Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістра
(освітньо-кваліфікаційний рівень)

на тему

Створення веб-застосунку для аудиту програмного коду з використанням штучного інтелекту

Виконав: студент 6 курсу, групи 602-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Маслак М.І.
(прізвище та ініціали)
Керівник Капітон А.М.
(прізвище та ініціали)
Консультант Скакаліна О.В.
(прізвище та ініціали)
Рецензент Яковенко А.А.
(прізвище та ініціали)

Полтава – 2025 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

спеціальність 122 «Комп'ютерні науки»

на тему

«Створення веб-застосунку для аудиту програмного коду з використанням штучного інтелекту»

Студента групи 602-ТН Маслака Михайла Ігоровича

Керівник роботи
д.е.н., професор Капітон А.М.

Консультант
Кандидат технічних наук,
доцент Скакаліна О.В.

Завідувач кафедри
Кандидат фізико-математичних наук,
доцент Двірна О.А.

Полтава – 2025 року

РЕФЕРАТ

Кваліфікаційна робота магістра: 99 с., 64 малюнка, 1 додаток, 23 літературних джерела.

Об’єкт дослідження: створення веб-застосунку для аудиту програмного коду з використанням штучного інтелекту.

Мета роботи: метою дипломної роботи є дослідження, демонстрація можливостей та переваг використання штучного інтелекту для аудиту програмного коду через створення веб-застосунку. Цей веб-застосунок допомагатиме розробникам автоматизовано перевіряти якість, безпеку та ефективність їхнього коду, а також надавати рекомендації для його покращення.

Для того, щоб розробити веб-застосунок для аудиту коду, було проведено аналіз літературних джерел, досліджень та сучасних рішень у галузі автоматизованого аналізу коду. Проєктування та розробка веб-застосунку включала створення інтерфейсу користувача, його структури та функціоналу, використовуючи такі технології, як React JS, Next.js, Tailwind CSS та TypeScript.

Ключовою частиною розробки було інтегрування OpenAI API для забезпечення можливості аналізу програмного коду з використанням методів штучного інтелекту. Ця технологія дозволяє отримувати глибокий аналіз коду з акцентом на виявлення вразливостей, покращення продуктивності та відповідність стандартам якості.

Ключові слова: веб-застосунок, аудит програмного коду, штучний інтелект, React, Next.js, Tailwind CSS, OpenAI API, TypeScript, аналіз коду, якість коду, безпека, продуктивність, автоматизація, тестування застосунку, розробка інтерфейсу.

ABSTRACT

Master's thesis: 99 pages, 64 figures, 1 appendice, 23 sources.

Object of research: creation of a web application for auditing software code using artificial intelligence.

Purpose: The purpose of the thesis is to study, demonstrate the possibilities and benefits of using artificial intelligence to audit program code by creating a web application. This web application will help developers to automatically check the quality, security, and efficiency of their code, as well as provide recommendations for its improvement.

In order to develop a web application for code auditing, we analysed literature, research, and current solutions in the field of automated code analysis. The design and development of the web application included the creation of the user interface, its structure and functionality, using technologies such as React JS, Next.js, Tailwind CSS and TypeScript.

A key part of the development was the integration of the OpenAI API to enable the analysis of programme code using artificial intelligence methods. This technology provides in-depth code analysis with a focus on identifying vulnerabilities, improving performance, and meeting quality standards.

Keywords: web application, software code audit, artificial intelligence, React, Next.js, Tailwind CSS, OpenAI API, TypeScript, code analysis, code quality, security, performance, automation, application testing, interface development.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект, технології, що дозволяють машинам виконувати завдання, які зазвичай потребують людського інтелекту, такі як навчання, розуміння мови та прийняття рішень.

ML – Machine Learning – метод штучного інтелекту, що дозволяє комп'ютерам навчатися з даних без явного програмування.

NLP – Natural Language Processing – обробка природної мови, підгалузь AI, що займається взаємодією між комп'ютерами та людською мовою.

JavaScript – скриптова мова програмування, використовується для створення інтерактивності на веб-сторінках.

TypeScript – надмножина JavaScript, що додає статичну типізацію, що дозволяє виявляти помилки на етапі компіляції та полегшує розвиток великих проєктів.

React – JavaScript бібліотека для побудови користувацьких інтерфейсів, що дозволяє створювати складні веб-застосунки за допомогою компонентного підходу.

Next.js – фреймворк для React, що спрощує розробку серверних і статичних веб-додатків.

Tailwind CSS – утилітарний CSS-фреймворк, що дозволяє швидко створювати кастомізовані дизайни за допомогою класів.

OpenAI API – інтерфейс програмування додатків, що надає доступ до потужних моделей штучного інтелекту для аналізу та генерації тексту.

Code Quality – якість програмного коду, що визначає, наскільки код є зрозумілим, підтримуваним і безпечним.

Security Vulnerabilities – вразливості безпеки в програмному забезпеченні, які можуть бути використані зловмисниками для компрометації системи.

Automated Code Review – автоматизований процес перевірки коду на наявність помилок, стилістичних проблем та вразливостей за допомогою спеціалізованих інструментів.

Performance Optimization – оптимізація продуктивності веб-додатків, спрямована на покращення швидкості завантаження та загальної ефективності програми.

Frontend – клієнтська частина веб-додатку, що відповідає за взаємодію з користувачем.

Backend – серверна частина веб-додатку, що обробляє запити та управляє даними.

API Integration – інтеграція інтерфейсів програмування додатків для забезпечення взаємодії між різними системами.

User Experience (UX) – досвід користувача, що описує, як користувачі взаємодіють з продуктом, включаючи їх задоволення і комфорт.

User Interface (UI) – користувацький інтерфейс, що визначає, як інформація відображається та взаємодіє з користувачами

Quality Assurance (QA) – процес забезпечення якості програмного забезпечення, що включає перевірку коду, тестування та виправлення помилок.

SaaS – Software as a Service – модель надання програмного забезпечення через Інтернет, яка дозволяє користувачам отримувати доступ до додатків без необхідності їх інсталяції.

ВСТУП

Сучасні технології розробки програмного забезпечення постійно вдосконалюються і штучний інтелект (ШІ) займає все більш значуще місце у цій сфері. Завдяки швидкому розвитку ШІ, автоматизація аналізу та перевірки програмного коду стає важливим аспектом як для малих, так і великих розробницьких команд.

Виявлення помилок на ранніх етапах, забезпечення безпеки та підвищення якості коду є критично важливими завданнями для сучасного програмного забезпечення, що зумовлює актуальність створення інструментів для аудиту коду.

Використання штучного інтелекту для аналізу програмного коду дозволяє скоротити час на ручну перевірку, виявляти складні вразливості та покращувати продуктивність програмних продуктів. Веб-застосунки, які використовують інтелектуальні технології для автоматизованого аналізу коду, допомагають підвищити ефективність розробників та забезпечити більш високий рівень якості коду. Це особливо важливо в умовах сучасного ринку програмного забезпечення, де час і якість є ключовими факторами успіху.

Основна мета цього випускного кваліфікаційного проєкту полягає у створенні веб-застосунку для аудиту програмного коду з використанням штучного інтелекту, який допоможе розробникам автоматично аналізувати код на відповідність стандартам якості, безпеки та продуктивності. Застосунок проводитиме детальний аналіз коду, виявлятиме вразливості та надаватиме рекомендації щодо їх усунення, спрощуючи процес перевірки програмного забезпечення.

Для досягнення цієї мети необхідно виконати наступні завдання:

1. Провести аналіз сучасних підходів та технологій для аудиту програмного коду.
2. Описати ключові особливості використання штучного інтелекту для аналізу коду.

3. Дослідити існуючі інструменти для автоматизованого аналізу коду та обґрунтувати вибір технологій.

4. Розробити структуру веб-застосунку з використанням React, Next.js, Tailwind CSS, TypeScript, OpenAI API.

5. Створити алгоритми для інтеграції інтелектуальних інструментів аналізу коду.

6. Провести тестування застосунку та оптимізувати його продуктивність.

Об’єктом дипломного дослідження є методи та інструменти автоматизованого аудиту програмного коду.

Предметом дослідження є веб-застосунки, які використовуються для автоматизації аналізу програмного коду з використанням штучного інтелекту.

Структура кваліфікаційної роботи складається з наступних частин: вступ, основна частина, висновки, список використаної літератури та додатки.

РОЗДІЛ 1
ОГЛЯД СИСТЕМ АВТОМАТИЗОВАНОГО АНАЛІЗУ ПРОГРАМНОГО КОДУ

1.1. Основні поняття, завдання та призначення автоматизованого аудиту коду

Аудит коду – це фундаментальний процес у сучасній розробці програмного забезпечення, який використовується для перевірки вихідного коду на наявність потенційних проблем, помилок, вразливостей та дотримання стандартів кодування. Він надає розробникам та інженерам з безпеки цінну інформацію про якість та безпеку їхньої кодової бази. Від незначних речей, таких як правила іменування змінних, до серйозних проблем з безпекою, що ховаються в коді та можуть спричинити фатальні наслідки,

Існує два основних підходи до перегляду коду: ручний та автоматизований. Ручний аудит коду, коли аудитори переглядають код вручну, дає уявлення, засновані на досвіді та інтуїції. Автоматизований аналіз, з іншого боку, використовує інструменти перегляду коду для сканування коду на наявність типових проблем, гарантуючи, що жодні загальні проблеми не будуть пропущені.

Автоматизований аудит коду – це процес розробки програмного забезпечення, в якому використовуються інструменти для автоматичного аудиту та аналізу вихідного коду на наявність будь-яких потенційних проблем. Ці інструменти зазвичай аналізують код статично з мінімальним втручанням людини або взагалі без нього, що дозволяє розробникам зосередитися на усуненні проблем, а не на трудомісткому процесі ручного перегляду всього коду вручну.

Важливість аудиту вихідного коду при розробці програмного забезпечення неможливо переоцінити. Він діє як превентивний захід, зменшуючи ризик збоїв

у роботі програмного забезпечення, ризики безпеки та порушень і гарантуючи, що програмний продукт є надійним, безпечним і має високоякісний код.

Однією із головних цілей аудиту коду є виявлення проблем безпеки.

В рамках аудиту коду аудитори старанно шукають будь-які слабкі місця або лазівки в програмному забезпеченні, які можуть зробити додаток вразливим до кіберзагроз. Цифрова сфера кишить загрозами, і певні вразливості є більш поширеними, ніж інші.

До найбільш поширених вразливостей відносяться:

- SQL-ін’єкції: це дозволяє зловмисникам втручатися в запити до вашої бази даних, потенційно отримуючи доступ до даних або змінюючи їх;
- міжсайтовий скриптинг (XSS): це відбувається, коли на веб-сайти впроваджуються скрипти зі шкідливим кодом, які впливають на користувачів, які нічого не підозрюють;
- небезпечні методи автентифікації: слабкі або недосконалі процеси автентифікації можуть призвести до несанкціонованого доступу до системи.

Забезпечення безпеки коду є дуже важливим аспектом. Програмному забезпеченню довіряють обробку даних, управління процесами та, часто, фінансові транзакції. Безпека коду є ключовим елементом для захисту даних, оскільки вона забезпечує збереження конфіденційності та цілісності інформації користувачів. Водночас, надійність програмного забезпечення є основою для підтримки довіри, оскільки користувачі повинні мати впевненість у тому, що можуть використовувати продукт без страху перед можливими порушеннями. Окрім цього, захист фінансових ресурсів і репутації компанії також значною мірою залежить від безпеки коду, адже вразливості можуть призвести до суттєвих фінансових втрат та негативно вплинути на імідж бренду.

Важливою метою аудиту коду також є і покращення якості коду. Якість коду виходить за рамки простої функціональності. Вона гарантує, що програмне забезпечення буде ефективним, стійким і зможе розвиватися відповідно до мінливих потреб.

Аудит коду також може допомогти виявити так звані «запахи коду» або «анти-патерни», які натякають на потенційні проблеми. Це не негайні проблеми, а ознаки того, що код може бути не оптимально структурований. З часом вони можуть перерости в серйозні проблеми, якщо їх не вирішити.

Перевірка також сприяє підвищенню зручності обслуговування та читання. Програмне забезпечення не є статичним. Зі зміною потреб неминучі зміни в коді.

Послідовність у кодуванні – це не лише питання естетики, а й ключовий елемент, що сприяє успішній розробці програмного забезпечення. Як граматика в мові допомагає зрозуміти зміст, так і стандартизований код сприяє кращій взаємодії між розробниками. Завдяки єдиному стилю коду, перевірки як серверної, так і клієнтської частини стають більш ефективними та швидкими. Крім того, така послідовність забезпечує безперебійну співпрацю в команді, дозволяючи кожному розробнику легко долучатися до проєкту та робити внесок незалежно від часу його приєднання.

Оптимізація якості та безпеки програмного забезпечення – це безперервна робота. Впровадження найкращих практик під час аудиту коду може впорядкувати процес аудиту коду, гарантуючи, що програмне забезпечення відповідає бажаним стандартам якості та розвивається разом з постійно мінливим ландшафтом технологій та загроз кібербезпеки.

Регулярне проведення аудиту програмного коду має важливе значення для забезпечення постійної працездатності та цілісності програмного забезпечення. Такі періодичні перевірки виявляють нові проблеми, які могли виникнути з часу останнього аудиту, і підтверджують, що раніше виявлені проблеми були вирішені. Регулярний аудит коду забезпечує послідовну основу для оцінки розвитку програмного забезпечення та його відповідності встановленим стандартам.

Аудит програмного коду – це не лише пошук недоліків. Це ще й можливість навчання. Активно залучаючи розробників до процесу аудиту коду, вони можуть на власні очі побачити сфери, які потребують покращення. Це сприяє розвитку мислення співпраці, що дозволяє розробникам краще розуміти

критерії якості, отримувати конструктивний зворотній зв’язок та ефективніше інтегрувати найкращі практики у свою роботу.

Хоча ручна перевірка коду забезпечує глибину і розуміння, автоматизація прискорює процес аудиту коду, особливо для великої захищеної кодової бази. Автоматизований інструмент, доповнений ручним аудитом, сканує код на наявність відомих шаблонів помилок або потенційних ризиків і вразливостей, гарантуючи, що легко ідентифіковані проблеми продуктивності будуть швидко виявлені. Поєднання переваг ручної перевірки з автоматизованою робить аудит більш комплексним та ефективним [1].

До основних переваг автоматизованого аудиту коду можна віднести наступні:

1. Швидкість: розробникам потрібен час для ручного перегляду коду; автоматизовані інструменти працюють набагато швидше. Автоматизовані аудити коду вирішують проблеми на ранніх стадіях і точно визначають, де саме в коді міститься помилка. Таким чином, ми можемо виправити ці помилки на більш ранніх стадіях життєвого циклу розробки, що дозволить також заощадити витрати в довгостроковій перспективі.

2. Глибина: ручний перегляд коду не може охопити всі можливі шляхи виконання коду, але автоматизований інструмент може. Це дозволить проаналізувати, де можуть бути потенційні проблеми у всьому коді на основі обраних правил.

3. Точність: ручний перегляд коду схильний до людських помилок і упередженості. Автоматизовані інструменти – ні: вони неупереджені і сканують кожен рядок коду. Це допомагає забезпечити найвищу якість коду.

4. Послідовність: ручна перевірка коду часто є суб’єктивною і залежить від навичок та досвіду рецензента. Менш досвідчений рецензент може витратити багато часу на перевірку кожного рядка коду і не виявити всіх проблем. З іншого боку, рецензент, який є другом автора, може бути готовий відправити код навіть з невирішеними проблемами. Автоматизувавши процес

рецензування коду, ці суб’єктивні фактори більше не є проблемою, і ви гарантуєте, що ваш код буде проаналізовано послідовно.

5. Відповідність вимогам: автоматизована перевірка коду може допомогти відповідати стандартам і отримати сертифікацію. Якщо галузь вимагає певних стандартів і практик кодування, інструмент автоматизованого перегляду коду може полегшити роботу, автоматично перевіряючи всю кодову базу.

6. Зворотній зв’язок у реальному часі: після завершення автоматизованого перегляду коду інструмент надає вичерпний зворотній зв’язок, відображаючи всі недоліки безпеки, порушення стилю коду та інші проблеми в коді. Таким чином, команда має практичний огляд, який вони можуть використовувати для негайного виправлення виявлених проблем. Крім того, цей зворотній зв’язок у реальному часі гарантує, що ми дотримуємось усіх найкращих практик ще до того, як ми витратимо час на ручний перегляд коду.

7. Масштабованість: якщо у нас велика команда розробників або ми швидко розвиваємось, можна без проблем автоматизувати процес перегляду коду. Зі збільшенням кількості розробників, зосереджених на створенні коду, кодова база буде зростати, але це не означає, що час, який ми витрачаємо на перевірку коду, також збільшиться. Автоматизовані інструменти для перегляду коду можуть обробляти велику кількість коду і при цьому неймовірно швидко його оцінювати.

8. Економічна ефективність: розробники витрачають до 25% свого часу на ручний перегляд коду. Можна значно скоротити цей час, включивши автоматизований перегляд коду в свій робочий процес, а це означає, що продуктивність підвищиться. Крім того, можна отримати автоматизовані інструменти для перегляду коду за низькою ціною і досягти кращої рентабельності інвестицій [2].

Хоча аудит коду є безцінним для забезпечення якості та безпеки програмного забезпечення, він супроводжується низкою викликів, які необхідно враховувати. Однією з головних проблем є неповне покриття коду, оскільки

навіть за найретельнішого підходу існує ризик, що деякі ділянки можуть залишитися не перевіреними. Це може статися через складність великих кодових баз або інтеграцій, що призводить до вразливостей. Також варто зазначити недоліки автоматизованих інструментів, які можуть давати як хибнопозитивні, так і хибнонегативні результати, що викликає додаткові труднощі у прийнятті рішень.

Крім того, важливим фактором є обмеженість часу та ресурсів, адже якісний аудит вимагає значних зусиль кваліфікованого персоналу, що може створювати додаткове навантаження на команди. Останнім, але не менш важливим викликом, є необхідність знайти баланс між безпекою та швидкістю розробки, оскільки додаткові перевірки безпеки часто уповільнюють процес розробки, вимагаючи ретельного планування і розподілу пріоритетів.

Компанія Codegrip замовила опитування 1000+ технічних ентузіастів, включаючи технічних директорів, генеральних директорів, технічних керівників і старших розробників з усього світу, щоб зрозуміти стан якості коду та порівняти найкращі практики. В опитуванні взяли участь компанії різного розміру – від стартапів до компаній зі списку Fortune 500. Деякими з найвідоміших компаній, які брали участь, є Oracle, Github, AMD тощо.

Вони виявили, що 84% компаній мають певний процес перевірки коду. З них 36% компаній сказали, що аудит коду є найкращим способом покращити якість коду. 26% компаній вважають, що інтеграційне тестування є найкращим способом покращити якість коду, і ті ж результати були отримані для модульного тестування.

Більше половини (57%) компаній сказали, що вони проводять перевірки коду на основі інструментів або використання комбінації інструментів.

Дивлячись на зростаючу конкуренцію, 67% компаній, які зараз не виконують перевірку коду, заявили, що хочуть запровадити перевірку коду як процес у своїй компанії. З тієї ж групи компаній 37% повідомили про відсутність робочої сили як про перешкоду номер один для впровадження перевірки коду в їхній компанії. Компанії з аутсорсинговими командами розробників повідомили,

що брак знань для виконання перевірки коду є фактором, що перешкоджає перевірці коду.

Загальні тенденції свідчать про те, що все більше компаній включають автоматизовані перевірки коду як частину процесу перевірки коду. З іншого боку, є невеликий відсоток компаній, які все ще не зробили перевірку коду частиною свого циклу розробки програмного забезпечення.

84% технологічних компаній мають визначений процес перевірки коду, тоді як 16% технологічних компаній досі не мають процесу перевірки коду. На перший погляд, хоча статистика здається обнадійливою, зміна типу вибірки представляє зовсім іншу картину. Для компаній, які мали менші команди, перегляд коду проводився менше ніж на 50%. Це означає, що менші команди набагато більш невимушені щодо якості коду, ніж великі.

Крім того, з 84% лише 35% вважали, що вони мають ефективний процес перевірки коду, тоді як інші зазначили, що відчувають потребу його вдосконалити.

Компанії, які мали ефективний процес перевірки коду відчули дві основні переваги виконання перевірки коду:

1. Підвищена задоволеність клієнтів завдяки наявності звітів про якість коду та меншій кількості помилок із кожним спринтом;

2. Оновлення версій було легше впроваджувати та стабільніше завдяки меншому технічному боргу.

Серед компаній, які перевіряють код, близько 36% компаній заявили, що кожен розробник відповідає за якість коду. Хоча трохи вище відсоток (38%) спостерігався для компаній, де керівник команди відповідає за якість коду. Приголомшливо низький відсоток у 7% спостерігався для компаній, де за те саме відповідав технічний директор або генеральний директор. Компанії, де відповідають старші розробники, також становлять лише 12%.

50% компаній витрачали 2-5 годин на тиждень на перевірку коду. 14% компаній витратили 5-10 годин, і те саме стосується компаній, які витратили 10+ годин. До 22% компаній витратили менше 2 годин щотижня на перевірку коду.

Більшість компаній, які витрачали менше часу на якість коду, відчули відставання часу або здатності виміряти використання часу, витраченого на якість коду [3].

1.2. Ключові переваги та виклики використання штучного інтелекту для аналізу коду

Штучний інтелект змінює професійний ландшафт, зокрема, сферу створення програмного забезпечення, дозволяючи розробникам писати код більш ефективно та результативно. Завдяки сучасним алгоритмам, інструменти на основі ШІ автономно аналізують код, пропонуючи вдосконалення та визначаючи часті помилки, ризики безпеки та перешкоди для ефективності. Це значно полегшує роботу розробників, оскільки інструменти ШІ базуються на даних, що знижує упередженість і дозволяє аналізувати великі обсяги коду за лічені секунди.

Доповнення людських зусиль за допомогою ШІ має низку переваг. По-перше, це підвищує загальну якість коду, виявляючи проблеми, які можуть бути непоміченими під час ручного перегляду, та знижує технічний борг. По-друге, це підвищує продуктивність, надаючи негайний зворотний зв’язок і економлячи час на ручні перевірки. Третє, автоматизований процес перевірки забезпечує відповідність стандартам кодування, що робить код більш читабельним і легким у підтримці. По-четверте, ШІ підвищує точність аналізу коду, зменшуючи вплив людських помилок та упереджень. Нарешті, він забезпечує масштабованість, дозволяючи ефективно обробляти великі проєкти і підтримувати високі стандарти якості коду навіть при зростанні його обсягів [4].

Незважаючи на прогрес у галузі штучного інтелекту, людський досвід залишається важливим. Хоча інструменти штучного інтелекту можуть допомогти у виявленні поширених помилок кодування або вразливості безпеки, вони можуть мати труднощі з більш тонкими аспектами програмування.

Наприклад, розуміння контексту певної частини коду або оцінка компромісів між різними підходами до впровадження часто потребує втручання людини.

Використовуючи сторонні хмарні інструменти штучного інтелекту, важливо враховувати питання конфіденційності. Хоча ці інструменти пропонують зручність і потужні можливості аналізу, вони передбачають обмін кодом і потенційно конфіденційною інформацією із зовнішнім постачальником послуг.

Підбиваючи підсумок, можна сказати, що використання можливостей штучного інтелекту в перевірці коду приносить численні переваги командам розробників програмного забезпечення. Розробники та організації повинні використовувати рішення для перевірки коду на основі штучного інтелекту, щоб залишатися попереду в цьому середовищі, що швидко розвивається. Інтегруючи ці інструменти у свої процеси розробки, вони можуть забезпечити надійні програмні продукти, заощаджуючи час і ресурси [5].

Що стосується вибору штучного інтелекту, то для даного проєкту було вирішено використовувати саме OpenAI, який використовує архітектуру трансформеру.

Трансформер – це тип архітектури нейронної мережі, який перетворює або змінює вхідну послідовність на вихідну. Вони роблять це, вивчаючи контекствідстежуючи зв’язки між компонентами послідовності.

На практиці використовується багато архітектур глибокого навчання, але трансформер є унікальним за своєю сферою застосування - це єдина архітектура, яку можна застосувати до величезної кількості завдань.

Трансформери обробляють довгі послідовності повністю за допомогою паралельних обчислень, що значно скорочує час навчання та обробки. Це уможливило навчання дуже великих мовних моделей (LLM), таких як GPT і BERT, які можуть вивчати складні мовні репрезентації. Вони мають мільярди параметрів, які охоплюють широкий спектр людської мови і знань, і підштовхують дослідження до більш узагальнених систем ШІ.

