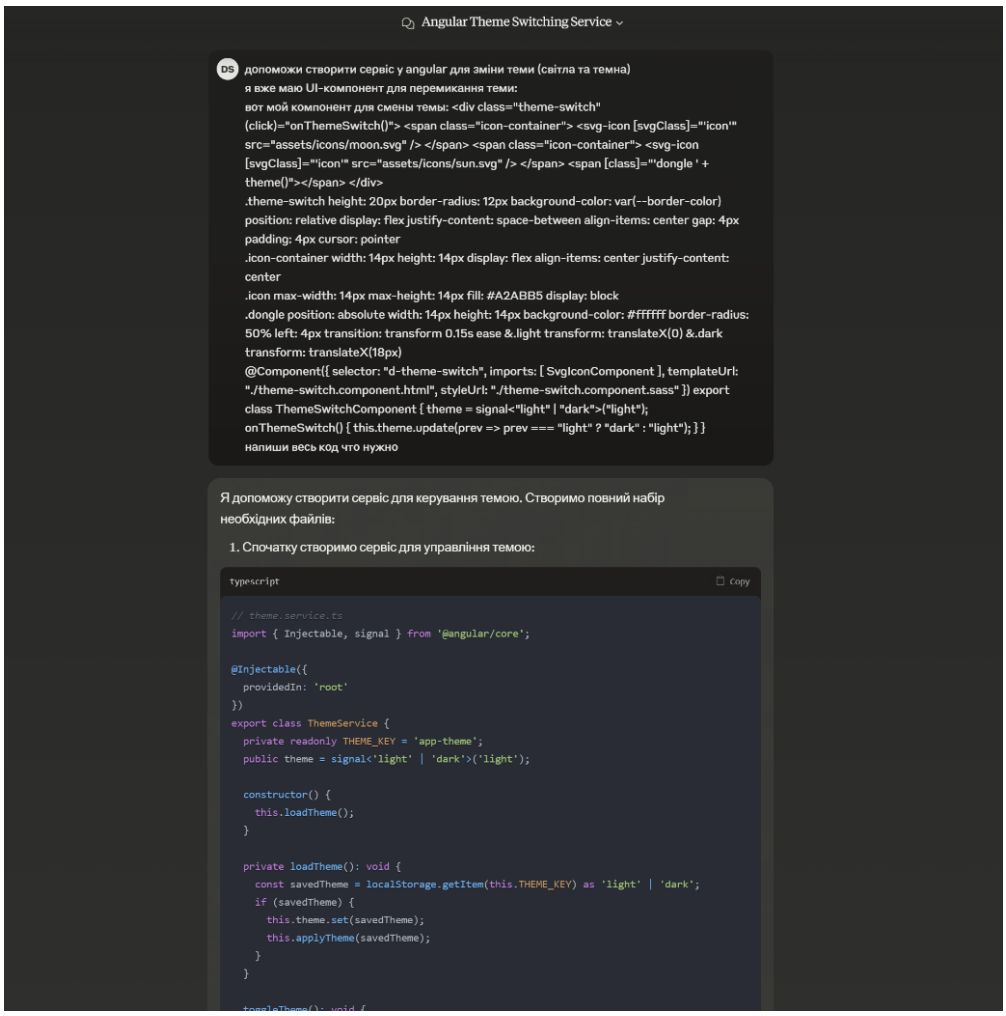


Рисунок 3.2 – Генерація коду з використанням Claude

Обсяг коду складав 140-170 рядків, при цьому код відрізнявся високим рівнем читабельності та якості документації. В процесі тестування виявлялось 4-6 помилок на компонент, на виправлення яких витрачалось 15-25 хвилин. Кількість ітерацій розробки також становила 2-3.

3.2.4 Порівняльний аналіз підходів

Аналіз показників всіх трьох підходів демонструє суттєві переваги використання ШІ-інструментів. У порівнянні з традиційним підходом, застосування GitHub Copilot забезпечило:

– скорочення загального часу розробки на 67,7%;
– зменшення кількості рядків коду на 16,7%;
– зниження кількості помилок на 41,7%;
– скорочення часу на виправлення помилок на 55,6%;
– зменшення кількості необхідних ітерацій на 50%.

Використання Claude AI показало схожу ефективність:

– скорочення загального часу розробки на 58,2%;
– зменшення кількості рядків коду на 22,2%;
– зниження кількості помилок на 50%;
– скорочення часу на виправлення помилок на 66,7%;
– зменшення кількості необхідних ітерацій на 50%.

При цьому важливо відзначити, що обидва ШІ-інструменти не лише прискорюють процес розробки, але й сприяють підвищенню якості коду за рахунок зменшення кількості помилок та покращення структурованості коду.

На основі даних, отриманих в ході дослідження, було проведено комплексний аналіз ефективності різних підходів до веб-розробки. Узагальнені результати представлені в таблиці 3.1, яка демонструє ключові показники для кожного з досліджених методів.

Для наочного представлення порівняння кількості рядків коду та помилок при різних підходах було створено діаграму (рис. 3.3).

Аналіз часу, витраченого на виправлення помилок, представлений у таблиці 3.2, демонструє значне підвищення ефективності при використанні ШІ-інструментів.

Таблиця 3.1 – Порівняння часових витрат на розробку компонента

Тип завдання	Традиційний підхід (хв)	З використанням ШІ (хв)	Економія часу (%)
Створення компоненту зі стилями	15-20	5-10	57,1
Написання складної логіки компонентів	60-120	20-65	52,8
Наповнення компоненту вмістом	40-60	10-15	75,0
Написання документації до компоненту	30-45	7-10	77,3

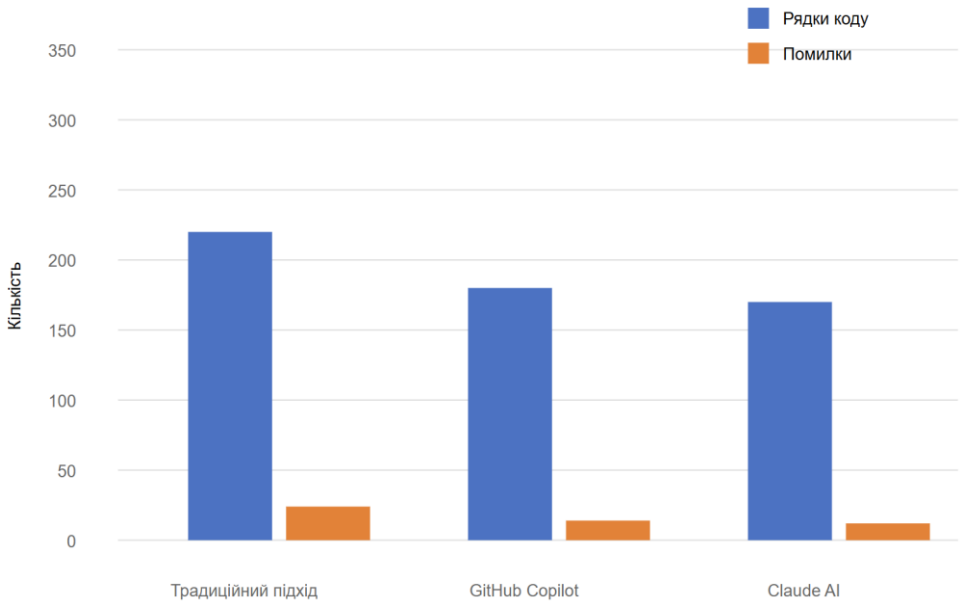


Рисунок 3.3 – Порівняння кількості рядків коду та помилок при різних підходах

Таблиця 3.2 – Час на виправлення помилок

Підхід	Середній час на виправлення помилок (хв)	Зниження часу (%)
Традиційний підхід	45-60	-
GitHub Copilot	20-30	55,6
Claude AI	15-25	66,7

На основі проведеного аналізу можна зробити наступні висновки щодо ефективності різних підходів. Використання ШІ-інструментів демонструє суттєве підвищення продуктивності розробки за всіма досліджуваними показниками. Найбільш значне покращення спостерігається у часових показниках, де економія досягає 77,3% при створенні документації та 75% при наповненні компонентів функціоналом.

Якісні показники також демонструють позитивну динаміку. Зменшення кількості рядків коду при збереженні функціональності свідчить про більш ефективну та оптимізовану структуру коду. Значне зниження кількості помилок та часу на їх виправлення вказує на підвищення загальної якості розробки.

Особливо важливим є те, що обидва досліджені ШІ-інструменти показали схожу ефективність, хоча і мають свої особливості застосування. GitHub Copilot демонструє кращі результати при роботі з типовими компонентами, тоді як Claude AI більш ефективний при розробці складних алгоритмічних рішень та створенні документації.

Загальна економія ресурсів при використанні ШІ-інструментів, враховуючи всі досліджені показники, складає:

– GitHub Copilot: в середньому 56,3%;

– Claude AI: в середньому 61,8%.

Ці результати підтверджують високу ефективність впровадження ШІ-інструментів у процес веб-розробки та вказують на доцільність їх подальшого використання та вдосконалення.

4 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ З ВИКОРИСТАННЯМ ШІ-ІНСТРУМЕНТІВ

4.1 Проєктування архітектури веб-застосунку

В рамках дослідження впливу ШІ-інструментів на процес веб-розробки було створено веб-застосунок, який не лише демонструє можливості сучасних технологій, але й сам використовує штучний інтелект для допомоги розробникам. Архітектура застосунку розроблялася з урахуванням результатів попередніх етапів дослідження та спрямована на максимальну ефективність взаємодії користувача з ШІ-інструментами. При проєктуванні архітектури веб-застосунку були враховані результати роботи описаної у попередньому розділі ефективності різних ШІ-інструментів, представлені у попередньому розділі.

