МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма: «Безпека комп'ютерних систем і мереж»

Група: 4КБ-02

Дипломний проект

здобувача освіти денної форми навчання
КБ.02.19.000.ДП

САГАЙДАКА
МАКСИМА ІВАНОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: «**Безпека комп'ютерних систем і мереж»**

Група: **4КБ-02**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка захищеної системи аналізу

та оптимізації програмного коду

Проектний матеріал складається з пояснювальної записки на 81 сторінках та графічного (презентаційного) матеріалу на 17 аркушах (слайдах)

Дипломник _____ (Сагайдак М.І.)

Керівник _____ (Закроєв Ю.М.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «21» червня 2025 р. Протокол ЕК № 2

Оцінка ЕК 5 (відмінно) / 95 з.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Безпека комп'ютерних систем і мереж»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"19" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Сагайдаку Максиму Івановичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка захищеної системи аналізу та оптимізації програмного коду*

затверджена наказом по коледжу від "14" *листопада* 2024р. № *246*

2. Термін здачі закінченого проекту 16.06.25.

3. Вихідні данні до проекту

1. Передбачити використання штучного інтелекту для аналізу та оптимізації коду;

2. Використати сучасні Front-End технології;

3. Використати сучасні Back-End технології;

4. Реалізувати захищену архітектуру, з урахуванням безпеки клієнтських даних.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

1. Аналіз предметної області. 2. Технології та засоби розробки. 3. Проєктування архітектури системи обміну повідомленнями. 4. Робота з JWT-токенами та refresh-токенами 5. Розробка серверної частини 6. Інтеграція з сервісами електронної ідентифікації 7. Реєстрація, вхід, управління сесіями 8. Реалізація модуля автентифікації 9. Економічний розрахунок. 10. Аспекти охорони праці та техніки безпеки

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Мета проєкту; Технології; Файлова схема Front-End; Файлова схема Back-End; User Flow; Дизайн; Розробка Front-End частини; Розробка Back-End частини; Сторінка авторизації; Головна сторінка; Сторінка з результатом аналізу коду; Сторінка генерації UUID/GUID; Сторінка генерації HASH; Сторінка з інформацією; Вивантаження на сервери.

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Закроєв Ю.М.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 12.05.2025

Керівник *Закроєв Ю.М.* (підпис)

Завдання прийняв до виконання *Сагайдак М.І* (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Формування вступу*	15.05.2025	виконано
2	*Аналіз предметної області*	16.05.2025	виконано
3	*Підбір технічної літератури*	18.05.2025	виконано
4	*Вибір технологій та архітектурних рішень*	24.05.2025	виконано
5	*Проєктування дизайну веб-інтерфейсу*	26.05.2025	виконано
6	*Реалізація Front-End частини*	01.06.2025	виконано
7	*Реалізація Back-End частини*	04.06.2025	виконано
8	*Тестування розробленого Full-Stack веб-застосунку*	05.06.2025	виконано
9	*Оформлення пояснювальної записки*	07.06.2025	виконано
10	*Оформлення графічної (презентаційної) частини*	08.06.2025	виконано
11	*Економічний розрахунок*	10.06.2025	виконано
12	*Опис охорони праці та техніки безпеки*	12.06.2025	виконано
13	*Аналіз результатів розробки*	13.06.2025	виконано
14	*Підготовка доповіді для захисту*	14.06.2025	виконано
15	*Формування вступу*	16.06.2025	виконано

Дипломник (підпис)

Керівник (підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КБ 02. 19 000. 00 ДП	Дипломний проект		
А4			КБ 02. 19 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КБ 02. 19 000. 00 ДП		
Розробив		Сагайдак М.І.		16.05	Розробка захищеної системи аналізу та оптимізації програмного коду	Літ.	Аркуш 4 / Аркушів 81
Перевірив		Закроєв Ю.М.		16.05		Н Д П	
Н. контр.		Петрашова В.І.		16.05		ВСП "ОТФК ОНТУ" гр.4КБ-02	
Затверд.		Кривченко Ю.В.		16.05			

ЗМІСТ

ВСТУП

Різноманітні стартапи поступово збільшують свою кількість у зв'язку зі стрімким зростом технологій, однак більша кількість навіть не доходить і до MVP (мінімально життєздатного продукту). Очевидно, що на успішний вихід стартапу на ринок впливає низка факторів, серед яких одним із ключових є швидкість розробки. Не менш важливою є й вартість створення продукту – надмірні витрати можуть повністю зупинити реалізацію ідеї. Виходячи з цих факторів, можна зробити висновок, що оптимальним підходом є створення MVP якомога швидше та з мінімальними витратами, що в свою чергу дозволить протестувати гіпотези та перевірити попит.

Найм Senior-розробника стартапу зазвичай дається доволі важко, в силу обмеженого бюджету. Крім того зазвичай досвідченим фахівцям також не дуже подобається працювати в нестабільному стартапі. Виходячи з цих факторів стартап зазвичай наймає Junior або Middle-розробників. Але тут з’являється інша проблема, як правило їхня продуктивність і досвід можуть бути суттєво нижчими, що критично для темпів розвитку.

Ось тут і вступає в силу ШІ (штучний інтелект). Спеціально навчена модель штучного інтелекту може значно підвищити продуктивність розробника, що позитивно вплине на загальну швидкість розробки та в подальшому на швидкість виходу на ринок. Тип паче і програмісту це також дає свої плюси, але звісно за умови його правильної інтеграції у робочий процес.

Даний проєкт – “AuditX”, являє собою модель штучного інтелекту взаємодія з якою відбувається через повноцінний веб-інтерфейс. Фактично “AuditX” використовує Chat GPT але з упором на взаємодію з програмним кодом.

Для розробки Front-End частини проєкту використовувався сучасний веб-фреймворк React, для стилізації бібліотека Tailwind CSS. Для Back-End частини Express та база даних MongoDB.

Мета даного проєкту – допомогти новим стартапам реалізовуватись, а розробникам-початківцям – працювати у більш комфортних умовах.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Огляд сучасного стану предметної галузі

1.1.1 Вивчення ринку існуючих аналогів

Розглянемо наступні існуючі аналоги ШІ-асистентів – “Claude”, “ChatGPT ” та “DeepSeek”. Усі вони являються мовними моделями LLM (Large Language Models), кожна з них спеціалізуються на генерації тексту та не тільки. В основі всіх зазначених асистентів лежить концепція мовних моделей, які є одним з основних напрямків розвитку сучасного штучного інтелекту [1].

“Claude” являє собою збалансовану генеративну модель яка розроблена компанією “Anthropic”. Модель видає комплексні відповіді на повсякденні запити користувача. Застосовується для вирішення корпоративних завдань: пошук відповідей на питання, обробка даних, парсинг тексту з зображень, переклад та інші завдання. Країна походження моделі США, актуальна версія Claude 3.5. Однією з особливостей “Claude” являється орієнтованість на етичність, безпечність та підвищену прозорість. Застосування таких рішень добре описано у сучасних навчальних посібниках з інформаційних технологій [2].

На рисунку нижче 1.1. зображено веб-інтерфейс “Claude”.

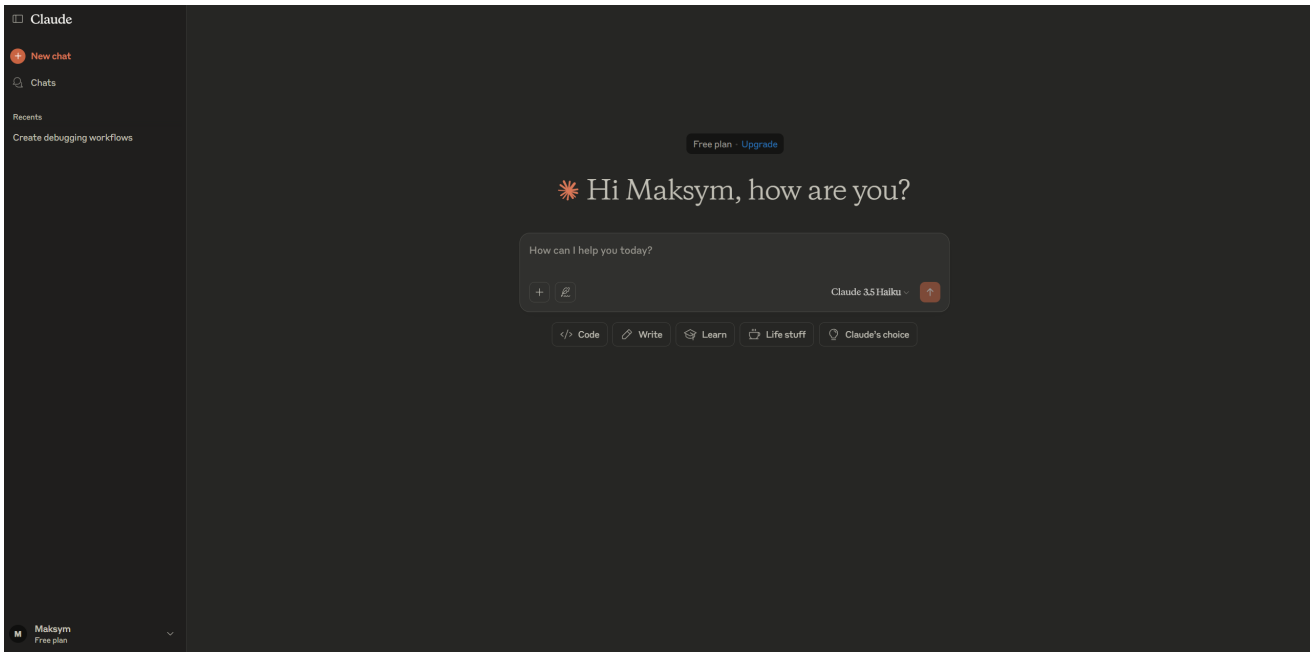


Рисунок 1.1. веб-інтерфейс “Claude”

Переваги:

1. Має змогу надати найбільший контекст (до 200 тисяч токенів у Claude 3 Opus).
2. Орієнтована на максимально безпечну та етичну взаємодію.
3. Часто краще за інші моделі обробляє навчальні запити, пояснює логіку своїх дій, чим, як найкраще підходить для навчання.
4. Завдяки змозі надати великий контекст (до 200 тисяч токенів), краще розуміє складні інструкції з багатьма умовами.
5. Необхідний API вже доступний для інтеграції в різноманітні продукти.

Недоліки:

1. Нажаль поки немає повноцінної підтримки голосової моделі (тільки текст). Але це вже в процесі розробки.
2. Має обмеження в деяких країнах.
3. Оскільки “Claude” зазвичай намагається надати як можна більш релевантну відповідь то обробка запиту може займати набагато більше часу ніж у інших моделей.
4. Якраз через свою місцями надто збільшену етичність, може уникати та не розкривати деякі теми в повному обсязі, в той час, як інші моделі цим не обмежуються.
5. Оскільки модель являється відносно новою, деякі інструменти відсутні, починаючи з голосової моделі, закінчуючи плагінами та розширеннями.

“ChatGPT” являє собою чат-бот с генеративним штучним інтелектом, розроблений компанією “OpenAI”. “ChatGPT” здатний працювати в діалоговому режимі, підтримуючи запити природними мовами. Система здатна відповідати на питання, генерувати тексти різними мовами, включаючи українську. Має найсильнішу модель для генерації коду “GPT-4 Turbo”. Має змогу обробляти як текстові файли так і зображення, тим більше і генерувати їх.

На рисунку нижче 1.2. зображено веб-інтерфейс “ChatGPT”.