Архітектура нейронної мережі-трансформера має кілька програмних рівнів, які працюють разом, щоб генерувати кінцевий результат. На рис.1.1 показані компоненти архітектури трансформації:

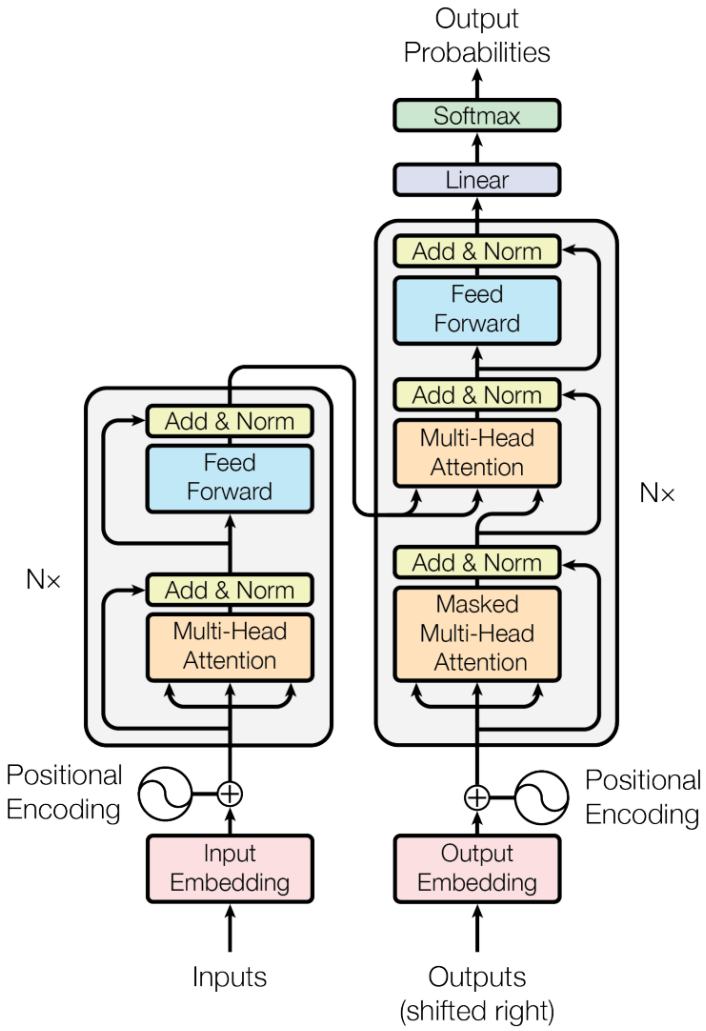


Рисунок 1.1 – Архітектура нейронної мережі-трансформера [6]

OpenAI використовує два основні методи: навчання з підкріпленням на основі людського зворотного зв’язку (Reinforcement Learning from Human Feedback, RLHF) та контрольоване точне налаштування (Supervised Fine-Tuning)

RLHF працює на основі простої, але потужної передумови: використання людського зворотного зв’язку для керування процесом навчання моделей ШІ. Цей принцип, як правило, включає в себе кілька ключових етапів:

1. Початкове навчання: модель проходить початкове навчання з використанням стандартного набору даних, щоб вивчити базову структуру задачі, що розглядається.

2. Взаємодія з людиною: після початкового навчання люди взаємодіють з моделлю, надаючи зворотній зв’язок щодо її результатів. Цей зворотний зв’язок може приймати різні форми, такі як ранжування, виправлення або бінарний вибір.

3. Інтеграція зворотного зв’язку: потім зворотний зв’язок використовується для коригування параметрів моделі або шляхом безпосереднього включення в процес навчання, або через модель винагороди, яка інтерпретує зворотний зв’язок і відповідно коригує майбутні прогнози.

4. Ітеративне вдосконалення: цей процес отримання зворотного зв’язку та оновлення моделі повторюється, з кожною ітерацією покращуючи продуктивність моделі для кращого узгодження з людськими очікуваннями.

Подорож починається з попереднього навчання мовної моделі (ММ) на великому масиві тексту. Провідні організації, такі як «OpenAI», «Anthropic» і «DeepMind», використовують моделі, що містять від 10 мільйонів до 280 мільярдів параметрів. Ця фундаментальна модель слугує основою для розуміння та генерування тексту, схожого на людський.

Ключовим кроком у RLHF є створення «моделі винагороди», відкаліброваної на основі людських уподобань. Ця модель оцінює текстові послідовності, присвоюючи бали, які відображають людську симпатію. Методи можуть бути різними: від точного налаштування вже існуючих ЛМ до навчання нових моделей з нуля на основі даних про вподобання. Метою є система, яка на основі будь-якого тексту може надати скалярну винагороду, що вказує на вподобання людини.

Етап точного налаштування передбачає коригування параметрів мовної моделі за допомогою навчання з підкріпленням, зокрема за допомогою таких алгоритмів, як проксимальна оптимізація політики (Proximal Policy Optimization, PPO). Цей процес уточнює вихідні дані моделі на основі оцінок «моделі

винагороди», прагнучи створити текст, який більше відповідає людським судженням.

Схема на рис. 1.2 ілюструє процес налаштування мовної моделі за допомогою підкріплювального навчання з людським зворотним зв'язком (RLHF). Вона складається з таких компонентів:

1. Prompts Dataset – набір даних із запитами.
2. Initial Language Model – початкова мовна модель, яка генерує базовий текст.
3. Tuned Language Model (RL Policy) – налаштована мовна модель після навчання.
4. Reward (Preference) Model – модель винагороди, яка оцінює текст.
5. Reinforcement Learning Update (e.g. PPO) – оновлення параметрів моделі за допомогою підкріплювального навчання.
6. KL prediction shift penalty – штраф за зсув передбачення.

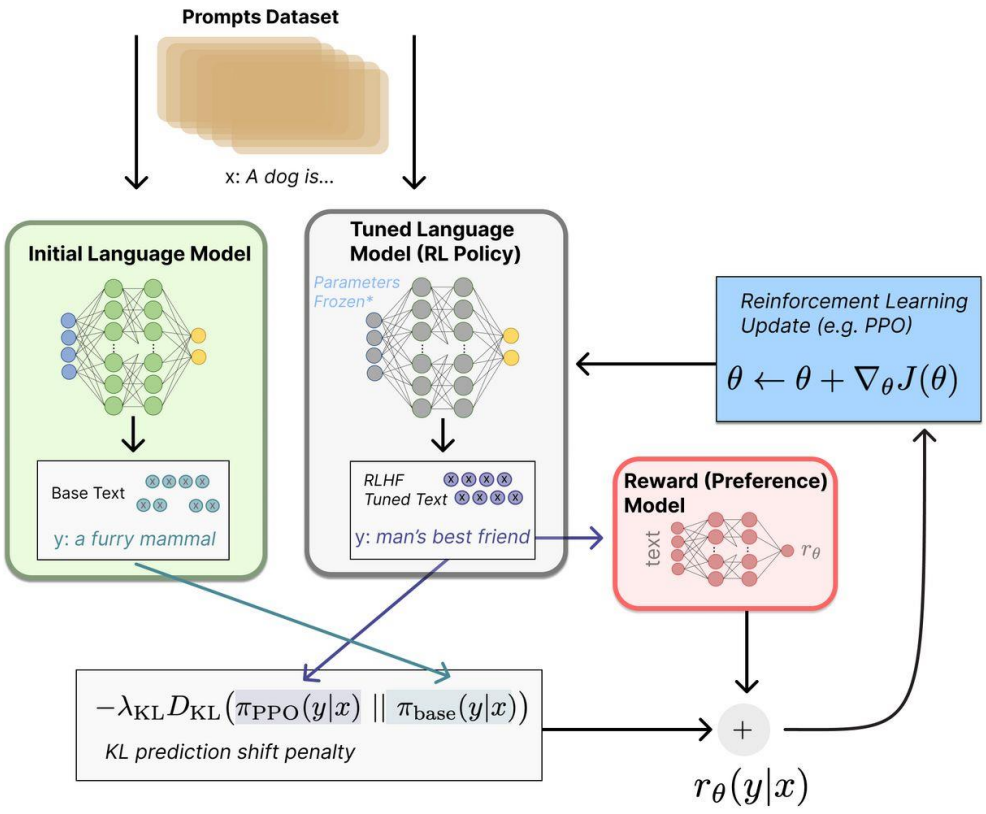


Рисунок 1.2 – Процес налаштування мовної моделі за допомогою RLHF

Основні виклики RLHF:

1. Масштабованість: збір та інтеграція зворотного зв’язку в масштабі вимагає значних ресурсів.

2. Упередженість та суб’єктивність: зворотний зв’язок може бути упередженим, що вимагає різноманітних вхідних даних.

3. Складне впровадження: включення зворотного зв’язку додає складності навчанню ШІ.

4. Якість зворотного зв’язку: ефективність RLHF залежить від точного і змістовного зворотного зв’язку.

5. Етичні проблеми: рішення про те, який зворотний зв’язок використовувати, пов’язане з важливими етичними рішеннями.

6. Конфіденційність даних: управління конфіденційними даними зворотного зв’язку вимагає суворих заходів конфіденційності.

7. Вартість: високі фінансові та часові витрати на збір і обробку зворотного зв’язку.

У галузі обробки природної мови (Natural Language Processing, NLP), що швидко розвивається, точне налаштування (SFT) стало потужною та ефективною технікою для адаптації попередньо навчених великих мовних моделей (Large Language Models, LLM) до конкретних завдань, що стоять перед ними в подальшому. Попередньо навчені великомасштабні мовні моделі (як сімейство GPT) продемонстрували значний прогрес у розумінні та генеруванні мови. Однак ці попередньо навчені моделі, як правило, навчаються на величезних обсягах текстових даних за допомогою неконтрольованого навчання і можуть бути не оптимізовані для конкретного завдання.

Точне налаштування заповнює цю прогалину, використовуючи загальне розуміння мови, отримане під час попереднього навчання, і адаптуючи його до цільового завдання за допомогою навчання під контролем. Зокрема, для великих мовних моделей точне налаштування має вирішальне значення, оскільки етап перенавчання з усіма даними є надто трудомістким.

Порівняння переднавчання та тонкого налаштування LLM зображено на рис. 1.3. Переднавчання потребує великого обсягу обчислювальних ресурсів та використовується на великих наборах даних без міток. Тонке налаштування, навпаки, вимагає менше обчислювальних ресурсів та застосовується на невеликих наборах даних з мітками.

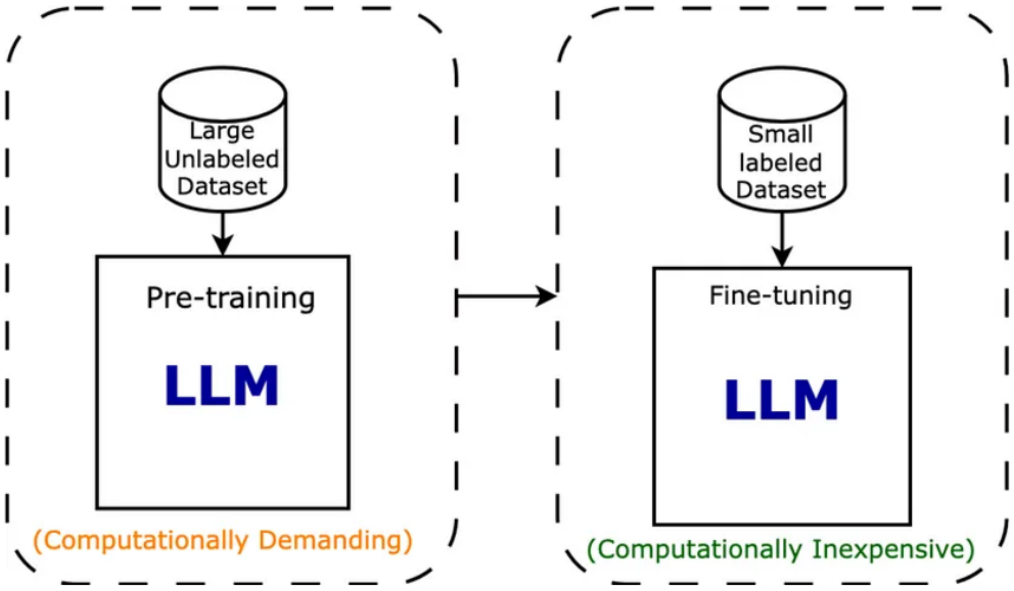


Рисунок 1.3 – Порівняння переднавчання та тонкого налаштування LLM

Успіх точного налаштування привів до численних найсучасніших результатів у широкому спектрі завдань NLP і став стандартною практикою в розробці високоефективних мовних моделей. Дослідники та практики продовжують досліджувати варіації та оптимізацію технік точного налаштування, щоб розширити межі можливостей NLP.

Порівнюючи OpenAI з іншими компаніями в цій галузі, можна виділити кілька ключових аспектів, які підтверджують його лідерство:

1. Якість моделей. OpenAI відомий своїми потужними мовними моделями, які демонструють надзвичайну здатність генерувати текст, розуміти контекст і виконувати різноманітні завдання. Конкуренти, такі як Google DeepMind або Anthropic, також пропонують потужні моделі, але OpenAI забезпечує більш узгоджений та реалістичний результат у різних сценаріях.

2. Безпека та етика. OpenAI активно працює над етичними аспектами штучного інтелекту та безпекою. Компанія зосереджена на запобіганні зловживанням своїми технологіями, тоді як деякі конкуренти не завжди вживають такі заходи. OpenAI публікує дослідження та звіти про вплив своїх технологій на суспільство, демонструючи свою відповідальність.

3. Інтерфейс і досвід користувача. Платформи OpenAI, такі як ChatGPT, мають зрозумілий і зручний інтерфейс, що робить їх доступними для користувачів різного рівня підготовки. У той час як конкуренти можуть пропонувати складніші інтерфейси або менш зрозумілі API, OpenAI забезпечує простоту використання та швидке впровадження в проєкти.

4. Спільнота та Підтримка. OpenAI активно розвиває спільноту розробників, пропонуючи різноманітні ресурси, включаючи документацію, приклади коду та форуми для обговорення. Це створює екосистему, яка сприяє швидкому навчанні та вдосконаленню користувачів. Конкуренти, хоча й мають свої ресурси, не завжди забезпечують таку ж рівень підтримки та взаємодії з користувачами.

5. Інновації та Дослідження. OpenAI постійно інвестує в дослідження та розвиток нових технологій, таких як DALL-E для генерації зображень, Codex для програмування та інші інноваційні продукти. Це дозволяє компанії залишатися на передовій штучного інтелекту, тоді як конкуренти часто зосереджені на вдосконаленні вже існуючих рішень.

Враховуючи всі наведені аспекти, OpenAI виявляється значним лідером у галузі штучного інтелекту. Його моделі відрізняються високою якістю, етикою використання, зручністю інтерфейсу, підтримкою спільноти та інноваційним підходом до розвитку технологій. Це робить OpenAI привабливим вибором для компаній і розробників, які прагнуть впровадити штучний інтелект у свої проєкти.

1.3. Огляд існуючих інструментів для автоматизованого аудиту програмного коду

Багато інструментів штучного інтелекту полегшують роботу розробників, залишаючись актуальними та автоматизуючи завдання. Ці програми зі штучним інтелектом розроблені таким чином, щоб легко виявляти такі проблеми, як потенційні помилки, запах коду, вразливості системи безпеки, вузькі місця тощо.

Отже, розглянемо існуючі інструменти для автоматизованого аудиту програмного коду, які використовують штучний інтелект:

1. Codacy – це безкоштовний інструмент перевірки коду штучного інтелекту, який аналізує оцінку в реальному часі та надає цінну інформацію та пропозиції щодо покращення. Використовуючи Codacy, DevOps може легко автоматизувати перевірку коду. Це дозволяє організаціям створювати та виконувати чітко визначений процес перевірки коду.

Вигляд інтерфейсу сайту зображено на рис. 1.4:

Рисунок 1.4 – Інтерфейс сайту Codacy

Можливості:

- добре інтегрується з Bitbucket, Slack, GitHub та Jira;
- забезпечує автоматичний перегляд коду для більш ніж 30 мов;

- цей інструмент може допомогти відстежувати прогрес, аналізувати комміти та захищати кодову базу в Git.

Таблиця 1.1 – Переваги і недоліки Codacy

Переваги	Недоліки
Він пропонує поради, керовані штучним інтелектом	Складність у роботі зі складними правилами
Він може перевіряти дублювання та погані метричні області коду	Він не сертифікований за стандартом SOC 2

*Складено за даними джерела [7]

Продукт пропонує безкоштовний план. Є два платних плани:

- Про – 15 доларів США/за розробника/місяць;
- Корпоративний – потрібно зв’язатись з відділом продажів.

2. Snyk – використовує орієнтований на розробників підхід до статичного тестування безпеки додатків (SAST). Він допомагає організаціям знаходити та виправляти вразливості в бібліотеках та контейнерах з відкритим кодом. Платформа використовує передові алгоритми машинного навчання для пошуку вразливостей безпеки та потенційних загроз.

Вигляд інтерфейсу сайту зображено на рис. 1.5:

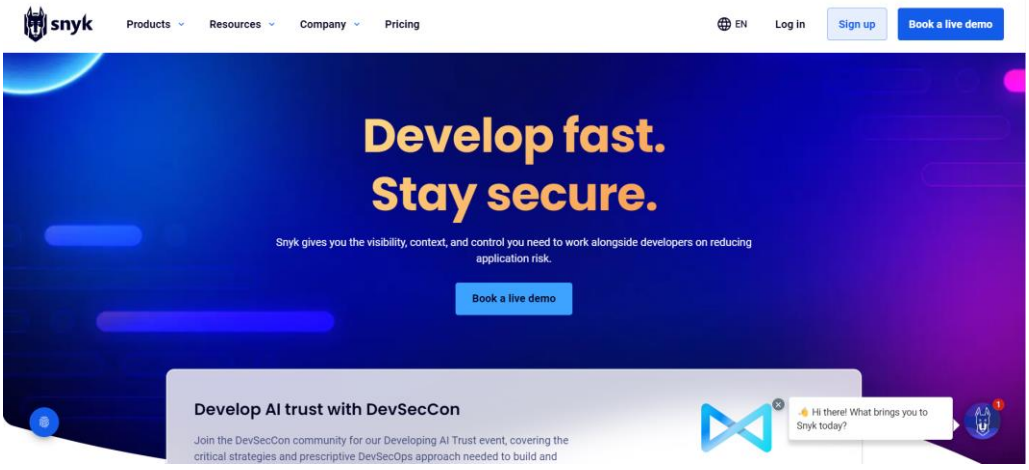


Рисунок 1.5 – Інтерфейс сайту Snyk

Можливості:

- дозволяє сканувати код в режимі реального часу;
- рекомендує виправлення для кожного знайденого рядка коду з проблемами;
- користувачі можуть встановлювати власні правила та додавати автоматизовані тести.

Таблиця 1.2 – Переваги і недоліки Snyk

Переваги	**Недоліки**
Висока точність сканування	Інтерфейс користувача може бути складним для початківців
Інтегрується з численними інструментами розробки	Обмежений фокус на загальній якості коду

*Складено за даними джерела [7]

Інструмент пропонує безкоштовний план і має два платні плани:

- Командний план – $25 за продукт/місяць;
- Корпоративний – зверніться до відділу продажів.

3. GitHub Copilot – використовує просунутий штучний інтелект для допомоги в кодуванні; він допомагає розробникам створювати високоякісний код більш ефективно, діючи як віртуальний дует програмістів. Інструмент навчається на великих наборах даних з різних джерел і підтримує безліч мов програмування (рис. 1.6).

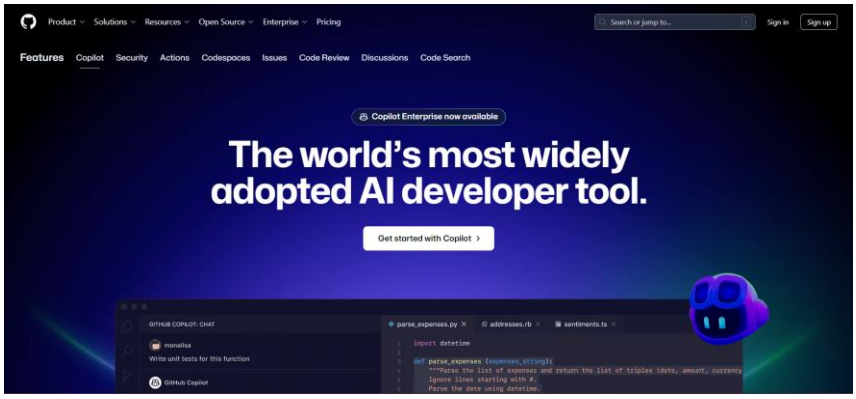


Рисунок 1.6 – Інтерфейс сайту GitHub

Можливості:

- забезпечує контекстне розуміння;
- цей інструмент допомагає орієнтуватися в нових мовах програмування, генеруючи фрагменти коду;
- має можливість виявляти будь-які можливі недоліки безпеки у вашому коді.

Таблиця 1.3 – Переваги і недоліки GitHub Copilot

Переваги	**Недоліки**
Інтегрується з багатьма IDE	Не пропонує функцію чату
Пропонує підказки під час написання коду	Іноді може пропонувати неефективний код

*Складено за даними джерела [7]

Пропонує безкоштовний план. Існує два платні плани:

- Командний план – $3.67/користувач/місяць;
- Корпоративний – $19.25/користувач/місяць.

4. Tabnine – це ефективний генератор коду для ШІ, що дозволяє програмістам швидше писати високоякісний код (рис. 1.7).

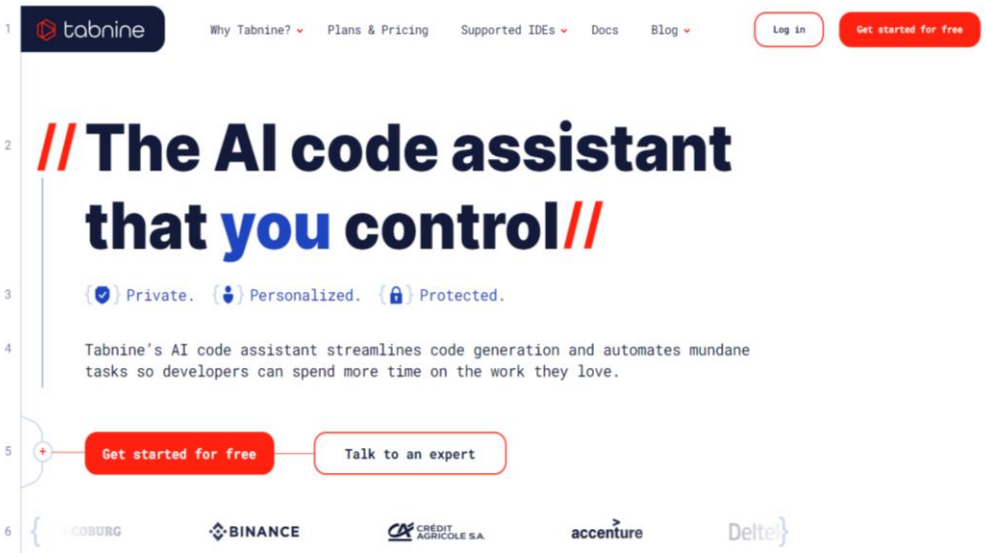


Рисунок 1.7 – Інтерфейс сайту Tabnine

Користувачі віддають перевагу цій платформі, тому що вона створена з сильним акцентом на конфіденційності та безпеці. Вона допомагає підвищити продуктивність і якість коду, надаючи розумні пропозиції щодо завершення та виявляючи потенційні помилки.

Можливості:

- сумісний з різними мовами програмування, такими як Python, Java, JavaScript тощо;
- він може виявляти помилки в режимі реального часу;
- інтеграція з Android Studio, Visual Studio Code, Eclipse тощо.

Таблиця 1.4 – Переваги і недоліки Tabnine

Переваги	**Недоліки**
Може згенерувати повний рядок коду на основі наданих вами підказок	Може не працювати з деякими мовами зі складною динамікою
Може виконувати юніт-тести на наданих фрагментах коду	Споживає багато ресурсів процесора при локальному запуску

*Складено за даними джерела [7]

Пропонує безкоштовний план. Існує два платні плани:

- Про – $12/користувач/місяць;
- Корпоративний – $39/користувач/місяць.

5. Amazon CodeWhisperer – один з найкращих інструментів зі штучним інтелектом для перегляду коду. Він пропонує рекомендації щодо кодування в режимі реального часу відповідно до вашого стилю. Платформа також відома тим, що надає API для безперешкодної інтеграції в існуючі робочі процеси програмістів (рис. 1.8).

Можливості:

- добре інтегрується з іншими хмарними сервісами Amazon;
- швидко виявляйте та повідомляйте про помилки кодування;
- цей інструмент може генерувати коментарі та документацію до коду.

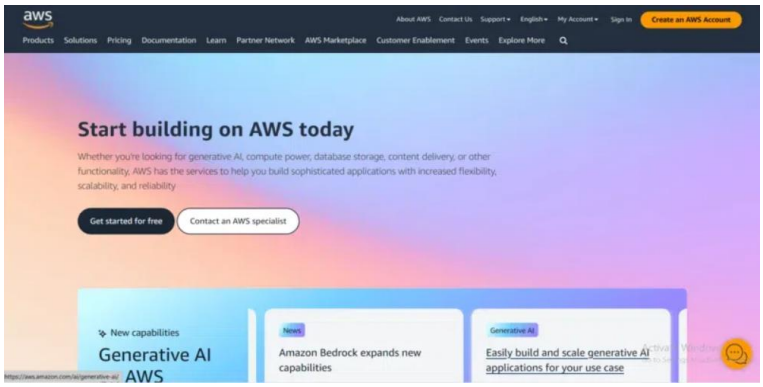


Рисунок 1.8 – Інтерфейс сайту AWS

Таблиця 1.5 – Переваги і недоліки Amazon CodeWhisperer

Переваги	**Недоліки**
Точні пропозиції щодо коду в реальному часі	Використовує старі бібліотеки
Працює з декількома IDE	Іноді не справляється зі складним кодом

*Складено за даними джерела [7]

Пропонує підхід до ціноутворення за принципом «плати по факту».

6. Replit – це інноваційний інструмент для створення коду, який надає користувачам інтерактивний простір для кодування та колективного навчання. Він пропонує індивідуальні пропозиції, які можуть добре узгоджуватися з вашою поточною роботою. Унікальність цієї платформи полягає в тому, що вона може надавати пояснення коду, а також модифікації коду (рис. 1.9).