Проєктуючи архітектуру застосунку було прийнято рішення використовувати підхід Single Page Application (SPA), що дозволяє створити більш динамічний та гнучкий інтерфейс. Для реалізації клієнтської частини обрано фреймворк Angular [5] з використанням standalone компонентів замість традиційних NgModule. Це рішення забезпечує більш легковесну та гнучку архітектуру, де кожен компонент може використовуватись незалежно, що значно спрощує тестування та підтримку коду. Серверна частина реалізована на платформі ASP.NET Core [6], що забезпечує високу продуктивність та надійність. Для зберігання даних використовується MS SQL Server.

Ключовою особливістю розробленої архітектури є пряма взаємодія клієнтської частини з OpenAI API. Таке рішення було прийнято для зменшення навантаження на сервер та забезпечення більш швидкого відгуку системи. Серверна частина відповідає лише за базовий функціонал: автентифікацію користувачів та збереження інформації про чати. Цей підхід дозволив створити гнучку систему, де користувачі можуть ефективно взаємодіяти з ШІ-сервісами без додаткових затримок на серверну обробку.

Клієнтська частина застосунку побудована на основі незалежних компонентів, кожен з яких відповідає за певну функціональність. Основні standalone компоненти включають компоненти автентифікації (SignInComponent, SignUpComponent, ProfileComponent), компоненти чату (ChatComponent, MessagesComponent, InputComponent) та спільні компоненти для всього застосунку (HeaderComponent, FooterComponent, LoaderComponent). Взаємодія між компонентами забезпечується через систему сервісів, які включають AuthService для роботи з автентифікацією, ChatService для взаємодії з серверним API, OpenAIService для прямої комунікації з OpenAI та StateService для управління станом додатку.

Серверна частина має лаконічну структуру, відповідаючи за обмежений набір функцій. Основними компонентами є контролери для обробки запитів автентифікації та управління даними користувачів, сервіси для роботи з базою даних, та системи валідації запитів. Важливим елементом є система управління JWT-токенами, яка забезпечує безпечну автентифікацію користувачів.

Особливу увагу при проєктуванні було приділено системі спеціалізованих чатів. В системі реалізовано п'ять основних напрямків:

– код-асистент для допомоги з написанням програмного коду;
– UI-компоненти для розробки користувацького інтерфейсу;
– оптимізація для покращення продуктивності коду;
– тестування для допомоги у створенні тестів;
– документація для автоматизації створення технічної документації.

Кожен тип чату має власні налаштування промптів та специфічні обмеження, що дозволяє отримувати більш релевантні відповіді для конкретних задач розробки.

В системі реалізовано різні рівні доступу для авторизованих та неавторизованих користувачів. Неавторизовані користувачі мають доступ до базового функціоналу, що дозволяє оцінити можливості системи. Після авторизації користувачам стає доступний розширений функціонал, включаючи збереження історії чатів та можливість продовжувати попередні діалоги.

При розробці архітектури активно використовувались результати попередніх етапів дослідження ефективності різних ШІ-інструментів. Було застосовано комбінований підхід до розробки, де GitHub Copilot використовувався для створення базових компонентів, Claude AI допомагав з проєктуванням складних архітектурних рішень, а критичні компоненти безпеки розроблялися традиційним способом.

Розроблена архітектура (рис. 4.1) забезпечує необхідну гнучкість для подальшого розвитку системи та можливість легкої інтеграції нових функцій та інструментів. Особливо важливим є те, що система може бути легко адаптована для роботи з новими моделями ШІ та іншими API, що з'являться в майбутньому. Використання standalone компонентів в Angular та пряма взаємодія з OpenAI API створюють оптимальну основу для подальшого масштабування та вдосконалення системи.

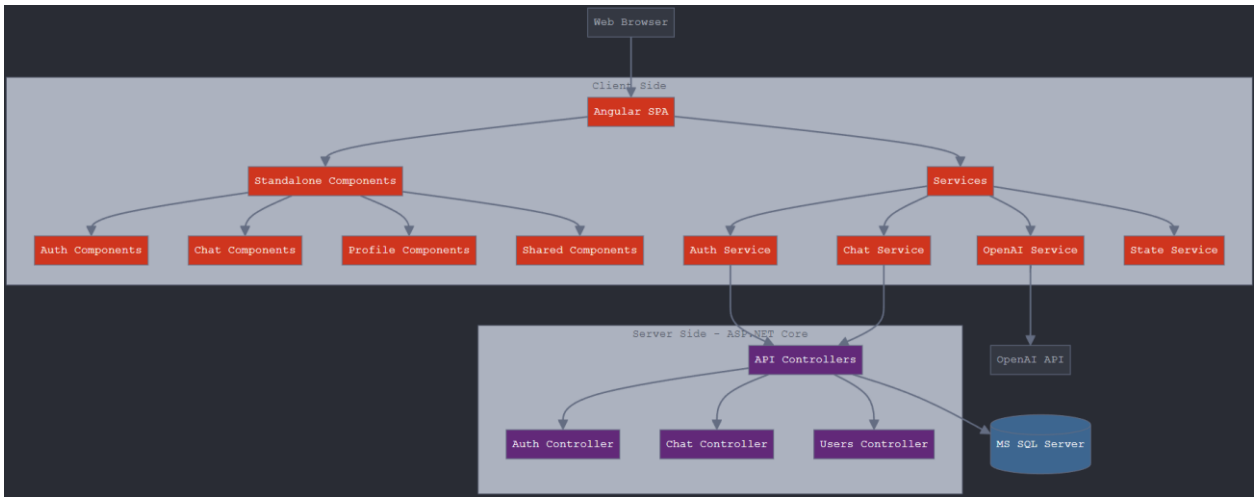


Рисунок 4.1 – Розроблена архітектура веб-застосунку

4.2 Реалізація клієнтської частини

Реалізація клієнтської частини веб-застосунку проводилась з використанням фреймворку Angular [5] версії 19, що дозволило застосувати найновіші можливості та підходи до розробки. Відповідно до результатів дослідження ефективності різних підходів до розробки, реалізація відбувалася

з активним використанням ШІ-інструментів, зокрема GitHub Copilot для стандартних компонентів та Claude AI для складних архітектурних рішень.

При розробці було використано новий підхід Angular з standalone компонентами, що дозволило створити більш оптимізовану структуру застосунку. Основною перевагою такого підходу є можливість створення незалежних компонентів без необхідності їх об'єднання в модулі, що значно спрощує процес розробки та тестування. В процесі розробки активно використовувались сигнали (signals) замість традиційних RxJS потоків, що дозволило створити більш прозору та ефективну систему управління станом компонентів.

Важливим компонентом системи є сервіс взаємодії з OpenAI API (рис. 4.2), який забезпечує основну функціональність генерації відповідей та обробки помилок.

```typescript
@Injectable({
    providedIn: "root"
})
export class OpenAIService {
    private readonly http = inject(HttpClient);
    private readonly stateMessagesService = inject(StateMessagesService);
    private readonly loadingService = inject(LoadingService);
    private readonly openai: OpenAI;
    private readonly config: OpenAIConfig = {
        apiKey: "...",
        model: "gpt-4o-mini",
        temperature: 0.1,
        maxTokens: 2048
    };
    private readonly prompts = signal<OpenAIPrompt[]>([]);

    constructor() {
        if (!this.config.apiKey) {
            this.stateMessagesService.showError("OpenAI API ключ не налаштований");
            throw new Error("OpenAI API key is not configured");
        }

        this.openai = new OpenAI({
            apiKey: this.config.apiKey,
            dangerouslyAllowBrowser: true
        });

        firstValueFrom(this.loadPrompts())
            .catch(() => this.stateMessagesService.showError("Помилка завантаження промптів"));
    }

    generateChatTitle(messages: OpenAIMessage[]): Observable<string> {
        const requestMessages = this.prepareMessages(messages, OpenAIPromptType.ChatTitle);

        return from(this.sendRequest(requestMessages)).pipe(
            map(this.getCompletionContent),
            catchError(error => {
                this.stateMessagesService.showError("Помилка при генерації назви чату");

                return throwError(() => error);
            })
        );
    }
```

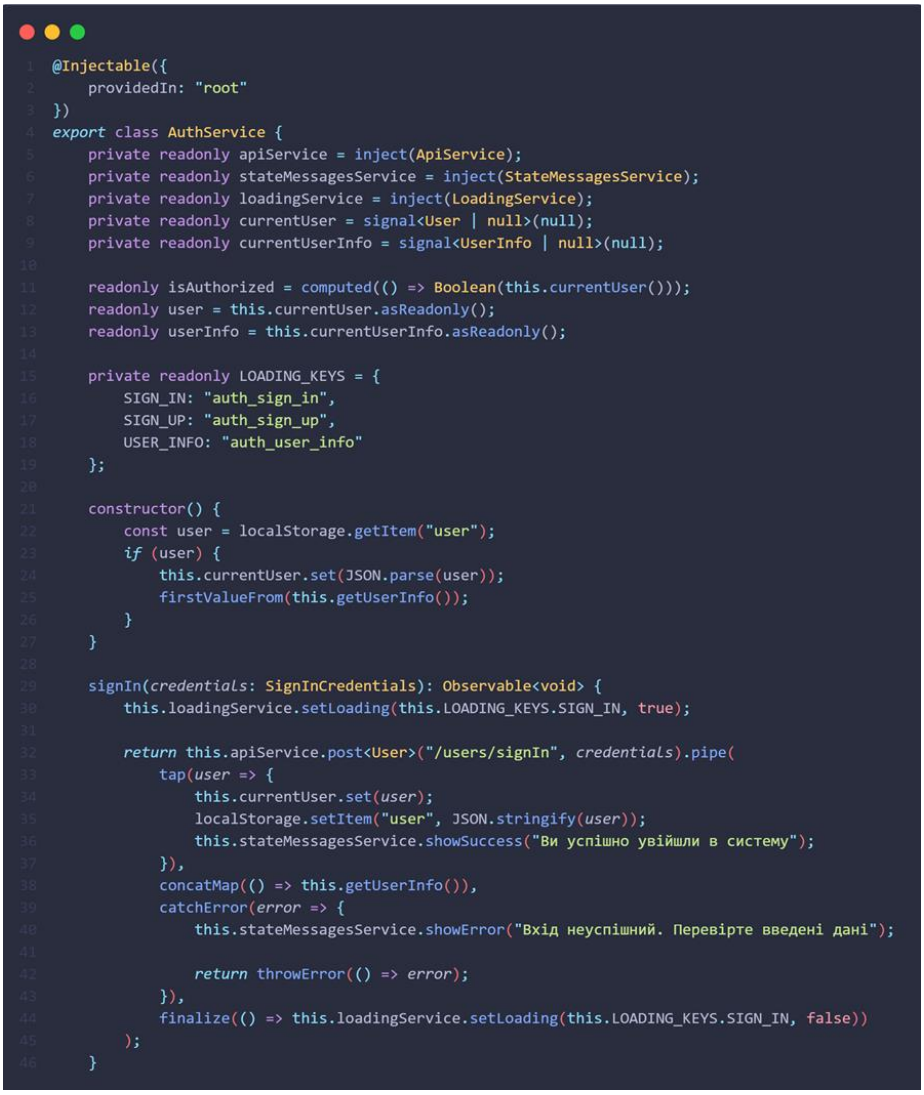
Рисунок 4.2 – Приклад коду сервіса взаємодії з OpenAI API