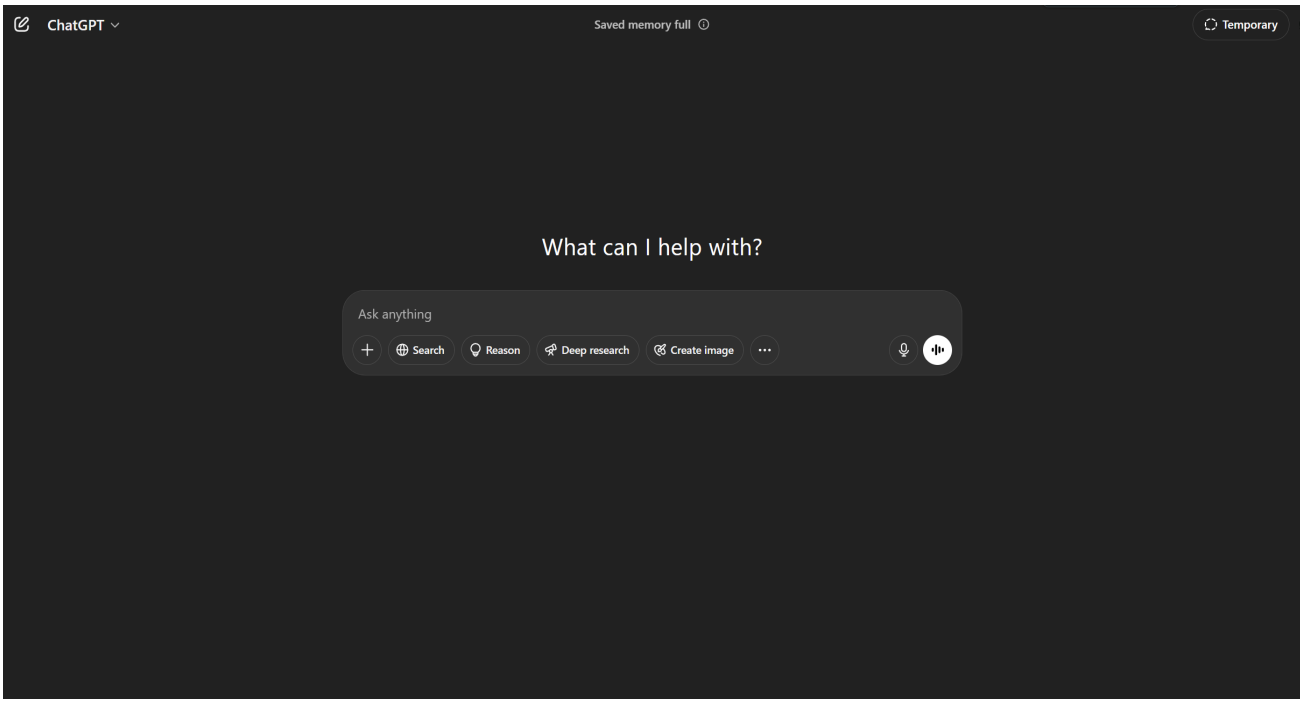


Рисунок 1.2. веб-інтерфейс “ChatGPT”

Переваги:

1. Однією з головних переваг являється підтримка голосової взаємодії (через “ChatGPT Voise”).
2. Має одну з найсильніших моделей для генерації коду (GPT-4 Turbo).
3. Підтримує обробку та навіть генерацію різноманітних файлів, зображень, таблиці.
4. Має неймовірно велику кількість інструментів, плагінів та розширень.
5. Являється однією з популярніших та найстабільніших моделей.

Недоліки:

1. Одна з найдорожчих моделей, саме в якій і доступний необмежений “GPT-4”, безкоштовна версія дозволяє використовувати “GPT-3.5” та “GPT-4” обмежений час.
2. Має найбільш виражені “галюцинації” – модель сама додумаю деякі факти, із-за чого можуть бути неточності у відповіді.
3. В деяких випадках модель не в змозі в повному обсязі пояснити хід своїх думок, що може стати вирішальним фактором при вибору моделі для навчання.

“DeepSeek” – це нейромережа, розроблена однойменною китайською

компанією. На відміну від більшості інших рішень, представлених на ринку, її перевага полягає в тому, що користувачам вона доступна абсолютно безкоштовно практично без обмежень щодо кількості та тематики запитів. Нейромережа пише тексти, аналізує документи, програмує і робить багато іншого, не вимагаючи купувати підписки.

На рисунку нижче 1.3. зображено веб-інтерфейс “DeepSeek”.

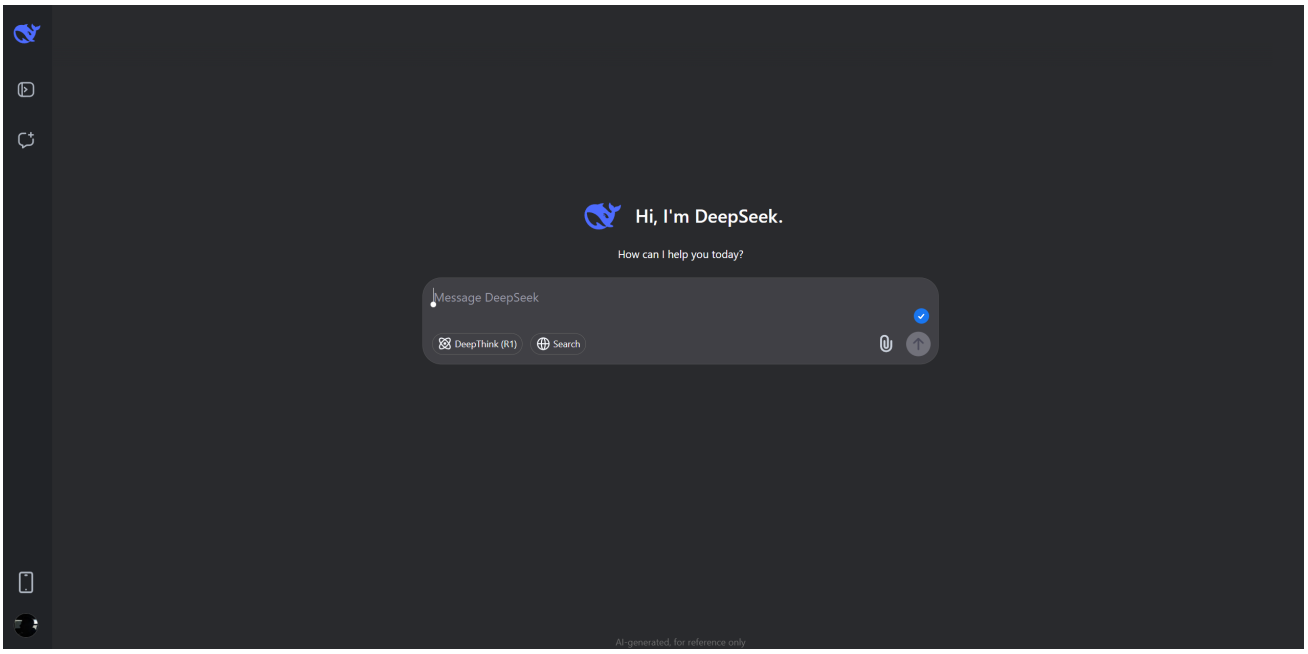


Рисунок 1.3. веб-інтерфейс “ DeepSeek”

Переваги:

1. Дає повністю безкоштовний доступ до доволі потужних моделей.
2. Славиться високою точністю у генерації коду та розв’язувані математичних задач. В більшому завдяки саме навчанню моделі на саме такого типу даних.
3. Модель орієнтована на open-source спільноту, відповідно має відкритий вихідний код.
4. Одна з найпрогресивніших моделей, постійно оновлюється. В більшості завдяки open-source спільноті.
5. Доволі ефективна у вузькопрофільних задачах, а саме у технічних.

Недоліки:

1. Повністю відсутня голосова взаємодія.
2. Оскільки модель вийшла в реліз доволі не давно, інтерфейс може здатися не

таким зручним.

3. Оскільки модель вважається більш гуманітарною, то в деяких випадках може не вистачати креативності.
4. Знов повертаючись то того, що модель вийшла в реліз не давно, на сьогодні вона не може похвалитися великою кількістю плагінів та розширень.
5. Менш відома поза технічною спільнотою та має менше довіри з боку бізнесу.

Порівняльну характеристику розглянутих рішень та їх функціоналу наведено у таблиці 1.1.

Таблиця 1.1. Порівняльна характеристика існуючих рішень.

Критерій порівняння	**Claude (Anthropic)**	**ChatGPT (OpenAI)**	**DeepSeek (DeepSeek AI)**
Країна походження	США	США	Китай
Модель (версія)	Claude 3.5	GPT-4 / GPT-3.5	DeepSeek-VL
Підтримка коду	Так, фокусується на поясненнях та безпечності	Так, один з найкращих у класі (кодова підмодель "Code Interpreter")	Так, із сильним фокусом на математику та код
Голосова взаємодія	Обмежено (на момент написання – у тестовій фазі)	Так (у ChatGPT Voice, через Whisper + TTS)	Ні
Контекстна довжина	До 200 тисяч токенів (Claude 3 Opus)	До 128 тисяч токенів (GPT-4 Turbo)	Обмежена, але з фокусом на ефективність
Платні функції	Так	Так	Ні
Критерій порівняння	**Claude (Anthropic)**	**ChatGPT (OpenAI)**	**DeepSeek (DeepSeek AI)**
Вартість	Висока	Висока	Відсутня
Відритий вихідний код	Ні	Ні	Так

Продовження таблиці 1.1

Критерій порівняння	Claude (Anthropic)	ChatGPT (OpenAI)	DeepSeek (DeepSeek AI)
Особливості	Орієнтована на етичність, безпечність, підвищену прозорість	Інтеграція з плагінами, можливість роботи з файлами, голосовий режим	Безкоштовна, розвивається з відкритим доступом до деяких моделей

1.2 Обґрунтування обраних архітектурних рішень та технологій

1.2.1 Font-End технології

При розробці Front-End частини мною були обрані найбільш релевантні та сучасні рішення. Звісно, за основу, без якої не обійшовся будь-який веб-застосунок, мною була обрана мова гіпертекстової розмітки – HTML. Для стилізації, мова каскадних стилів CSS. Та щоб перейти зі стадії “картинки” в інтерактивний веб-застосунок мною використовувалась мова програмування JavaScript. Всі попередньо перераховані технології є базою, які не мають аналогів. Є безліч різноманітних бібліотек, фреймворків, але всі вони все одно інтерпретуються в ці три могучих слони, які стоять на черепасі під назвою браузер.

Саме браузер інтерпретує HTML, CSS та JavaScript, перетворюючи їх на взаємодійну сторінку, яку бачить користувач. Усі сучасні інструменти фронтенду – від React і Vue до Tailwind чи Webpack – у підсумку лише спрощують, структурують або автоматизують роботу з цими трьома базовими технологіями. Вони є незамінним фундаментом будь-якого веб-інтерфейсу, і розуміння їх роботи – ключ до створення якісного та ефективного користувацького досвіду.

Хоч мені й довелося згадати базові веб-технології, без яких не обходиться жоден сучасний застосунок, проте у роботі використовував їх не зовсім у нативному вигляді. Попередньо це можна обґрунтувати тим, що світ Front-End розробки вже давно перейшов за базові межі “чистих” HTML, CSS та JavaScript, в силу швидкості розробки та в деякій мірі зменшення рутини Front-End розробників.