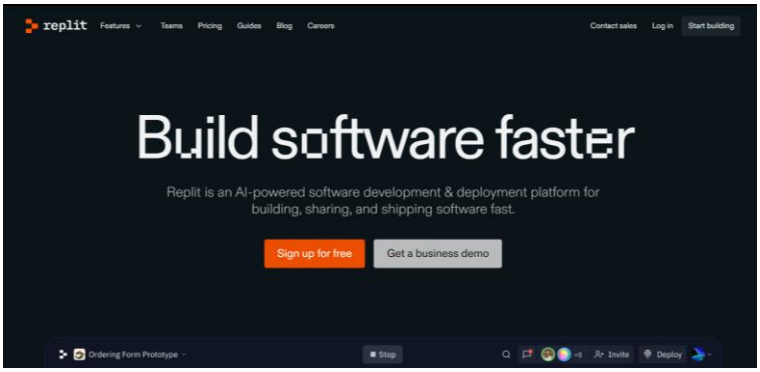


Рисунок 1.9 – Інтерфейс сайту Replit

Можливості:

- швидко знаходить помилки та допомагає їх виправити;
- має можливість уточнювати пропозиції за допомогою коментарів до коду;
- інструмент підтримує більше 20 мов.

Таблиця 1.6 – Переваги і недоліки Replit

Переваги	**Недоліки**
Дозволяє користувачам кодувати безпосередньо в браузері	Він не пропонує офлайн-рішення
Забезпечує спільне кодування в реальному часі	Іноді не справляється зі складним кодом

*Складено за даними джерела [7]

Пропонує безкоштовний тарифний план. Існує один платний план – Replit Core – $12/місяць.

7. Cody (Sourcegraph) – це комплексний помічник для кодування, який може допомогти розробникам програмного забезпечення писати, редагувати та переглядати код. Він також може надавати інформацію про те, де знаходиться код і хто його оновлював. Цей інструмент робить це в усіх гілках і на всіх хостах коду. Вигляд інтерфейсу сайту зображено на рис. 1.10.

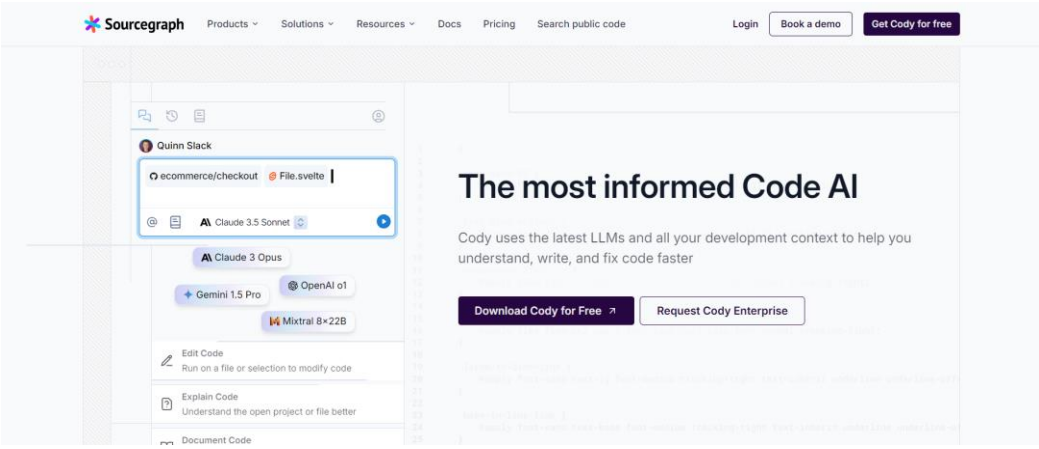


Рисунок 1.10 – Інтерфейс сайту Sourcegraph

Можливості:

- користувачі можуть шукати змінні у будь-якій області свого проєкту;
- він може знаходити певні компоненти та виявляти потенційні проблеми із запропонованими виправленнями;
- може бути інтегрований у різні відомі IDE.

Таблиця 1.7 – Переваги і недоліки Cody

Переваги	**Недоліки**
Він надає find-посилання для коду	Працює тільки в IDE
Він може впоратися зі складними завданнями кодування	Іноді Cody потребує кількох підказок

*Складено за даними джерела [7]

Пропонує безкоштовний план. Існує два платні плани:

- Про – $9 за користувача/місяць;
- Корпоративний – $19 за користувача/місяць.

8. AskCodi – ще один чудовий ШІ оглядач коду, який значно спростив процес кодування. Він пропонує розумні рекомендації та підтримку розробникам, надаючи їм пропозиції щодо коду та усуваючи можливість помилок (рис. 1.11).

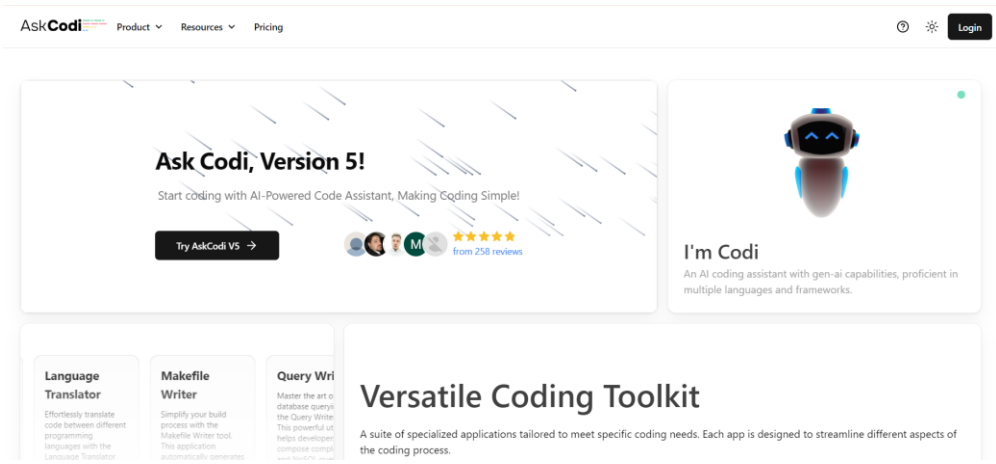


Рисунок 1.11 – Інтерфейс сайту AskCodi

Можливості:

- інтегрується з різними IDE, включаючи IntelliJ IDEA, Visual Studio Code та PyCharm;
- відповідає на питання програмування природною мовою, щоб полегшити програмістам-початківцям розуміння та усунення проблем;
- генерує коди на різних мовах програмування, таких як Java, Kotlin, Python, Ruby тощо.

Таблиця 1.8 – Переваги і недоліки AskCodi

Переваги	**Недоліки**
Інтуїтивно зрозумілий інтерфейс	Іноді може бути повільним у реагуванні
Економічно ефективний	Належне форматування запитів може бути складним завданням, що призводить до неточних результатів

*Складено за даними джерела [7]

Пропонує безкоштовний та два платні плани: Преміум – $9.99 на місяць та Максимальна – $29.99 на місяць.

CodeWP – допомагає розробникам WordPress створювати швидкі фрагменти коду за лічені секунди. Він оновлюється найновішими плагінами та документацією WordPress і генерує правильний код для таких завдань, як розробка плагінів, створення користувацьких типів постів і розширення основних функцій потрібних продуктів WordPress (рис. 1.12).

Можливості:

- графічний аналіз веб-сайтів за допомогою штучного інтелекту;
- призначений для творців і розробників WordPress;
- підтримує різні завдання, такі як розробка плагінів і створення кастомних типів постів.

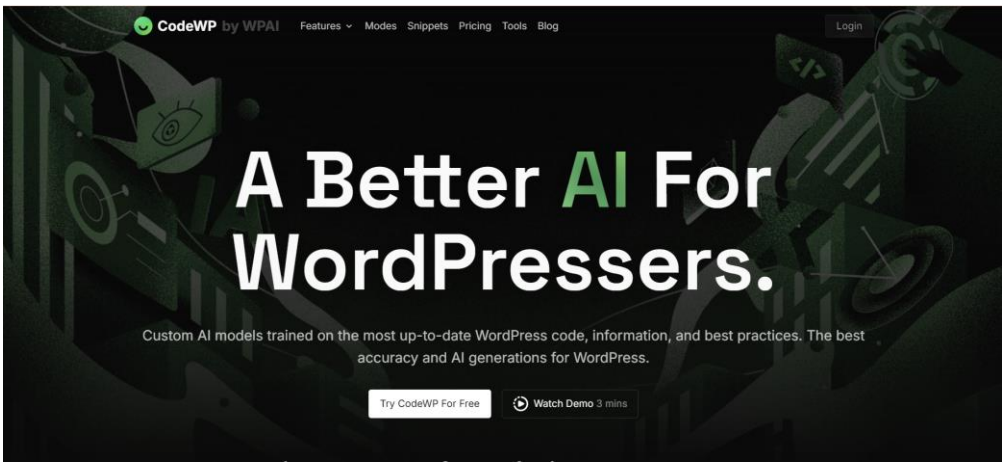


Рисунок 1.12 – Інтерфейс сайту CodeWP

Таблиця 1.9 – Переваги і недоліки CodeWP

Переваги	**Недоліки**
Допомагає розробникам швидко генерувати код	Згенеровані коди іноді містять помилки
Готові та перевірені фрагменти коду для різних варіантів використання	Відсутній генератор шаблонів

*Складено за даними джерела [7]

Пропонує безкоштовний та два платні плани:

- Про – $18 на місяць;
- Агентський – $48 на місяць.

Інструменти штучного інтелекту для перевірки та генерації коду значно спрощують роботу розробників, аналізуючи вихідний код програмного проєкту та надаючи професійний і конструктивний зворотний зв’язок.. Вони можуть інтегруватися з різними платформами та забезпечувати рекомендації в реальному часі, підвищуючи продуктивність та безпеку коду. Однак кожен інструмент має свої переваги та недоліки, тому вибір залежить від конкретних потреб розробників, проєктів та їх бюджету [7].

1.4. Огляд сучасних технологій для розробки веб-застосунків та обґрунтування їх вибору

У сучасній розробці веб-додатків важливим аспектом є вибір мови програмування або її варіантів, які забезпечують оптимальну продуктивність, зручність та безпеку. JavaScript протягом багатьох років залишається стандартом для фронтенд-розробки, однак, з появою TypeScript, розробники отримали новий інструмент, що розширює можливості JavaScript. Давайте розглянемо основні відмінності між JavaScript та TypeScript, їх переваги та недоліки, а також визначимо, чому TypeScript часто стає вибором сучасних команд розробників для складних проєктів.

JavaScript, який спочатку був розроблений як клієнтська мова, розширився до використання на стороні сервера. Однак його зростаюча складність та обмеження в об’єктно-орієнтованому програмуванні перешкоджали його впровадженню на підприємствах. Щоб вирішити цю проблему, було створено TypeScript, який вдосконалив JavaScript, надавши статичну типізацію та функції для надійної розробки на стороні сервера на рівні підприємства.

Можливості TypeScript:

1. Трансляція в JavaScript: код TypeScript трансплантується в JavaScript для інтерпретації браузером, що дозволяє браузерам читати і відображати код.

2. Перетворення JavaScript в TypeScript: код на JavaScript можна перетворити на TypeScript, змінивши розширення файлу з .js на .ts.

3. Універсальне використання TypeScript: TypeScript можна скомпілювати для запуску в будь-якому браузері, пристрої або операційній системі, що робить його адаптивним до різних середовищ.

4. Підтримка бібліотек JavaScript: TypeScript безперешкодно підтримує існуючий JavaScript-код, дозволяє інтегрувати популярні JavaScript-бібліотеки та полегшує виклик коду TypeScript з рідного JavaScript.

Таблиця 1.10 – Порівняння TypeScript та JavaScript

Функція	**TypeScript**	**JavaScript**
Типізація	Забезпечує статичну типізацію	Динамічну типізацію
Інструментарій	Постачається з IDE та редакторами коду	Обмежений вбудований інструментарій
Синтаксис	Подібний до JavaScript, з додатковими можливостями	Стандартний синтаксис JavaScript
Сумісність	Зворотна сумісність з JavaScript	Не можна запускати TypeScript у файлах JavaScript
Налагодження	Сильніша типізація може допомогти виявити помилки	Може вимагати більше налагодження та тестування
Час навчання	Може знадобитися час для вивчення додаткових функцій	Стандартний синтаксис JavaScript знайомий

*Складено за даними джерела [8]

Отже, розглянувши відмінності між JavaScript та TypeScript, можна зробити висновок, що TypeScript є більш надійним інструментом для сучасної розробки складних веб-додатків. Завдяки статичній типізації, можливості виявлення помилок під час компіляції та зворотній сумісності з JavaScript, TypeScript надає розробникам додаткові засоби для підвищення якості коду та зниження кількості помилок на етапі розробки.

Сучасні веб-додатки – це набагато більше, ніж прості інтерфейси, що відображають статичний контент. Це складні, багатофункціональні системи, призначені для виконання широкого спектру завдань, від забезпечення зв’язку в режимі реального часу до обробки величезних обсягів даних. Однією з головних характеристик сучасних веб-додатків є їхня здатність інтегрувати новітні технології, такі як штучний інтелект (ШІ).

Інтеграція ШІ дозволяє веб-додаткам автоматизувати різноманітні процеси, покращувати взаємодію з користувачами завдяки персоналізованому досвіду, аналізувати великі масиви даних і навіть виконувати завдання, які традиційно вимагали втручання людини, такі як аудит коду або моніторинг безпеки. Це не тільки підвищує ефективність і надійність веб-додатків, але й

розширює їх функціональність, дозволяючи компаніям і розробникам створювати більш інтелектуальні та адаптивні системи.

Виходячи з досвіду, використання навіть одного фреймворку у фронтенд-розробці має велике значення. Сьогодні вони є необхідними для створення веб-сайтів та веб-додатків. Використовуючи їх у своїй роботі, розробники можуть заощадити час і зусилля, а також гарантувати, що веб-сайт буде зручним для користувача і візуально привабливим.

Однією з головних переваг використання таких інструментів є те, що вони можуть допомогти покращити загальну продуктивність веб-сайту. Ці пакети оптимізовані для швидкості та ефективності, що може допомогти зменшити час завантаження та покращити користувацький досвід.

Ще одна перевага використання фреймворків полягає в тому, що вони можуть зробити розробку більш послідовною та ефективною. Це може бути особливо корисно для компаній, які хочуть скористатися послугами аутсорсингу розробки, оскільки дозволяє їм гарантувати, що їхній веб-сайт буде створений і підтримуватиметься за високими стандартами.

Ці набори включають готові макети, логіку, поведінку та дизайн у вигляді папок і файлів, які використовуються в розробці. Вони надають заздалегідь написаний код та інструменти, що допомагають розробникам швидко і легко створювати користувацький інтерфейс.

Крім того, вони пропонують стандартизовану структуру коду і часто включають набір готових компонентів, таких як кнопки, форми та навігаційні меню, які можна легко інтегрувати у веб-сайт, щоб він був адаптивним і підлаштовувався під різні розміри екранів і пристрої.

Багато фреймворків також включають такі функції, як підтримка препроцесорів CSS і вбудований функціонал JavaScript, що може ще більше спростити процес розробки. Головна мета цих інструментів – зробити процес розробки швидшим, простішим і більш послідовним.

Розглянемо декілька найпопулярніших фреймворків на сьогоднішній день.

Розроблений і підтримуваний компанією Facebook, React (рис. 1.13) – який є JavaScript фреймворком для створення користувацьких інтерфейсів. Цей компонентний фреймворк дозволяє розробникам створювати веб-сайт або веб-додаток, розбиваючи його на менші, багаторазово використовувані компоненти.

Рисунок 1.13 – Логотип React.js

За останніми даними GitHub, React має 1644+ дописувачів і отримав 216K+ зірок.

Завдяки цій величезній спільноті розробників, яка постійно розвивається, існує понад 13,55 мільйонів веб-сайтів, створених з використанням React.

Також, якщо говорити про вибір розробників, то згідно з опитуванням StackOverflow 2023, React.js виявився найпопулярнішою технологією для веб-фронтенду з 42,87% голосів (рис. 1.14) [9].

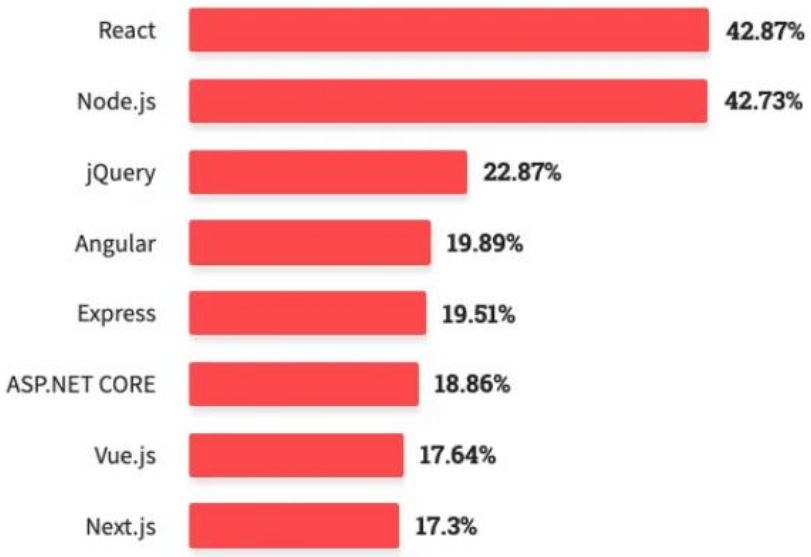


Рисунок 1.14 – Результати опитування StackOverflow 2023

За допомогою React.js можна реалізувати додатки різних масштабів, наприклад:

- односторінкові додатки (SPA);
- медіа-сайти;
- платформи для потокового відео;
- рішення SaaS;
- системи управління контентом (CMS);
- прогресивні веб-додатки (PWA);
- інтерактивні інформаційні панелі;
- веб-сайти електронної комерції;
- освітні додатки;
- додатки для бронювання подорожей;
- десктопні додатки;
- мобільні додатки.

У цифровому ландшафті Європи та США, що постійно розвивається, використання можливостей ReactJS стало стратегічним питанням. Завдяки своїй майстерності у створенні динамічних користувацьких інтерфейсів, ReactJS стає наріжним каменем сучасної веб-розробки.

Рисунок 1.15 – Ключові переваги ReactJs

ReactJS забезпечує взаємодію з будь-яким макетом інтерфейсу користувача і є простим у використанні. Крім того, він дозволяє швидко та якісно розробляти та рендерити додатки, заощаджуючи час для клієнтів та React-розробників.

Створення динамічних веб-додатків з використанням HTML-рядків було складним завданням, оскільки вимагало складного скриптингу. React JS вирішив цю проблему і спростив її, вимагаючи менше коду і надаючи більше можливостей. Він розробив JSX (розширення JavaScript), синтаксис, який дозволяє HTML-тегам і лапкам відображати певні підкомпоненти. Крім того, це полегшує створення машинозчитуваних кодів.

Також неабияким плюсом є можливість повторного використання створених компонентів. Кожен компонент має свою логіку та елементи керування, створюючи шматок HTML-коду, який можна використовувати в будь-якому місці. Ця можливість допомагає розробляти адаптивний веб-дизайн. Переваги повторного використання включають коротший цикл розробки, підтримку стандартів продукту, збільшення потенціалу платформи, легше управління додатками, зниження витрат на розробку і забезпечення бездоганної роботи.

ReactJS використовує віртуальну модель об'єктів документа (Virtual DOM), яка є кросплатформним інтерфейсом програмування API, щоб підвищити продуктивність у веб-розробці. При створенні додатку з великою кількістю взаємодій користувача і змін даних, важливо протестувати, як структура додатку впливатиме на швидкість. Незважаючи на швидкі клієнтські платформи та JavaScript-двигуни, велика кількість маніпуляцій з DOM може впливати на швидкість роботи платформи і створювати незручний користувацький досвід. React вирішує цю проблему, використовуючи віртуальну модель об'єктів документа, яка представляє DOM веб-браузера, що живе в пам'яті. Цей підхід уникає запису в DOM під час створення компонентів React, замість цього створюючи віртуальні компоненти, які відповідають і перетворюються у реальний DOM, що призводить до більш плавної і швидшої роботи платформи.

ReactJS широко впроваджений багатьма компаніями для створення їхніх додатків, веб-сайтів та внутрішніх проєктів. Серед цих компаній Walmart Labs, Tencent, Tesla, Bloomberg, Airbnb, Baidu Mobile, GoDaddy, Gyroscope та інші, які розробили мобільні додатки за допомогою даного фреймфорка. Також веб-сайти, які використовують React, включають Coursera, PayPal, Netflix, Dailymotion, IMDb, Chrysler, BBC, American Express, Khan Academy, Lyft, New York Times, Dropbox, Reddit та інші.

Перераховані вище причини більш ніж виправдовують популярність бібліотеки React і те, чому її впроваджує велика кількість організацій та бізнесів [10].

Vue.js – це фреймворк який має відкритий вихідний код та створений для реалізації користувацьких інтерфейсів. Його розробив Еван Ю (Evan You), а зараз він підтримується командою розробників в організації Vue.js. Як і React, Vue також є компонентним фреймворком, який дозволяє розробникам створювати веб-сайт або веб-додаток, розбиваючи його на менші, багаторазово використовувані компоненти.

Рисунок 1.15 – Логотип Vue.js

На Reddit спільнота Vue.js налічує 99 тис. учасників і входить у топ-5% спільнот за розміром.

Згідно з опитуванням BuiltWith, наразі існує понад 1 689 786 сайтів, що використовують Vue. Крім того, Vue.js посідає 4-те місце серед 10 тисяч найпопулярніших сайтів у категорії «Бібліотека JavaScript».

Згідно з опитуванням StackOverflow, Vue.js віддають перевагу близько 17,64% розробників по всьому світу.

Типи інтерфейсних веб-додатків, які можна створити за допомогою Vue.js:

- односторінкові додатки (SPA);
- прогресивні веб-додатки;
- сайти електронної комерції;
- 2D настільні ігри, такі як chess.com;
- онлайн-сховища;
- платформи онлайн-бронювання/резервування квитків;
- платформи соціальних мереж;
- системи управління контентом (CMS);
- веб-додатки в режимі реального часу;
- гібридні мобільні додатки.

Vue.js використовують багато відомих компаній для розробки своїх додатків та веб-сайтів. Серед цих компаній можна знайти Facebook, Netflix, Google, Apple, Zoom, Microsoft, Nintendo, BMW, UpWork, Behance, Xiaomi, Adobe, GitLab, Alibaba, Trivago, 9GAG та інші1. Це демонструє, наскільки популярним та надійним є Vue.jsсеред великих брендів.

Vue.js легко вивчити та використовувати, знаючи лише основи HTML, CSS і JavaScript. Можна писати код у компонентах, що знаходяться в одному файлі, що робить їх більш читабельними та зручними для підтримки. Цей фреймворк є швидким і ефективним завдяки віртуальному DOM та реактивній системі. Його невеликий розмір, близько 18 КБ у стиснутому вигляді, дозволяє швидко завантажувати та не впливає на швидкість роботи сайту чи SEO. Vue.js є гнучким та адаптивним, підходить для будь-яких проєктів – від простих веб-сайтів до складних веб-додатків. Можна легко інтегрувати його з іншими бібліотеками або існуючими проєктами. Vue.js також підтримує компонентно-орієнтовану розробку, що полегшує повторне використання, тестування та масштабування коду.

Ось кілька обмежень або, можна сказати, мінусів Vue.js:

1. Часті оновлення: порівняно з іншими інтерфейсними технологіями, Vue.js отримує оновлення занадто часто, що вимагає занадто багато для вивчення та дослідження, що робить його складним для початківців.

2. Проблема адаптації: Vue.js був створений і розгорнутий у Китаї, і більшість членів його спільноти є вихідцями звідти. Отже, більшість документації не перекладена іншими мовами, що створює мовний бар'єр для розробників, які хочуть ним користуватися.

3. Невеликий досвід: Vue.js все ще перебуває на стадії розвитку, тому у нього не так багато учасників, і ви знайдете лише кількох VueJS розробників, які зможуть задовольнити ваші потреби.

4. Обмежений вибір плагінів: порівняно з іншими фреймворками, Vue.js має меншу кількість доступних плагінів. І це може бути певною перешкодою для розробників, які звикли мати у своєму розпорядженні інструменти підтримки.

5. Проблеми з мобільною підтримкою: додатки, створені за допомогою Vue, часто стикаються з проблемами несумісності мобільних браузерів з їх застарілими версіями, чого зазвичай не повинно бути [10].

Наступним є Angular, ще один комплексний фреймворк JavaScript для створення надійних і масштабованих веб-додатків. Цей фреймворк розроблений і підтримується компанією Google і використовується для створення динамічних і складних веб-додатків.

Рисунок 1.16 – Логотип Angular

Angular використовує компонентну архітектуру, розділяючи додаток на менші багаторазові компоненти, якими можна легко керувати та підтримувати. Це дозволяє розробникам створювати великі, масштабовані додатки з високим ступенем повторного використання коду.

Angular також використовує декларативний підхід до побудови користувацьких інтерфейсів, що полегшує розробникам розуміння та підтримку коду. Вбудована система ін’єкції залежностей дозволяє Angular ефективно керувати компонентами та сервісами додатку. Він також має потужну систему шаблонів, яка дозволить розробникам легко визначати та контролювати структуру користувацького інтерфейсу додатку.

Крім того, Angular підтримує двостороннє зв’язування даних, що дозволяє при зміні моделі автоматично оновлювати представлення, а при зміні представлення – модель. Це допомагає спростити процес створення складних, керованих даними додатків.

React.js, Vue.js, та Angular мають свої особливості, що пояснюють вибір React. Перший простий в освоєнні та гнучкий, дозволяючи розробникам самостійно обирати додаткові бібліотеки. Другий також легко засвоюється, але має меншу екосистему, ніж React. Angular ж складніший і вимагає більше часу для навчання, але пропонує повну екосистему. React забезпечує високу продуктивність через використання Virtual DOM, тоді як Angular покладається на реальний DOM, що може знижувати швидкість в деяких випадках. React також має найбільшу спільноту, що спрощує розробку й пошук ресурсів.