Центральним елементом застосунку є система чатів, реалізована через набір взаємопов'язаних компонентів. Основний компонент чату забезпечує відображення історії повідомлень, обробку введення користувача та взаємодію з OpenAI API. Компонент вводу повідомлень реалізований як окремий елемент з підтримкою автоматичного розширення текстового поля (рис. 4.3).

```
@Component({
    selector: "d-input",
    imports: [
        SvgIconComponent,
        FormsModule,
        LoaderComponent
    ],
    templateUrl: "./input.component.html",
    styleUrl: "./input.component.sass"
})
export class InputComponent {
    private readonly textareaRef = viewChild<ElementRef<HTMLTextAreaElement>>("textarea");

    disabled = input.required<boolean>();
    messageSubmit = output<string>();

    protected value = signal("");

    autoResize() {
        const el = this.textareaRef()?.nativeElement;
        if (el) {
            el.style.height = "auto";
            el.style.height = `${el.scrollHeight}px`;
        }
    }

    protected send() {
        if (this.value().trim() && !this.disabled()) {
            this.messageSubmit.emit(this.value().trim());
            this.value.set("");
            this.autoResize();
        }
    }

    protected onEnter(event: KeyboardEvent) {
        if (event.key === 'Enter' && !event.shiftKey) {
            event.preventDefault();
            this.send();
        }
    }
}
```

Рисунок 4.3 – Приклад коду компоненту вводу

Важливою частиною системи є сервіс автентифікації, який відповідає за управління станом користувача та взаємодію з серверним API. Сервіс реалізує функціонал входу, виходу та отримання інформації про користувача (рис. 4.4).



```typescript
@Injectable({
    providedIn: "root"
})
export class AuthService {
    private readonly apiService = inject(ApiService);
    private readonly stateMessagesService = inject(StateMessagesService);
    private readonly loadingService = inject(LoadingService);
    private readonly currentUser = signal<User | null>(null);
    private readonly currentUserInfo = signal<UserInfo | null>(null);

    readonly isAuthorized = computed(() => Boolean(this.currentUser()));
    readonly user = this.currentUser.asReadonly();
    readonly userInfo = this.currentUserInfo.asReadonly();

    private readonly LOADING_KEYS = {
        SIGN_IN: "auth_sign_in",
        SIGN_UP: "auth_sign_up",
        USER_INFO: "auth_user_info"
    };

    constructor() {
        const user = localStorage.getItem("user");
        if (user) {
            this.currentUser.set(JSON.parse(user));
            firstValueFrom(this.getUserInfo());
        }
    }

    signIn(credentials: SignInCredentials): Observable<void> {
        this.loadingService.setLoading(this.LOADING_KEYS.SIGN_IN, true);

        return this.apiService.post<User>("/users/signIn", credentials).pipe(
            tap(user => {
                this.currentUser.set(user);
                localStorage.setItem("user", JSON.stringify(user));
                this.stateMessagesService.showSuccess("Ви успішно увійшли в систему");
            }),
            concatMap(() => this.getUserInfo()),
            catchError(error => {
                this.stateMessagesService.showError("Вхід неуспішний. Перевірте введені дані");

                return throwError(() => error);
            }),
            finalize(() => this.loadingService.setLoading(this.LOADING_KEYS.SIGN_IN, false))
        );
    }
```

Рисунок 4.4 – Приклад коду сервісу автентифікації

Для покращення користувацького досвіду було реалізовано спеціалізований сервіс StateMessagesService, який відповідає за відображення повідомлень про стан системи. Цей сервіс забезпечує уніфікований підхід до сповіщення користувача про успішні операції, помилки та важливі події в системі. Повідомлення відображаються у вигляді спливаючих нотифікацій з різними типами оформлення залежно від типу повідомлення (success, error, warning, info).

Окремої уваги заслуговує реалізація системи тем оформлення. ThemeService забезпечує можливість перемикання між світлою та темною темами, а також зберігає вибір користувача між сесіями. Сервіс враховує системні налаштування користувача та автоматично застосовує відповідну

тему при першому запуску застосунку. Зміна теми відбувається через зміну CSS-змінних на рівні кореневого елементу, що забезпечує миттєве оновлення інтерфейсу без необхідності перезавантаження сторінки.

Система маршрутизації реалізована з використанням нового підходу Angular, де кожен маршрут асоційований з окремим standalone компонентом. Це забезпечує ефективне розділення коду та lazy loading для оптимізації завантаження. На кожній сторінці реалізовано окремий тип чату зі специфічними налаштуваннями промптів для конкретної задачі розробки.

В процесі розробки значна увага приділялася оптимізації продуктивності. Використання standalone компонентів та сигналів дозволило зменшити розмір файлів кінцевих файлів коду та покращити швидкість роботи застосунку.

Результати розробки клієнтської частини підтвердили ефективність використання ШІ-інструментів у процесі розробки. Зокрема, було досягнуто значного скорочення часу розробки при збереженні високої якості коду та оптимальної архітектури застосунку. Використання сучасних можливостей Angular 19 у поєднанні з ШІ-інструментами дозволило створити високопродуктивний та зручний у використанні веб-застосунок.

4.3 Реалізація серверної частини

Реалізація серверної частини веб-застосунку проводилась з використанням платформи ASP.NET Core 8.0 [6], що забезпечує високу продуктивність та надійність системи. Розробка серверної частини, як і клієнтської, виконувалась з використанням комбінованого підходу до застосування ШІ-інструментів, що було обґрунтовано результатами попередніх етапів дослідження.

Серверна частина застосунку виконує досить обмежений набір функцій, оскільки основна взаємодія з OpenAI API відбувається безпосередньо з клієнтської частини. Основними задачами серверної частини є забезпечення

автентифікації користувачів та збереження даних про чати в базі даних. Таке розділення відповідальності дозволило створити більш оптимізовану та ефективну архітектуру системи.

База даних реалізована з використанням Entity Framework Core та MS SQL Server. Структура бази даних включає таблиці для зберігання інформації про користувачів, чати та повідомлення. При проєктуванні схеми бази даних [9] (рис. 4.5) особлива увага приділялася оптимізації запитів та забезпеченню цілісності даних.