Як основу для створення інтерфейсу було прийнято рішення обрати бібліотеку – React, хоч із кожною мажорною версією все більше і більше React

схожий на повноцінний фреймворк, але і досі продовжує позиціонувати себе як бібліотеку. Завдяки React можна не мислити окремими сторінками, оскільки React дозволяє продумати інтерфейси як компоненти, які є ізольованими, повторно використовуваними одиницями логіки та вигляду, що складаються у складну, але впорядковану структуру інтерфейсу. Такий підхід значно спрощує розробку, тестування та масштабування веб-застосунку, особливо у середніх та великих проектах. Також можна зазначити т ще мільйонів діючих Front-End розробників, кіллер фічою React є JSX, який поєднує в собі JavaScript та HTML-подібний синтаксис в єдине ціле. JSX дозволяє описувати структуру інтерфейсу прямо всередині коду, використовуючи звичну для розробника логіку та змінні, що значно підвищує читабельність та підтримуваність проєкту. Завдяки цьому підходу, логіка та візуальне представлення компонента більше не розкидані по різних файлах чи шаблонах, а зібрані разом – в одному місці, як цілісна функціональна одиниця. Це дає змогу швидко орієнтуватися в коді, легко передавати дані між компонентами, керувати їхнім життєвим циклом та гнучко реагувати на зміни стану програми. А з підтримкою сучасних хуків, таких як useState, useEffect, useContext та багато інших, React став ще більш зручним для розробки складних інтерфейсів з чіткою структурою та прогнозованою поведінкою.

Щоб вибір виглядав не як сліпе слідування моді, а як усвідомлене рішення, було проведено аналіз найпопулярніших інструментів, які використовуються для створення сучасних UI.

Нижче наведено порівняльну таблицю 1.2. Що охоплює ключові аспекти таких рішень, як React, Angular, Vue та Svelte.

Таблиця 1.2. Порівняння фреймворків React, Angular, Vue та Svelte.

Критерій	**React**	**Angular**	**Vue**	**Svelte**
Тип	Бібліотека для UI	Повноцінний фреймворк	Прогресивний фреймворк	Компілер (компілює в чистий JS)
Старт року	2013	2010	2014	2016

Продовження таблиці 1.2

Критерій	React	Angular	Vue	Svelte
Підхід до UI	Компонентний + JSX	Компонентний + шаблони	Шаблони + опціональний JSX	Компоненти з автооптимізацією
Синтаксис	JSX (JavaScript + HTML)	HTML шаблони + TypeScript	HTML шаблони та JSX	HTML-подібний, простий синтаксис
Типізація	JavaScript, TypeScript	TypeScript	JavaScript / TypeScript (через обгортки)	JavaScript / часткова підтримка TS
Продуктивність (runtime)	Висока, але з віртуальним DOM	Добра, але важчий фреймворк	Висока, з оптимізованим VDOM	Дуже висока – без VDOM
Масштабованість	Висока (із правильним підходом)	Висока (все включено)	Висока	Середня
Реактивність	Через стейт-менеджмент / хуки	Через служби та RxJS	Проста та декларативна	Автоматична, на рівні компіляції
Рендерінг на сервері (SSR)	Next.js	Angular Universal	Nuxt.js	SvelteKit

Кожне з рішень має свої унікальні переваги:

1. React – ідеально підходить для гнучкої побудови UI, має найбільшу спільноту та широку екосистему.
2. Angular – вибір для корпоративного рівня, із суворою структурою та інтеграцією всього "з коробки".
3. Vue – відомий своєю простотою та адаптивністю, особливо для швидкого старту проєктів.
4. Svelte – радикально новий підхід, де більшість логіки переноситься на етап компіляції, що дає мінімальний runtime overhead.

До речі, повертаючись до React, потрібно відмітити, що тут немає обмеження використовувати лише React як такого – до стеку було також додано TypeScript, що стало кроком у напрямку більш строгої, безпечної та масштабованої розробки. TypeScript яляє собою надбудову над JavaScript, яка вводить статичну типізацію та дозволяє виявляти помилки ще на етапі розробки, до того, як код потрапить у браузер. Такий тандем – React та TypeScript – на сьогодні є де-факто стандартом у професійному середовищі Front-End розробки. Саме він дозволяє будувати складні, надійні, безпечні та підтримувані веб-застосунки, де кожен компонент чітко знає, які дані він приймає, що очікує на виході, і яку роль відіграє в загальній архітектурі інтерфейсу. І саме цей підхід, поєднаний із сучасними практиками було обрано для реалізації даного проєкту.

Для стилізації було використано Tailwind CSS – утилітарну CSS-бібліотеку нового покоління, яка кардинально змінює підхід до написання стилів. Tailwind не змушує створювати окремі CSS-файли з купою класів – замість цього, він дозволяє використовувати вже готові класи безпосередньо в HTML (або JSX), забезпечуючи надзвичайно швидкий та інтуїтивно зрозумілий процес верстки. Tailwind дає змогу створювати сучасні, адаптивні інтерфейси з мінімальною кількістю коду та максимальною гнучкістю. Його філософія "design in markup" ідеально пасує до компонентного підходу React [3].

Та не дивлячись на велику популярність Tailwind, він не єдиний варіант для розробників. Існують як класичні фреймворки, на кшталт Bootstrap, так і нові, більш “атомарні” альтернативи, які підходять для інших підходів до дизайну. Для оцінки переваг та недоліків різних рішень, подивимось нижче на таблицю 1.3. з порівнянням Tailwind CSS, Bootstrap та Bulma.

Таблиця 1.3. Порівняння Tailwind CSS, Bootstrap, Bulma.

Критерій	**Tailwind CSS**	**Bootstrap**	**Bulma**
Тип	Утилітарний CSS	Компонентний CSS + JS	Компонентний CSS

Продовження таблиці 1.3

Критерій	**Tailwind CSS**	**Bootstrap**	**Bulma**
Автори	Tailwind Labs	Twitter	Nicolas Gallagher / інші
Рік релізу	2017	2011	2016
Філософія	Стилізація прямо в HTML через класи	Використання готових компонентів	Прості класи для компонентів
Готові компоненти	Мінімум (через Headless UI / Flowbite)	Багато, одразу “з коробки”	Є базові
Особливості	Стилі прямо в класах, мобільно першим	Найпопулярніший класичний фреймворк	Простий та гнучкий
Використовується в	Vercel, GitHub	Airbnb, Spotify, NASA	VueJS Docs, JAMstack

Підведемо підсумок табличної інформації:

1. Tailwind CSS – дає повний контроль і чудово масштабуються в великих проєктах, але вимагає звикання.
2. Bootstrap – ідеально підходить для швидкого прототипування та корпоративних рішень, але часто “виглядає як Bootstrap”.
3. Bulma – легкий у вивченні, без зайвого JavaScript, добре підходить для невеликих проєктів.

Оскільки кожна бібліотека має свою філософію та цінності, потрібно підбирати ту яка якомога краще співпадає з цінностями проєкта. Оскільки мені була важлива швидкість, гнучкість та масштабованість, було вирішино зупинитися на Tailwind CSS.

Доволі важливо обрати саме ті технології, які якомога найкраще відповідатимуть специфіці проєкту, але навіть найкращого вибору технологій не вистачить для проєкту який за розміром середній або більше середнього. Тут

важливо обдумати і архітектурні рішення.

Для даного проєту “AuditX” було вирішено використовувати архітектуру FSD (Feature-Sliced Design), яка являє собою архітектурну методологію для проектування Front-End-застосунків [4]. Простіше кажучи, це зведення правил та угод щодо організації коду. Головна мета методології – зробити проект зрозумілим та структурованим, особливо за умов регулярної зміни вимог бізнесу.

Архітектурна методологія FSD має такі переваги як:

1. Явна бізнес-логіка. Архітектуру легко освоювати, оскільки вона складається з доменних модулів.
2. Адаптивність. Компоненти архітектури можна гнучко замінювати, додавати до нових умов.
3. Технічний обов'язок. Кожен модуль можна незалежно модифікувати або переписати без сайд-ефектів.
4. Явна перевикористання. Зберігається баланс між DRY та локальною кастомізацією.

Також FSD пропагує такі концепції:

1. Public API. Кожен модуль повинен мати на верхньому рівні декларацію свого публічного API.
2. Ізоляція. Модуль не повинен залежати безпосередньо від інших модулів того ж шару або вище шарів
3. Розуміння потреб. Орієнтування на потреби бізнесу та користувача

У FSD архітектура проєкту поділяється на три ключові рівні:

1. Layers (Слої) — логічне групування модулів за роллю: від загальних (shared) до конкретних застосунків (app).
2. Slices (Слайси) — розподіл за бізнес-сутностями, наприклад: user, post, comment.
3. Segments (Сегменти) — частини всередині кожного слайсу: ui (інтерфейс), model (логіка), api (запити).

На рисунку нижче 1.4. зображено як слої, слайси та сегменти утворюють ієрархію:

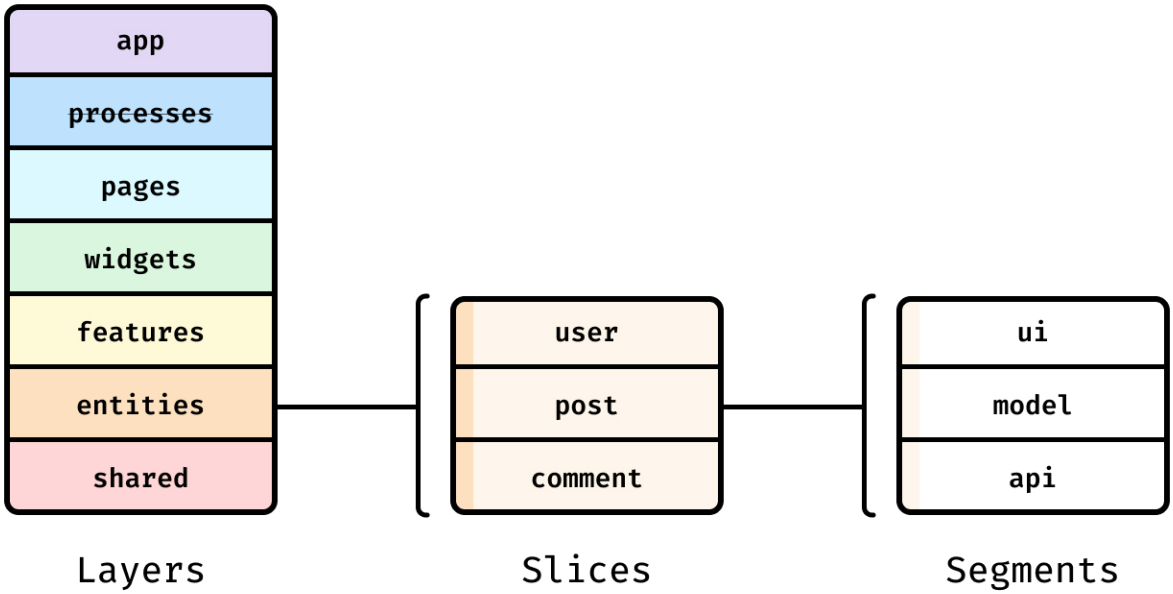


Рисунок 1.4. Шари, слайси та сегменти FSD

Оскільки, не в перший раз використовується дана архітектурна методологію FSD, і доволі тісно відомі всі її плюсами та мінуси, було вирішено, що FSD є найбільш підходящою архітектурною методологією для даного дипломного проєкту.

Підводячи підсумок до цього розділу, хочеться підкреслити доцільність і обґрунтованість обраного технічного стеку та архітектурної методології. За основу взятий React та TypeScript, стилізується за допомогою бібліотеки Tailwind CSS, а для масштабування та швидкого змінювання бізнес-логіку була обрана архітектурна методологія FSD. Як показала подальша розробка цей вибір виявися не помилковим.

1.2.2 Back-End технології

При розробці Back-End частини, мною була обрана мова програмування NodeJS [5]. NodeJS це той самий JavaScript але з можливістю працювати не тільки у веб-середовищі в браузері на стороні клієнта, але і поза браузером. В більшості вибір був оснований на тому, що вже є здобуті знання з мови програмування JavaScript, і не доведеться вчити новий синтаксис нової мови програмування.