Також у даному проєкті було вирішено використовувати Next.js, що надає додаткові можливості для оптимізації продуктивності та зручності розробки. Next.js дозволяє виконувати рендеринг на стороні сервера (SSR) та генерацію статичних сторінок (SSG), що значно покращує швидкість завантаження сторінок і SEO-оптимізацію [11].

Tailwind CSS використовується як основний інструмент для швидкого та ефективного створення адаптивного і чистого інтерфейсу, що дозволяє значно

прискорити розробку завдяки його утилітарному підходу до стилізації без необхідності написання кастомних CSS-класів.

Tailwind CSS – чудовий вибір для веб-розробників з кількох вагомих причин. По-перше, Tailwind пропонує спрощений та ефективний підхід до веб-розробки завдяки своїй методології, орієнтованій на практичність. Цей підхід дозволяє розробникам швидко створювати сучасні та адаптивні веб-сайти без необхідності писати великі кастомні стилі CSS. Маючи у своєму розпорядженні багатий набір утиліт-класів, можна легко розробляти макети, застосовувати стилі та керувати інтерактивністю з мінімальними зусиллями. Це значно зменшує потребу в написанні та підтримці складних CSS-файлів, що призводить до більш чистого та зручного для підтримки коду. Крім того, утилітарні класи Tailwind дуже виразні, що дозволяє легко зрозуміти призначення та функціональність кожного класу [12].

Рисунок 1.17 – Логотип Tailwind CSS

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ АУДИТУ КОДУ

2.1. Визначення цілей та вимог

Проєкт має на меті створити веб-додаток для автоматизованого аудиту коду з використанням штучного інтелекту. Основні цілі проєкту включають покращення якості програмного забезпечення, виявлення потенційних вразливостей, підвищення ефективності процесу розробки та забезпечення відповідності стандартам безпеки й якості коду. Додаток також надасть користувачам глибокий аналіз коду, рекомендації для оптимізації та безперебійний процес інтеграції з іншими розробницькими інструментами.

Функціональні вимоги:

- можливість завантаження та аналізу вихідного коду користувача;
- автоматичне виявлення проблем у коді, таких як помилки безпеки, нестандартні практики та проблеми продуктивності;
- генерація звіту з результатами аналізу, включаючи рекомендації щодо покращення коду;
- генерація звіту у PDF-форматі;
- модульна архітектура з можливістю інтеграції із системами контролю версій (наприклад, Git) та CI/CD процесами.

Нефункціональні вимоги:

- забезпечення стабільної роботи при великих обсягах коду;
- продуктивний аналіз коду для оптимізації процесів розробки;
- інтуїтивно зрозумілий інтерфейс для розробників різного рівня;
- захист конфіденційних даних користувача та результатів аудиту;
- можливість розширення функціональності та підтримки нових стандартів.

Цільова аудиторія проєкту включає команди розробників, тестувальників, інженерів з безпеки, а також окремих фрілансерів, які прагнуть підвищити якість

і безпеку свого коду. Основна аудиторія – це IT-спеціалісти з досвідом розробки, зацікавлені в автоматизованому інструменті, який може інтегруватись у їх робочі процеси та забезпечити систематичний аудит коду.

2.2. Архітектура додатку

Для побудови архітектури додатку обрано підхід Model-View-Controller (MVC). Це дозволяє розділити логіку програми на три основні компоненти: Model, View, Controller.

Model (Модель): Відповідає за управління даними та бізнес-логікою. У нашому випадку це аудитор, який виконує аналіз коду та зберігає результати.

View (Подання): Відповідає за відображення даних. Це React-компоненти, що відповідають за рендеринг інтерфейсу користувача.

Controller (Контролер): Відповідає за взаємодію між поданням та моделлю. Контролер приймає запити від користувача, обробляє їх через модель, та повертає результати у вигляді подання [13].

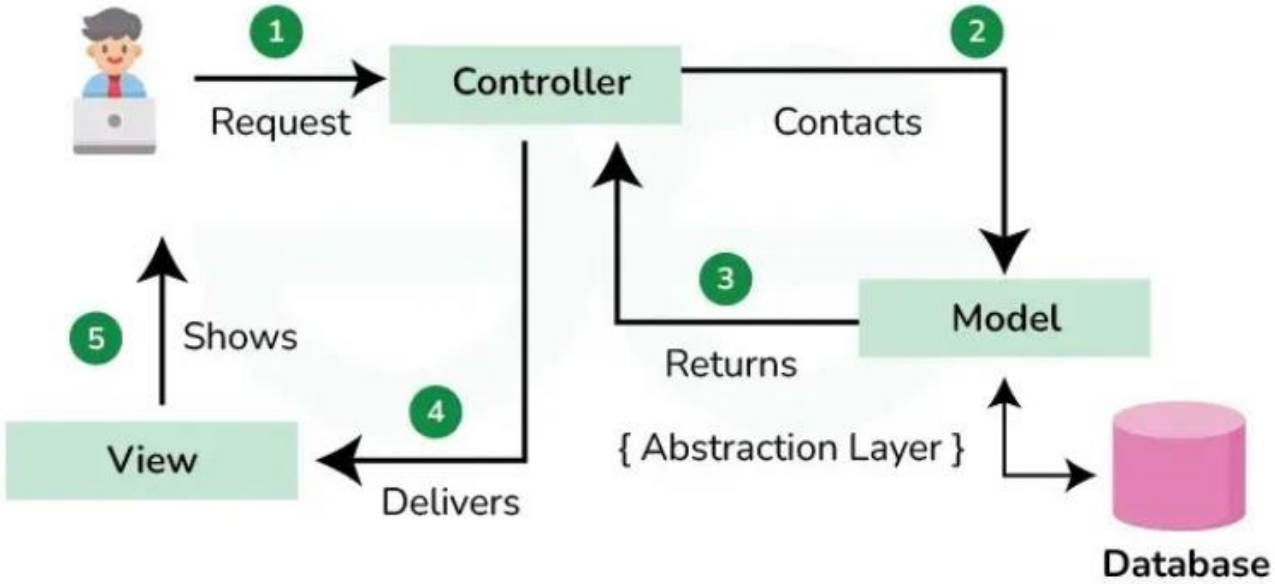


Рисунок 2.1 – Шаблон проєктування MVC

Для реалізації додатку використано React для компонентного підходу та управління станом, TypeScript для статичної типізації й уникнення помилок, а також Next.js для серверного рендерингу та генерації статичних сторінок, що знижує навантаження на клієнт і підвищує продуктивність.

Структура проєкту складається з наступних каталогів:

1. src/ – є основним каталогом, де зберігається весь код додатку. Це включає в себе різні компоненти, службові функції та інші важливі файли, які забезпечують функціонування додатку.

2. app/ – Каталог app/ містить основні файли та підкаталоги додатку:

2.1. Підкаталог fonts/ - цей каталог призначений для зберігання файлів шрифтів, які використовуються в додатку. Це дозволяє забезпечити консистентний та привабливий вигляд тексту на всіх сторінках.

2.2. Файл «favicon.ico» – Цей файл містить іконку для браузера, яка відображається на вкладці браузера поруч із назвою сторінки. Це невелике зображення формату .ico.

2.3. Файл «globals.css» – файл глобальних стилів CSS, який застосовується до всього додатку. Тут визначені загальні стилі, які будуть використовуватися на всіх сторінках додатку, такі як основні кольори, шрифти, відступи та інші стилістичні елементи.

2.4. Файл «layout.tsx» – файл компонента Layout. Він зазвичай використовується для визначення загальної структури сторінок додатку. У ньому можуть міститися заголовок (header), підвал (footer) та основний контент, що змінюється залежно від маршрутизації.

2.5. Файл «page.tsx» – файл компонента сторінки. Це основний файл, який відповідає за відображення вмісту на конкретній сторінці. Він включає в себе логіку для відображення різних компонентів та даних.

3. components/ – призначений для зберігання усіх окремих компонентів, які складають інтерфейс користувача додатку. Має підкаталог ui/ який у свою чергу містить файл «wavy-background.tsx». Файл «wavy-background.tsx» містить компонент для створення анімованого хвилястого фону,

що додає візуальний ефект і покращує естетику сторінки. Також каталог components/ містить в собі наступні файли:

3.1. Файл «code-input.tsx» – містить компонент, який надає інтерфейс для вводу та редагування коду з функціями підсвічування синтаксису та валідації.

3.2. Файл «header.tsx» – містить компонент, який відповідає за відображення заголовка сторінки. Цей компонент використовує інший компонент «WavyBackground» для додавання хвилястого фону та забезпечення привабливого візуального ефекту.

3.3. Файл «result-modal.tsx» – містить компонент, який відповідає за відображення модального вікна з результатами аудиту коду.

4. utils/ - містить допоміжні функції та утиліти, які використовуються в різних частинах додатку для спрощення коду та підвищення його зручності в обслуговуванні. Має наступні файли:

4.1. Файл «ai-prompt.ts» – містить функцію, яка використовується для виконання аудиту коду за допомогою OpenAI API.

4.2. Файл «cn.ts» – містить функцію, яка використовується для об’єднання класів CSS. Вона забезпечує зручний спосіб управління класами, особливо у випадках умовного застосування стилів.

4.3. Файл «exportToPDF.ts» – містить функціонал генерації PDF-звіту результатів аудиту коду.

Окрім каталогів і файлів, які описані вище, ми також маємо й інші файли кореневої папки, які є конфігураційними та службовими:

1. «.env» – файл змінних оточення в Node. Використовуються для зберігання конфіденційних даних, таких як паролі, облікові дані API та інша інформація, яку не слід писати безпосередньо в коді.

2. «.gitignore» – це текстовий файл, який повідомляє Git, які файли чи папки ігнорувати в проєкті.

3. «next.config.mjs» – конфігураційний файл, який налаштовує параметри для проєкту Next.js.

4. «package.json» – містить різні метадані про проєкт, включаючи залежності, скрипти, які потрібні для запуску проєкта. Даний файл є ключовим для будь якого Node.js проєкту.

5. «package-lock.json» – містить інформацію про версії залежностей проєкту. Це гарантує однаковість версій для всіх розробників, які будуть мати справу з проєктом.

6. «postcss.config.mjs» – містіть конфігураційні параметри для PostCss, який відповідає за перетворення css за допомогою плагінів

7. «tsconfig.json.» – містить конфігураційні опції для компілятора TypeScript. Це включає налаштування шляхів, модулів, параметри компіляції та інші аспекти TypeScript проєкту [14].

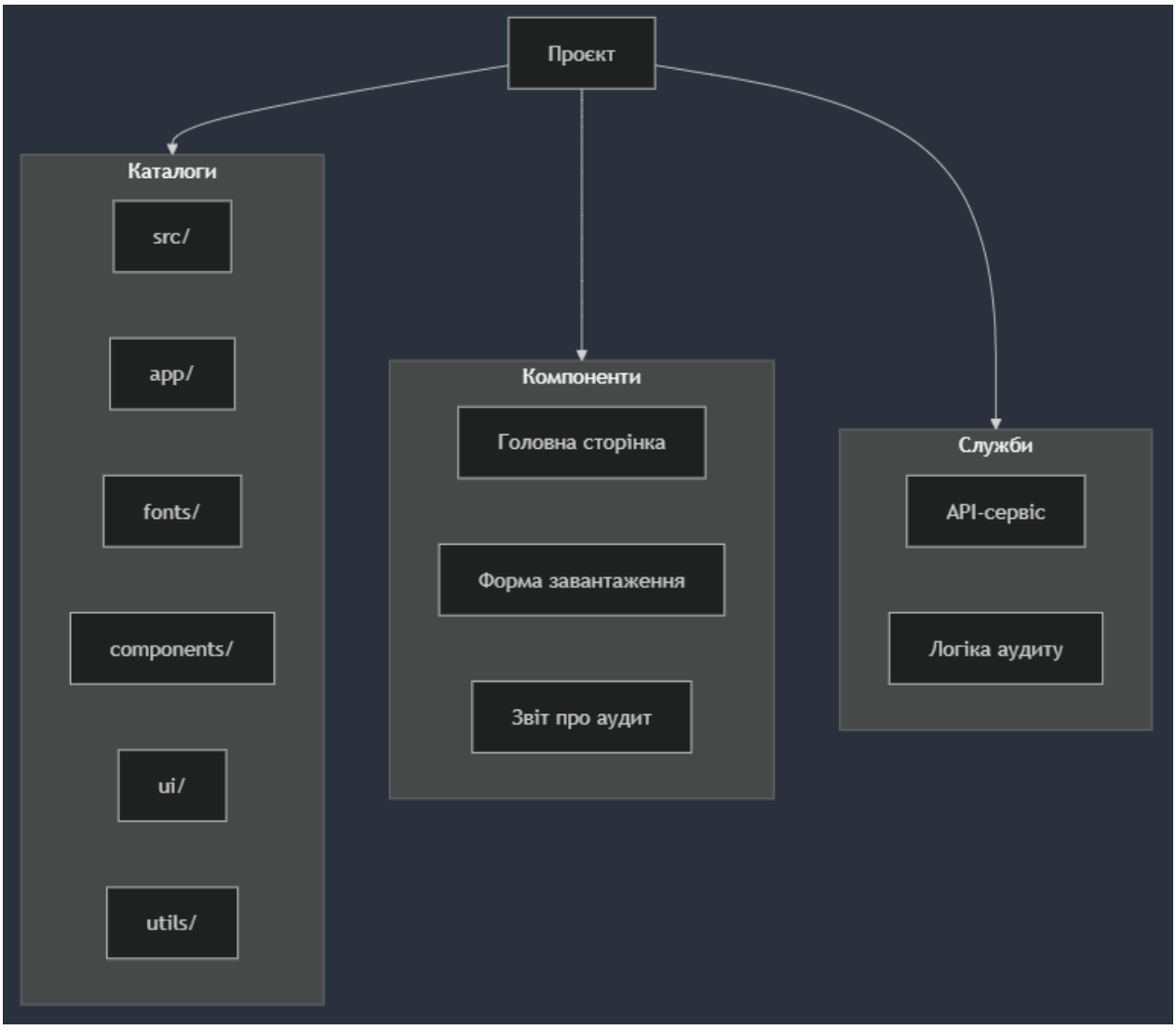


Рисунок 2.2 – Структура проєкту

2.3. Проєктування дизайну інтерфейсу

Дизайн інтерфейсу користувача (UI) та досвіду користувача (UX) цього додатку заснований на принципах зручності, ефективності та привабливості. Інтерфейс побудований максимально просто, без зайвих елементів, що дозволяє користувачам легко орієнтуватися та не відволікатися. Навігація інтуїтивно зрозуміла: користувач може швидко знаходити потрібну інформацію завдяки зручним меню та кнопкам. Усі елементи інтерфейсу узгоджені між собою за стилем та поведінкою, що забезпечує консистентність і запобігає плутанині.

Для розробки шаблону проєкту використовувся інструмент Figma. Figma - це хмарний інструмент для дизайну, схожий на Sketch за функціональністю та можливостями, але з великими відмінностями, які роблять Figma кращим інструментом для командної роботи.

Figma зробила революцію в тому, як дизайнери та команди співпрацюють над проектами, пропонуючи безліч функцій, які спрощують весь процес проектування. Однією з найважливіших переваг Figma є її здатність безперешкодно працювати на будь-якій платформі. Незалежно від того, чи використовуєте ви macOS, Windows або навіть Linux, веб-природа Figma гарантує, що можна отримати доступ до файлів дизайну з будь-якого пристрою, підключеного до Інтернету. Ця крос-платформенна сумісність усуває бар'єри, традиційно пов'язані з програмним забезпеченням, яке обмежене певними операційними системами. Співпраця у Figma інтуїтивно зрозуміла і звична, подібна до досвіду використання інших популярних інструментів для спільної роботи. Кілька членів команди можуть одночасно працювати над одним і тим же файлом проекту, а оновлення в реальному часі миттєво відображаються для всіх користувачів [15].

Процес роботи над шаблоном даного проєкту:

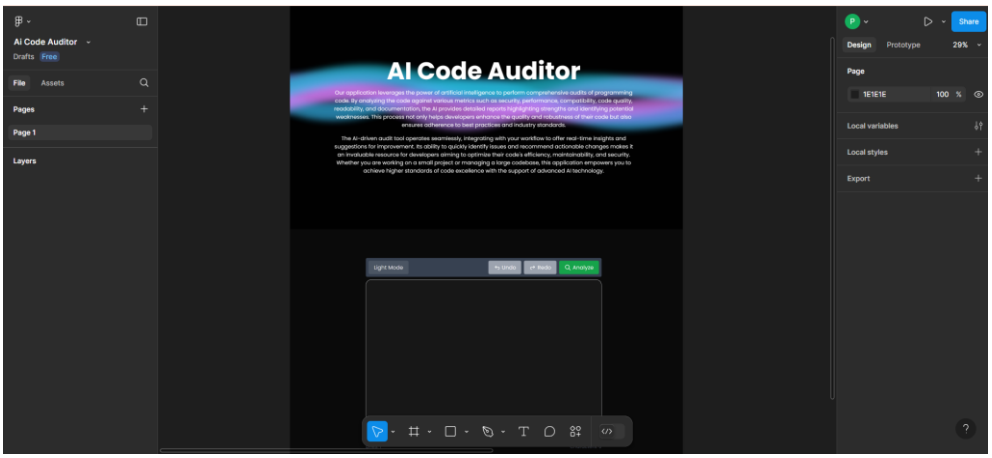


Рисунок 2.3 – Інтерфейс Figma

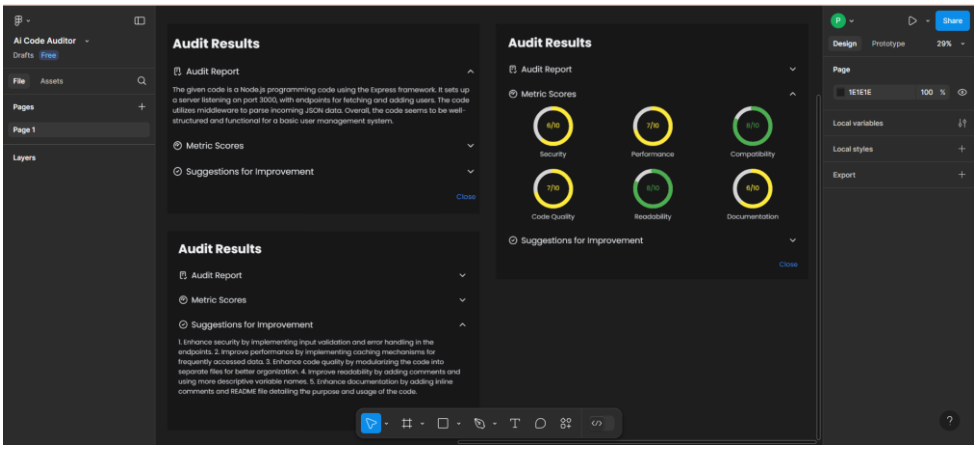


Рисунок 2.4 – Інтерфейс Figma

Інтерфейс додатку складається з декількох основних компонентів. Головна сторінка відображає результати аудиту та загальну інформацію про проєкт. Вона може містити графіки, діаграми та інші візуалізації, що допомагають користувачеві зрозуміти результати аудиту.

Форма завантаження дозволяє користувачам завантажувати код для аудиту за допомогою полів для вводу, кнопок завантаження та індикаторів процесу. Цей компонент спроектований з урахуванням зручності користувача, забезпечуючи легке та швидке завантаження файлів. Підтримка різних форматів файлів та можливість перетягування забезпечують додатковий комфорт для користувачів.

Звіт про аудит відображає детальну інформацію про результати аудиту, включаючи оцінки за різними метриками (безпека, продуктивність, якість коду тощо) та рекомендації щодо поліпшення коду.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ АУДИТУ ПРОГРАМНОГО КОДУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

3.1. Підготовка середовища

Перший етап передбачає підготовку середовища для розробки. Це включає встановлення необхідного програмного забезпечення та інструментів, створення проєкту і налаштування базової конфігурації. Для розробки ми будемо використовувати IDE PhpStorm (рис. 3.1), яка надає потужні засоби для написання та налагодження коду, а також підтримує інтеграцію з популярними інструментами та бібліотеками.

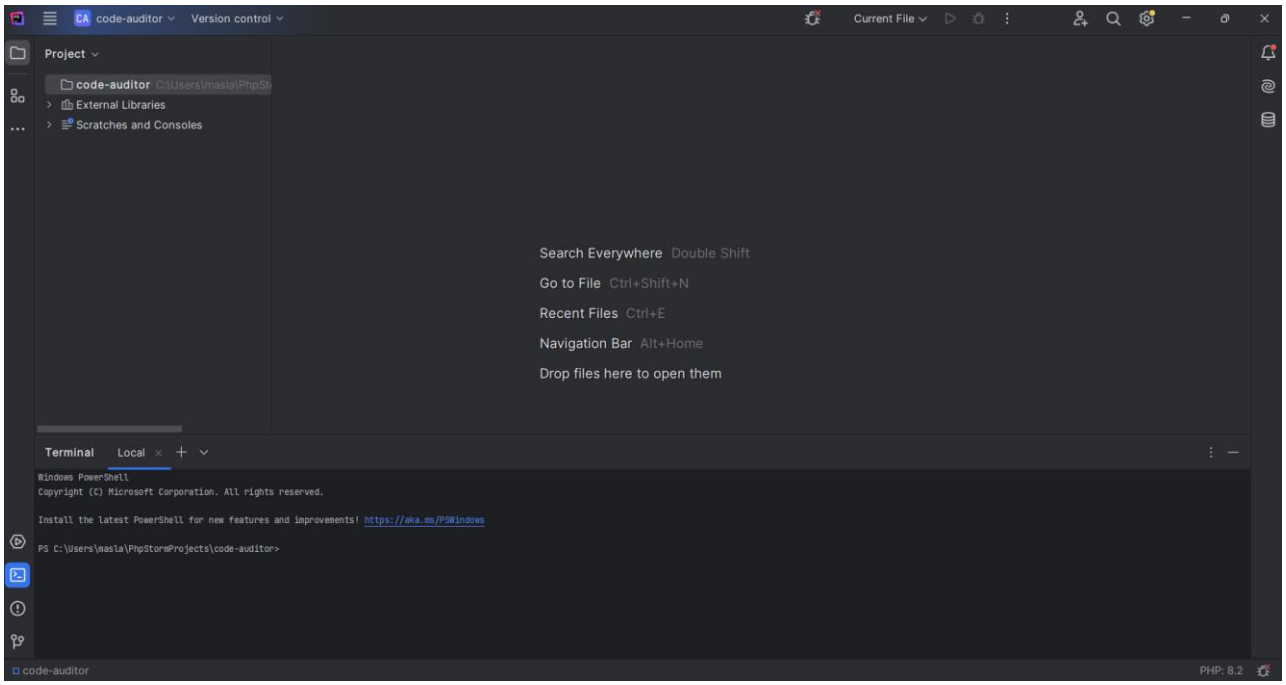


Рисунок 3.1 – Інтерфейс IDE PhpStorm

Першим кроком буде встановлення Node.js та npm.

Node.js – це середовище виконання JavaScript, яке дозволяє запускати JavaScript-код поза браузером. npm (Node Package Manager) – це менеджер

пакетів для Node.js, який використовується для встановлення та управління залежностями проєкту.

Отже, для встановлення переходимо на офіційний сайт Node.js та завантажуємо рекомендовану LTS (Long Term Support) версію:

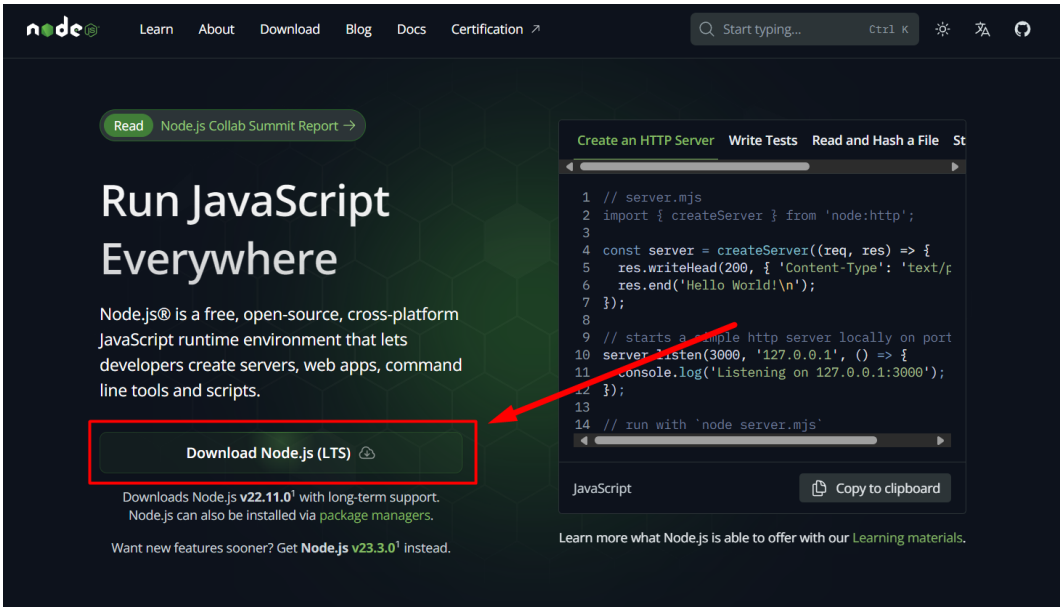


Рисунок 3.2 – Офіційний сайт Node.js [16]

Після завантаження інсталятора слідуємо інструкціям нашої операційної системи продовж встановлення.