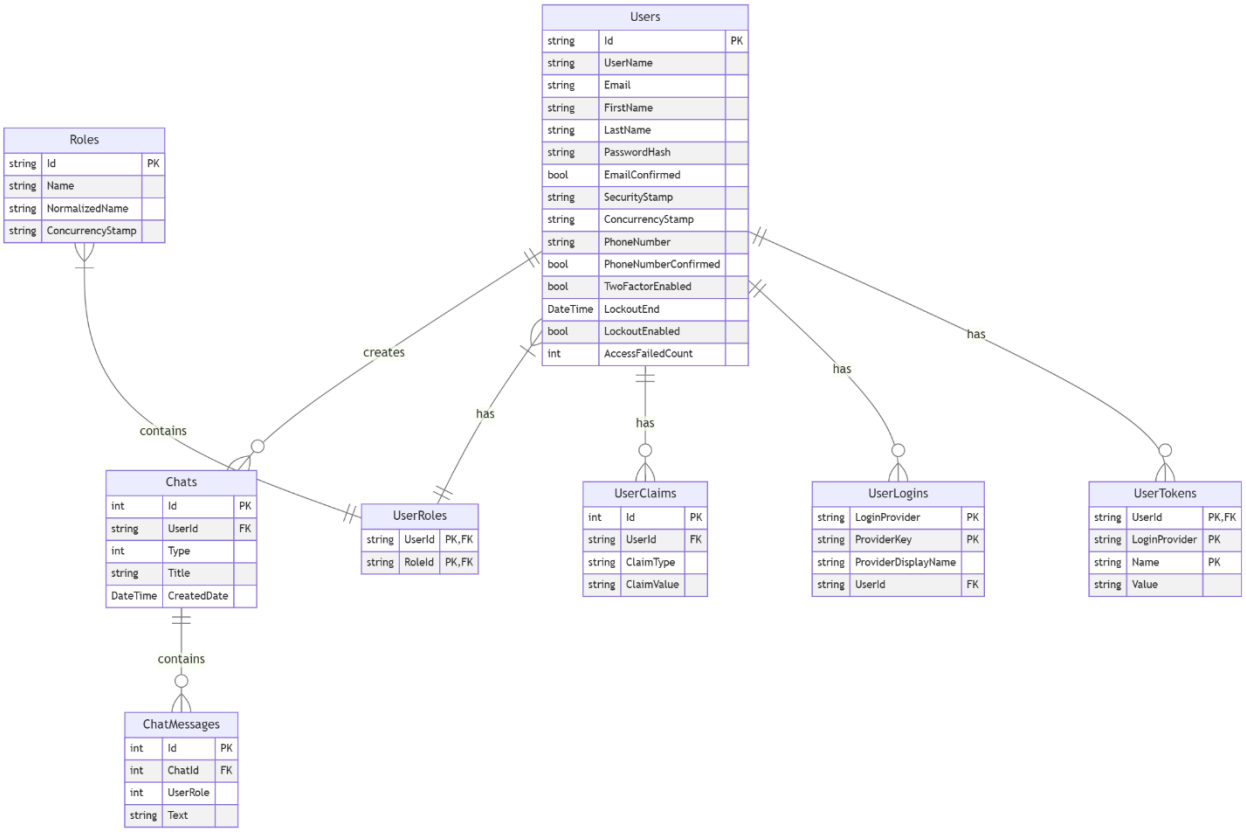


Рисунок 4.5 – Схема бази даних

Система автентифікації реалізована з використанням ASP.NET Core Identity, що забезпечує надійний механізм управління користувачами та їх ролями. Для авторизації використовується механізм JWT-токенів, який забезпечує безпечну взаємодію між клієнтом та сервером. Кожен запит до захищених ендпоінтів API перевіряється на наявність валідного токену, а також на відповідність користувача доступним ресурсам.

Важливою частиною розробки стала реалізація системи обробки помилок. Створено спеціалізований middleware (рис. 4.6), який перехоплює всі виключення, що виникають під час обробки запитів, та формує уніфіковані відповіді з відповідними HTTP-статусами та описами помилок. Це значно спрощує діагностику проблем та забезпечує зручний формат повідомлень про помилки для клієнтської частини.

```
namespace DEVAI.WebAPI.Middlewares
{
    4 references
    public class ExceptionMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<ExceptionMiddleware> _logger;
        private readonly IHostEnvironment _env;

        0 references
        public ExceptionMiddleware(RequestDelegate next, ILogger<ExceptionMiddleware> logger, IHostEnvironment env)
        {
            this._next = next;
            this._logger = logger;
            this._env = env;
        }

        0 references
        public async Task InvokeAsync(HttpContext context)
        {
            try
            {
                await this._next(context);
            }
            catch (Exception ex)
            {
                this._logger.LogError(ex, ex.Message);
                context.Response.ContentType = "application/json";
                context.Response.StatusCode = StatusCodes.Status500InternalServerError;

                var response = new ProblemDetails
                {
                    Status = StatusCodes.Status500InternalServerError,
                    Detail = this._env.IsDevelopment() ? ex.StackTrace : null,
                    Title = ex.Message
                };

                var options = new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase };

                var json = JsonSerializer.Serialize(response, options);

                await context.Response.WriteAsync(json);
            }
        }
    }
}
```

Рисунок 4.6 – Приклад коду спеціалізованого middleware

Для оптимізації продуктивності реалізовано систему кешування часто запитуваних даних. Використовується вбудований механізм кешування ASP.NET Core з налаштованими політиками інвалідації кешу для різних типів даних. Це дозволило значно зменшити навантаження на базу даних та прискорити відгук системи на повторювані запити.

В процесі розробки активно використовувались можливості ШІ-інструментів. GitHub Copilot виявився особливо корисним при написанні типових CRUD-операцій та базових валідацій. Claude AI використовувався для генерації більш складних компонентів, таких як системи автентифікації та обробки помилок. При цьому всі згенеровані рішення ретельно перевірялися та за необхідності оптимізувалися вручну.

Особлива увага приділялася безпеці даних. Реалізовано механізми захисту від найпоширеніших типів атак, включаючи SQL-ін'єкції, XSS та CSRF-атаки. Всі конфіденційні дані, такі як паролі користувачів, зберігаються в захищеному вигляді з використанням сучасних алгоритмів хешування.

Розгортання серверної частини налаштовано з використанням різних конфігурацій для development та production середовищ. Всі налаштування, включаючи параметри підключення до бази даних та ключі для JWT-токенів, винесені в конфігураційні файли та змінні середовища, що забезпечує гнучкість при розгортанні та підтримці застосунку. Результати розробки серверної частини підтвердили ефективність використання ШІ-інструментів для генерації типового коду та вирішення стандартних задач. При цьому для критично важливих компонентів, таких як система безпеки та обробка конфіденційних даних, використовувався більш консервативний підхід з ретельною перевіркою та тестуванням кожного рішення.

4.4 Інтеграція з OpenAI API

Інтеграція з OpenAI API [4] є ключовим компонентом розробленого веб-застосунку. API (Application Programming Interface) – це набір визначених методів та протоколів, які дозволяють різним програмним компонентам взаємодіяти між собою. У контексті нашого застосунку, OpenAI API надає доступ (рис. 4.7) до потужних мовних моделей, зокрема GPT-4o, що дозволяє генерувати відповіді на запити користувачів та допомагати у вирішенні різноманітних задач розробки.

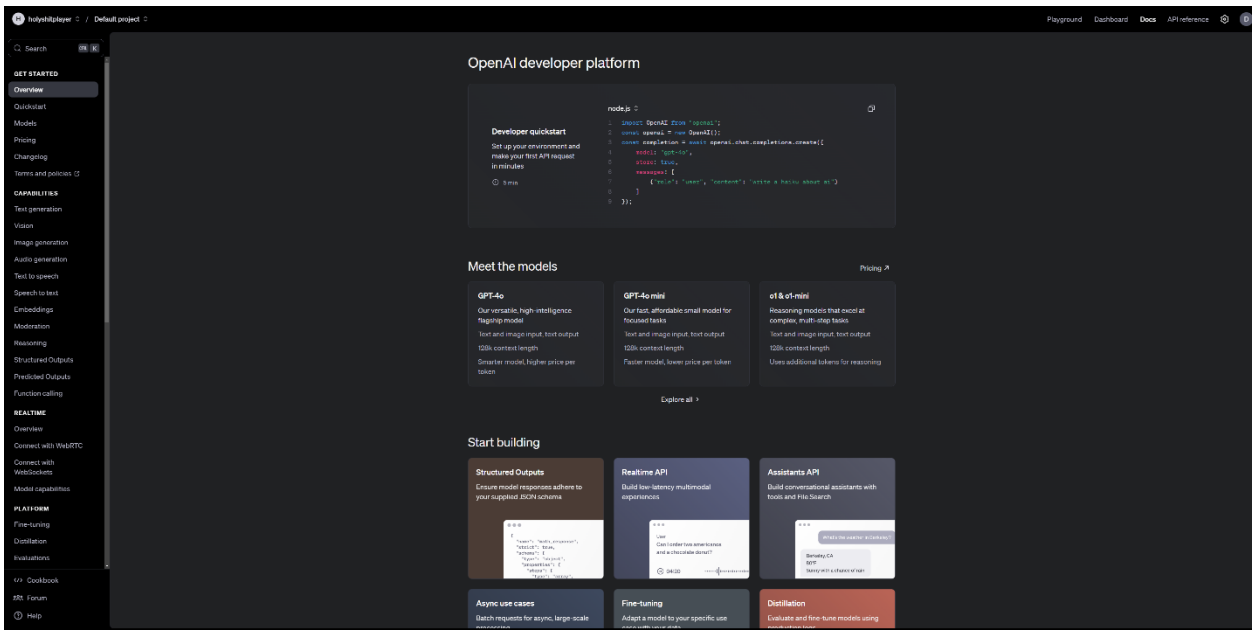


Рисунок 4.7 – Інтерфейс управління доступу до OpenAI API

Важливою частиною роботи з мовними моделями є правильне формування промптів. Промпт – це спеціально сформульований текст, який визначає контекст та напрямок спілкування з моделлю штучного інтелекту. Можна сказати, що промпт виступає своєрідною інструкцією для ШІ, яка визначає, як саме він має інтерпретувати запити користувача та формувати відповіді. В нашому застосунку розроблено спеціалізовані промпти для різних типів задач розробки.

При розробці системи промптів було враховано специфіку кожного напряму роботи. Наприклад, для допомоги з написанням коду промпт налаштовує модель на роль досвідченого розробника, який надає практичні рекомендації та приклади коду з урахуванням сучасних практик програмування. Для роботи з UI компонентами промпт орієнтує модель на експертизу в області розробки інтерфейсів, з акцентом на принципи дизайну та доступності. У процесі інтеграції з OpenAI API було прийнято рішення реалізувати взаємодію безпосередньо з клієнтської частини застосунку. Це рішення базувалося на кількох факторах. По-перше, такий підхід дозволяє зменшити навантаження на наш сервер, оскільки запити до OpenAI надсилаються напряму з браузера користувача. По-друге, це забезпечує більш

швидкий відгук системи, адже відсутня додаткова затримка на обробку запитів нашим сервером.

Для забезпечення ефективної роботи з API було впроваджено систему управління контекстом розмови. Це означає, що модель "пам'ятає" попередні повідомлення в рамках одного діалогу, що дозволяє підтримувати послідовну та змістовну бесіду. Наприклад, якщо користувач обговорює конкретний фрагмент коду, модель враховує цей контекст при наданні подальших рекомендацій та відповідей на уточнюючі запитання.

Особлива увага приділялася обробці різних сценаріїв роботи з API. Система вміє коректно обробляти ситуації, коли сервер OpenAI не відповідає, коли вичерпано ліміт запитів або виникають інші технічні проблеми. В таких випадках користувач отримує зрозуміле повідомлення про помилку та рекомендації щодо подальших дій.