Хоч даний вибір навряд чи від чогось змінився, все одно є сенс порівняти NodeJS з аналогами. Нижче на таблиці 1.4. зображено порівняльну характеристику мов програмування, які ставлять на одну полицю з NodeJS, а саме Python та PHP.

Таблиця 1.4. Порівняння NodeJS, Python, PHP.

Критерій	**Node.js (JavaScript)**	**Python (Django / Flask)**	**PHP (Laravel / Symfony)**
Популярність	Дуже популярний, особливо в стартапах і для SPA-додатків	Дуже популярний у Data Science, веб-розробці	Популярний для веб-сайтів, CMS (WordPress)
Швидкодія	Висока (асинхронна модель, події)	Середня (інтерпретована мова)	Середня (синхронна модель)
Масштабованість	Добре масштабується	Підходить для малих/середніх проєктів	Можливе масштабування, але складніше
Фреймворки	Express, Nest.js, Koa	Django, Flask, FastAPI	Laravel, Symfony, CodeIgniter
Готовність до високих навантажень	Висока (особливо з кластеризацією)	Обмежена, потрібно оптимізувати	Є Не завжди підходить для великих навантажень
Сфера застосування	Веб, API, реального часу (чат, сокети)	Веб, API, машинне навчання	Веб-сайти, CMS, блоги
Підтримка спільноти	Дуже велика	Дуже велика	Дуже велика

Оскільки великої різниці між ними немає, було вирішини зупинити вибір на більш комфортному для мене NodeJS. Мова повністю закриває мої потреби, тим більше знайомий для мене синтаксис.

Доволі довго йшли міркування над потрібністю бази даних, і скоріше всього зупинив вибір був би зупининий на об’єктно орієнтованій базі даних MongoDB. Але було прийнято рішення не зберігати на сервері ніяких даних користувача. Дане рішення можна обґрунтувати посилаючись на контекст використання “AuditX”.

“AuditX” позиціонує себе як помічник розробнику, а в цій практиці дуже важлива безпека даних та конфіденційність особливо стосовно програмного коду. Варто зауважити, що справді серйозно розглядалася ця технологія. По-перше, вона дозволяє зберігати гнучкі схеми даних, що дуже зручно у випадках, коли структура збереженої інформації може змінюватись з часом. Це важливо в контексті розробки інструменту типу “AuditX”, де потенційно можна було б зберігати історію змін, логів або навіть результати проходження певних навчальних завдань з безпеки. Але, як вже було зазначено, на даному етапі пріоритетом є саме відсутність централізованого зберігання. Це не лише про безпеку, а й про довіру: користувач має бути впевнений, що після завершення сесії жодні його дії не залишають цифрового сліду на стороні сервера.

Для авторизації у даному проєкті було вирішено використовувати модель “OAuth2.0” оскільки вона дозволяє реалізувати авторизацію без потреби зберігати паролі користувачів безпосередньо в базі даних, або взагалі уникати обробки облікових даних на стороні сервера. Це добре вписується в загальну концепцію “AuditX” як інструменту, який не накопичує чутливу інформацію, а навпаки – забезпечує повну прозорість і мінімізацію ризиків для користувача. У даному випадку на даний момент користувач може авторизуватися через Google – а моя система вже отримує обмежений токен доступу, який можна використати лише у межах сесії. Вибір OAuth2 ще й підкріплений тим, що він дозволяє легко масштабувати систему в майбутньому. Якщо згодом виникне потреба додати, наприклад, окремий рівень ролей, або інтегрувати систему з корпоративними середовищами, то модель авторизації вже буде готова до цього, без необхідності перебудови архітектури з нуля.

У зв’язку з цим архітектура “AuditX” базується на так званому stateless-підході. Уся інформація, що необхідна для роботи користувача, зберігається локально а всі обчислення і перевірки відбуваються на стороні клієнта. Це дозволяє значно підвищити рівень приватності та мінімізувати ризики витоку інформації – навіть у разі компрометації сервера, жодні важливі дані не можуть бути викрадені, бо їх там просто немає. Такий підхід також спрощує питання GDPR та інших

регуляторних обмежень – адже, технічно, персональні дані користувача не обробляються, тому зникає потреба в згоді на їх обробку, а також у тривалому зберіганні даних.

Але важливо відмітити, що саме сервер “AuditX” не збирає та не зберігає жодних даних користувача. Але від капотом є сервіс “Google” – “OAuth2.0” для авторизації та сервіс “OpenAI” – “ChatGPT” при взаємодії з кодом.

Варто зазначити, що відмова від серверного зберігання – це не лише спрощення, а й свідома архітектурна стратегія. Вона підкреслює основні цінності “AuditX”: прозорість, автономність користувача та орієнтацію на безпеку з перших рядків коду.

Поглянемо нижче на рисунок 1.5, де зображено обрані мною технології, для подальшої розробки веб-застосунку.

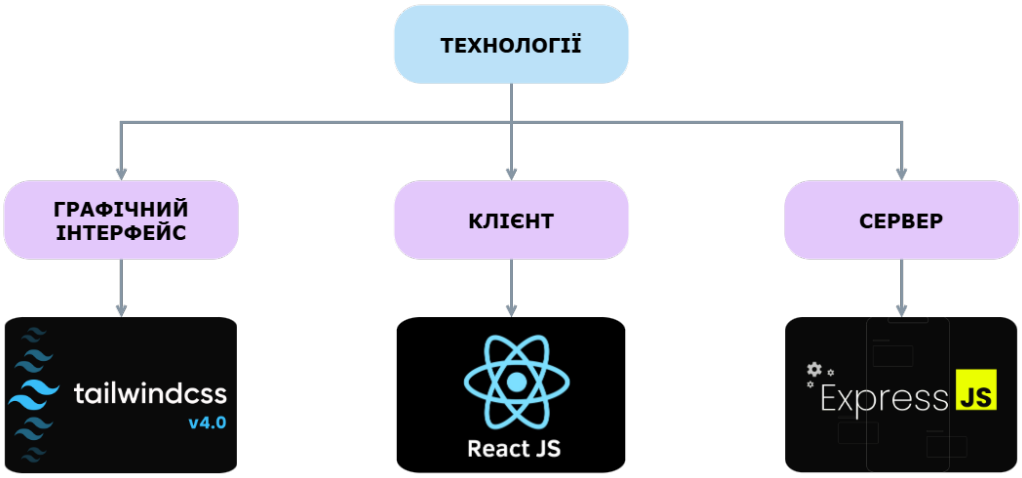


Рисунок 1.5. Обрані технології

1.3 Проєктування веб-системи

1.3.1 Технічне завдання

Виходячи з того, що даний проєкт “AuditX” позиціонується як сервіс для допомоги діючим програмістам, або ж початківцям, було вирішено закласти в нього ідею безпеки як основну цінність. Тобто не як додаток «на майбутнє», не як гарну фічу для презентації, а саме як фундаментальну складову архітектури. Безпека не повинна бути додатком до системи, вона повинна бути в її серці – саме

так видно ідеальну структуру подібного інструменту. “AuditX” не тільки про зручність, а й про довіру. У мене не було в планах створити чергову платформу, де користувач щось вводить і не знає, куди воно потім потрапить. Навпаки – хотілось, щоб навіть найскептичніший користувач відчував: тут його дані в безпеці, а ще краще – тут їх взагалі ніхто не бачить. Це рішення не тільки ідеологічне, а й практичне. У сучасному вебі важко довіряти будь-кому, особливо коли мова йде про код, а тим більше – про потенційно вразливий код.

Саме тому з самого початку було вирішено уникати зберігання будь-яких даних на сервері “AuditX”. Усе, що потрібно для роботи, має оброблятись локально, наскільки це можливо. Але все ж таки оскільки “AuditX” не має своєї ШІ моделі він поки використовує “ChatGPT API”, з цього можна зробити висновок, що все одно як би не хотілось зробити “AuditX” повноцінним монолітним безпечним сервісом, поки це не вийде, с силу фінансової неспроможності, але про цю інформацію “AuditX” попереджає користувача перед відправкою його даних сторонньому сервісу (ChatGPT), в той же час, де все відбувається без взаємодії зі сторонніми сервісами такого попередження не буде.

Моє бачення: максимально stateless. Тобто сервер “AuditX”, якщо і використовується, то виключно як посередник, як провідник, а не як сховище. Усі сесії тимчасові, всі дані зникають після закриття вкладки, жодних слідів, жодних архівів, ніякої аналітики користувача сервіс “AuditX” не зберігає собі.

Для авторизації було обрано модель OAuth2. Це сучасний, перевірений підхід, який дозволяє користувачеві залишатися у звичному середовищі авторизації. “AuditX” не обробляє паролі, не зберігає логіни, не бере на себе відповідальність за збереження таких даних. Усе максимально просто, чесно і безпечно.

Окремо варто сказати про фічі, які було закладено у функціонал. По-перше, це інтеграція з інструментами, які можуть справді допомогти програмісту в щоденній роботі: генерація UUID та хешів – не просто як прикольні кнопки, а як практичний інструмент. UUID часто потрібен під час розробки API або при створенні об’єктів, які мають унікальні ідентифікатори. Хешування – незамінне в

практиці безпечного зберігання даних або навіть при побудові мікросервісної архітектури. Це речі, які використовуються щодня, і якщо вони під рукою – то це просто зручно.

Ще одне – це інтеграція ШІ-помічника. Ідея не в тому, щоб AI писав код за користувача – ні. Основна мета – пояснення, підказк та підтримка. Це може бути як щось середнє між ментором і компілятором: з одного боку – підказує, коли є помилка, з іншого – пояснює, чому вона сталася. І головне – робить це в контексті безпеки, а не просто синтаксису. Тобто користувач не просто дізнається, що десь не вистачає дужки, а що ця дужка в такому-то місці може дати XSS або SQL-ін’єкцію. Звісно одразу потрібно нагадати, що “AuditX” не має своєї AI моделі, тому використовує “ChatGPT API”, тому весь код який відправляється, все одно потрапляє до стороннього сервісу, а вже відповідальність за роботу сторонніх сервісів “AuditX” не несе, але обов’язково перед такою дією користувач буде про це завчасно попереджений та вже сам буде приймати рішення чи довіряє він свої дані “OpenAI”.

Освітній елемент – одна з цілей “AuditX”. Також у технічному завданні було враховано розширюваність. Тобто одразу було заплановано структуру проєкту так, щоб його можна було доповнювати: новими модулями, новими типами перевірок, новими підказками та сценаріями. Наприклад, у майбутньому можна буде додати аналіз коду на основі AST (Abstract Syntax Tree), або перевірку залежностей із точки зору вразливостей. Усе це можливо завдяки початково правильному технічному фундаменту.

Ще один важливий момент “AuditX” – бути легким, зручним веб-інструментом, який запускається миттєво і працює швидко, навіть на слабкому пристрої. Для цього було використано сучасні фреймворки, оптимізую завантаження ресурсів і будую логіку взаємодії з користувачем таким чином, щоб усе працювало на клієнті. Це не лише зручно, а й відповідає моїй головній ідеї: контроль залишається у користувача.

Підводячи підсумок: технічне завдання “AuditX” – це зробити такий інструмент, який буде чесним, надійним, корисним. І саме на цих принципах

плануємо будувати кожен наступний етап реалізації.

1.3.2 Проектування дизайну веб-інтерфейсу

Насправді при замовлені веб-сайту, замовник очікує отримати прототипи для подальшого узгодження. Це важливий етап, де визначається дизайн та функціональність перед реалізацією [6]. При розробці прототипів враховують технічні вимоги та предметну область, а макети основних елементів, як футер чи навігаційне меню, розробляються у Figma, або інших графічних редакторах, таких як Photoshop або Adobe XD.

Розглянемо порівняльну таблицю аналогів для розробки дизайну у сучасному світі. Порівняльна таблиця представлена на таблиці 1.5.

Таблиця 1.5. Порівняння Figma, Photoshop та Adobe XD.