Щоб перевірити, що інсталяція пройшла успішно, відкриваємо термінал та виконуємо команди «node –v» та «npm –v». Якщо у результаті виконання даних команд ми отримали інформацію про встановлені версії, то інсталяція пройшла успішно.

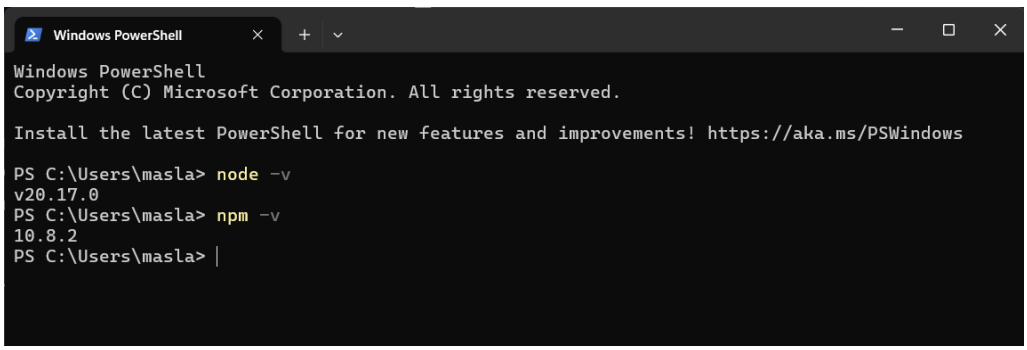


Рисунок 3.3 – Перевірка встановлення Node.js та npm

Щоб створити веб-застосунок, ми скористаємось фреймворком для React, який має назву Next.js. Він забезпечує інструменти для створення швидких та зручних для пошукових систем веб-застосунків.

Офіційна документація рекомендує запускати новий Next.js-додаток за допомогою команди «npx create-next-app», який автоматично все налаштовує для нас:

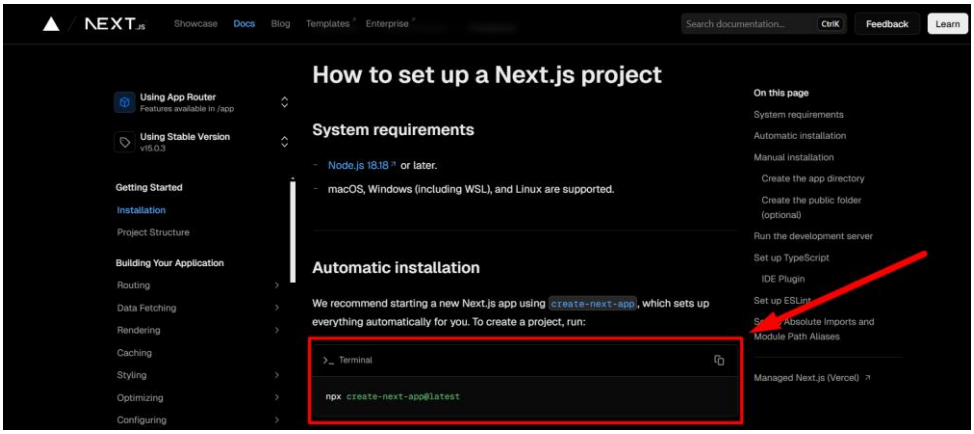


Рисунок 3.4 – Офіційна веб-сторінка документації Next.js [17]

Під час встановлення ми бачимо перелік конфігураційних запитань для нашої програми:

```
PS C:\Users\masla\PhpStormProjects\code-auditor> npx create-next-app@latest
Need to install the following packages:
create-next-app@15.0.3
Ok to proceed? (y) y

√ What is your project named? ... code-auditor
√ Would you like to use TypeScript? ... No / Yes
√ Would you like to use ESLint? ... No / Yes
√ Would you like to use Tailwind CSS? ... No / Yes
√ Would you like your code inside a `src/` directory? ... No / Yes
√ Would you like to use App Router? (recommended) ... No / Yes
√ Would you like to use Turbopack for next dev? ... No / Yes
√ Would you like to customize the import alias (@/* by default)? ... No / Yes
Creating a new Next.js app in C:\Users\masla\PhpStormProjects\code-auditor\code-auditor.

Using npm.

Initializing project with template: app-tw


Installing dependencies:
- react
- react-dom
- next

Installing devDependencies:
- typescript
- @types/node
- @types/react
- @types/react-dom
- postcss
- tailwindcss


added 143 packages, and audited 144 packages in 34s
```

Рисунок 3.5 – Процес створення Next.js додатку

В результаті успішного створення початкового додатку, ми бачимо наступну структуру файлів у нашому IDE:

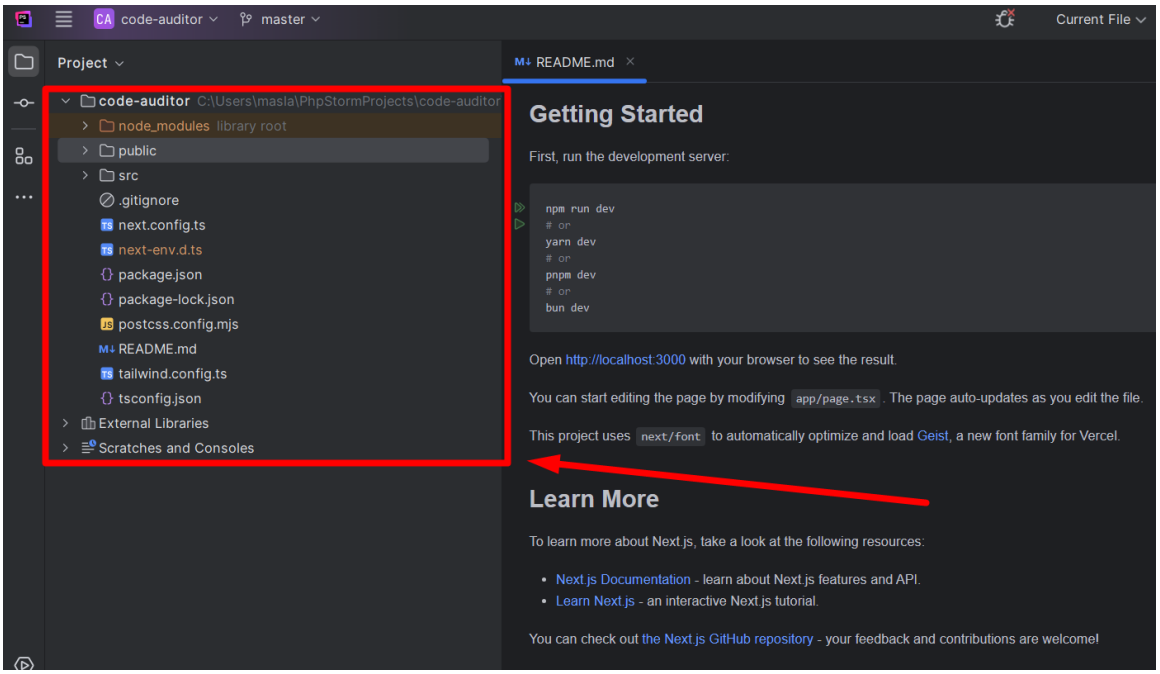


Рисунок 3.6 – Початкова файлова структура проєкту

Почнемо з роботи над файлом «layout.tsx», який знаходиться за замовчуванням у проєкті на базі Next.js та відповідає за загальний вигляд структури нашого додатку та його глобальних стилей. Це дозволяє нам централізовано налаштувати такі речі як шрифти, метадані або ж просто глобальні CSS-стилі.

Спершу у даному файлі ми описуємо метадані, а саме встановлюємо заголовок та опис програми через змінну «metadata». Цей етап є важливим для SEO та представлення сторінок.

Також ми інтегруємо потрібний нам шрифт «Poppins» через бібліотеку «next/font/google», що спрощує підключення та оптимізацію шрифтів.

Останнім етапом редагування даного файлу є створення базового шаблону, включаючи такі HTML-теги як <html> та <body> та додавання до останнього класу, який містить налаштування шрифту всієї програми.

```tsx
import type {Metadata} from "next";
import {Poppins} from "next/font/google";
import "./globals.css";

const poppins = Poppins({weight: ["400", "700"], subsets: ["latin"]});

no usages  Maslak-M *
export const metadata: Metadata = {
    title: "Code Auditor",
    description: "AI Code Auditor",
};

no usages  Maslak-M *
export default function RootLayout({
                                       children,
                                   }: Readonly<{
    children: React.ReactNode;
}>) {
    return (
        <html lang="en">
        <body className={poppins.className}>{children}</body>
        </html>
    );
}
```

Рисунок 3.7 – Налаштований файл «layout.tsx»

Тепер переходимо до файлу «page.tsx», який є точкою входу для рендерингу головної сторінки додатку. Він відповідає за відображення інтерфейсу та інтеграцію компонентів, які містять функціонал програми.

Для зручності ми створюємо окрему папку «components», де у нас будуть зберігатись усі файли компонентів.

3.2. Розробка компонентів додатку

Першим компонентом, над яким ми працюватимемо, буде «Header». Він відповідає за відображення заголовка та опису функцій додатку. Це є важливою частиною користувацького інтерфейсу, адже він відіграє ключову роль у першому враженні користувача при відвідуванні додатку.

Даний компонент має декоративний фон, який реалізовано за допомогою «WavyBackground» – кастомного візуального елементу, що додає привабливий анімований градієнтний фон. Щоб застосувати даний візуальний елемент, ми використовуємо бібліотеку «Acertenity UI», яка надає прості, але потужні інструменти створення сучасного та привабливого UI. Її компоненти легко інтегрувати та стилізувати відповідно до потреб проєкту.

Отже, відвідуємо офіційний сайт «Acertenity UI», знаходимо елемент, який нас цікавить, та слідуємо інструкціям по використанню, де вказано, що ми маємо інсталювати залежності та додати новий файл з назвою «utils.ts», проте у нашому проєкті було вирішено назвати даний файл «cn.tsx», що є скороченою назвою «class names».

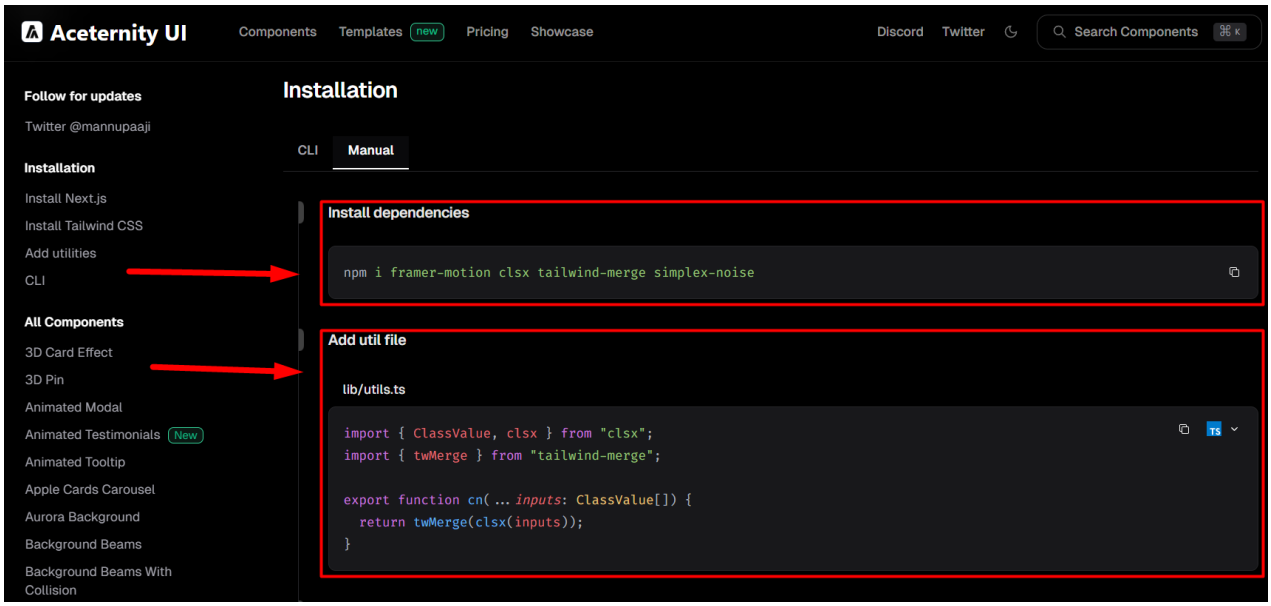


Рисунок 3.8 – Документація по використанню візуального елементу «WavyBackground» [18]

Далі ми створюємо окремий файл компоненти з назвою «wavy-background.tsx», куди ми копіюємо код нашого візуального елемента:

Copy the source code

components/ui/wavy-background.tsx

```
"use client";
import { cn } from "@/lib/utils";
import React, { useEffect, useRef, useState } from "react";
import { createNoise3D } from "simplex-noise";

export const WavyBackground = ({
  children,
  className,
  containerClassName,
  colors,
```

Expand

Рисунок 3.9 – Код візуального елемента «WavyBackground»

Тепер ми інтегруємо наш створений компонент у вже існуючий компонент «Header» та додаємо текстові елементи контенту. Важливо зазначити, що текст стилізовано з використання класів Tailwind CSS, що забезпечує гарний вигляд на різних розмірах екрану. Код компоненту «Header» має наступний вигляд:

```
import {WavyBackground} from "@/components/ui/wavy-background";

Show usages
export default function Header() {
    return (
        <WavyBackground className="max-w-4xl mx-auto pb-20">
            <p className="text-4xl md:text-4xl lg:text-7xl text-white font-bold inter-var text-center">
                AI Code Auditor
            </p>
            <p className="text-base md:text-1xl mt-4 text-white font-normal inter-var text-center">
                Our application leverages the power of artificial intelligence to perform comprehensive audits of
                programming code. By analyzing the code against various metrics such as security, performance,
                compatibility, code quality, readability, and documentation, the AI provides detailed reports
                highlighting strengths and identifying potential weaknesses. This process not only helps developers
                enhance the quality and robustness of their code but also ensures adherence to best practices and
                industry standards.
            </p>
            <p className="text-base md:text-1xl mt-4 text-white font-normal inter-var text-center">
                The AI-driven audit tool operates seamlessly, integrating with your workflow to offer real-time insights
                and suggestions for improvement. Its ability to quickly identify issues and recommend actionable changes
                makes it an invaluable resource for developers aiming to optimize their code's efficiency,
                maintainability, and security. Whether you are working on a small project or managing a large codebase,
                this application empowers you to achieve higher standards of code excellence with the support of
                advanced AI technology.
            </p>
        </WavyBackground>
    );
}
```

Рисунок 3.10 – Код компонента «Header»

Після додавання компоненти «Header» у файл «page.tsx», запускаємо «localhost» – процес розгортання локального сервера для тестування веб-

додатків на комп’ютері. Це забезпечується таким серверним програмним забезпеченням, як Node.js.

Для запуску користуємось командою «npm run dev», після чого сервер буде доступний за адресою http://localhost:3000.

Тепер ми можемо бачити наш робочий веб додаток із першим доданим компонентом:

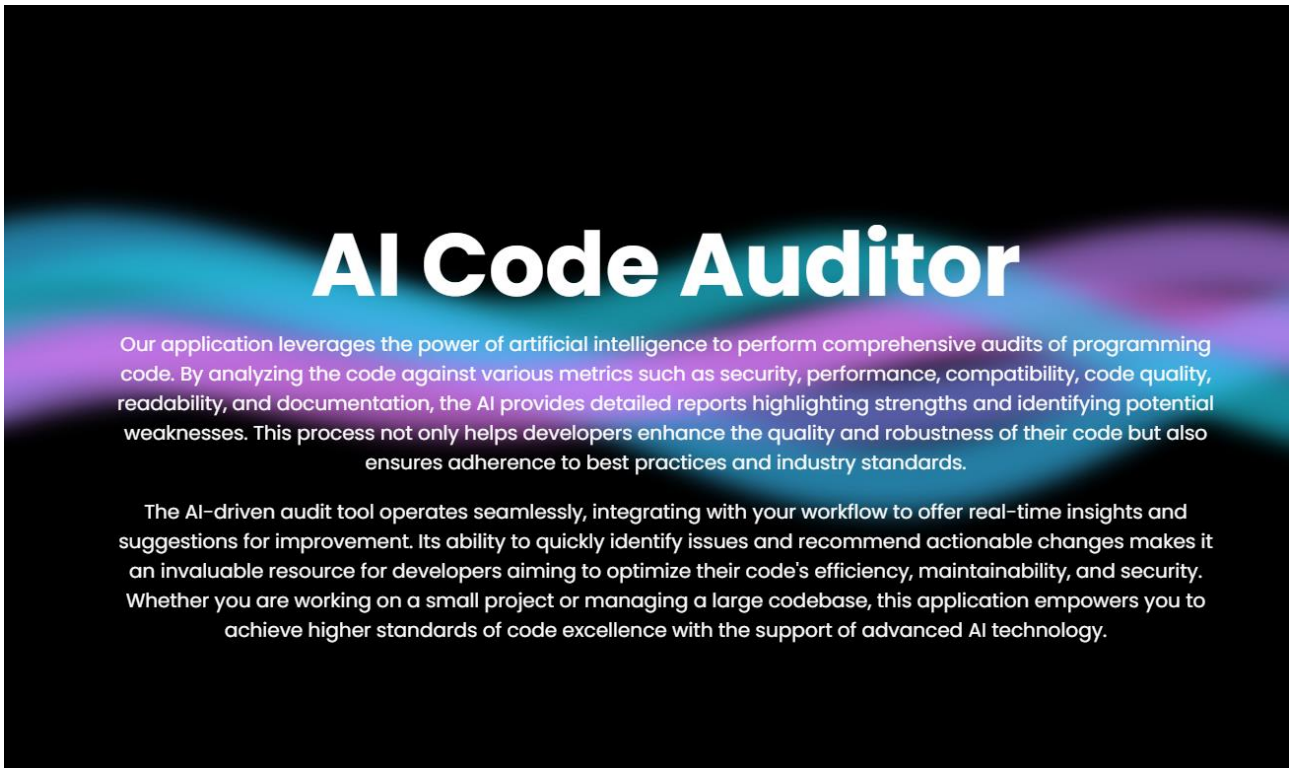


Рисунок 3.11 – Візуальний вигляд компонента «Header»

Далі ми приступаємо до розробки компонента «CustomCodeEditor», який реалізовано у файлі «code-input.tsx», який відповідає за створення інтерактивного поля вводу для коду.

Для успішної реалізації даного компонента, нам знадобиться допомога деяких бібліотек, а саме «react-simple-code-editor», яка надає зручний редактор для роботи з текстом, та «prismjs», яка відповідає за підсвічування синтаксису коду з використанням визначеного мовою стилю [19, 20].

Отже, для інсталяції прописуємо команди «npm install react-simple-code-editor» та «npm install prismjs», та додаємо до нашого файлу «code-input.tsx» за

допомогою ключового слова «import», яке використовується в JavaScript для імпортування модулів. Також було встановлено стилі «prism.css» і теми, які дозволяють змінювати тему редактора на світлу або темну

```
import React, { useState } from "react";
import Editor from "react-simple-code-editor";
import Prism from "prismjs";
import "prismjs/components/prism-javascript";
import "prismjs/themes/prism-tomorrow.css"; // Темна тема
import "prismjs/themes/prism.css"; // Світла тема
import { IconArrowBackUp, IconArrowForwardUp, IconSearch,} from "@tabler/icons-react";
```

Рисунок 3.12 – Імпорт бібліотек для редактора коду

Оскільки у нашому додатку згідно дизайну мають використовуватись іконки, було вирішено використовувати іконки із бібліотеки «Tabler». Це сучасна бібліотека інтерфейсних компонентів, які дозволяють забезпечити гнучкість та простоту веб-додатку. Але у нашому випадку нас цікавлять лише іконки, які надає дана бібліотека. Отже, щоб їх використовувати, інсталюємо відповідний пакет:

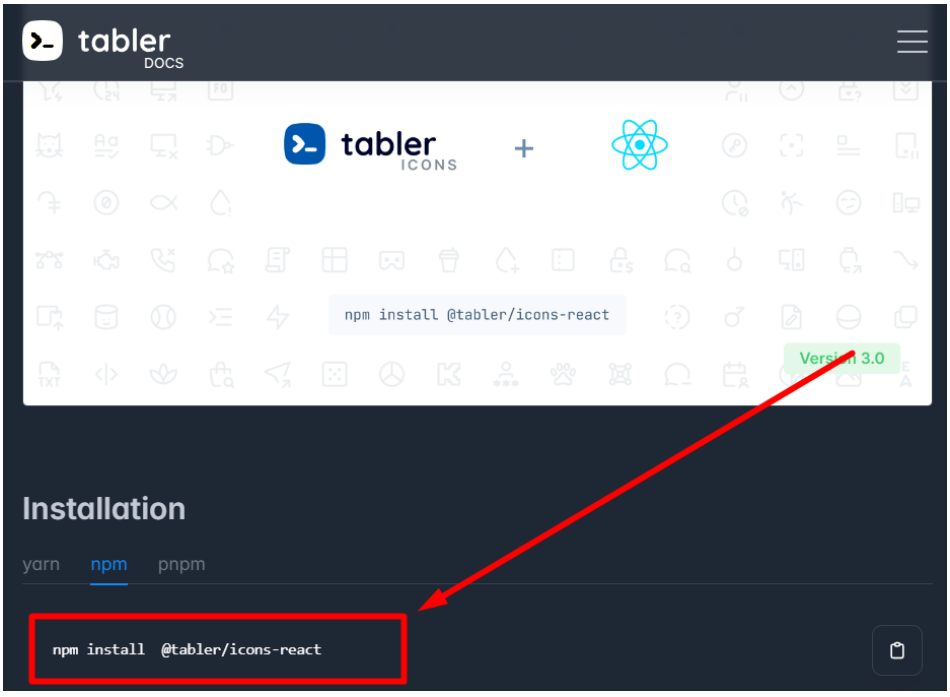


Рисунок 3.13 – Інсталяція готових іконок з бібліотеки «Tabler» [21]

Роботу над компонентом розпочинаємо з визначення його інтерфейсу. Для цього створюємо типи для основних пропсів:

- «code» – текст коду, який відображається та змінюється у редакторі;
- «setCode» – функція, що відповідає за оновлення тексту коду;
- «analyze» – функція, що аналізує введений код.

Також у редакторі коду ми реалізували функцію підсвічування синтаксису коду. Спеціально для цього створено функцію «highlightWithPrism», яка обробляє введений текст за допомогою «Prism.js» та змінює колір ключових слів, змінних операторів тощо. Ця функція значно полегшує орієнтування користувача у коді.

```
const highlightWithPrism = (code: string) => {
    return Prism.highlight(code, Prism.languages.javascript, "javascript");
};
```

Рисунок 3.14 – Код функції «highlightWithPrism»

Компонент «CustomCodeEditor» використовує різні стани, які забезпечують його функціональність та гнучкість. Стан у ReactJS – це об’єкт, який зберігає динамічні дані компонента і визначає його поведінку. Це екземпляр класу React-компонентів, який можна визначити як об’єкт з набором спостережуваних властивостей, що керують поведінкою компонента.

```
const [theme, setTheme] = useState("dark");
const [history, setHistory] = useState<string[]>([code]);
const [historyIndex, setHistoryIndex] = useState(0);
```

Рисунок 3.15 – Оголошення станів компоненти «CustomCodeEditor»

Почнемо з першого стану «theme», який відповідає за зміну теми у нашому редакторі. Для перемикання теми використовується функція toggleTheme, яка оновлює стан теми:

```
const toggleTheme = () =>{
    setTheme((prevTheme) => (prevTheme === "dark" ? "light" : "dark"));
}
```

Рисунок 3.16 – Код функції «toggleTheme»

Також додаємо стани «history» та «historyIndex», які допоможуть у реалізації функціоналу історії версій коду, щоб користувач міг повернутись до попередніх версій коду, або навпаки повернутись до найновіших.