Також було реалізовано систему зворотного зв'язку для покращення користувацького досвіду. Користувач завжди бачить (рис. 4.8), коли система очікує відповіді від API через індикатори завантаження. При отриманні відповіді або виникненні помилки, система відображає відповідні повідомлення, що допомагає користувачу розуміти поточний стан взаємодії з ШІ.

Рисунок 4.8 – Індикатор очікування відповіді від ChatGPT-4o

Важливим аспектом інтеграції стала оптимізація використання API. Оскільки кожен запит до OpenAI має свою вартість, було впроваджено механізми для ефективного використання ресурсів. Це включає оптимальне налаштування параметрів запитів, таких як максимальна кількість токенів та температура генерації, що впливає на креативність та точність відповідей.

Результати впровадження інтеграції з OpenAI API підтвердили ефективність обраного підходу. Користувачі отримали зручний інструмент для вирішення різноманітних задач розробки, від написання коду до створення документації. Спеціалізовані промпти для різних типів задач забезпечують отримання релевантних та корисних відповідей, а пряма інтеграція з клієнтської частини гарантує швидкий та зручний доступ до можливостей штучного інтелекту.

4.5 Підготовка інформаційного вмісту

Підготовка якісного інформаційного наповнення було важливим етапом у розробці веб-застосунку, оскільки система не лише надає інструменти для роботи з ШІ, але й містить освітній компонент щодо використання штучного інтелекту в процесі веб-розробки.

Інформаційний контент застосунку розділено на кілька ключових категорій. В першу чергу, це теоретичні матеріали про інтеграцію ШІ у веб-розробку. Цей розділ містить детальну інформацію про сучасні ШІ-інструменти, їх можливості та особливості застосування. Особлива увага приділяється порівнянню різних підходів та інструментів, таких як GitHub Copilot, Amazon CodeWhisperer та Claude AI.

Другу категорію складають практичні поради та рекомендації, де зібрано практичний досвід використання різних ШІ-інструментів для конкретних задач розробки. Це включає написання та оптимізацію коду, створення та налаштування UI компонентів, розробку тестів та генерацію технічної документації.

Для створення контенту використовувався комбінований підхід, який включав аналіз документації провідних ШІ-інструментів, узагальнення практичного досвіду розробки, використання Claude AI для структурування та оптимізації інформації, а також включення реальних прикладів використання та результатів досліджень.

Візуальне оформлення застосунку реалізовано з використанням мінімалістичного дизайну, що фокусується на функціональності та зручності використання. Кольорова схема розроблена з акцентом на контрастність та читабельність, реалізовано адаптивну типографіку для різних розмірів екранів та інтуїтивно зрозумілу систему навігації.

Особлива увага приділялася створенню інтерактивних прикладів, які демонструють можливості системи. Кожен тип чату супроводжується демонстраційними діалогами, що показують типові сценарії використання та очікувані результати. Система також включає інтерактивні підказки та спливаючі повідомлення, які допомагають користувачам краще зрозуміти функціонал та можливості різних типів чатів. Це особливо важливо для користувачів, які тільки починають працювати з ШІ-інструментами.

Такий комплексний підхід до підготовки інформаційного вмісту забезпечив створення не просто інструменту для роботи з ШІ, а повноцінної освітньої платформи, яка допомагає розробникам ефективно інтегрувати технології штучного інтелекту у свій процес розробки.

4.6 Попереднє тестування web-застосунку

Забезпечення якості та надійності веб-застосунку вимагає комплексного підходу до тестування на всіх етапах розробки. Процес тестування був організований таким чином, щоб гарантувати не лише технічну справність, але й відповідність потребам розробників, які використовуватимуть систему для оптимізації своєї роботи.

В процесі розробки застосовувався принцип поступового тестування, починаючи з окремих компонентів системи. Кожен standalone-компонент Angular проходив ретельну перевірку на відповідність функціональним вимогам, візуальним стандартам та вимогам до взаємодії з іншими частинами системи. Такий підхід дозволив виявляти та усувати потенційні проблеми на найбільш ранніх етапах, що значно знизило витрати на подальше виправлення помилок.

Особлива увага приділялася функціональному тестуванню взаємодії з OpenAI API. Перевірялася коректність формування запитів, обробка різних типів відповідей та поведінка системи при виникненні помилок. Важливим аспектом було тестування роботи спеціалізованих промптів для різних типів задач розробки – від написання коду до створення документації.

Тестування користувацького інтерфейсу включало перевірку:

– зручності навігації між різними типами чатів;

– коректності відображення історії повідомлень;

– роботи системи сповіщень про статус операцій;

– функціонування механізмів збереження контексту розмови;

– коректності роботи системи автентифікації.

Для забезпечення максимальної доступності застосунку було проведено всебічне тестування на різних платформах та пристроях. Тестування охопило основні веб-браузери, включаючи Google Chrome, Mozilla Firefox, Microsoft Edge та Safari. Перевірка проводилася на різних апаратних конфігураціях:

– настільні комп'ютери з різною роздільною здатністю екрану (від 1920x1080 до 4K);

– ноутбуки різних виробників;

– планшети з різними операційними системами;

– мобільні пристрої з різними розмірами екрану.

На всіх тестових платформах застосунок продемонстрував стабільну роботу та коректне відображення інтерфейсу. Особлива увага приділялася перевірці адаптивного дизайну, який забезпечує комфортне використання застосунку на пристроях з різними розмірами екрану.

Процес тестування підтвердив результати експертного дослідження щодо зменшення кількості помилок при використанні ШІ-інструментів. Як і було виявлено в ході дослідження, застосування комбінованого підходу до розробки дозволило знизити кількість помилок на 41,7-50% та суттєво скоротити час на їх виправлення. Всі виявлені в процесі тестування проблеми були ретельно задокументовані та виправлені, що дозволило створити

надійний та зручний інструмент для підтримки процесу веб-розробки з використанням штучного інтелекту.

4.7 Публікація web-застосунку

У межах розробки та тестування веб-застосунку було реалізовано локальне розгортання з використанням сучасних інструментів для створення тестового середовища. Такий підхід дозволив ефективно перевірити функціональність системи в умовах, наближених до реального розгортання.

Для реалізації тестового доступу до застосунку через мережу Інтернет було використано Ngrok [8] – інструмент для створення захищених тунелів до локальних серверів. Цей інструмент надав можливість налаштувати тимчасовий публічний доступ до локально розгорнутого застосунку, що було критично важливим для проведення тестування з різних локацій та пристроїв.

Клієнтська частина застосунку, розроблена на Angular, була розгорнута за допомогою Angular CLI з використанням оптимізованої production-збірки. Процес збірки включав такі етапи оптимізації як мініфікацію коду, tree-shaking для видалення невикористаного коду та попереднє завантаження критичних ресурсів. Усі статичні ресурси, включаючи JavaScript бандли та CSS файли, було оптимізовано для швидкого завантаження.

Серверна частина застосунку, реалізована на ASP.NET Core, була розгорнута в локальному середовищі з налаштованим підключенням до бази даних Microsoft SQL Server. База даних MS SQL Server була розгорнута локально з повною конфігурацією для забезпечення надійного зберігання даних користувачів та історії чатів. Особлива увага приділялася налаштуванням безпеки та оптимізації продуктивності бази даних.

API-ключі та інші чутливі налаштування зберігаються в захищеному вигляді з використанням змінних середовища та конфігураційних файлів, які не включаються до системи контролю версій. Це забезпечує додатковий рівень безпеки при розгортанні застосунку.

Використання Ngrok забезпечило можливість тестування застосунку з різних пристроїв та локацій, що дозволило переконатися в його стабільній роботі в різних умовах мережевого з'єднання. Це також дало змогу провести повноцінне тестування користувацького досвіду в умовах, максимально наближених до реального використання.

4.8 Експертна оцінка розробленого застосунку

Деякі з розроблюваних елементів (рис. 4.9) застосунку було перевірено та оцінено групою експертів з 5 людей, за такими критеріями:

```
import { Injectable } from "@angular/core";

@Injectable({
    providedIn: "root"
})
export class ThemeService {
    THEME_KEY = "selected-theme";
    theme: string = "light";

    constructor() {
        let theme = localStorage.getItem(this.THEME_KEY);
        let dark = window.matchMedia("(prefers-color-scheme: dark)");

        if (theme) {
            this.theme = theme;
        } else if (dark.matches) {
            this.theme = "dark";
        }

        this.updateTheme();
    }

    updateTheme() {
        document.documentElement.setAttribute("data-theme", this.theme);
        localStorage.setItem(this.THEME_KEY, this.theme);
    }

    toggleTheme() {
        if (this.theme === "light") {
            this.theme = "dark";
        } else {
            this.theme = "light";
        }
        this.updateTheme();
    }

    getTheme() {
        return this.theme;
    }
}
```

Рисунок 4.9 – Код написаний традиційним методом (без використання ШІ)

– функціональність (чи виконує код поставлене завдання?);

– ефективність (наскільки код оптимізований?);

– відповідність стилю (наскільки код відповідає стандартам і має єдиний стиль?);