Критерій	**Figma**	**Photoshop**	**Adobe XD**
Призначення	Веб-дизайн, UI/UX, прототипування	Растрова графіка, фотоманіпуляції	UI/UX дизайн, прототипування
Співпраця в реальному часі	Підтримує (онлайн, кілька користувачів)	Не підтримує	Обмежена (через Creative Cloud)
Прототипування	Вбудоване	Немає (потрібні сторонні інструменти)	Вбудоване
Навчальна крива	Помірна, інтуїтивний інтерфейс	Висока, багато функцій і складний інтерфейс	Легка, зручний інтерфейс
Платформа	Онлайн (браузер, десктоп-додатки)	Десктоп (Windows, macOS)	Десктоп (Windows, macOS)
Ціна	Безкоштовний план + платний Pro	Платна підписка через Adobe Creative Cloud	Платна, але часто входить у пакет Creative Cloud

Figma це ідеальний для командної роботи та швидкого прототипування [7]. Photoshop краще підходить для роботи з графікою, а не UI/UX. Adobe XD зручний

для UI/UX, але трохи менш популярний за Figma і має обмеження щодо спільної роботи.

Тому було вирішено зупинити вибір на Figma. Вона більше підходить для швидкого прототипування та більш зрозуміла для тих, хто в дизайні не так давно.

Основні переваги Figma включають:

1. Figma дозволяє працювати в реальному часі над одним і тим же проєктом декільком дизайнером одночасно. Роблячи зміни, які одразу відображаються у всій команді, писати коментарі, відповідати на коментарі. Це доволі сильно прискорює процес розробки.
2. Незважаючи на відносно простий інтерфейс Figma дозволяє робити як зовсім простий дизайн, так і високодеталізовані та інтерактивні макети. Тим самим покриває потреби великої кількості дизайнерів.
3. Figma дозволяє робити не тільки статичний макет, а і вдихнути в нього життя, показати як має поводити себе вже повністю готовий проєкт. Досягається це завдяки різноманітним інструментам, обробникам дій, переходам та адаптивністю. Все це вже є в базовій версії Figma.
4. Зовсім не давно було додано AI помічника у Figma. Зараз він використовується для перетворення макетів Figma на повністю готовий сайт. Хоч це доволі сира реалізація, але в подальшому це може стати зручним інструментом для швидких тестів прототипів.

У даному проєкті було вирішино використати сучасний стиль дизайну, який називається Glassmorphism. Це стиль, який виглядає як скло – прозорий, з розмиттям заднього фону. Він зараз доволі популярний і часто використовується у мобільних застосунках, сайтах і навіть в операційних системах, наприклад, в iOS. Основна фішка в тому, що блоки не просто мають фон, а виглядають ніби зроблені зі скла. Крізь них можна трохи бачити те, що позаду, але все розмите. Це створює відчуття простору, легкості, і інтерфейс виглядає акуратно і сучасно.

Ще мені цей стиль подобається тим, що він не перевантажує очі. Елементи не виглядають важкими або нав’язливими. Все ніби парить над фоном, виглядає повітряно. Особливо круто це виглядає, якщо задній фон кольоровий або з

градієнтом – тоді ефект розмиття стає ще помітнішим і приємнішим.

Я пробував різні варіанти, але саме glassmorphism дав найкращий результат. Він одразу робить дизайн більш стильним, навіть якщо сама структура макету проста.

У Figma реалізувати цей стиль дуже легко. Там є готові ефекти розмиття, можна налаштувати прозорість, межі, тіні – і все це видно одразу. Мені не треба було довго розбиратись, бо інтерфейс інтуїтивний, і всі налаштування робляться в пару кліків. Навіть було знайдено плагіни, які допомагають ще швидше зробити елементи у стилі glassmorphism, але більшість з них і не потрібні, бо у Figma вже і так все є.

Можна зазначити, що цей стиль зараз є одним з найкращих варіантів для інтерфейсів, бо він одночасно і простий, і ефектний. Він додає трохи “дорогого” вигляду сайту або застосунку, але при цьому не потребує якихось суперскладних рішень. До того ж, він ідеально лягає на адаптивний дизайн – все виглядає красиво як на комп’ютері, так і на телефоні. У даному випадку доцільно зробити щось сучасне, але не переускладнювати – тому цей стиль підійшов якнайкраще.

Розглянемо дизайн хедеру веб-застосунку. Хедер є одним із ключових елементів інтерфейсу, він задає тон всьому веб-застосунку та забезпечує користувачам швидкий доступ до основних функцій.

Мною було прийнято рішення не робити хедер занадто забитим функціоналом, тому хедер даного веб-застосунку має:

1. В лівій частині логотип, він виконує функцію ідентифікації веб-сайту і є посиланням на головну сторінку.
2. В правій частині вже після авторизація з’являється випадаюче меню зі змогою перейти на сторінку інформації та вийти з аккаунту.

На рисунку 1.6. нижче зображено кінцеву версію хедеру, вже після авторизації користувача.

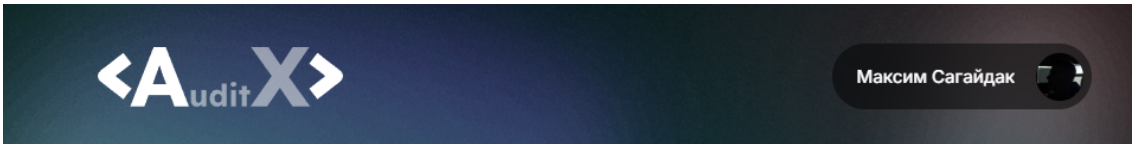


Рисунок 1.6. Дизайн хедеру веб-застосунку, що розробляється

Розглянемо дизайн секції з посиланням на сторінки, так званий navbar. Оскільки в подальшому ймовірно буде додаватися новий функціонал, важливо, щоб було місце під розташування посилань на ці сторінки.

На даний момент структура navbar така:

1. Першим йде посилання на сторінку UUID/GUID генератора.
2. Другим посилання на генерацію HASH на основі вхідних даних від користувача.

На рисунку 1.7. нижче зображено поточну версію navbar.

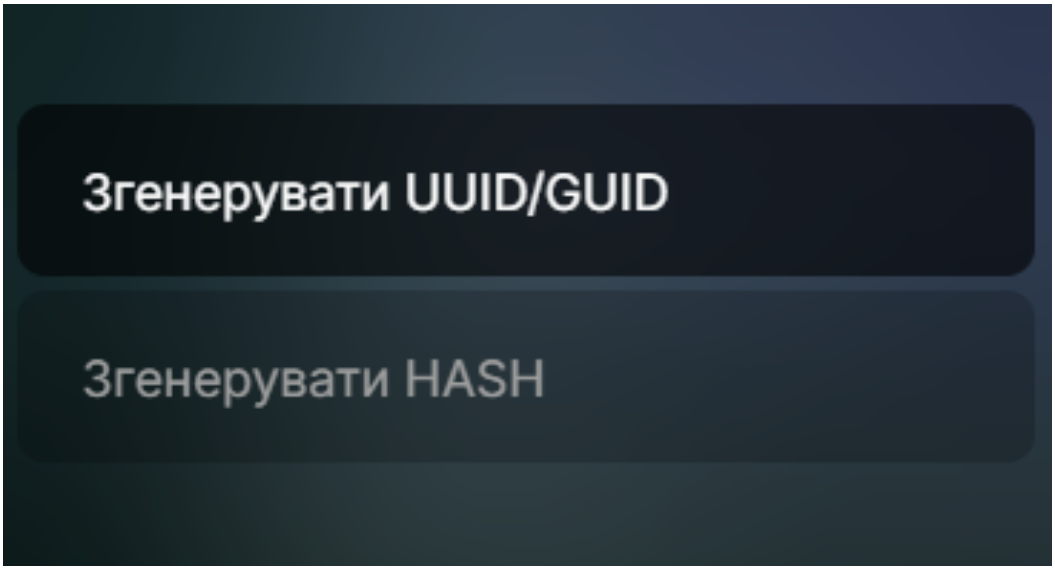


Рисунок 1.7. Дизайн navbar веб-застосунку, що розробляється

Хоча і досі не розповідалося про саму функціональність “AuditX”, розглянемо це в наступному розділі, але зараз важливо зазначити про таку візуальну частину сервісу “AuditX” як відправка користувацького коду на аналіз та оптимізацію [8]. Для цього використовується поле вода, в середині якого є функціональні кнопки для налаштування та відправки коду.

Поле вода користувацького коду має таку структуру:

1. Велике поле під написання\вставку користувацького коду.
2. Функціональні кнопки налаштування аналізу коду від ШІ, змога увімкнути оптимізацію коду, кнопка поради є обов’язковою.
3. Кнопка відправки коду. Якщо все вірно та нарікань від системи немає, залишається тільки натиснути на кнопку відправити код та чекати відповіді від ШІ.

Також це поле повністю адаптивне та залежно від пристрою змінює свої розміри. Не менш цікавим функціоналом це те, що поле змінюється по висоті залежно від кількості коду, що вставив користувач, але при цьому не більше висоти його пристрою.

Нижче на рисунку 1.8. зображено поле вводу користувацького коду для аналізу та оптимізації від ШІ.

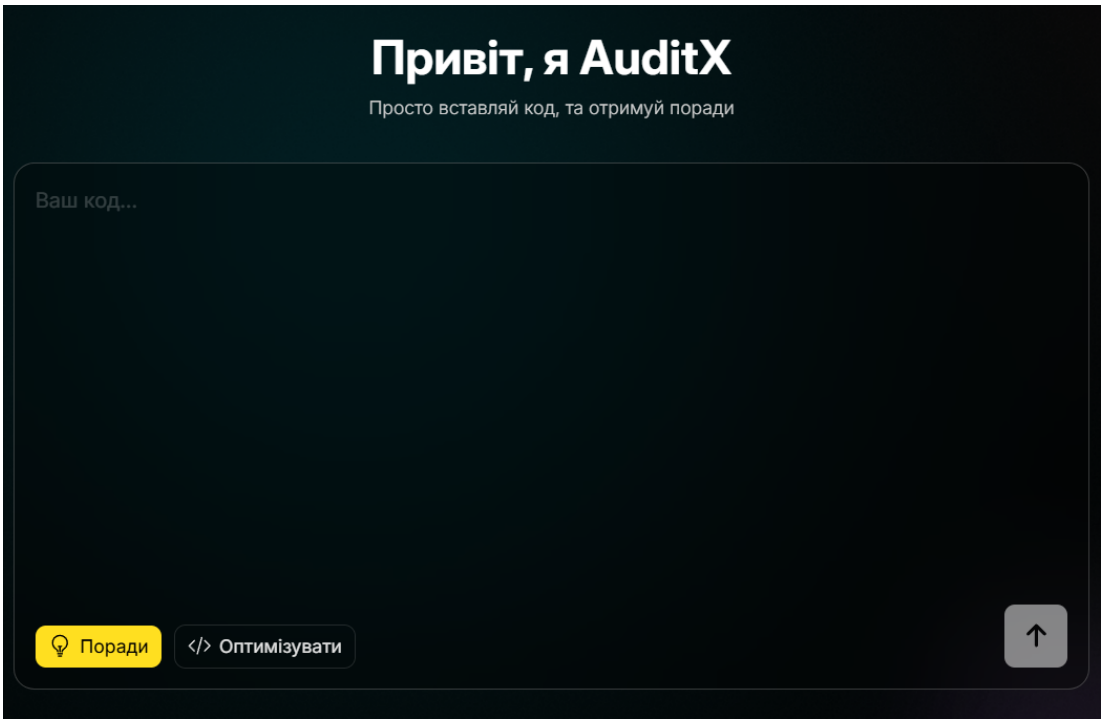


Рисунок 1.8. Дизайн поля воду користувацького коду

1.3.3 Проектування архітектури веб-застосунку

На Front-End частині я, як раніше вже зазначав використовую FSD (Feature-Sliced Design), яка являє собою архітектурну методологію для проектування Front-End-застосунків. Головна мета методології – зробити проект зрозумілим та структурованим, особливо за умов регулярної зміни вимог бізнесу. У методології завчасно є назви для всіх ймовірних слоїв, зроблено це для стандартизації та правильно побудови імпортів та експорті.