Для цього було створено три функції:

1. «saveToHistory» – відповідає за зберігання змін коду в масив історії. Вона викликається щоразу, коли користувач редагує код.
2. «handleUndo» – дозволяє повернутись до попердньої версії коду.
3. «handleRedo» – відновлює наступну версію коду, які така версія існує.

```
28      // Undo
        Show usages
29      const handleUndo = () => {
30          if (historyIndex > 0) {
31              setHistoryIndex(historyIndex - 1);
32              setCode(history[historyIndex - 1]);
33          }
34      };
35
36      // Redo
        Show usages
37      const handleRedo = () => {
38          if (historyIndex < history.length - 1) {
39              setHistoryIndex(historyIndex + 1);
40              setCode(history[historyIndex + 1]);
41          }
42      };
43
44      // Save history
        Show usages
45      const saveToHistory = (newCode: string) => {
46          const updatedHistory = [...history.slice(0, historyIndex + 1), newCode];
47          setHistory(updatedHistory);
48          setHistoryIndex(updatedHistory.length - 1);
49      };
```

Рисунок 3.17 – Код функцій «saveToHistory», «handleUndo», «handleRedo»

Коли функціональна складова виконана, приступаємо до розробки візуального інтерфейсу компоненти. Для стилізації було використано класи «TailwindCSS». Код панелі інструментів зображено нижче:

```
{/* Toolbar */}
<div className="flex justify-between mb-2 p-2 bg-gray-200 dark:bg-gray-700 rounded">
    <button
        onClick={toggleTheme}
        className="px-4 py-2 rounded bg-gray-300 dark:bg-gray-600"
    >
        {theme === "dark" ? "Light Mode" : "Dark Mode"}
    </button>
    <div className="flex gap-x-2">
        <button
            onClick={handleUndo}
            className="flex items-center gap-x-1 px-4 py-2 rounded bg-gray-400 text-white"
        >
            <IconArrowBackUp size={20} /> Undo
        </button>
        <button
            onClick={handleRedo}
            className="flex items-center gap-x-1 px-4 py-2 rounded bg-gray-400 text-white"
        >
            <IconArrowForwardUp size={20} /> Redo
        </button>
        <button
            onClick={analyze}
            className="flex items-center gap-x-1 px-4 py-2 rounded bg-green-600 text-white"
        >
            <IconSearch size={20} /> Analyze
        </button>
    </div>
</div>
```

Рисунок 3.18 – Код панелі інструментів з кнопками

Також було вирішено додати можливість завантаження локальних файлів з кодом та відображати їх у редакторі з метою подальшого аудиту. Для цього було створено додатковий обробник та форму завантаження файлів:

```
const handleFileUpload = (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0];
    if (file) {
        const reader = new FileReader();
        reader.onload = () => {
            const fileContent = reader.result as string;
            setCode(fileContent);
            saveToHistory(fileContent);
        };
        reader.readAsText(file);
    }
};
```

Рисунок 3.19 – Обробник завантаження локальних файлів

Код, який відповідає за вивід текстового поля для редактора, виглядає наступним чином:

```
{/* Editor */}
<div
    className={`border rounded-2xl p-6 ${
        theme === "dark"
            ? "dark:bg-neutral-900 dark:text-neutral-200"
            : "bg-white text-black"
    }`}
    style={{ height: "450px", overflowY: "auto" }}
>
    <Editor
        value={code}
        onValueChange={handleCodeChange}
        highlight={(code) => highlightWithPrism(code)}
        padding={15}
        textareaId="code-editor"
        className="textarea-editor"
        textareaClassName="outline-none"
        style={{
            fontFamily: '"Fira Mono", monospace',
            fontSize: 17,
            minHeight: "100%",
            background: "transparent",
            color: "inherit",
        }}
    />
</div>
```

Рисунок 3.20 – Код текстового поля редактора коду

У результаті ми отримуємо такий вигляд редактора коду:

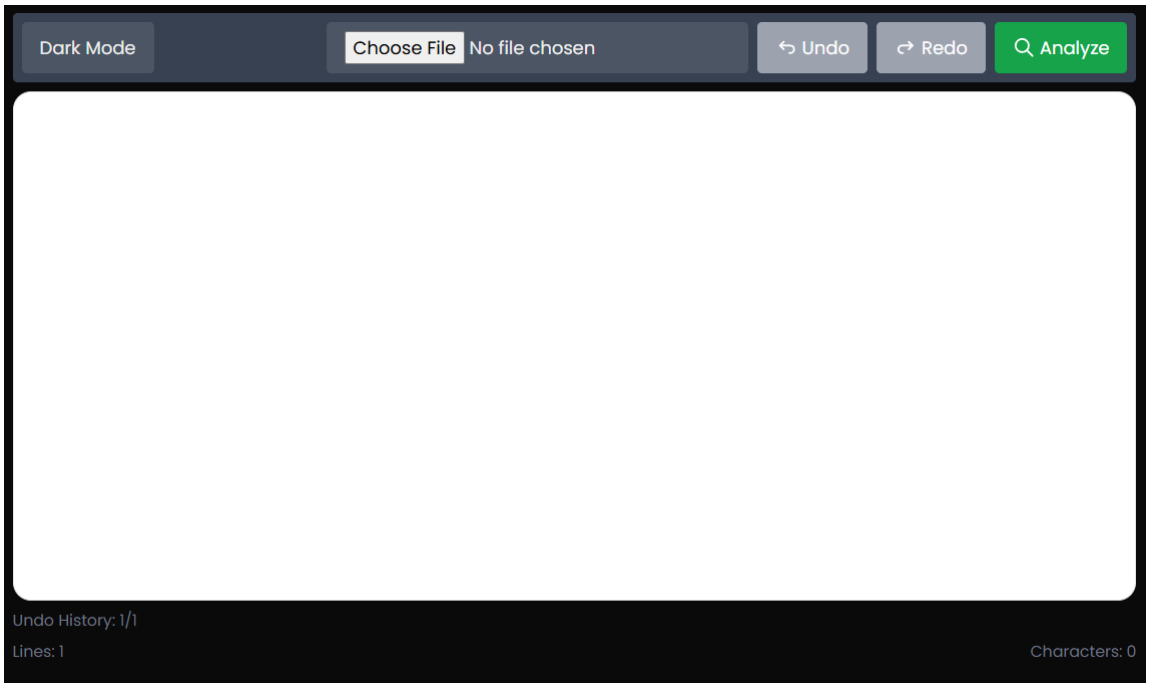


Рисунок 3.21 – Користувацький інтерфейс редактора коду

Файл «ai-prompt.tsx» є ключовою частиною даного проєкту, що забезпечує функціонал аналізу коду з використанням мовної моделі від OpenAI. Головною метою даного файлу є забезпечення обміну інформацією між користувацьким інтерфейсом та штучним інтелектом, який виконує аудит програмного коду.

Перш за все, ми маємо встановити необхідну для роботи з OpenAI залежність, перед цим виконавши команду «npm install openai».

Після успішного встановлення та імпорту пакету, переходимо до налаштування OpenAI API. Для цього створюємо екземпляр клієнта OpenAI:

```
const apiKey = process.env.NEXT_PUBLIC_API_KEY;

const openai = new OpenAI({
    apiKey: apiKey,
    dangerouslyAllowBrowser: true,
});
```

Рисунок 3.22 – Код конфігурації OpenAI API

На скріншоті вище ми вказуємо унікальний користувацький API ключ, який зберігається у змінні середовища «NEXT_PUBLIC_API_KEY» та параметр «dangerouslyAllowBrowser», який використовується для доступу по API браузера.

Для того щоб отримати API ключ для роботи з OpenAI, ми маємо перейти до офіційного сайту:

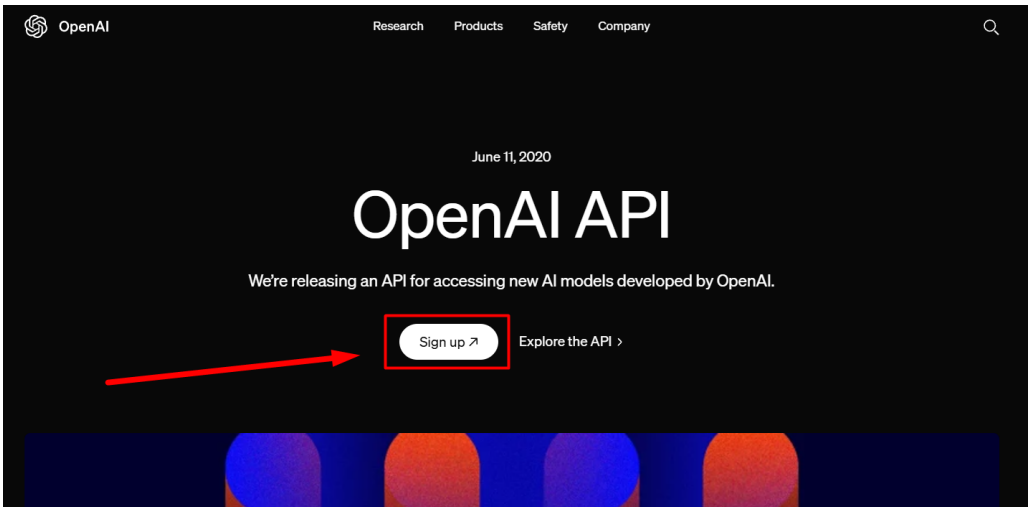


Рисунок 3.23 – Офіційний сайт OpenAI [22]

Якщо немає облікового запису, реєструємось, використовуючи email, «Google» або ж «Microsoft».

Щоб відкрити панель керування API, натискаємо на своє ім’я або аватар у правому куті сторінки та переходимо до свого профілю.

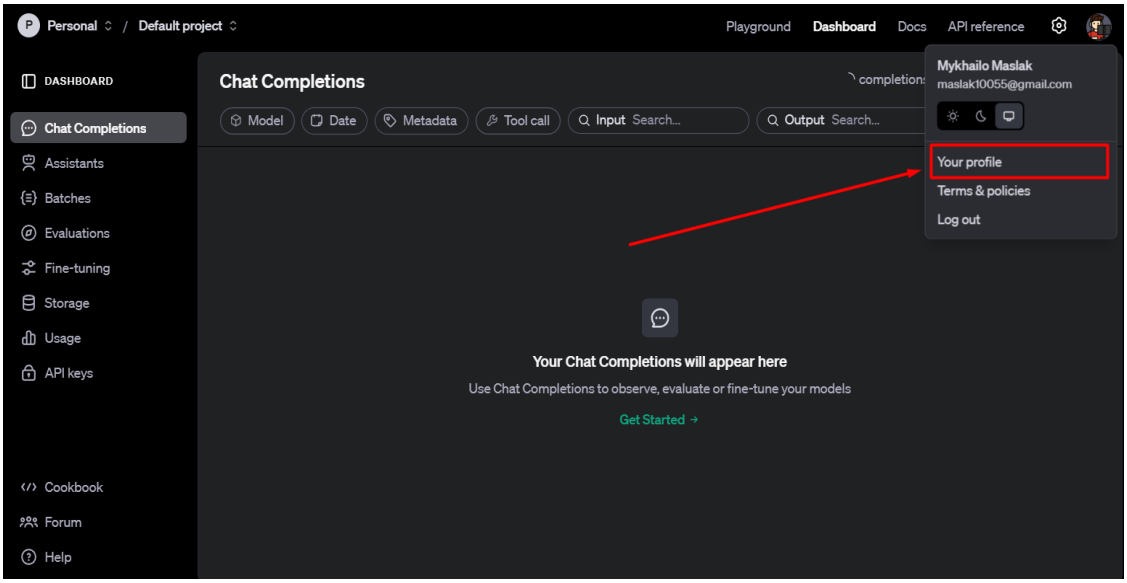


Рисунок 3.24 – Відкриття профілю

Після відкриття профілю, переходимо до пункту «API Keys» та створюємо новий секретний ключ, натиснувши кнопку «create new secret key», та заповнивши необхідні поля у діалоговому вікні, яке з’являється після натискання кнопки. Важливо не забути зберегти ключ відразу після його генерації, адже його видно лише один раз.

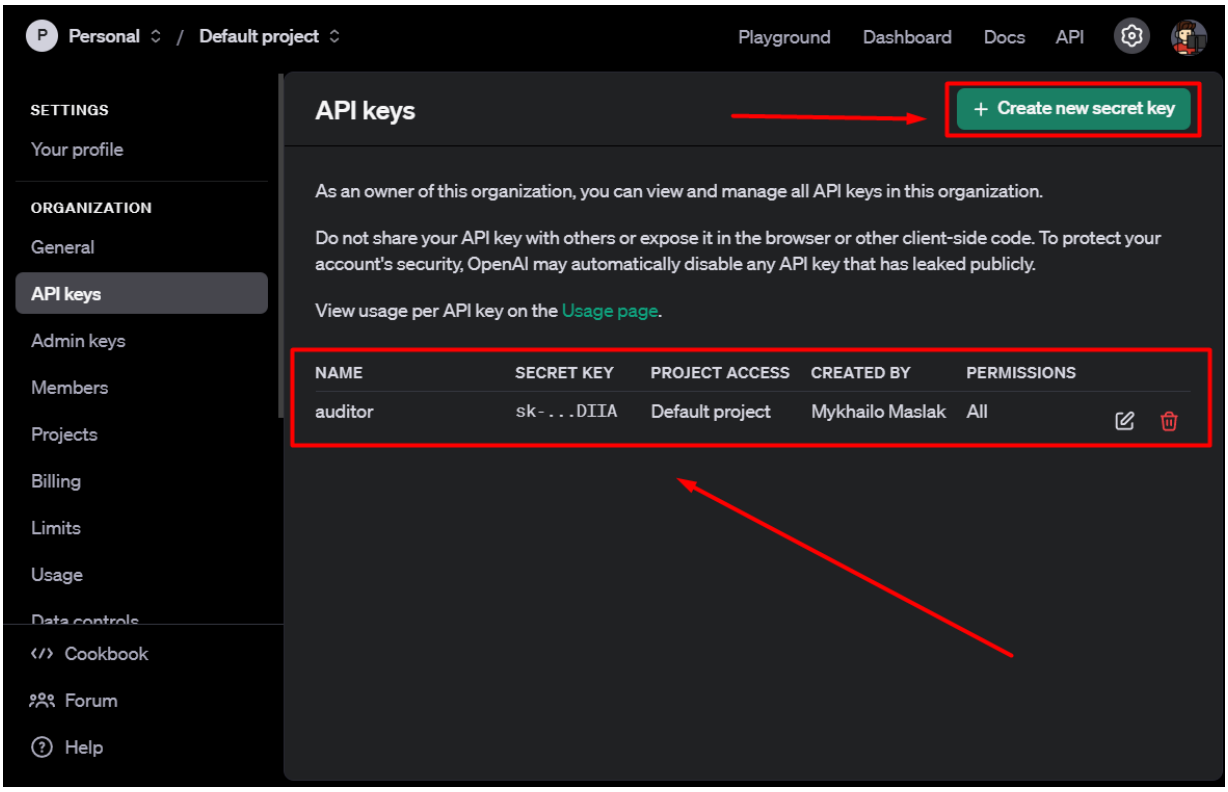


Рисунок 3.25 – Створення API ключа

Згенерований ключ копіюємо та додаємо до проєкту. У нашому випадку, в файл «.env», а саме у змінну «NEXT_PUBLIC_API_KEY».

Наступним етапом буде перевірка квоти. Якщо обліковий запис щойно створено, то OpenAI надає обмежену безкоштовну квоту на суму 18 доларів США для використання мовної моделі. Важливо зазначити, що ця квота має обмеження по часу, тому вона згорає через певний період. Квота вимірюється кількістю використаних токенів: кожен запит до моделі генерує токени, які обчислюються в залежності від довжини введеного тексту та результату. Токени складаються з частин слів, тому чим довшим є текст, тим більше токенів він використовує.

Для перевірки актуальної квоти, переходимо у розділ «Usage». На скріншоті нижче можемо побачити інформацію про квоти: термін дії, доступна сума тощо. Оскільки у безкоштовної закінчився термін дії, тому було придбано додаткову на суму 5 доларів США.

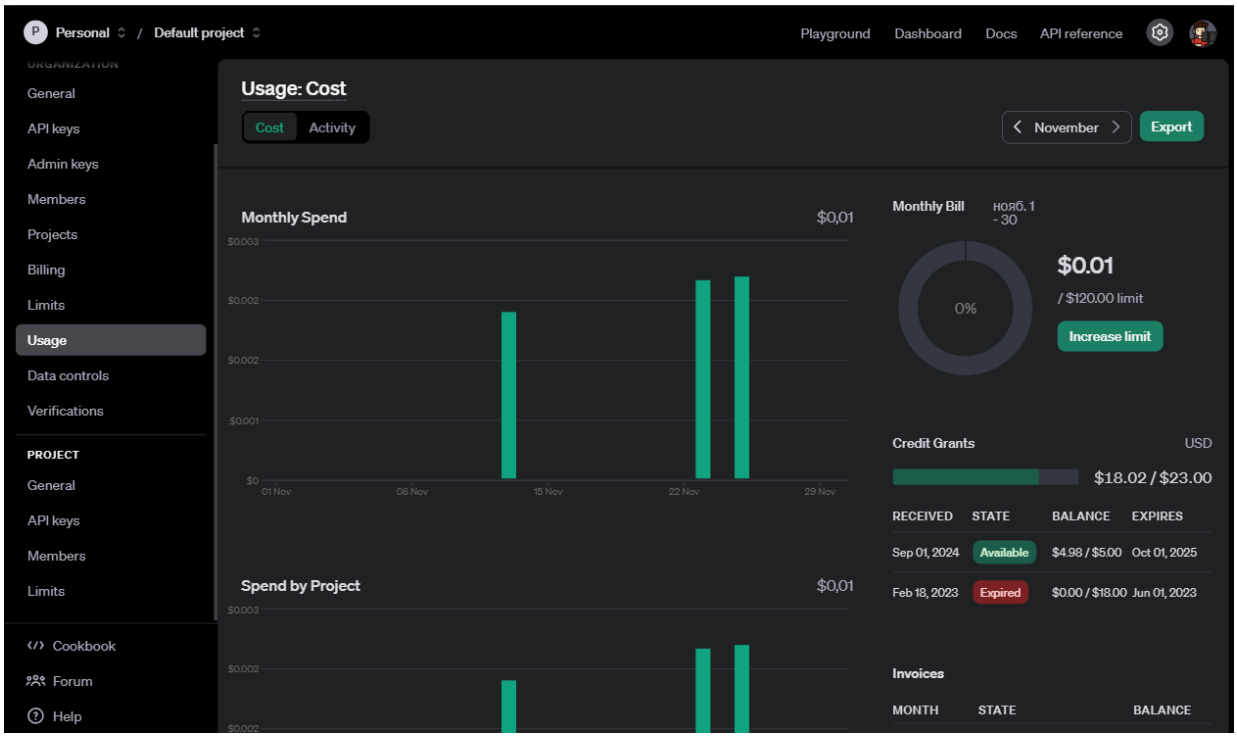
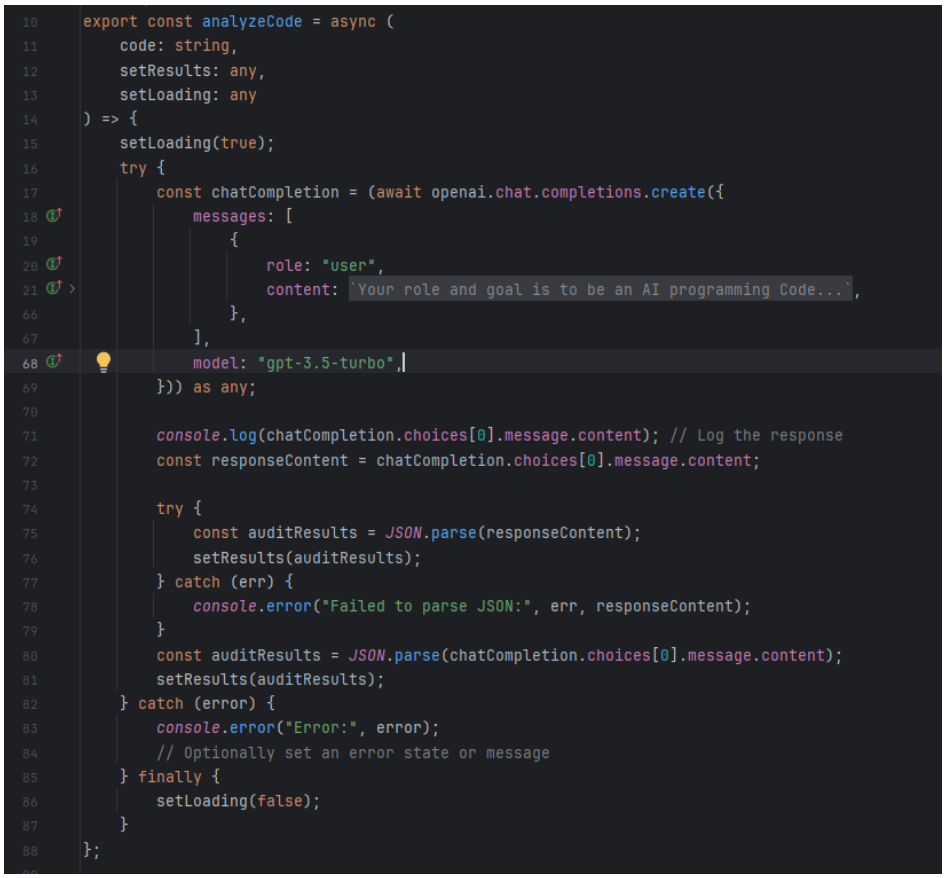


Рисунок 3.26 – Інформація про використання OpenAI API

Коли квота безкоштовного використання вичерпується, тоді потрібно оплачувати подальше використання, відповідно до тарифів OpenAI, які можуть залежати від версії мовної моделі та обсягу використаних токенів.

Отримавши працюючий ключ, приступаємо до реалізації функції «analyzeCode», яка відповідальна за процес взаємодії з моделлю та обробки результатів.

Функція analyzeCode починається з встановлення стану завантаження, що дозволяє користувачеві зрозуміти, що процес аналізу коду розпочато. Далі формується запит до API OpenAI, який містить код для аудиту. Після отримання відповіді функція намагається перетворити її на формат JSON. Якщо цей процес успішний, оновлюються результати аналізу, які відображаються в інтерфейсі. Якщо сталася помилка при обробці відповіді, вона фіксується. У будь-якому випадку стан завантаження вимикається, відновлюючи нормальний вигляд інтерфейсу. Функція має наступний вигляд:



```
export const analyzeCode = async (
    code: string,
    setResults: any,
    setLoading: any
) => {
    setLoading(true);
    try {
        const chatCompletion = (await openai.chat.completions.create({
            messages: [
                {
                    role: "user",
                    content: `Your role and goal is to be an AI programming Code...`,
                },
            ],
            model: "gpt-3.5-turbo",
        })) as any;

        console.log(chatCompletion.choices[0].message.content); // Log the response
        const responseContent = chatCompletion.choices[0].message.content;

        try {
            const auditResults = JSON.parse(responseContent);
            setResults(auditResults);
        } catch (err) {
            console.error("Failed to parse JSON:", err, responseContent);
        }
        const auditResults = JSON.parse(chatCompletion.choices[0].message.content);
        setResults(auditResults);
    } catch (error) {
        console.error("Error:", error);
        // Optionally set an error state or message
    } finally {
        setLoading(false);
    }
};
```

Рисунок 3.27 – Код функції «analyzeCode»

Останнім етапом створення додатку є реалізація компонента, який відповідає за вивід результати аудиту коду. Мова йде про створення компонента «ResultsModal».

Першим кроком було визначення структури модального вікна за допомогою компонента «Dialog» з бібліотеки «@headlessui/react», який забезпечує просте створення модальних вікон.

Для організації результатів було обрано три основні розділи: звіт аудиту, оцінки метрик і пропозиції щодо покращення. Кожен розділ має окрему можливість для розгортання або згортання, що забезпечує зручний інтерфейс користувача.