– зрозумілість (наскільки легко зрозуміти код?).

```typescript
import { Injectable } from "@angular/core";
import { BehaviorSubject, Observable } from "rxjs";

@Injectable({
    providedIn: "root"
})
export class ThemeService {
    private readonly THEME_KEY = "selected-theme";
    private currentTheme: BehaviorSubject<"light" | "dark">;

    constructor() {
        this.currentTheme = new BehaviorSubject<"light" | "dark">("light");
        this.initTheme();
    }

    private initTheme(): void {
        const savedTheme = localStorage.getItem(this.THEME_KEY) as "light" | "dark";
        const prefersDark = window.matchMedia("(prefers-color-scheme: dark)").matches;
        const theme = savedTheme || (prefersDark ? "dark" : "light");

        this.currentTheme.next(theme);
        this.applyTheme(theme);
    }

    private applyTheme(theme: "light" | "dark"): void {
        localStorage.setItem(this.THEME_KEY, theme);
        document.documentElement.setAttribute("data-theme", theme);
    }

    public toggleTheme(): void {
        const newTheme = this.currentTheme.getValue() === "light" ? "dark" : "light";
        this.currentTheme.next(newTheme);
        this.applyTheme(newTheme);
    }

    public getCurrentTheme(): Observable<"light" | "dark"> {
        return this.currentTheme.asObservable();
    }
}
```

Рисунок 4.10 – Код написаний з використанням GitHub Copilot

В таблиці 4.1 наведено показники ваги обраних критеріїв та коефіцієнт узгодженості експертів.

```typescript
import { Injectable, signal } from "@angular/core";

type Theme = "light" | "dark";

@Injectable({
    providedIn: "root"
})
export class ThemeService {
    private readonly THEME_KEY = "selected-theme";
    private readonly currentTheme = signal<Theme>("light");

    constructor() {
        this.initializeTheme();
    }

    toggleTheme(): void {
        const newTheme = this.currentTheme() === "light" ? "dark" : "light";
        this.setTheme(newTheme);
    }

    getCurrentTheme(): Theme {
        return this.currentTheme();
    }

    private initializeTheme(): void {
        const savedTheme = localStorage.getItem(this.THEME_KEY) as Theme || "light";
        const prefersDark = window.matchMedia("(prefers-color-scheme: dark)").matches;

        this.setTheme(savedTheme || (prefersDark ? "dark" : "light"));
    }

    private setTheme(theme: Theme): void {
        this.currentTheme.set(theme);
        localStorage.setItem(this.THEME_KEY, theme);
        document.documentElement.setAttribute("data-theme", theme);
    }
}
```

Рисунок 4.11 – Код написаний з використанням Claude AI

Таблиця 4.1 – Критерії оцінки

Критерії	Оцінка експерта					Строкова сума	Відх від строков.	Кв. відх.	Коеф. конкорд	Вага
	1	2	3	4	5					
Функціональність	3	3	3	4	3	16	-3.5	12.25	0.098	0.32
Ефективність	4	4	4	3	4	19	-6.5	42.25	0.338	0.38
Відповідність стилю	1	2	2	1	2	8	4.5	20.25	0.162	0.16
Зрозумілість	2	1	1	2	1	7	5.5	30.25	0.242	0.14
Сума						50		109		1

Для визначення коефіцієнту конкордації, тобто узгодженості експертів необхідно зробити розрахунки за (4.1).

$$W = \sqrt{\frac{12 \times S}{n^2 \times (m^3 - m)}}, \quad (4.1)$$

де m – кількість альтернатив,

n – кількість експертів.

Виходячи з отриманих результатів у таблиці, маємо що коефіцієнт конкордації до 1. Отже можна казати про достатню узгодженість думок експертів. У таблицях 4.2-4.5 наведено результати розрахунків за кожним з обраних критеріїв для кожного види написаного коду.

Таблиця 4.2 – Оцінка функціональності

	Оцінка експерта					Строкова сума	Відх від строков.	Кв. відх.	Коеф. конкорд
	1	2	3	4	5				
Код 1	1	1	2	1	1	6	4	16	0.32
Код 2	2	2	1	2	3	10	0	0	0
Код 3	3	3	3	3	2	14	-4	16	0.32

Таблиця 4.3 – Оцінка ефективності

	Оцінка експерта					Строкова сума	Відх від строков.	Кв. відх.	Коеф. конкорд
	1	2	3	4	5				
Код 1	1	2	2	1	2	8	2	4	0.08
Код 2	2	1	1	2	1	7	3	9	0.18
Код 3	3	3	3	3	3	15	-5	25	0.5

Таблиця 4.4 – Оцінка відповідності стилю

	Оцінка експерта					Строкова сума	Відх від строков.	Кв. відх.	Коеф. конкорд
	1	2	3	4	5				
Код 1	3	1	2	2	1	9	1	1	0.02
Код 2	2	2	1	1	2	8	2	4	0.08
Код 3	1	3	3	3	3	13	-3	9	0.18

Таблиця 4.5 – Оцінка зрозумілості

	Оцінка експерта					Строкова сума	Відх від строков.	Кв. відх.	Коеф. конкорд
	1	2	3	4	5				
Код 1	1	1	1	2	1	6	4	16	0.32
Код 2	2	2	2	1	2	9	1	1	0.02
Код 3	3	3	3	3	3	15	-5	25	0.5

По результатам оцінки думки експертів визначено узгодженими.

У таблиці 4.6 наведено зведені результати що до оцінювання експертами розробленого елементу трьома способами.

Таблиця 4.6 – Результати

	Код 1	Код 2	Код 3
Функціональність	1.2	2	2.8
Ефективність	1.6	1.4	3
Відповідність стилю	1.8	1.6	2.6
Зрозумілість	1.2	1.8	3

Після отриманих результатів можна сказати що на основі проведеного експертного оцінювання трьох варіантів реалізації коду можна зробити висновок, що код, написаний з використанням Claude AI (Код 3), отримав найвищі оцінки за всіма критеріями – від 2,6 до 3 балів. GitHub Copilot (Код 2) показав середні результати з оцінками від 1,4 до 2 балів, а код, написаний традиційним способом (Код 1), отримав найнижчі оцінки майже за всіма критеріями, окрім відповідності стилю.

Отримані результати демонструють, що використання інструментів штучного інтелекту, особливо Claude AI, дозволяє створювати більш якісний та ефективний код порівняно з традиційним підходом до розробки.

5 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ШІ-ІНСТРУМЕНТІВ

5.1 Вибір та використання ШІ-інструментів

На основі проведеного дослідження та практичного досвіду розробки веб-застосунку можна сформулювати методику ефективного вибору та використання ШІ-інструментів у процесі веб-розробки. Важливо розуміти, що кожен проєкт має свої особливості та вимоги, тому підхід до вибору інструментів повинен бути гнучким та адаптивним.

При виборі ШІ-інструментів для веб-розробки необхідно враховувати декілька ключових факторів. Першим фактором є масштаб та складність проєкту. Для невеликих проєктів достатньо використання базових інструментів автодоповнення коду та простих ШІ-асистентів. Для великих проєктів рекомендується комплексний підхід з використанням спеціалізованих інструментів для різних аспектів розробки.

Другим важливим фактором є технологічний стек. Різні ШІ-інструменти мають різний рівень підтримки різних мов програмування та фреймворків. Наприклад, GitHub Copilot демонструє найкращі результати при роботі з популярними технологіями, такими як JavaScript, Python та Java.

Третім фактором є бюджетні обмеження. Необхідно враховувати вартість ліцензій та підписок на різні інструменти. Для невеликих команд можуть бути більш доцільними безкоштовні альтернативи або комбінація платних та безкоштовних інструментів.

Четвертим ключовим фактором є вимоги до безпеки. При роботі з конфіденційними даними важливо обирати інструменти, які забезпечують належний рівень захисту коду та інформації. У таких випадках може бути доцільним використання локальних рішень замість хмарних сервісів.

На основі цих факторів розроблено поетапну методику вибору інструментів. На першому етапі проводиться аналіз вимог проєкту:

визначення основних напрямків використання ШІ, оцінка обсягу та складності задач, аналіз технологічного стеку, визначення бюджетних обмежень.

На другому етапі відбувається оцінка доступних інструментів: складання списку потенційних інструментів, аналіз їх можливостей та обмежень, оцінка вартості використання, перевірка сумісності з обраними технологіями.

Третій етап включає тестування обраних інструментів: створення тестових проєктів, оцінка швидкості та якості розробки, аналіз зручності використання, виявлення потенційних проблем.

На четвертому етапі відбувається прийняття рішення та впровадження: вибір оптимальної комбінації інструментів, розробка стратегії впровадження, налаштування робочого процесу, навчання команди.

Для різних типів проєктів рекомендуються різні комбінації інструментів. Для невеликих проєктів достатньо використання GitHub Copilot для основної розробки та ChatGPT для консультацій та документації. Для середніх проєктів рекомендується використання GitHub Copilot для розробки, Amazon CodeWhisperer для оптимізації та Claude AI для складних архітектурних рішень. Великі проєкти зазвичай потребують повного набору спеціалізованих інструментів, власних налаштувань та інтеграцій, а також комбінації різних ШІ-сервісів.

5.2 Рекомендації щодо навчання розробників

Ефективне використання ШІ-інструментів вимагає системного підходу до навчання розробників. На основі досвіду впровадження таких інструментів можна виділити кілька ключових напрямків навчання.

Базова підготовка повинна включати розуміння принципів роботи ШІ, знання можливостей та обмежень різних інструментів, навички ефективного формулювання запитів та розуміння безпекових аспектів.

Практичні навички мають охоплювати роботу з різними типами ШІ-асистентів, ефективне використання автодоповнення коду, валідацію та

оптимізацію згенерованого коду, інтеграцію ШІ-інструментів у робочий процес. Програма навчання повинна включати теоретичну частину, що охоплює основи роботи нейронних мереж, принципи роботи мовних моделей, огляд сучасних ШІ-інструментів та методи оптимізації запитів.

Практична частина має складатися з роботи з реальними проєктами, вирішення типових задач, аналізу складних випадків та оптимізації робочого процесу. Важливим компонентом є перевірка знань через тестові завдання, практичні проєкти, аналіз результатів та отримання зворотного зв'язку.