Розберемо структуру “AuditX” у зворотному від найвищого слою app до найвищого shared. Зроблено це щоб більш конкретно потім побачити цю структуру на рисунку структури “AuditX”.

Все починається зі шару app. Тут зосереджена глобальна конфігурація

проєкту: налаштування роутингу, провайдери контекстів, глобальні стилі, базові лейаути і компоненти, які обгортають увесь застосунок. Це наче основа, точка входу, звідки запускається і збирається весь Front-End.

Далі йде pages – це конкретні сторінки застосунку, кожна з яких відповідає за свій окремий екран або маршрут. Тут збираються усі необхідні віджети, фічі та сутності, щоб сформувати повноцінний інтерфейс для користувача. По суті, сторінка – це кінцева точка, яку бачить користувач.

Після pages йдуть widgets – це великі, готові компоненти інтерфейсу, які складаються з декількох features і entities. Вони служать своєрідними блоками інтерфейсу, які можна повторно використовувати, наприклад, панель навігації, інформаційна картка чи складна таблиця. Widgets допомагають об’єднати дрібніші функціональні частини в цілісні елементи.

Далі шар features – тут реалізовані конкретні функції або дії, які виконує користувач, наприклад: авторизація, пошук, фільтри, редагування профілю. Features працюють з сутностями, виконують бізнес-логіку і реалізують поведінку системи.

Наступний рівень – це entities. Тут описані ключові бізнес-сутності, типу користувач, продукт, замовлення. В цьому шарі знаходяться компоненти, моделі, хелпери, які безпосередньо працюють із цими об’єктами. Entities – це як фундамент, на якому базується вся логіка.

І в кінці – shared. Це універсальна бібліотека загальних компонентів, утиліт, хуків і констант, які використовуються у всіх шарах проєкту. Вони максимально ізольовані від бізнес-логіки і можуть застосовуватися де завгодно.

Важливо, що кожен зі слоїв може використовувати дочірні, тобто, наприклад, сторінки використовують widgets, ті в свою чергу – features і entities, а всі разом можуть звертатися до shared. Це створює чітку ієрархію залежностей і допомагає підтримувати код чистим і організованим.

Такою структурою керувалися при побудові Front-End частини проєкту “AuditX”, і це реально допомагає розробляти зрозумілий і масштабований код. Нижче на рисунку 1.8. файлову структуру Front-End частини проекту “AuditX”.

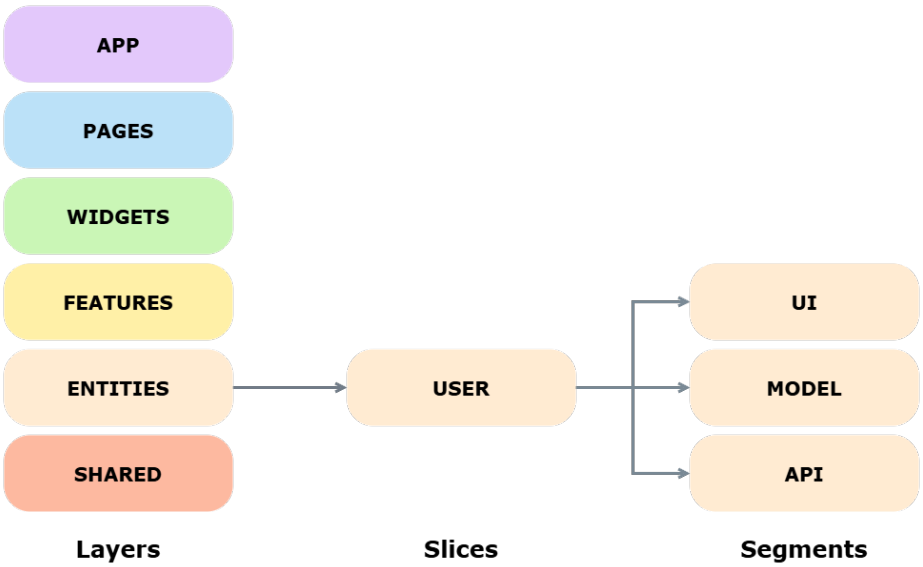


Рисунок 1.8. Структура папок та файлів Front-End частини “AuditX”

Стосовно Back-End частини, якихось особливих чи специфічних методології або архітектурних рішень не було застосовано. Зв’язано це з тим, що Back-End не має якоїсь тяжкої логіки, чи буде якось кардинально маштабуватися чи змінюватися, банальної архітектури вистачить з головою. Тому Back-End частина складається з базових слоїв.

У файлі index.ts налаштовується сервер. Тут створюється інстанс додатку, додаються всі потрібні мідлвари, налаштовуються базові шляхи до маршрутів, підключення до бази даних і запуск сервера. Це, можна сказати, точка входу на бекенді, де все стартує.

Далі йдуть routes. Тут описуються всі доступні ендпоінти, які обробляє сервер. Для зручності і порядку в мене маршрути поділені за функціональністю, наприклад /auth, /users, /operations. Вони просто перенаправляють запити до потрібного контролера. Це така собі мапа для бекенду.

У controllers вже відбувається обробка логіки запиту. Тобто сюди приходить запит з маршруту, тут дістаються параметри, відправляються потрібні дані в сервіс, обробляються помилки, і формується відповідь. Контролери не містять ніякої важкої логіки, вони просто керують потоком.

Middlewares використовується для перевірки токенів, обробки помилок, валідації і всяких таких речей, які повторюються. Це допомагає не дублювати один і той самий код по всьому проєкту. Вони стоять між запитом і контролером і

фільтрують чи обробляють його.

У services винесена бізнес-логіка. Вона не зав'язана на HTTP-запити і може використовуватись будь-де. Тут відбуваються запити до бази, всяка перевірка, фільтрація, створення записів. Цей шар дозволяє легко тестувати функціональність і не мішати її з контролером.

І останнє – shared. Це спільна папка, де зберігаются утиліти, константи, помічники, типи – все, що може знадобитись у будь-якому місці проєкту. Наприклад, функція генерації токена, універсальні обробники помилок, або якісь кастомні валідатори.

В результаті, архітектура бекенду хоч і доволі проста, але поділена на логічні шари, що робить код чистим, зрозумілим і зручним для підтримки.

Всі ці файли та папки знаходяться в батьківській папці src, так реалізовано як і на Front-End частині так і на Back-End. Звичайно окрім цих папок та файлів є багато допоміжних та конфінураційних, які знаходяться вже поза папкою src, наприклад такі як .env, .gitignore, .prettier, але оскільки це є вже прям базою для будь якого веб-застосунку і не тільки, не бачу великого сенсу про них розповідати. Нижче на рисунку 1.9. зображено файлову структуру Back-End частини проекту "AuditX".

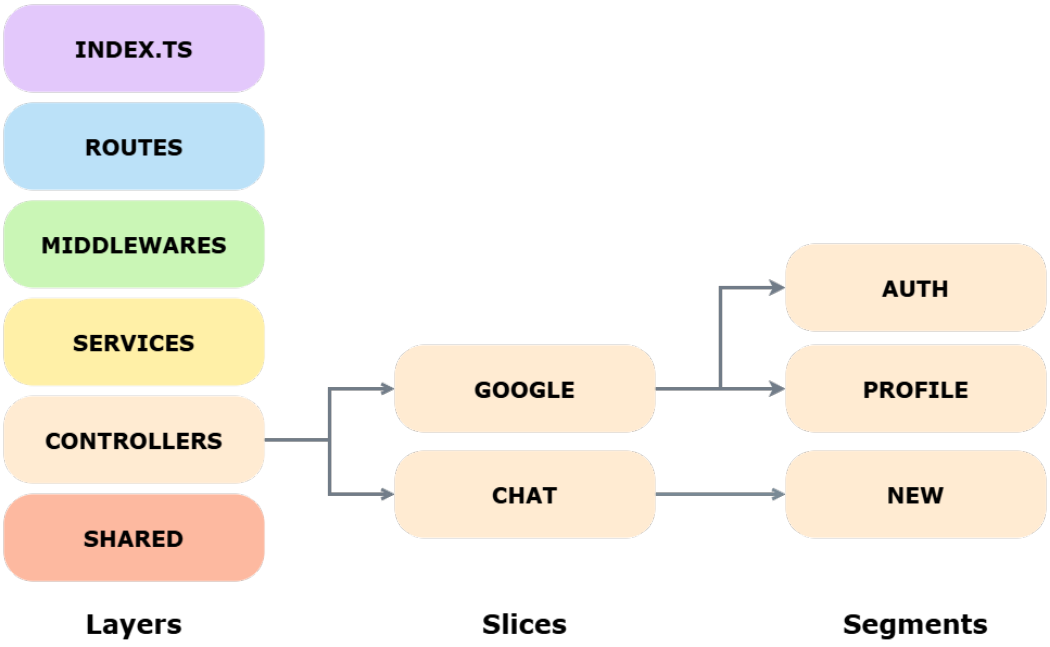


Рисунок 1.9. Структура папок та файлів Back-End частини "AuditX"

1.4 Реалізація веб-застосунку

1.4.1 Реалізація Front-End

Для розуміння реалізації веб застосунку, для початку потрібно розібрати поетапно кожну папку та більш менш важливі файли та їх наповнення. Звісно знати як там написані css-стилі розповідати великого сенсу немає, але ось як налаштований звичайний роутинг та приватний роутинг, як працює інтерсептор axios для перевірки та оновлення токенів, реалізація генерації UUID/GUID та HASH, відправка коду на сервер та може ще якісь технічні моменти слід оговорити [9][10]. Почнемо по порядку.

Поглянемо нижче, що у нас знаходиться в папці “./app”, рисунок 1.10.