Однією з важливих частин є інтеграція індикаторів завантаження. Це реалізовано через умовний рендеринг: якщо дані ще не отримані (запит триває), відображається спіннер, який дає користувачу зрозуміти, що процес аналізу триває. Коли дані отримані, вони відображаються у відповідних секціях:

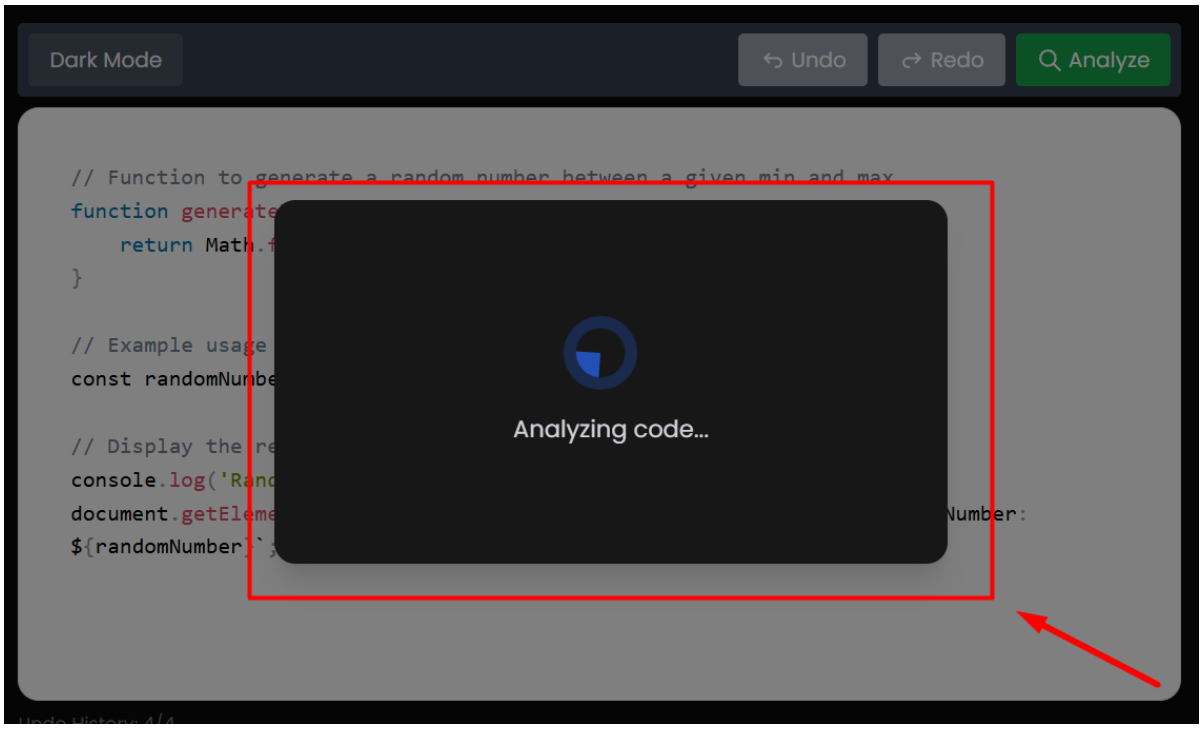


Рисунок 3.28 – Індикатор завантаження

Особливу увагу приділено візуалізації метрик. Кожна метрика відображається як круглий індикатор прогресу, використовуючи бібліотеку «react-circular-progressbar». Кольори цього індикатора змінюються залежно від значення метрики: зелений для високих оцінок, жовтий для середніх і червоний для низьких. Це допомагає користувачам швидко оцінювати результат:

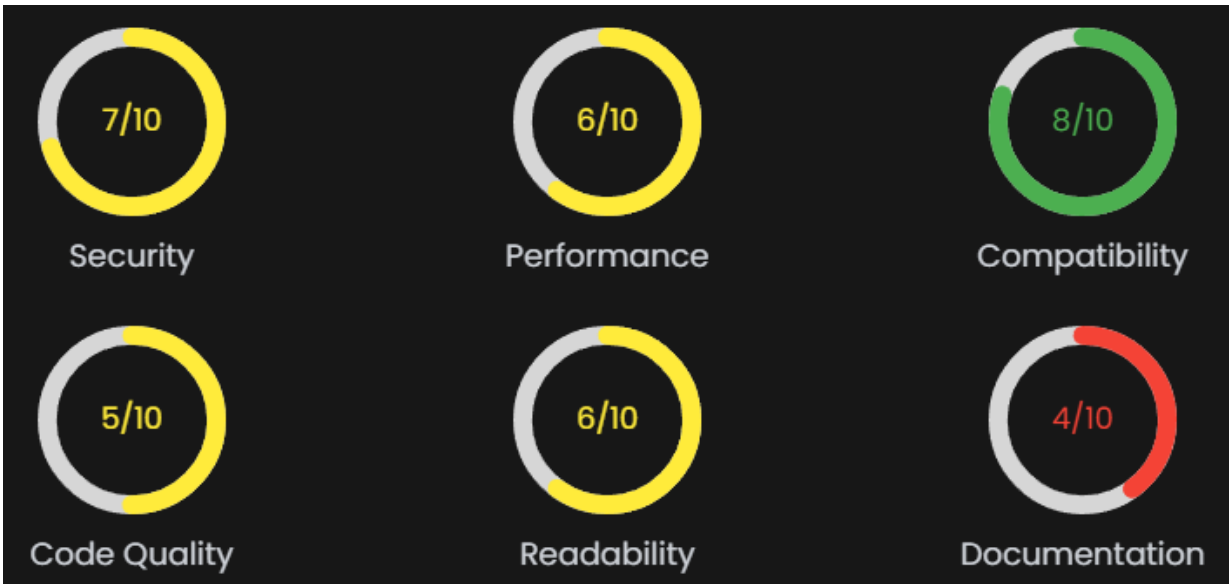
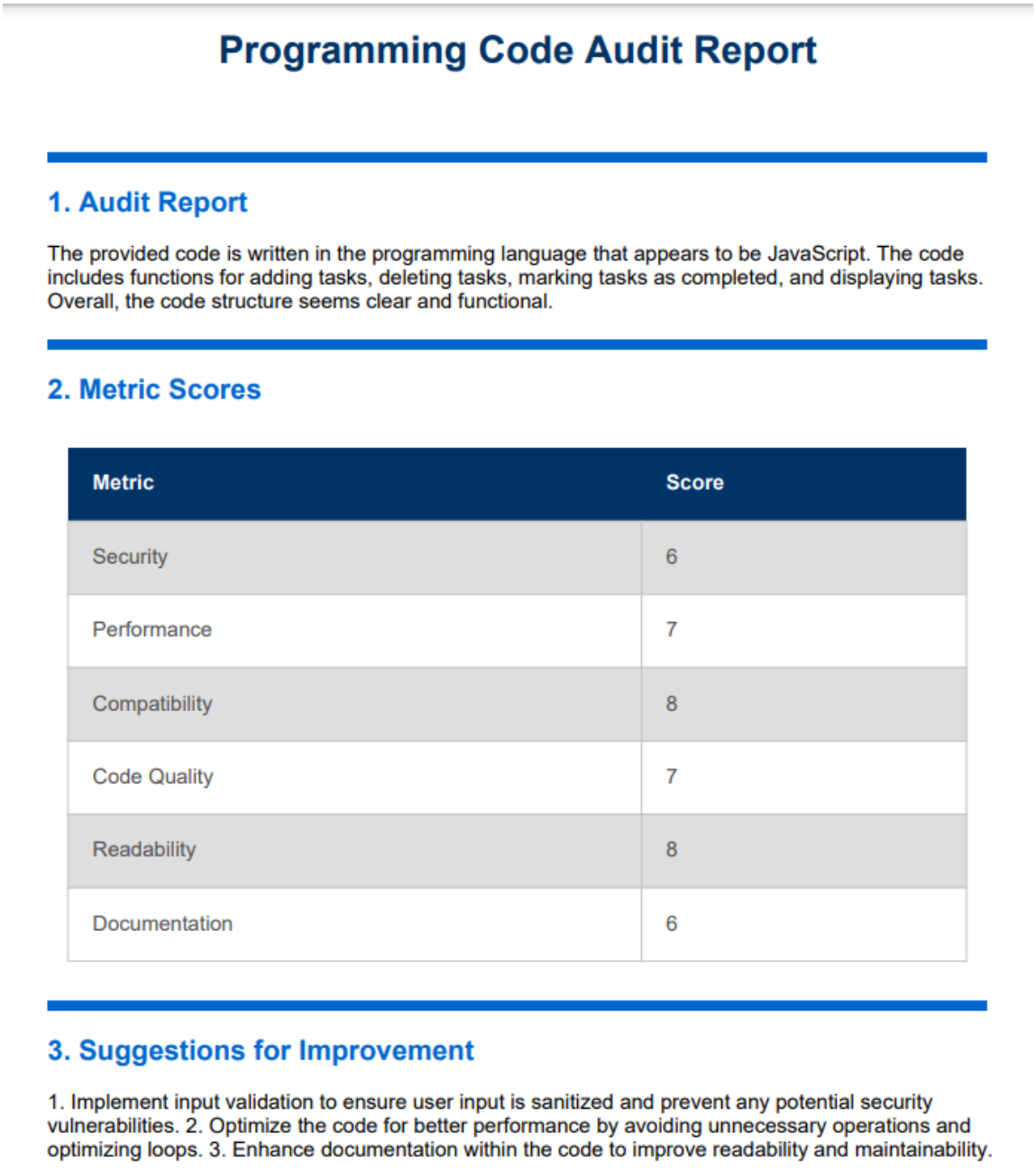


Рисунок 3.29 – Візуалізація метрик

Далі реалізовуємо функціонал для генерації PDF-звітів з використанням бібліотеки «jsPDF» та плагіну «jspdf-autotable». Перш за все встановлюємо залежності з допомогою команди «npm install jspdf jspdf-autotable».

Тепер ми створюємо функцію «generatePDFReport», яка прийматиме на вхід масив даних і генеруватиме PDF-звіт.

Згенерований звіт у разі успішно виконаного аудиту має наступний вигляд:



Programming Code Audit Report

1. Audit Report

The provided code is written in the programming language that appears to be JavaScript. The code includes functions for adding tasks, deleting tasks, marking tasks as completed, and displaying tasks. Overall, the code structure seems clear and functional.

2. Metric Scores

Metric	Score
Security	6
Performance	7
Compatibility	8
Code Quality	7
Readability	8
Documentation	6

3. Suggestions for Improvement

1. Implement input validation to ensure user input is sanitized and prevent any potential security vulnerabilities. 2. Optimize the code for better performance by avoiding unnecessary operations and optimizing loops. 3. Enhance documentation within the code to improve readability and maintainability.

Рисунок 3.30 – PDF-звіт результатів аудиту

3.3. Демонстрація роботи додатку

Оскільки додаток не розміщений на віддаленому хостингу, запускаємо додаток вже відомою нам командою «npm run dev».

На головному екрані бачимо поле для введення тексту. Вставляємо текстовий код у відповідне поле або завантажуємо файл. Код може бути простим або складним. Для демонстрації роботи було обрано код функції для множення матриць. Вона приймає на вхід дві матриці та обчислює їх добуток:

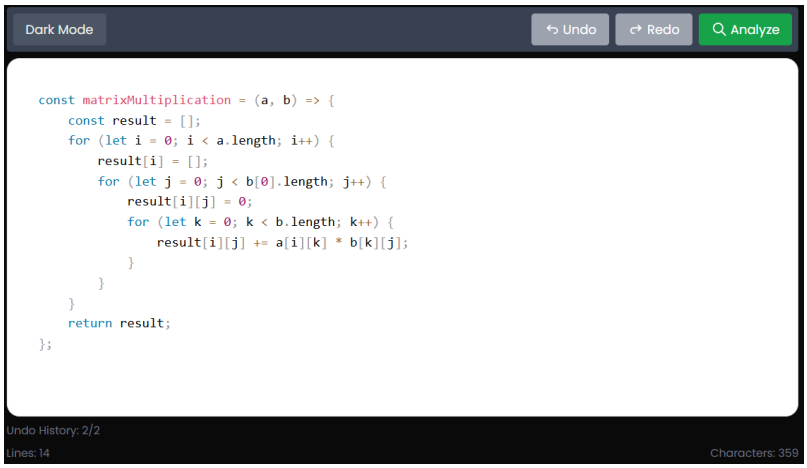


Рисунок 3.31 – Введення коду для аудиту

Також додаток має можливість зміни колірної теми. Так, наприклад, виглядає темна колірна тема:

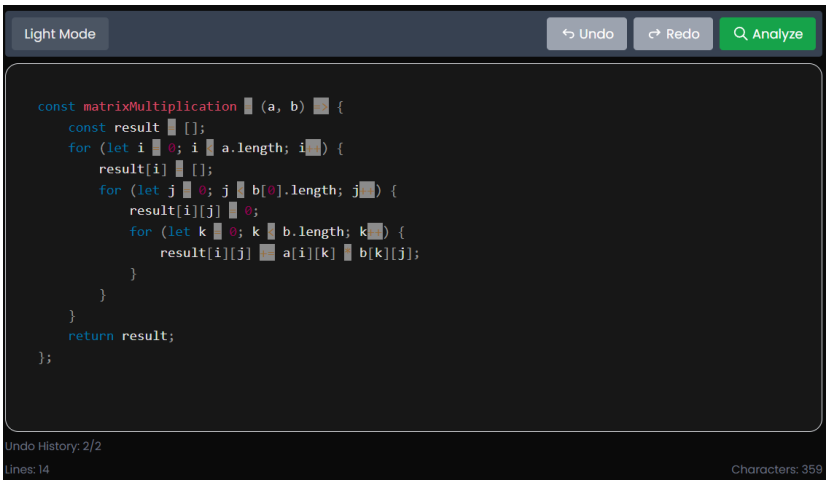


Рисунок 3.32 – Зміна колірної теми додатку

Натискаємо кнопку «Аналізувати», після чого програма передає дані до OpenAI API для обробки, після чого ми спостерігаємо за індикатором завантаження, поки додаток виконує запит. Час відповіді залежить від розміру та складності коду.

В результаті успішно виконаного запиту на екрані відображаються результати аналізу, які включають загальний звіт, метрики якості та рекомендації щодо покращень.

На рисунку 3.33 показано, що додаток зумів визначити, що введений код написаний мовою програмування Javascript і виконує матричне множення. Також вказано як саме названа функція, які параметри вона приймає і що повертає у результаті

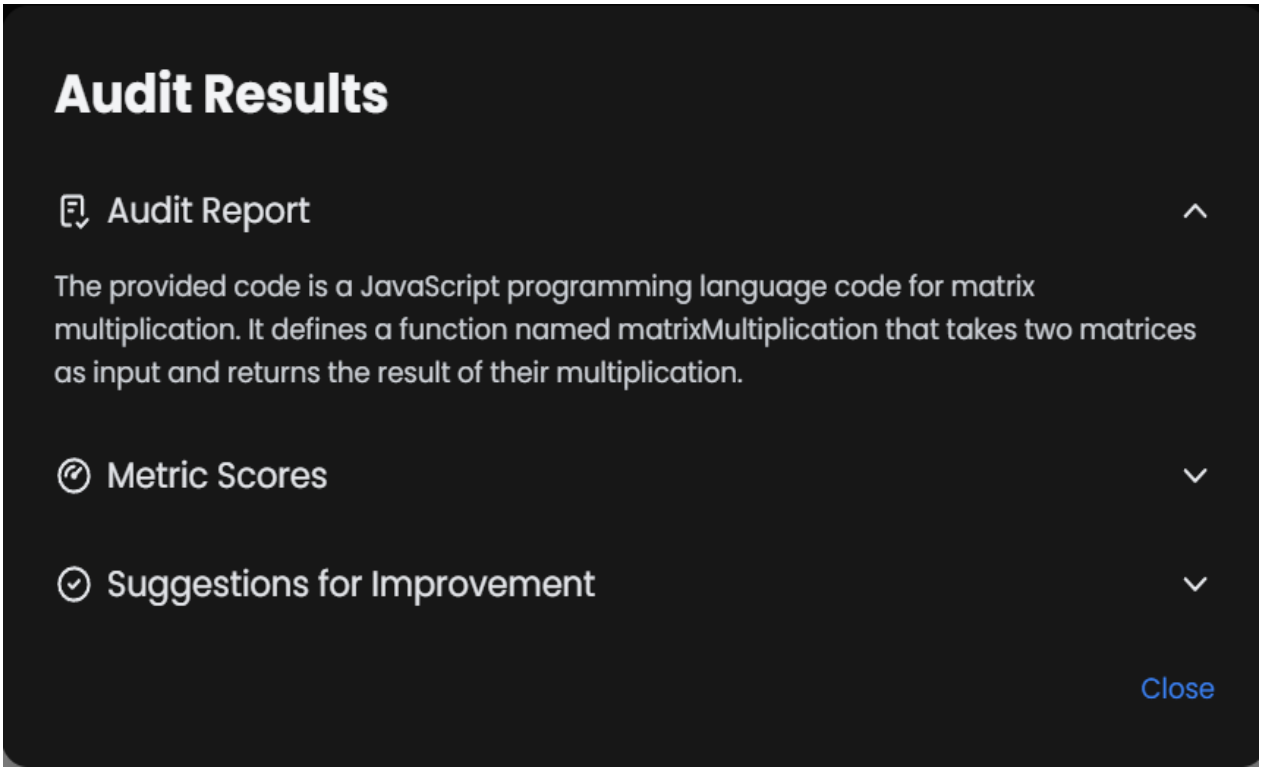


Рисунок 3.33 – Загальний звіт по аудиту

На рисунку 3.34 показано результати метричних показників. Серед них ми можемо побачити такі показники як: безпека, продуктивність, сумісність, якісь коду, читабельність, документація.

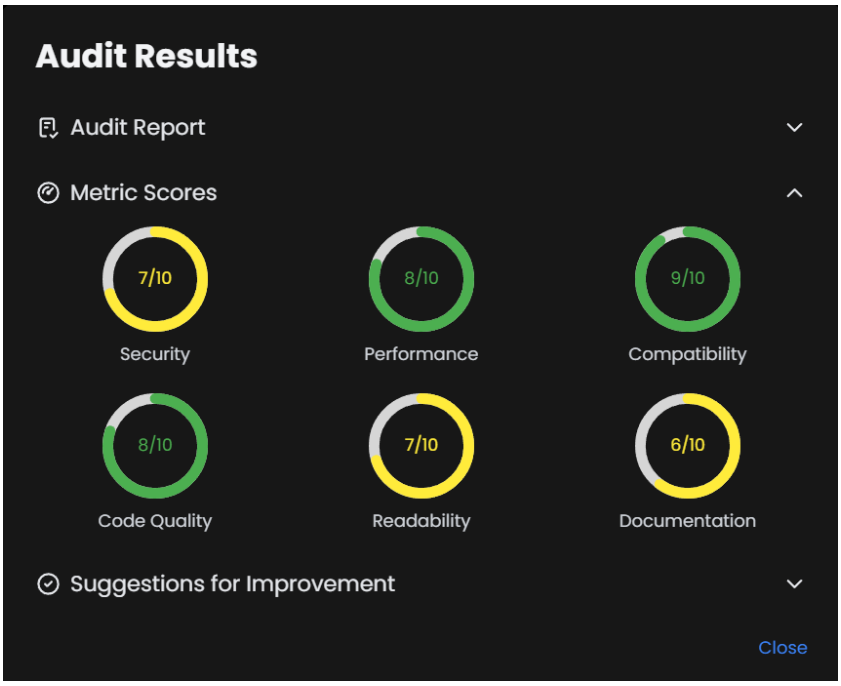


Рисунок 3.34 – Метричні показники

На рисунку 3.35 зображено звіт, який стосується пропозицій для покращень. Серед них є порада щодо підвищення заходів безпеки шляхом перевірки вхідних матриць та обробки крайових випадків. Також отримано рекомендацію оптимізувати продуктивність, враховуючи розмір матриці та ефективність алгоритм тощо

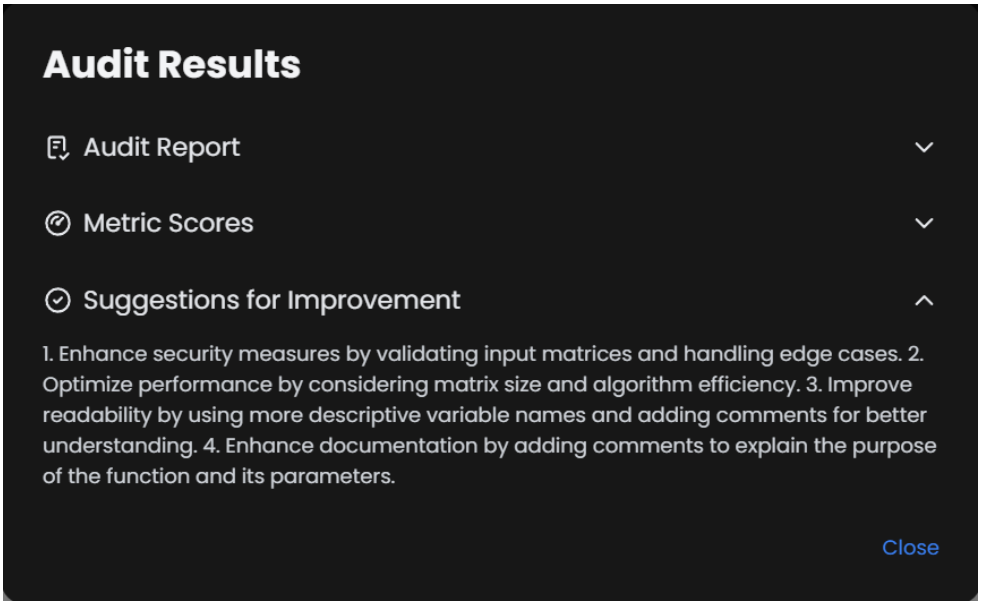


Рисунок 3.35 – Пропозиції по покращенню

РОЗДІЛ 4
ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ АУДИТУ ПРОГРАМНОГО КОДУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

4.1. Переваги тестування на основі чек-листів

Тестування на основі чек-листів – це тип тестування програмного забезпечення, що базується на заздалегідь спланованому списку завдань, який називається чек-листом. Професійні тестувальники, які мають достатній технічний досвід, зазвичай складають такі списки. Інженери QA (Quality Assurance) використовують такі контрольні списки, щоб керувати тестуванням.

Тестування на основі чек-листів широко використовується QA-експертами, які відстоюють його зручність і продуктивність. Їхні знання та досвід є основою для якісного тестування продуктивності. Основні причини використання цього типу тестів:

1. Гнучкість. Цей вид перевірки можна використовувати в усіх типах тестування.
2. Легкість у створенні. Створення, використання та підтримка чек-листів не є складним процесом.
3. Аналіз результатів. Контрольні списки легко перевіряти та аналізувати.
4. Інтеграція команди. Чек-лист може бути готовим керівництвом і допомогти новим тестувальникам інтегруватися в роботу.
5. Контроль дедлайнів. Цей тип тестування допомагає контролювати виконання тестів і не пропустити деякі критичні помилки до дедлайну [23].

4.2. Розробка та використання чек-листів

Таблиці нижче демонструють різні аспекти перевірок, зокрема функціональні, продуктивні та UX/UI-тести.

Таблиця 4.1 – Тестування на різних наборах даних та оцінка результатів

Тестовий сценарій	Очікуваний результат	Результат
Аналіз простого коду	Виявлення помилок або підтвердження правильності коду	Успішно
Аналіз коду з дрібними синтаксичними помилками	Виявлення синтаксичних помилок	Успішно
Аналіз великого обсягу коду (>1000 рядків)	Час виконання знаходиться в допустимих межах	Успішно
Реакція на порожній файл	Показ відповідного повідомлення без помилок у роботі	Успішно
Реакція на некоректний код	Показ відповідного повідомлення без краху програми	Успішно

Таблиця 4.2 – Оцінка продуктивності та якості аналізу коду

Тестовий сценарій	Очікуваний результат	Результат
Час аналізу коду для 10 рядків	Аналіз виконується менше ніж за 1 секунду	Успішно
Час аналізу коду для 100 рядків	Аналіз виконується менше ніж за 3 секунди	Успішно
Час аналізу коду для 1000 рядків	Аналіз виконується менше ніж за 10 секунд	Успішно
Якість виявлення помилок	Виявлення 90-100% відомих помилок у тестових даних	Успішно
Рекомендації щодо оптимізації коду	Генерація релевантних рекомендацій	Успішно

Таблиця 4.3 – Тестування UI/UX:

Тестування UI/UX	Опис тесту	Очікуваний результат	Результат
Перевірка адаптивності	Перевірка вигляду інтерфейсу на різних пристроях	Інтерфейс має коректно виглядати на всіх пристроях	Успішно
Колірна схема	Перевірка колірної схеми в темному та світлому режимах	Колірна схема має відповідати вибраному режиму	Успішно
Зручність навігації	Тестування інтуїтивно зрозумілої навігації	Користувач має безперешкодно навігувати по сайту	Успішно
Перевірка кнопок	Перевірка кнопок для всіх доступних дій	Кнопки мають бути доступні для натискання	Успішно

4.3. Дослідження продуктивності та затримок у роботі додатку

Потужні мовні моделі OpenAI зробили революцію в застосунках зі штучним інтелектом, але їхня ефективність залежить від одного критичного фактора: затримки.

Затримка API - це час, необхідний OpenAI API для обробки вашого запиту і повернення відповіді. Висока затримка може суттєво вплинути на взаємодію з користувачем, особливо в додатках у режимі реального часу. На час відгуку OpenAI API впливає декілька факторів:

Рисунок 4.1 – Фактори впливу на час відгуку OpenAI API

Складність моделі: коли ми говоримо про затримку, перше, що, ймовірно, спадає на думку, це швидкість виведення. Мова йде про те, як швидко мовна модель обробляє токени, і зазвичай вона вимірюється в токенах за хвилину (TPM) або токенах за секунду (TPS). Одним з найбільших факторів, що

впливають на швидкість виведення, є розмір моделі. Як правило, менші моделі працюють швидше і є більш економічно ефективними.

Довжина та складність вхідних даних можуть істотно впливати на час відповіді від OpenAI API. Тривалі запити, що містять великі обсяги тексту або складні обчислення, можуть збільшити навантаження на модель. Це призводить до більш тривалої обробки, оскільки модель повинна пройти через більший обсяг даних та виконати складнішу аналітику. Також синтаксичні помилки або складні структури можуть спричинити додаткові затримки, оскільки модель намагається зрозуміти та обробити ці помилки або обчислення, що займає більше часу.

Вихідні дані, які генерує OpenAI API, також можуть впливати на час обробки та затримки. Наприклад, якщо запит передбачає складні, багатошарові або великі результати, це може спричинити затримки в відповіді. Об’ємні результати, що містять детальні пояснення, рекомендації чи багатоступеневі розрахунки, потребують більше часу для генерації.

Навантаження на сервер: коли ми маємо справу з відповідями API, навантаження на сервер може відігравати значну роль. Високий трафік може призвести до уповільнення відповідей.

Мережеві умови: умови мережі відіграють вирішальну роль у швидкості передачі даних між додатком та API. Якщо існує сильне, стабільне з’єднання, користувач отримує швидший час відгуку і більш плавну взаємодію. З іншого боку, слабке або нестабільне з’єднання може призвести до затримок і переривань, що зробить користувацький досвід менш приємним.

Проведемо дослідження швидкості генерації та отримання відповіді від програми в залежності від розміру та складності вхідних даних.