У даному розділі було розроблено комплексні рекомендації щодо впровадження ШІ-інструментів у процес веб-розробки. Запропонована методика вибору та використання інструментів враховує різні аспекти розробки та може бути адаптована під конкретні потреби проєктів. Особлива увага приділена питанням навчання розробників, як ключовому фактору успішного впровадження нових технологій.

Розроблені рекомендації базуються на практичному досвіді створення веб-застосунку та враховують як технічні, так і організаційні аспекти впровадження ШІ-інструментів. Запропоновані підходи до навчання розробників забезпечують системне засвоєння необхідних знань та навичок.

Отримані результати можуть бути використані як основа для розробки конкретних планів впровадження ШІ-інструментів у різних організаціях та проєктах. При цьому важливо враховувати специфіку конкретної ситуації та адаптувати рекомендації відповідно до наявних потреб та обмежень.

6 ЕКОНОМІЧНА ЧАСТИНА

6.1 Характеристика науково-дослідної роботи

Метою даного розділу є економічне обґрунтування витрат на проведення науково-дослідної роботи (НДР), в межах якої досліджується використання інструментів штучного інтелекту для оптимізації процесу розробки веб-сайтів, їх вплив на зменшення витрат часу та коштів при створенні веб-проєктів.

У роботі досліджено сучасні методи та інструменти штучного інтелекту, що застосовуються при розробці веб-сайтів, проаналізовано проблеми та обмеження існуючих рішень, розроблено та реалізовано веб-застосунок для інтеграції ШІ-інструментів у процес розробки. На основі експериментальних даних проведено розрахунок економічної ефективності запропонованого рішення.

Під час такого обґрунтування буде здійснено: розрахунок трудовитрат та заробітної плати працівникам, розрахунок одноразових витрат і прибутку, оцінку результатів НДР.

Етапи виконання НДР:

– аналіз існуючих ШІ-інструментів для розробки веб-сайтів;
– визначення методології інтеграції ШІ в процес розробки;
– проєктування та розробка веб-застосунку;
– інтеграція API OpenAI та налаштування спеціалізованих промптів;
– тестування розробленого рішення;
– проведення експериментів для оцінки ефективності;
– аналіз та оцінка результатів НДР.

6.2 Етапи виконання НДР, їх трудомісткість та заробітна плата

Умовно НДР можна розділити на три послідовні етапи: підготовчий, основний і заключний.

На підготовчому етапі було проведено аналіз предметної області та існуючих рішень. Здійснено огляд сучасних інструментів штучного інтелекту, що використовуються при розробці веб-сайтів, включаючи GitHub Copilot, Amazon CodeWhisperer та інші. Визначено їх переваги, недоліки та обмеження. Сформовано критерії оцінки ефективності ШІ-інструментів, які включають швидкість розробки, якість генерованого коду, зручність інтеграції та економічну доцільність.

В основній частині було спроєктовано та розроблено веб-застосунок з використанням Angular, ASP.NET Core та MS SQL Server. Реалізовано інтеграцію з API OpenAI та налаштовано спеціалізовані промпти для різних аспектів розробки – написання коду, тестування, створення документації. Проведено тестування розробленого рішення на реальних проєктах для оцінки його ефективності порівняно з традиційними методами розробки.

У заключній частині виконано аналіз отриманих результатів, розраховано економічний ефект від впровадження розробленого рішення, підготовлено та захищено фінальний звіт з НДР. Дану роботу виконували троє фахівців: веб-розробник, інженер-дослідник та ШІ-інженер. За даними сайту dou.ua, середня заробітна плата веб-розробника становить 120 000,00 грн, інженера-дослідника – 140 000,00 грн, ШІ-інженера – 130 000,00 грн.

Проведемо розрахунок трудовитрат і заробітної плати виконавців робіт. Середньоденна заробітна плата виконавця робіт (Зср.дн.):

$$З_{ср.дн.} = \frac{З_{ср.міс.}}{n}, \tag{6.1}$$

де $З_{ср.дн.}$ – середньомісячна зарплата виконавця роботи;

n – число робочих днів у місяці, (n = 22).

Підставивши дані до формули (6.1), отримаємо середньоденну заробітну плату веб-розробника у розмірі 5454,55 грн, інженера-дослідника – 6363,64 грн, ШІ-інженера – 5909,09 грн.

Етапи виконання НДР, перелік і зміст робіт, трудомісткість їх виконання, заробітна плата виконавців робіт представлені у таблиці 6.1.

Таблиця 6.1 – Розрахунок трудовитрат і заробітної плати виконавців робіт

№	Перелік робіт	Кіль-кість вико-навців	Посада виконавця	Трудо-місткість робіт, люд.-днів	Середньо-денна заро-бітна плата, грн	Сума заро-бітної плати, грн
1	Аналіз ринку ШІ-інструментів	1	Інженер-дослідник	5	6363,64	31818,20
2	Розробка методології інтеграції ШІ	1	Інженер-дослідник	3	6363,64	19090,92
3	Проєктування архітектури застосунку	1	Веб-розробник	4	5454,55	21818,20
4	Розробка базової версії застосунку	1	Веб-розробник	8	5454,55	43636,40
5	Інтеграція OpenAI API	1	ШІ-інженер	2	5909,09	11818,18
6	Налаштування промптів	1	ШІ-інженер	2	5909,09	11818,18
7	Тестування та оптимізація	2	Веб-розробник, ШІ-інженер	3	5454,55 5909,09	34090,92
8	Аналіз результатів	1	Інженер-дослідник	2	6363,64	12727,28
	Усього:			29		186818,28

Таким чином, сума витрат на заробітну плату в межах виконання НДР складе 186818,28 грн.

6.3 Розрахунок одноразових витрат на розробку НДР

Калькуляція собівартості розраховується відповідно до існуючих нормативних актів України. До складу калькуляції входять такі статті витрат:

– матеріальні витрати;

– витрати на оплату праці;

– єдиний соціальний внесок;

– амортизація основних засобів (вартість машинного часу);

– витрати на спожиту електроенергію;

– інші витрати.

Визначення повної собівартості науково-дослідної роботи вимагає розрахунку всіх витрат, пов'язаних з її виконанням. Відповідно до діючих нормативів, калькуляція має враховувати всі види витрат, які виникають під час проведення дослідження та розробки проєкту.

Основною статтею витрат є оплата праці членів команди розробників. За результатами попередніх розрахунків, ця сума складає 186818,28 грн. Дані витрати включають заробітну плату трьох ключових спеціалістів – веб-розробника, інженера-дослідника та ШІ-інженера, які працювали над проєктом протягом 29 робочих днів.

Згідно з чинним законодавством України, на фонд оплати праці нараховується єдиний соціальний внесок (ЄСВ) за ставкою 22 %. Таким чином, сума відрахувань на соціальне страхування складає 41100,02 грн.

Важливою складовою витрат є оплата електроенергії, спожитої комп'ютерним обладнанням під час розробки. Для реалізації проєкту використовувалося три робочі станції із середньою споживаною потужністю 0,6 кВт/год кожна. При поточному тарифі на електроенергію 4,32 грн/кВт×год та 8-годинному робочому дні, загальні витрати на електроенергію (Ве):

$$B_e = M \cdot t \cdot T_{кВт}, \tag{6.2}$$

де М – потужність устаткування, тобто кількість енергії, споживаної за одиницю часу (кВт/година);

t – кількість годин використання устаткування за період проведення науково-дослідницької роботи;

$T_{кВт}$ – тариф, тобто вартість використання 1 кВт електроенергії.

Підставивши значення у (6.2), визначимо величину витрат ($В_е$) на спожиту електроенергію:

$$B_e = (0{,}6\times80\times4{,}32) + (0{,}6\times120\times4{,}32) + (0{,}6\times56\times4{,}32) = 663{,}55 \text{ грн.}$$

Розрахунок витрат на обслуговування комп'ютерного обладнання базується на двох ключових параметрах: початковій вартості обладнання та періоді його експлуатації до заміни. Термін експлуатації зазвичай становить не більше трьох років, при цьому обладнання використовується протягом 254 робочих дні на рік. На основі цих параметрів розраховується річна амортизація основних засобів:

$$AB = \sum_{k=1}^{L} \frac{BO_k}{TE_k} \times T \,, \tag{6.3}$$

$$AB = \frac{105000{,}00\times29}{762} = 3996{,}06 \text{ грн.}$$

де АВ – сума амортизаційних відрахувань, нарахованих під час проведення НДР;

BO_k – вартість основних засобів k-го виду;

TE_k – термін експлуатації основних засобів k-го виду, днів;

Т – термін НДР, днів;

L – кількість видів обладнання.

При виконанні НДР застосовувалися 3 комп’ютери вартістю по 35000,00 грн кожен. Підставивши відомі значення у (6.3), визначимо величину амортизаційних відрахувань:

Калькуляція собівартості розраховується відповідно до існуючих нормативних актів України. До складу калькуляції входять основні статті витрат, які включають матеріальні витрати, витрати на оплату праці, єдиний

соціальний внесок, амортизацію основних засобів, витрати на електроенергію та інші додаткові витрати.

У структурі інших витрат важливе місце посідають адміністративні витрати, які охоплюють витрати на водопостачання, водовідведення, освітлення та опалення. Відповідно до прийнятих нормативів, адміністративні витрати встановлюються у розмірі 20 % від загального фонду оплати праці, що в даному випадку становить 37363,66 грн від базової суми 186818,28 грн.

Для реалізації науково-дослідної роботи було залучено сучасне програмне забезпечення та онлайн-сервіси. Зокрема, розробка проєкту потребувала використання середовища Visual Studio з місячною ліцензією вартістю 1900,00 грн. Важливим компонентом проєкту стала інтеграція з API OpenAI, витрати на яке склали 3000,00 грн на місяць. Для тестування функціоналу було орендовано хостинг вартістю 800,00 грн на квартал. Забезпечення стабільного інтернет-зв'язку коштувало 350,00 грн.