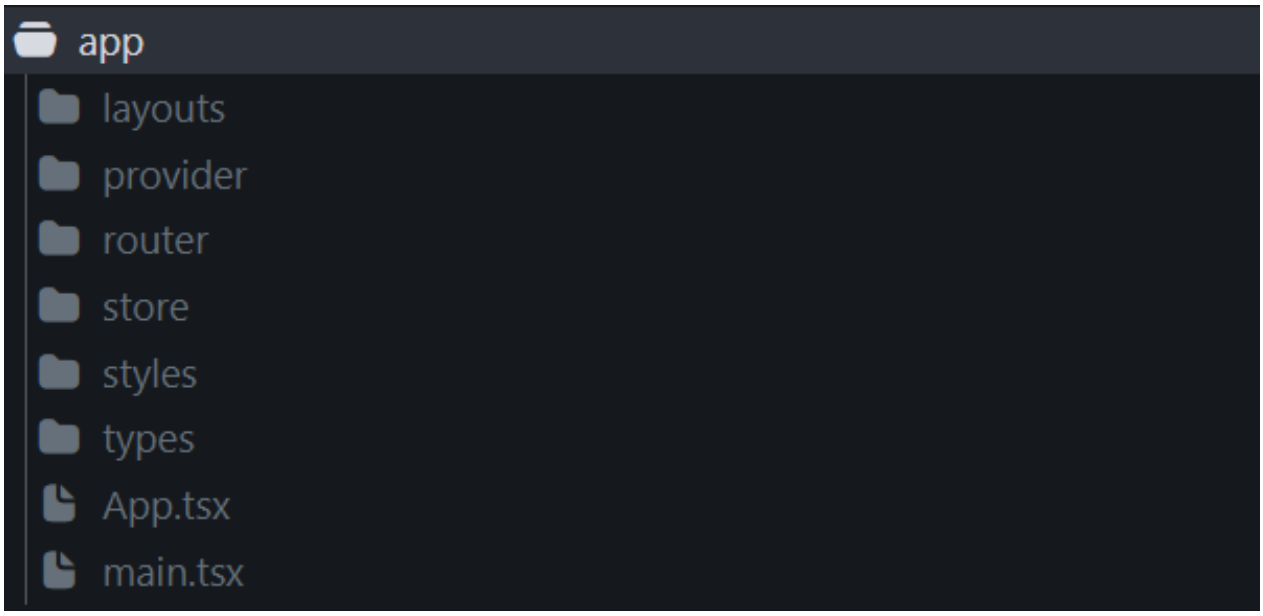


Рисунок 1.10. Структура папок та файлів а папці “./app”

Перша папка в папці “./app” – це “./layouts”, в ній зберігаються два лаяути, один для всього застосунку, а інший вже для сторінок для захищених сторінок, які зазвичай доступні вже пісня авторизації, зроблено це так, тому що до авторизації користувач не може відкрити навбар та перейти на захищені сторінки, а після авторизації йому це дозволяється, і доволі зручно мати в такому випадку глобальний лаяут і локальний. Розглянемо код глобального лаяуту:

```
import { Outlet } from 'react-router-dom'

import { Background } from 'shared/ui/Background'
import { Line } from 'shared/ui/Line'
import { PanelBlur } from 'shared/ui/Panels/Blur'
```

```
import { Header } from 'widgets/header/ui'

const Layout = () => {
  return (
    <div className="relative px-3 sm:px-4 md:px-10">
      <div  className="max-w-8xl  relative  mx-auto  my-0  grid  min-h-dvh  grid-cols-1
grid-rows-[auto_1fr] ">
        <Line position="left" />
        <Header />
        <Outlet />
        <Line position="right" />
      </div>
      <Background />
      <PanelBlur
        opacity={60}
        blur="none"
        className="md:rounded-t-4xl absolute inset-0 top-[var(--height-header)] -z-
10 rounded-t-2xl sm:rounded-t-3xl"
      />
    </div>
  )
}

export { Layout }
```

Тепер наглядно видно як працює глобальній лаяут “AuditX”. В глобальному лаяуті відразу є хедер, задній фон, блюр панель, все це в контейнері з відступами та має медіа запити для адаптації при зміні розміру екрану.

І тепер поглянемо на локальний лаяут, який використовується для налаштування висоти сторінки, відступів в середині головної секції та відповідає за показ navbar. Нижче код другого більш локального лаяуту.

```
import { Outlet } from 'react-router-dom'

import { Navbar } from 'widgets/navbar/ui'

const LayoutMain = () => {
  return (
    <div className="flex min-h-[calc(100dvh-var(--height-header))]">
      <Navbar />
      <div className="grow-1 p-8">
        <Outlet />
      </div>
    </div>
  )
}

export { LayoutMain }
```

Як бачимо перший div задає мінімальні розміри по висоті, потім висиавляжмо компонент Navbar про який ще обов’язково розглянемо пізніше та потім йде div в який вже завдяки компоненту Outlet буде динамічно вставлено наповнення.

Далі у нас йде папка “./provider” в середині неї файл “Provider.tsx” для налаштування глобальних провайдерів проєкту. У даному випадку це всього три провайдери, провайдер для “redux” – “Provider” , потім для “react-router-dom” – “BrowserRouter” та для “hot-toasts” – “Toaster”. Перший потрібен для збереження глобального стану, другий для роутінгу, третій для відображення тостів\повідомлень. Кожен із провайдерів приймає свої налаштування та може мати обов’язкові параметри. Саме завдяки провайдерам можна з будь якої частини застосунку викликати потрібні нам методи та не займатися props drilling, головне щоб викликати потрібні методи вже в середині провайдерів.

Наступна папка “./router”, і доволі логічно, що в середині налаштування роунтингу застосунку. Розберемо більш детально роунтингу. Нижче приведено код з файлу роутингу веб-застосунку який розробляється.

```
import { Route, Routes } from 'react-router-dom'

import { Layout, LayoutMain } from 'app/layouts'
import { AuthPage } from 'pages/auth'
import { HashPage } from 'pages/hash'
import { InfoPage } from 'pages/info'
import { MainPage } from 'pages/main'
import { ResultPage } from 'pages/result'
import { UuidPage } from 'pages/uuid'
import { configRouter } from 'shared/configs/router'
import { ProtectedRoute } from 'widgets/auth-protect'
import { SplashScreen } from 'widgets/splash'

export const AppRouter = () => {
  return (
    <Routes>
      <Route element={<SplashScreen />}>
        <Route element={<Layout />}>
          <Route path={configRouter.auth.path} element={<AuthPage />} />
          <Route element={<LayoutMain />}>
            <Route element={<ProtectedRoute />}>
              <Route path={configRouter.main.path} element={<MainPage />} />
              <Route
                path={configRouter.main.children.result.path}
                element={<ResultPage />}
              />
              <Route path={configRouter.uuid.path} element={<UuidPage />} />
              <Route path={configRouter.hash.path} element={<HashPage />} />
              <Route path={configRouter.info.path} element={<InfoPage />} />
            </Route>
          </Route>
        </Route>
      </Route>
    </Routes>
  )
}
```

Тепер розберемо більш детально, тут більше зрозуміло, що малося на увазі під глобальним та не глобальним лаяутом та захищеними та не захищеними сторінками. Компонент “Routes” та “Route” – це компоненти з бібліотеки “react-router-dom” перший обгортає всі роути, другий дає змогу створити новий роут. На справді такий спосіб створення роутів вже трохи застарілий, але в сили не великого веб-застосунку це більше ніж допустимо, тим паче, більше ніш 50% всі користувачів все ще використовують саме такий підхід.

Також тут одразу можна помітити які сторінки доступні до авторизації, це лише одна, інформаційна з кнопкою авторизації та які доступні після авторизації, це всі інші.

Далі йде папка “./store” в сторі у нас всього то одна сутність це user та в нього лише один action це записати отриманий код. До речі це доволі спірне рішення, але так було зроблено для того, щоб не показувати якісь дані в рядку пошуку як параметри та залишити посилання чистими. Про це також буде вже пізніше коли дійдемо до роботи ШІ.

Папка “./styles” зупинятися немає сенсу, бо це просто глобальні стилі, підключення “Tailwind CSS” та його налаштування.

Папка “./types” також, просто глобальні типи, не забуваємо, що весь проєкт написаний на “Typescript”.

Залишилося два файли це “App.tsx” в якому беремо компонент провайдерів та обгортаємо над налаштованим роутінгом. Та “main.tsx”, що є точкою входу в застосунок, найголовніший файл, ось нижче його код.

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'

import App from 'app/App'
import 'app/styles'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

Все, що тут робиться доволі просто, але без цього в браузері нічого би не відмалювалось. Береться функція “createRoot” в неї пережається компонент який отримали з DOM дерева завдяки “document.getElementById('root')” знак оклику “!”

каже, що компонент 100% буде отримано, і після того як функція “createRoot” поверне результат, викликаємо метод “render” та передаємо “JSX” розмітку в якій зазначаємо, щоб застосунок працював к режимі “StrictMode” та передаємо раніше обговорений компонент “App.tsx”.

Наступним важливим елементом у структурі застосунку є налаштування та використання axios interceptor, який відповідає за перевірку валідності токену, його автоматичне оновлення при потребі та забезпечення коректної авторизації всіх запитів до серверу. У даному випадку, використовується axios – популярна бібліотека для роботи з HTTP-запитами у JavaScript/TypeScript, яку обгортаємо у власну конфігурацію з авторизацією. Розглянемо реалізацію цього механізму.

Axios дозволяє створити так звані "interceptors" – перехоплювачі запитів або відповідей, які дозволяють виконувати додаткову логіку до того, як запит буде надіслано на сервер або відповідь буде оброблена. Це ідеально підходить для автоматичного додавання токенів авторизації в заголовки, перевірки чи не протух токен, його автоматичного оновлення при необхідності, обробки глобальних помилок, як-от 401 Unauthorized.

У даному проєкті це реалізовано у вигляді окремого конфігурованого екземпляру axios під назвою axiosAuthorized. Цей екземпляр одразу має вказаний базовий URL до API, а також прапорець withCredentials: true, щоб браузер передавав куки при кожному запиті.

Код реалізації перехоплювача запитів:

```
import axios from 'axios'
import { checkIsTokenExpired } from 'shared/lib/jwt'
import { getCookie } from 'shared/lib/utils'
import { refreshGoogleIdToken } from './refreshGoogleIdToken'

const axiosAuthorized = axios.create({
  baseURL: import.meta.env.VITE_SERVER_API_URL + '/api',
  withCredentials: true,
})

axiosAuthorized.interceptors.request.use(
  async (config) => {
    try {
      const idToken = getCookie('ID_TOKEN')

      if (!idToken) {
        throw new Error('Id token not defined')
      }
```

```
        if (checkIsTokenExpired(idToken)) {
          await refreshGoogleIdToken()
        }

        return config
      } catch (error) {
        console.error('Request cancelled:', error)
        return Promise.reject(error)
      }
    },
    (error) => {
      return Promise.reject(error)
    },
  )

  export { axiosAuthorized }
```

Перед кожним запитом перевіряється, чи існує ID токен. Якщо його немає або він протермінований (визначається за допомогою утиліти checkIsTokenExpired), запускається функція refreshGoogleIdToken, яка автоматично оновлює токен через бекенд.

Такий підхід дозволяє значно спростити логіку авторизованих запитів. Усі перевірки винесені в єдине місце, і компоненти, які відправляють запити, не потребують повторення цієї логіки кожного разу.

Тепер перейдемо до самого функціоналу, спочатку розберемо роботу генерації UUID/GUID. Для генерації використано бібліотеку “uuid” та просто викликаю функцію наприклад для генерації UUID версії 7 – “uuidv7()”, а вже в відповіді функція повертає рядок UUID. Саме тут якоїсь кастомної логіки з моє сторони написано не було. Ну звичайно не враховуючи саму сторінку і тд.

А ось для генерації хешу на основі вхідних даних від користувача, мені вже довелось написати свою кастомну функцію. Розглянемо мою кастомну функцію для генерації хешу. Звісно, технологія існує давно, тому велосипед не вигадується, але сам процес обгортання в зручну функцію та правильна робота з Web Crypto API – це важлива робота. Ця функція дозволяє обчислити хеш від будь-якого переданого рядка за допомогою алгоритму SHA-256. Ось вона:

```
const generateHash = async (value: string) => {
  const encoder = new TextEncoder()
  const data = encoder.encode(value)

  const hashBuffer = await crypto.subtle.digest('SHA-256', data)
  const hashArray = Array.from(new Uint8Array(hashBuffer))

  const hashHex = hashArray.map((b) => b.toString(16).padStart(2, '0')).join('')
  return hashHex
```

```
}

export { generateHash }
```

Спочатку створюється енкодер (TextEncoder()), який перетворює звичайний текстовий рядок у формат, придатний для хешування – у даному випадку, у Uint8Array. Це необхідно, оскільки функція crypto.subtle.digest працює саме з байтами, а не з рядками. crypto.subtle.digest Тут використовується вбудоване API від браузера – Web Crypto API. Воно дозволяє обчислювати криптографічні хеші без підключення сторонніх бібліотек, і що важливо – воно працює асинхронно.

У даному випадку було використано SHA-256, який є стандартом і доволі стійким алгоритмом. Він широко використовується у вебі – наприклад, для збереження паролів або перевірки цілісності даних. Преобразування хешу в читабельний вигляд.