Для цього у коді додатку було додано спеціальний код, який виводить час виконання запиту після його успішного виконання та отримання відповіді:

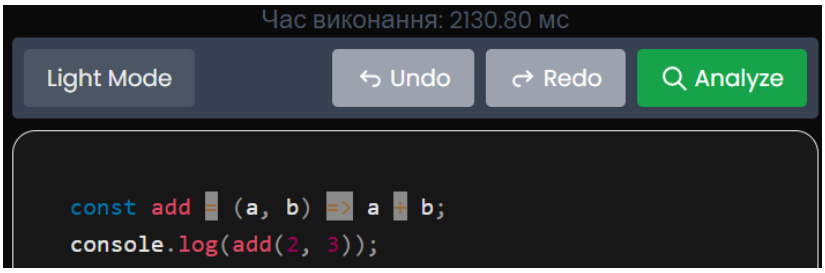


Рисунок 4.2 – Тестування часу виконання запитів

Також підготовано добірку тестових вхідних даних із різним розміром та складністю:

1. Простий код з одним завданням. Час виконання: 2286.10 мс.

```
const add = (a, b) => a + b;
console.log(add(2, 3));
```

Рисунок 4.3 – Тестові вхідні дані

2. Код з довгим коментарем. Час виконання: 2773.30 мс

```
// This is a very detailed comment explaining the
purpose of the following function,
// how it handles various edge cases, and how it
interacts with other parts of the codebase.
// It includes examples of usage and notes on
performance implications.
const sortArray = (arr) => arr.sort((a, b) => a - b);
```

Рисунок 4.4 – Тестові вхідні дані (2)

3. Складний код з кількома вкладеними циклами. Час виконання: 2957.00 мс

```
const matrixMultiplication = (a, b) => {
    const result = [];
    for (let i = 0; i < a.length; i++) {
        result[i] = [];
        for (let j = 0; j < b[0].length; j++) {
            result[i][j] = 0;
            for (let k = 0; k < b.length; k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return result;
};
```

Рисунок 4.5 – Тестові вхідні дані (3)

4. Обчислення всіх простих чисел до великого числа. Час виконання: 3694.90 мс

```
function isPrime(num) {
  for (let i = 2; i <= Math.sqrt(num); i++) {
    if (num % i === 0) return false;
  }
  return num > 1;
}

function generatePrimes(limit) {
  let primes = [];
  for (let i = 2; i < limit; i++) {
    if (isPrime(i)) primes.push(i);
  }
  return primes;
}

const largePrimes = generatePrimes(100000);
console.log(largePrimes);
```

Рисунок 4.6 – Тестові вхідні дані (4)

5. Код з глибокими рекурсивними викликами. Час виконання: 6150.50 мс

```
function generateRecursiveData(depth) {
  if (depth === 0) return "Base case";
  const data = generateRecursiveData(depth - 1);
  return `Nested data at depth ${depth}: ${data}`;
}

function createLargeDataset() {
  let result = '';
  for (let i = 0; i < 1000; i++) {
    result += generateRecursiveData(100) + '\n';  //
Глибока рекурсія
  }
  return result;
}

const largeDataset = createLargeDataset();
console.log(largeDataset);
```

Рисунок 4.7 – Тестові вхідні дані (5)

Можемо бачити наступну залежність часу виконання від складності та довжини вхідних даних:

Рисунок 4.8 – Гістограма часу виконання запитів додатком

ВИСНОВКИ

У процесі виконання дипломного проєкту було розроблено веб-додаток для автоматизованого аудиту коду із застосуванням сучасних технологій та методів штучного інтелекту. Основною метою було створення інструменту, який дозволяє аналізувати код з точки зору його якості, безпеки та продуктивності, що є актуальним завданням у сучасній розробці програмного забезпечення. Для реалізації цієї мети використовувався такий технологічний стек: React.js, Next.js, Tailwind CSS, TypeScript та API OpenAI. Вибір цих інструментів був обумовлений їх потужністю, масштабованістю та сумісністю, що дозволило створити гнучкий і продуктивний додаток.

На першому етапі роботи проведено аналіз існуючих технологій автоматизованого аудиту, включно з інструментами для аналізу коду та їх обмеженнями. Це дозволило визначити необхідний функціонал, включаючи перевірку структурного та синтаксичного складу коду, виявлення потенційних вразливостей і оцінку ефективності алгоритмів. Було розроблено архітектуру додатку, яка базується на концепції серверно-клієнтної взаємодії через Next.js, що дозволяє обробляти дані на серверній стороні та швидко відображати результати на клієнтському інтерфейсі.

Особлива увага приділялася інтеграції OpenAI API, що надало можливість застосування штучного інтелекту для глибокого аналізу коду. Це дозволило реалізувати унікальну функцію — генерацію рекомендацій щодо покращення коду, які є зрозумілими навіть для розробників-початківців. Для забезпечення сучасного та зручного дизайну у проєкті застосовано Tailwind CSS, який дозволив створити адаптивний інтерфейс. Завдяки його утилітарному підходу до стилізації, вдалося досягти балансу між ефективністю розробки та естетичною привабливістю. Такий підхід забезпечив інтуїтивність навігації й позитивний досвід для користувачів незалежно від типу пристрою, на якому відкривається додаток.

На етапі тестування додатка було перевірено його стабільність, точність аналізу та відповідність функціоналу заявленим вимогам. Проведені випробування дозволили виявити й усунути низку технічних недоліків, забезпечуючи стабільну роботу додатку навіть за умов інтенсивного використання. Зокрема, тестування охоплювало як базові перевірки коректності коду, так і складні сценарії аналізу великих проектів.

Практична цінність виконаної роботи полягає у створенні інструменту, що дозволяє автоматизувати рутинні завдання розробників і забезпечувати високий рівень контролю якості коду. Додаток можна легко інтегрувати у робочі процеси як незалежний сервіс чи як частину більших платформ, таких як GitHub. Завдяки використанню TypeScript знижено ризики появи помилок у самому коді додатка, а використання React.js забезпечило зручну компонентну структуру для його підтримки та розвитку.

Результати дипломного проєкту свідчать про доцільність використання штучного інтелекту у задачах аналізу та оптимізації коду. Додаток має потенціал для подальшого вдосконалення: планується розширення підтримуваних мов програмування, інтеграція з популярними системами управління проєктами та впровадження більш глибоких алгоритмів аналізу. Ці вдосконалення дозволять значно підвищити ефективність розробки програмного забезпечення та зробити додаток ще більш корисним для професійної спільноти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OneSevenTech Blog: Code Audit. OneSevenTech. URL: https://www.oneseventech.com/blog/code-audit (дата звернення: 14.09.2024).

2. Automated Code Reviews: Part of Workflow. Codacy Blog. URL: https://blog.codacy.com/automated-code-reviews-part-of-workflow (дата звернення: 16.09.2024).

3. Product Downloadables. TrustRadius. URL: https://media.trustradius.com/product-downloadables/DD/D7/XID8MVZTH0JF.pdf (дата звернення: 18.09.2024).

4. AI in the Code Review Process. Typo Blog. URL: https://typoapp.io/blog/ai-in-the-code-review-process (дата звернення: 20.09.2024).

5. AI Code Review: A Comprehensive Guide. Dev3lop. URL: https://dev3lop.com/ai-code-review-a-comprehensive-guide/ (дата звернення: 22.09.2024).

6. What is Transformers in Artificial Intelligence? AWS. URL: https://aws.amazon.com/what-is/transformers-in-artificial-intelligence/ (дата звернення: 24.09.2024).

7. AI Tools for Reviewing Code. GeeksforGeeks. URL: https://www.geeksforgeeks.org/ai-tools-for-reviewing-code/ (дата звернення: 26.09.2024).

8. Difference Between TypeScript and JavaScript. GeeksforGeeks. URL: https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/ (дата звернення: 28.09.2024).

9. Most Popular Webframe Technologies. Stack Overflow Survey 2023. URL: https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe-prof (дата звернення: 30.09.2024).

10. React.js vs Vue.js. MindInventory. URL: https://www.mindinventory.com/blog/reactjs-vs-vuejs/ (дата звернення: 02.10.2024).

11. What is Next.js and its Benefits? Medium. URL: https://medium.com/devsphere/what-is-next-js-and-its-benefits-8b13aab56bfd (дата звернення: 04.10.2024).

12. A Comprehensive Introduction to Tailwind CSS. Medium. URL: https://medium.com/@alifm2101/a-comprehensive-introduction-to-tailwind-css-36bc9cb81a1c (дата звернення: 06.10.2024).

13. Demystifying MVC Architecture. Medium. URL: https://medium.com/@finzyphinzy/demystifying-mvc-understanding-the-model-view-controller-architecture-85c88a558951 (дата звернення: 08.10.2024).

14. Getting Started with Next.js. Next.js Documentation. URL: https://nextjs.org/docs/app/getting-started/project-structure (дата звернення: 10.10.2024).

15. What is Figma?. Noble Desktop. URL: https://www.nobledesktop.com/learn/figma/what-is-figma (дата звернення: 12.10.2024).

16. Node.js Official Website. URL: https://nodejs.org/uk (дата звернення: 14.10.2024).

17. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 16.10.2024).

18. Wavy Background Components. UI Aceternity. URL: https://ui.aceternity.com/components/wavy-background (дата звернення: 18.10.2024).

19. React Simple Code Editor. npm. URL: https://www.npmjs.com/package/react-simple-code-editor (дата звернення: 20.10.2024).

20. PrismJS. npm. URL: https://www.npmjs.com/package/prismjs (дата звернення: 22.10.2024).

21. Tabler Icons React. Tabler. URL: https://tabler.io/docs/icons/react (дата звернення: 24.10.2024).

22. OpenAI Official Website. URL: https://openai.com/ (дата звернення: 26.10.2024).

23. QA Checklist. MuukTest Blog. URL: https://muuktest.com/blog/qa-checklist (дата звернення: 28.10.2024).

ДОДАТОК А

Код файлу «layout.tsx»:

```
import type {Metadata} from "next";
import {Poppins} from "next/font/google";
import "./globals.css";

const poppins = Poppins({weight: ["400", "700"], subsets: ["latin"]});

export const metadata: Metadata = {
    title: "Code Auditor",
    description: "AI Code Auditor",
};

export default function RootLayout({
                                       children,
                                   }: Readonly<{
    children: React.ReactNode;
}>) {
    return (
        <html lang="en">
        <body className={poppins.className}>{children}</body>
        </html>
    );
}
```

Код файлу «page.tsx»:

```
"use client";
import { useState } from "react";
import Header from "@/components/header";
import CustomCodeEditor from "@/components/code-input";
import ResultsModal from "@/components/result-modal";
import { analyzeCode } from "../../utils/ai-prompt";

export default function Home() {
  const [loading, setLoading] = useState(false);
  const [code, setCode] = useState("");
  const [results, setResults] = useState(null);
  const [isModalOpen, setIsModalOpen] = useState(false);
    const [executionTime, setExecutionTime] = useState<number | null>(null);
    const analyze = async () => {
        setIsModalOpen(true);
        await analyzeCode(code, setResults, setLoading, setExecutionTime);
    };

  return (
      <main className="flex min-h-screen w-full flex-col items-center justify-
between p-24">
        <Header />
          {executionTime && (
              <p className="text-center text-gray-600">
                  Час виконання: {executionTime.toFixed(2)} мс
              </p>
          )}

          <CustomCodeEditor
```

```
                code={code}
                setCode={setCode}
                analyze={analyze}
            />
            <ResultsModal
                isOpen={isModalOpen}
                closeModal={() => setIsModalOpen(false)}
                loading={loading}
                results={results}
            />
          </main>
    );
  }
```

Код файлу «globals.css»:

```
@tailwind base;
@tailwind components;
@tailwind utilities;

:root {
  --background: #ffffff;
  --foreground: #171717;
}

@media (prefers-color-scheme: dark) {
  :root {
    --background: #0a0a0a;
    --foreground: #ededed;
  }
}

body {
  color: var(--foreground);
  background: var(--background);
  font-family: Arial, Helvetica, sans-serif;
}

@layer utilities {
  .text-balance {
    text-wrap: balance;
  }
}
```

Код файлу «code-input.tsx»:

```
import React, { useState } from "react";
import Editor from "react-simple-code-editor";
import Prism from "prismjs";
import "prismjs/components/prism-javascript";
import "prismjs/themes/prism-tomorrow.css"; // Темна тема
import "prismjs/themes/prism.css"; // Світла тема
import { IconArrowBackUp, IconArrowForwardUp, IconSearch } from "@tabler/icons-
react";

interface CustomCodeEditorProps {
    code: string;
    setCode: React.Dispatch<React.SetStateAction<string>>;
    analyze: () => Promise<void>;
}
```

```
const highlightWithPrism = (code: string) => {
    return Prism.highlight(code, Prism.languages.javascript, "javascript");
};

const CustomCodeEditor: React.FC<CustomCodeEditorProps> = ({
                                                               code,
                                                               setCode,
                                                               analyze,
                                                           }) => {
    const [theme, setTheme] = useState("dark");
    const [history, setHistory] = useState<string[]>([code]);
    const [historyIndex, setHistoryIndex] = useState(0);

    const handleUndo = () => {
        if (historyIndex > 0) {
            setHistoryIndex(historyIndex - 1);
            setCode(history[historyIndex - 1]);
        }
    };

    const handleRedo = () => {
        if (historyIndex < history.length - 1) {
            setHistoryIndex(historyIndex + 1);
            setCode(history[historyIndex + 1]);
        }
    };

    const saveToHistory = (newCode: string) => {
        const updatedHistory = [...history.slice(0, historyIndex + 1), newCode];
        setHistory(updatedHistory);
        setHistoryIndex(updatedHistory.length - 1);
    };

    const handleCodeChange = (newCode: string) => {
        setCode(newCode);
        saveToHistory(newCode);
    };

    const handleFileUpload = (event: React.ChangeEvent<HTMLInputElement>) => {
        const file = event.target.files?.[0];
        if (file) {
            const reader = new FileReader();
            reader.onload = () => {
                const fileContent = reader.result as string;
                setCode(fileContent);
                saveToHistory(fileContent);
            };
            reader.readAsText(file);
        }
    };

    const toggleTheme = () => {
        setTheme((prevTheme) => (prevTheme === "dark" ? "light" : "dark"));
    };

    const lineCount = code.split("\n").length;
    const charCount = code.length;

    const lineNumbers = Array.from({ length: lineCount }, (_, i) => i + 1);

    return (
        <div className={`relative lg:w-4/6 w-full mx-auto ${theme}`}>
```

```
                <div  className="flex  justify-between  mb-2  p-2  bg-gray-200  dark:bg-
gray-700 rounded">
                    <button
                        onClick={toggleTheme}
                        className="px-4 py-2 rounded bg-gray-300 dark:bg-gray-600"
                    >
                        {theme === "dark" ? "Light Mode" : "Dark Mode"}
                    </button>
                    <div className="flex gap-x-2">
                        <div className="flex justify-between items-center">
                            <input
                                type="file"
                                accept=".txt,.js,.ts,.php"
                                onChange={handleFileUpload}
                                className="px-4 py-2 rounded bg-gray-300 dark:bg-gray-
600 cursor-pointer"
                            />
                        </div>
                        <button
                            onClick={handleUndo}
                            className="flex items-center gap-x-1 px-4 py-2 rounded bg-
gray-400 text-white"
                        >
                            <IconArrowBackUp size={20} /> Undo
                        </button>
                        <button
                            onClick={handleRedo}
                            className="flex items-center gap-x-1 px-4 py-2 rounded bg-
gray-400 text-white"
                        >
                            <IconArrowForwardUp size={20} /> Redo
                        </button>
                        <button
                            onClick={analyze}
                            className="flex items-center gap-x-1 px-4 py-2 rounded bg-
green-600 text-white"
                        >
                            <IconSearch size={20} /> Analyze
                        </button>
                    </div>
                </div>

                <div
                    className={`border rounded-2xl p-6 ${
                        theme === "dark"
                            ? "dark:bg-neutral-900 dark:text-neutral-200"
                            : "bg-white text-black"
                    }`}
                    style={{ height: "450px", overflowY: "auto", display: "flex" }}
                >
                    <div  className="line-numbers"  style={{  paddingRight:  "10px",
userSelect: "none",paddingTop:'15px'}}>
                        {lineNumbers.map((line) => (
                            <div    key={line}    className="line-number"    style={{
paddingLeft: "10px",lineHeight:'1.5856'}}>
```

Код файлу «ai-prompt.ts»:

```
import OpenAI from "openai";
import { generatePDFReport } from "./exportToPDF";

const apiKey = process.env.NEXT_PUBLIC_API_KEY;

const openai = new OpenAI({
    apiKey: apiKey,
    dangerouslyAllowBrowser: true,
});

export const analyzeCode = async (
    code: string,
    setResults: any,
    setLoading: any,
    setExecutionTime: (time: number) => void // Новий параметр для збереження часу
виконання
) => {
    setLoading(true);
    const startTime = performance.now(); // Початок заміру часу

    try {
        const chatCompletion = (await openai.chat.completions.create({
            messages: [
                {
                    role: "user",
                    content: `Your role and goal is to be an AI programming Code
Auditor. Your job is to perform an audit on the given programming code. Here is the code
${code}.

                    Provide detailed and comprehensive answers
                    Please STRICTLY provide the results in only the following JSON
format for easy front-end display:
                    Please provide more detailed answers
                    [
                      {
                        "section": "Audit Report",
                        "details": "A detailed audit report of the programming
code, covering that it is a programming language, security, performance, and any other
relevant aspects."
                      },
                      {
                        "section": "Metric Scores",
                        "details": [
                          {
                            "metric": "Security",
                            "score": "0-10"
                          },
                          {
                            "metric": "Performance",
                            "score": "0-10"
                          },
                          {
                            "metric": "Compatibility",
                            "score": "0-10"
                          },
                          {
                            "metric": "Code Quality",
                            "score": "0-10"
                          },
                           {
```

```
                                "metric": "Readability",
                                "score": "0-10"
                            },
                            {
                                "metric": "Documentation",
                                "score": "0-10"
                            }
                        ]
                    },
                    {
                        "section": "Suggestions for Improvement",
                        "details": "Suggestions for improving the programming code
in terms of security, performance, and any other identified weaknesses."
                    }
                ]

                Thank you.`,
            },
        ],
        model: "gpt-3.5-turbo",
    })) as any;

    console.log(chatCompletion.choices[0].message.content);    //    Log    the
response
    const responseContent = chatCompletion.choices[0].message.content;
    try {
        const auditResults = JSON.parse(responseContent);
        setResults(auditResults);
        generatePDFReport(JSON.parse(responseContent));
    } catch (err) {
        console.error("Failed to parse JSON:", err, responseContent);
    }
} catch (error) {
    console.error("Error:", error);
} finally {
    const endTime = performance.now(); // Кінець заміру часу
    setExecutionTime(endTime - startTime); // Обчислення часу виконання
    setLoading(false);
}
};
```

Код файлу «exportToPDF.ts»:

```
import jsPDF from "jspdf";
import autoTable from "jspdf-autotable";

type ReportData = {
    section: string;
    details: string | { metric: string; score: string }[];
};

export const generatePDFReport = (data: ReportData[]): void => {
    const pdf = new jsPDF();
    const pageWidth = pdf.internal.pageSize.getWidth();
    const margin = 10;
    let cursorY = 20;

    cursorY += 30;

    // Заголовок звіту
    pdf.setFont("helvetica", "bold");
```

```
            pdf.setFontSize(22);
            pdf.setTextColor(0, 51, 102); // Темно-синій колір
            pdf.text("Programming Code Audit Report", pageWidth / 2, cursorY, { align:
"center" });

            cursorY += 20;

            // Перебір JSON-даних
            data.forEach((item, index) => {
                // Додаємо кольорові розділювачі
                pdf.setFillColor(0, 102, 204); // Синій колір
                pdf.rect(margin, cursorY - 2, pageWidth - 2 * margin, 2, "F");

                cursorY += 10;

                pdf.setFont("helvetica", "bold");
                pdf.setFontSize(16);
                pdf.setTextColor(0, 102, 204); // Синій колір
                pdf.text(`${index + 1}. ${item.section}`, margin, cursorY);

                cursorY += 10;

                if (typeof item.details === "string") {
                    // Якщо `details` — це текст
                    pdf.setFont("helvetica", "normal");
                    pdf.setFontSize(12);
                    pdf.setTextColor(0, 0, 0); // Чорний колір
                    const textLines = pdf.splitTextToSize(item.details, pageWidth - 2 *
margin);
                    pdf.text(textLines, margin, cursorY);
                    cursorY += textLines.length * 6;
                } else if (Array.isArray(item.details)) {
                    // Якщо `details` — це список метрик
                    autoTable(pdf, {
                        startY: cursorY,
                        head: [['Metric', 'Score']],
                        body: item.details.map(detail => [detail.metric, detail.score]),
                        theme: 'grid',
                        headStyles: { fillColor: [0, 51, 102], textColor: 255 },
                        alternateRowStyles: { fillColor: [221, 221, 221] },
                        styles: { cellPadding: 5, fontSize: 12 },
                    });

                    cursorY = (pdf as any).previousAutoTable.finalY + 10;
                }

                // Додаємо нову сторінку, якщо місця на поточній сторінці бракує
                if (cursorY > pdf.internal.pageSize.getHeight() - 20) {
                    pdf.addPage();
                    cursorY = 20;
                }
            });

            // Збереження PDF
            pdf.save("audit_report.pdf");
        };
```

Код файлу «result-modal.tsx»:

```
import React, {
  useState
}
from "react";
import {
  Dialog
}
from "@headlessui/react";
import {
  IconChecklist,
  IconCircleCheck,
  IconGauge,
  IconChevronDown,
  IconChevronUp,
  IconTool,
}
from "@tabler/icons-react";
import {
  CircularProgressbar,
  buildStyles
}
from "react-circular-progressbar";
import "react-circular-progressbar/dist/styles.css";
interface ResultsModalProps {
  isOpen: boolean;
  closeModal: () => void;
  loading: boolean;
  results: any;
}
const ResultsModal: React.FC < ResultsModalProps > = ({
  isOpen,
  closeModal,
  loading,
  results,
}) => {
  const [expandedSection, setExpandedSection] = useState < string | null > (null);
  const toggleSection = (section: string) => {
    setExpandedSection((prevSection) =>
      prevSection === section ? null : section
    );
  };
  return ( < Dialog open = {
      isOpen
    }
    onClose = {
      closeModal
    }
    className = "fixed z-10 inset-0 overflow-y-auto" >
    < div className = "flex items-center justify-center min-h-screen px-4 text-
center" >
    < div className = "fixed inset-0 bg-black opacity-50"
    aria - hidden = "true" > < /div> {
      loading ? ( < div className = "bg-white dark:bg-neutral-900 rounded-2xl
overflow-hidden shadow-xl transform transition-all max-w-lg w-full p-8 space-y-8" >
        < div className = "flex py-14 flex-col items-center" >
        < svg className = "animate-spin -ml-1 mr-3 h-14 w-14 text-blue-600"
        xmlns = "http://www.w3.org/2000/svg"
        fill = "none"
        viewBox = "0 0 24 24" >
        < circle className = "opacity-25"
```

```
              cx = "12"
              cy = "12"
              r = "10"
              stroke = "currentColor"
              strokeWidth = "4" > < /circle> < path className = "opacity-75"
              fill = "currentColor"
              d = "M4 12a8 8 0 018-8v8h8a8 8 0 01-8 8V4a8 8 0 00-8 8z" > < /path> < /svg>
< p className = "mt-4 text-lg text-gray-700 dark:text-gray-300" >
              Analyzing code... < /p> < /div> < /div>
            ) : (
              results && ( < div className = "bg-white dark:bg-neutral-900 rounded-2xl
overflow-hidden shadow-xl transform transition-all max-w-3xl w-full p-8 space-y-8" >
                < div className = "space-y-8" >
                < div className = "flex justify-between items-center" >
                < h2 className = "text-3xl font-bold text-gray-900 dark:text-gray-100"
>
                Audit Results < /h2> < /div> < div className = "text-left" >
                < h3 className = "text-xl space-x-2 cursor-pointer flex items-center
justify-between dark:text-gray-200 mb-4"
                onClick = {
                  () => toggleSection("auditReport")
                } >
                < div className = "flex items-center space-x-2" >
                < IconChecklist size = {
                  24
                }
                /> < span > Audit Report < /span> < /div> {
                  expandedSection === "auditReport" ? ( < IconChevronUp size = {
                      24
                    }
                    />
                  ) : ( < IconChevronDown size = {
                      24
                    }
                    />
                  )
                } < /h3> {
                  expandedSection === "auditReport" && ( < p className = "text-base text-
gray-700 dark:text-gray-300" > {
                    results.find((r: any) => r.section === "Audit Report")
                    .details
                  } < /p>)
                } < /div> < div className = "text-left" >
                < h3 className = "text-xl space-x-2 cursor-pointer flex items-center
justify-between dark:text-gray-200 mb-4"
                onClick = {
                  () => toggleSection("metricScores")
                } >
                < div className = "flex items-center space-x-2" >
                < IconGauge size = {
                  24
                }
                /> < span > Metric Scores < /span> < /div> {
                  expandedSection === "metricScores" ? ( < IconChevronUp size = {
                      24
                    }
                    />
                  ) : ( < IconChevronDown size = {
                      24
                    }
                    />
                  )
```

```
                } < /h3> {
                  expandedSection === "metricScores" && ( < div className = "grid grid-
cols-1 md:grid-cols-3 gap-6" > {
                    results
                    .find((r: any) => r.section === "Metric Scores")
                    .details.map((metric: any, metricIndex: number) => {
                      let color;
                      if (metric.score >= 8) color = "#4caf50"; // green
                      else if (metric.score < 5) color = "#f44336"; // red
                      else color = "#ffeb3b"; // yellow
                      return ( < div key = {
                          metricIndex
                        }
                        className = "flex flex-col items-center" >
                        < div className = "w-24 h-24" >
                        < CircularProgressbar value = {
                          metric.score * 10
                        }
                        text = {
                          `${metric.score}/10`
                        }
                        strokeWidth = {
                          10
                        }
                        styles = {
                          buildStyles({
                            textSize: "16px",
                            pathColor: color,
                            textColor: color,
                            trailColor: "#d6d6d6",
                          })
                        }
                        /> < /div> < p className = "text-center mt-2 text-gray-700
dark:text-gray-300" > {
                          metric.metric
                        } < /p> < /div>
                      );
                    })
                  } < /div>)
                } < /div> < div className = "text-left" >
                < h3 className = "text-xl space-x-2 cursor-pointer flex items-center
justify-between dark:text-gray-200 mb-4"
                onClick = {
                  () => toggleSection("suggestions")
                } >
                < div className = "flex items-center space-x-2" >
                < IconCircleCheck size = {
                  24
                }
                /> < span > Suggestions
                for Improvement < /span> < /div> {
                  expandedSection === "suggestions" ? ( < IconChevronUp size = {
                      24
                    }
                    />
                  ) : ( < IconChevronDown size = {
                      24
                    }
                    />
                  )
                } < /h3> {
                  expandedSection === "suggestions" && ( < >
```

```
              < p className = "text-base text-gray-700 dark:text-gray-300" > {
                results.find(
                  (r: any) =>
                  r.section === "Suggestions for Improvement"
                ).details
              } < /p> < />
            )
          } < /div> < div className = "flex justify-end" >
          < button onClick = {
            closeModal
          }
          className = "text-blue-500 hover:text-blue-700" >
          Close < /button> < /div> < /div> < /div>
        )
      )
    } < /div> < /Dialog>
  );
};
export default ResultsModal;
```