За період виконання науково-дослідної роботи не виникало потреби у відрядженнях, залученні сторонніх виконавців чи проведенні маркетингових заходів. Матеріальні витрати також були відсутні.

Детальний розрахунок усіх витрат на виконання науково-дослідної роботи представлено у таблиці 6.2.

Таблиця 6.2 − Кошторис витрат на розробку НДР

№ з/п	Стаття витрат	Сума, грн
1	Заробітна плата	186818,28
2	Єдиний соціальний внесок (22 % від п.1)	41100,02
3	Матеріальні витрати	-
4	Амортизація основних засобів	3996,06
5	Витрати на спожиту електроенергію	663,55
6	Інші витрати, у тому числі:	
6.1	Адміністративні витрати (20% від п.1)	37363,66
6.2	Ліцензія Visual Studio	1900,00
6.3	Плата за використання API OpenAI	3000,00
6.4	Хостинг	800,00
6.5	Послуги інтернету	350,00
	Усього витрати на розробку (*Вр*)	275991,57

Таким чином, кошторис витрат на виконання даної НДР складає 266884,2 грн.

6.4 Оцінка результатів науково-дослідної роботи

Результат – це підсумковий наслідок послідовності виконаних дій, що може бути виражений як якісними, так і кількісними показниками. Загальна оцінка результатів науково-дослідної роботи полягає у визначенні ефективності отриманих рішень порівняно з існуючим науково-технічним рівнем.

Відповідно до теми даного дослідження, ключовим результатом впровадження НДР є оптимізація процесу розробки веб-сайтів через використання інструментів штучного інтелекту. Для оцінки ефективності розробленого рішення було проведено порівняльний аналіз часу, необхідного для виконання типових завдань веб-розробки із застосуванням традиційних методів та з використанням розробленого ШІ-інструменту.

Результат від впровадження НДР визначається за формулою:

$$\Delta P_j = \left| Хб_j - Хн_j \right|, \tag{6.4}$$

де ΔP_j – покращення j-ої характеристики досліджуваного процесу за рахунок впровадження результатів НДР (j = 1,m);

m – кількість досліджуваних характеристик;

$Хб_j$ – базове значення j-ої характеристики;

$Хн_j$ – нове значення j-ої характеристики після впровадження НДР.

В експериментальній частині було проведено аналіз часових витрат на виконання ключових завдань розробки веб-сайту. Для забезпечення об'єктивності оцінки вимірювання проводились для різних типів завдань з різним рівнем складності. Результатами тестування є зафіксований час для виконання певного типу завдань над компонентами веб-застосунку. Кількість

компонентів залежить від розміру проєкту. У даному випадку кількість компонентів – 40, тому в таблиці наведено результати вимірювань для одного компонента та для усіх компонентів веб-застосунку. Результати тестування наведено у таблиці 6.3.

Таблиця 6.3 – Час, необхідний для виконання задач у СКВ

Тип завдання	Традиційний підхід (хв)	З використанням ШІ (хв)	Традиційний підхід для всіх компонентів (хв)	З використанням ШІ для всіх компонентів (хв)
Створення компоненту зі стилями	15-20	5-10	600-800	200-400
Написання складної логіки компонентів	60-120	20-65	2400-4800	800-2600
Наповнення компоненту вмістом	40-60	10-15	1600-2400	400-600
Написання документації до компоненту	30-45	7-10	1200-1800	280-400

Підставивши відповідні значення до формули (6.4), визначимо результат від впровадження НДР у чисельному вигляді:

$$\Delta P_{створення\ компонентів} = |700\text{-}300| = 400 \text{ хв},$$

$$\Delta P_{\text{написання складної логіки}} = |3600\text{-}1700| = 1900 \text{ хв},$$

$$\Delta P_{\text{наповнення вмістом}} = |2000\text{-}500| = 1500 \text{ хв},$$

$$\Delta P_{\text{написання документації}} = |1500\text{-}340| = 1160 \text{ хв}.$$

На основі проведених розрахунків можна зробити висновок, що використання розробленого ШІ-інструменту забезпечує значне скорочення часу на виконання всіх основних завдань веб-розробки. Найбільш суттєве покращення спостерігається при написанні складної логіки компонентів (економія 300 хвилин або 52,8 % часу) та наповненні компонента вмістом (економія 37,5 хвилин або 75 % часу). Створення компонентів прискорюється

в середньому на 10 хвилин (57,1 % часу), а написання документації – на 29 хвилин (77,3 % часу). Загальна економія часу на повному циклі розробки одного компонента складає 124 хвилини, що в середньому становить 65,6 % порівняно з традиційним підходом. На створення усіх компонентів було зекономлено близько 5000 хвилин.

Такі результати свідчать про високу ефективність розробленого рішення та його значний потенціал для оптимізації процесів веб-розробки.

6.5 Визначення економічної ефективності результатів НДР

Для оцінки економічної ефективності результатів науково-дослідної роботи необхідно зіставити витрати на її проведення з отриманими результатами. Основним критерієм для оцінки ефективності слугує коефіцієнт "ефект-витрати", який обчислюється за формулою:

$$K_{ев} = \frac{\Delta P_j}{B_p}, \tag{6.5}$$

де B_p – витрати (кошторисна вартість) на виконання НДР, грн;

$K_{ев}$ – коефіцієнт «ефект-витрати», який відбиває, наскільки кожна гривня витрат НДР змінює j-ту характеристику досліджуваного процесу.

Підставивши раніше визначені значення до (6.5), розрахуємо чисельне значення коефіцієнту «ефект-витрати» розробленого рішення порівняно з загальними рішеннями:

$$\text{К}_{\text{ев}(\textit{створення компонентів})} = \frac{400}{275991{,}57} \times 100\ \% = 0{,}145\ \%,$$

$$\text{К}_{\text{ев}(\text{написання складної логіки})} = \frac{1900}{275991{,}57} \times 100\ \% = 0{,}688\ \%,$$

$$К_{ев(наповнення\ вмістом)} = \frac{1500}{275991,57} \times 100\ \% = 0,543\ \%,$$

$$К_{ев(написання\ документації)} = \frac{1160}{275991,57} \times 100\ \% = 0,420\ \%.$$

У результаті проведеного дослідження можна констатувати, що впровадження ШІ-інструментів для веб-розробки демонструє позитивну економічну ефективність. Це впровадження підтверджує, що кожна інвестована гривня забезпечує оптимізацію часових витрат. Розрахунки показують, що коефіцієнт ефективності становить: 0,145 % при створенні компонентів, 0,688 % при розробці складної логіки, 0,543 % при наповненні контентом та 0,420 % при написанні документації. Загальний показник ефективності дорівнює близько 1,796 %, що свідчить про високу практичну цінність використання ШІ-інструментів у веб-розробці. Роботу можна вважати ефективною та такою, що має науковий і технічний рівень.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було проведено комплексне дослідження процесу розробки веб-сайтів з використанням інструментів на базі штучного інтелекту, що дозволило досягти поставленої мети та вирішити всі визначені завдання.

Проведено глибокий аналіз сучасного стану розвитку штучного інтелекту та його впливу на процеси веб-розробки. Досліджено та порівняно ефективність різних ШІ-асистентів (GitHub Copilot, Claude AI) у розробці веб-сайтів. Розроблено та впроваджено методику комплексного використання різних ШІ-інструментів у процесі веб-розробки.

В рамках практичної частини розроблено веб-застосунок з використанням сучасного стеку технологій (Angular, ASP.NET Core, MS SQL Server). При розробці застосовано комбінований підхід до використання ШІ-інструментів, що дозволило досягти значного підвищення ефективності розробки при збереженні високої якості коду. Реалізовано інтеграцію з OpenAI API для створення спеціалізованих чат-інтерфейсів з різними аспектами веб-розробки.

На основі проведеного дослідження розроблено практичні рекомендації щодо вибору та використання ШІ-інструментів у веб-розробці. Рекомендації враховують результати експертного оцінювання та пропонують конкретні підходи до ефективної інтеграції ШІ-інструментів у процеси розробки.

Результати дослідження створюють основу для подальшого вивчення можливостей використання штучного інтелекту в розробці програмного забезпечення та можуть бути використані при впровадженні ШІ-інструментів у різних організаціях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки з виконання кваліфікаційної роботи здобувачів вищої освіти на другому (магістерському) рівні для студентів усіх форм навчання спеціальності 186 «Видавництво та поліграфія» / Н.Є. Кулішова, В.П. Ткаченко. Харків: ХНУРЕ, 2020. 51 с.

2. Методичні рекомендації до виконання економічної частини дипломних проектів, робіт для студентів денної та заочної форми навчання усіх спеціальностей / Л.В. Соколова, О.І. Горбач, С.В. Гришко, Є.В. Діденко, Л.В. Левченко, Г.М. Путятіна, В.Г. Харченко. Харків: ХНУРЕ, 2015. 49 с.

3. Copilot G. Документація GitHub Copilot. URL: https://docs.github.com/en/copilot (дата звернення: 06.01.2025).

4. Документація OpenAI API. URL: https://platform.openai.com/docs/ (дата звернення: 08.01.2025).

5. Посібник з розробки Angular. URL: https://angular.dev/overview (дата звернення: 04.01.2025).

6. Документація ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 05.01.2025).

7. Документація Claude AI. URL: https://docs.anthropic.com/claude/ (дата звернення: 16.01.2025).

8. Публікація локального веб-застосунку у мережу за допомогою Ngrok. URL: https://ngrok.com/docs/getting-started/ (дата звернення: 14.01.2025).

9. Посібник зі структурування та проєктування бази даних. URL: https://www.lucidchart.com/pages/database-diagram/database-design (дата звернення: 06.01.2025).