Результатом роботи digest є ArrayBuffer, який доведеться спочатку конвертувати в Uint8Array, а вже потім у масив звичайних чисел (Array.from). Далі відбувається найцікавіше: кожен байт переводиться в шістнадцятковий формат (hex), де .padStart(2, '0') потрібен для того, щоб значення завжди мали два символи. Наприклад, 7 стане 07. Збирання результату Зібравши всі hex-значення в один рядок (.join('')), отримаємо саме той хеш, який вже можемо повертати або зберігати, або передавати кудись далі.

Така функція дозволяє швидко перевірити унікальність даних, зробити fingerprint певного значення чи створити власну реалізацію збереження паролів (звісно, з додатковими заходами безпеки). По-друге, вона асинхронна – тобто не блокує головний потік, що важливо для сучасних веб-застосунків.

Тепер про відправку коду на сервер, для обробки ШІ, на справді о скільки тут великої логікі для Front-End частини не має, то й велокої кількості коду тут не було. Фактично є асинхронна функція для відправки коду:

```
import { axiosAuthorized } from 'shared/api/auth'

import { SendCodeResponse } from './types'

interface SendCode {
  code: string
  options: {
    optimize: boolean
    explain: boolean
```

```
  }
}

const sendCode = async ({
  code,
  options,
}: SendCode): Promise<SendCodeResponse> => {
  try {
    const response = await axiosAuthorized.post<SendCodeResponse>('chat', {
      code,
      options,
    })
    return response.data
  } catch (error) {
    console.error(error)
    throw new Error('Error send code')
  }
}

export { sendCode, type SendCode }
```

Функція приймає код користувача та параметри, потім відправляє дані на бєк. Але раніше вже були згадки про “Redux” і саме тут він використовується. Таким чином, був створений слайс користувача, який виглядає наступним чином:

```
import { createSlice } from '@reduxjs/toolkit'
import { StatusType } from 'shared/types'

import { selectCode } from './selectors/selectCode'
import { selectCodeStatus } from './selectors/selectCodeStatus'
import { sendCodeThunk } from './thunks/sendCodeThunk'
import { UserState } from './type'

const initialState: UserState = {
  user: {
    code: null,
  },
  codeStatus: StatusType.INIT,
}

const slice = createSlice({
  name: 'user',
  initialState,
  selectors: {
    selectCode,
    selectCodeStatus,
  },
  reducers: {},
  extraReducers: (builder) =>
    builder
      .addCase(sendCodeThunk.pending, (state) => {
        state.codeStatus = StatusType.LOADING
      })
      .addCase(sendCodeThunk.fulfilled, (state, actions) => {
        state.user.code = actions.payload.data
        state.codeStatus = StatusType.SUCCESS
      })
      .addCase(sendCodeThunk.rejected, (state) => {
        state.user.code = null
        state.codeStatus = StatusType.ERROR
      }),
```

```
})

export const userReducer = slice.reducer
export const userSelectors = slice.selectors
```

Це саме специфічний код для налаштування слайсу в “Redux Toolkit”. Якщо не вдаватися прямо в подробиці як це все працює, то потрібно згадати, що раніше згадувалося про “Redux”, він потрібен для того, щоб десь зберігати якісь дані та мати до них доступ з будь якої частинки застосунку. Тепер дивимось на той код, що вище, бачимо таку зміну “initialState” – це об’єкт в якому збережені самі перші, базові дані даного глобального стейту, доступ до якого можна отримати з будь якої частинки застосунку, ну звісно при правильному підключені редаксу.

Далі бачимо функцію “createSlice” яку викликаємо з першим параметром об’єктом в якому вказуємо назву слайсу “user”, даний стейт “initialState”, потім йдуть “selectors” якраз завдяки цим функціям, є можливість дістати дані, просто викликав їх, передавши стейт. Потім йдуть “reducers, завдяки ним, є можливість не тільки отримувати дані зі стейту а ще й змінювати, ось якраз для цього і потрібні функції редюсери. Фактично функція редюсер це чиста функція, яка завжди повертає стейт та приймає payload (потрібні дані для зміни стейту), до речі так працюють функції редюсери в “Redux Toolkit”, а ось в старому\звичайному “Redux” концепція трохи була інша, приймалися екшени і залежно від типу викликалась потрібна функція для зміни стейту, хоча казати, що прямо концепція інша то ні, просто в новому “Redux Toolkit” це все від нас приховано за синтаксичним цукром.

І тепер про “extraReducers” на справді ті ж самі звичайні редюсери, але ще з більшою обгорткою над собою. “extraReducers” дозволяє зручно змінювати стани при асинхронних діях, наприклад, зробив запит до бєку, статус на LOADING, отримав відповідь – SUCCESS, стався error – ERROR. Саме це й видно в прикладі з sendCodeThunk.

Докладніше розглянемо, що тут відбувається. Коли викликажться sendCodeThunk, це асинхронна дія (thunk), ось код sendCodeThunk:

```
import { isAxiosError } from 'axios'
import { SendCode, sendCode } from 'entities/user/api/sendCode'
import { createAppAsyncThunk } from 'shared/lib/redux'
import { SendCodeThunkResponse } from '../type'
```

```
const sendCodeThunk = createAppAsyncThunk<SendCodeThunkResponse, SendCode>(
  'send-code',
  async ({ code, options }, { rejectWithValue }) => {
    try {
      const response = await sendCode({ code, options })
      return { ...response, data: JSON.parse(response.data) }
    } catch (e: unknown) {
      if (isAxiosError(e)) {
        return rejectWithValue(e.message)
      }
      const knowError = e as IResponse<null>
      return rejectWithValue(knowError.message)
    }
  },
)

export { sendCodeThunk }
```

У цей момент змінюється статус на LOADING. Це дає змогу, наприклад, показати на інтерфейсі лоадер, щоб користувач розумів, що щось відбувається у фоновому режимі. .addCase(sendCodeThunk.fulfilled) – цей кейс викликається коли запит пройшов успішно. В такому випадку береться payload, який нам повернув thunk (по суті відповідь з бекенду), і зберігаємо data в state.user.code. Також, звичайно, міняємо статус на SUCCESS, що може бути тригером для показу якогось повідомлення, переходу на іншу сторінку або ще якоїсь дії .addCase(sendCodeThunk.rejected) – тут усе очевидно: сталася помилка, тому ставимо code назад у null, а статус на ERROR.

Це може бути корисно, щоб вивести помилку на екран, або запропонувати повторити дію. Використання extraReducers дозволяє дуже зручно і централізовано контролювати стани асинхронних процесів у застосунку. Це особливо корисно для таких речей як запити на авторизацію, підтвердження, отримання списку товарів, створення замовлень і так далі – тобто всюди, де є запити до сервера.

Це частина основної логіки Front-End частини, звісно можна розповісти ще багато про що. В даному випадку навіть не про всю основну логіку була можливість розповісти, але для повноти картини доцільним буде поглянути на рисунок 1.11, де схематично зображено головне флоу користувача при взаємодії в ШІ асистентом. Цей рисунок ілюструє послідовність дій, яку виконує користувач, починаючи від ініціації запиту й завершуючи отриманням відповіді. Він допомагає краще зрозуміти, як функціонують ключові компоненти інтерфейсу та як

забезпечується злагоджена взаємодія між фронтендом і бекендом.

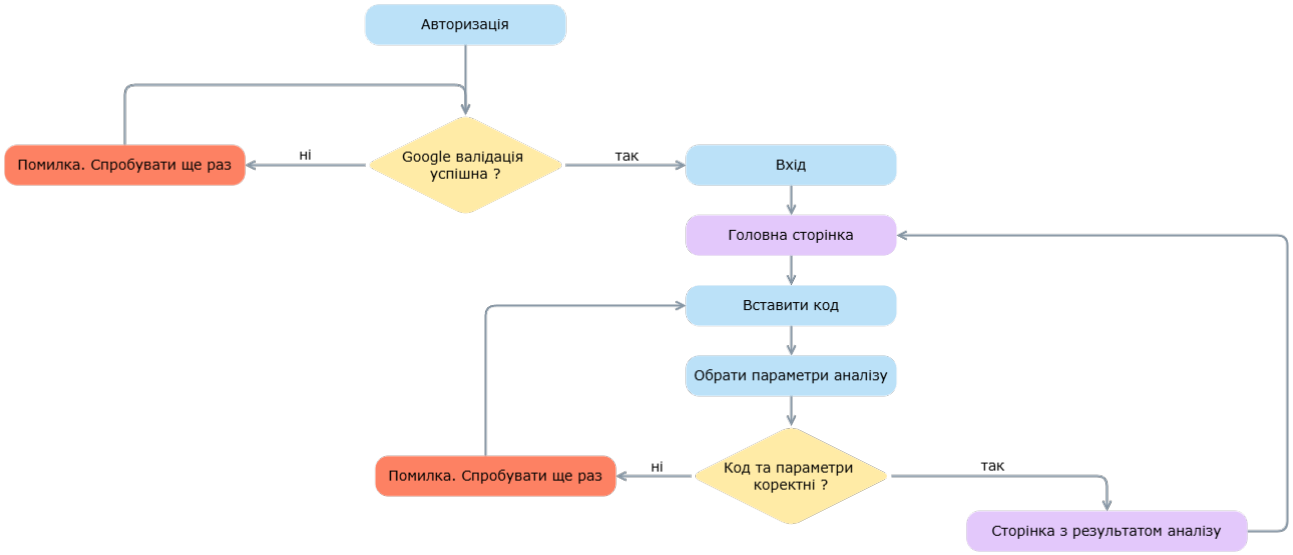


Рисунок 1.11. User flow при взаємодії з ШІ

1.4.2 Реалізація Back-End

Тепер поговоримо про розробку Back-End частини веб-застосунок “AuditX”. Насправді оскільки Back-End не планувався великим і не передбачує збереження клієнтських даних, то і сам по собі він вийшов невеликий, але все ж таки розберемо основний код та основні фічі.

Оскільки генерація UUID/GUID відбувається на клієнтській частині, то з беком це ніяк не зв’язано, тож одразу перейдемо до третьої фічі, самої цікавої, це роз’яснення та оптимізація клієнтського коду.

Як вже згадувалося раніше, для аналізу коду використовується “ChatGPT API”. Тому тут є про що розповісти, почнемо спочатку.

Розберемо головний файл серверу, де він ініціалізується, базово налаштовується, нижче код файлу “index.ts”:

```
import cookieParser from 'cookie-parser'
import cors from 'cors'
import express from 'express'
import nocache from 'nocache'

import { controllers } from './controllers'
import { env } from './shared/utils/env'

const start = () => {
  const app = express()
  const port = env.SERVER_PORT

  app.use(
```

```
      cors({
        origin: env.CLIENT_URL,
        credentials: true,
      }),
    )
    app.use(express.json())
    app.use(cookieParser())
    app.use(express.urlencoded({ extended: true }))

    app.use('/api', controllers)

    app.listen(port, () =>
      console.log(`Server running at http://localhost:${port}`),
    )
  }

  start()
```

В середині функції “start” створюємо екземпляр серверу під назвою “app”, далі створюємо зміну “port” в яку записуємо порт на якому буде працювати сервер. Далі неодноразово викликаємо функції “use” у екземпляра сервера.

Перший виклик з налаштуваннями “CORS”. Cors – це механізм безпеки сучасних браузерів, який дозволяє веб-сторінкам, що завантажуються з одного домену, взаємодіяти з ресурсами, розміщеними на іншому домені. хоча в подальшому не виключено, що клієнтська частина та серверна будуть на одному домені, все ж таки важливо дозволити завчасно просто вказати в файлі “ENV” домен, якому сервер одразу зможе без проблем довіряти.

Другий виклик “use” додає серверу змогу самому парсити стандартний формат представлення та обміну даними, який базується на синтаксисі об'єктів JavaScript – “JSON”.

Наступний виклик “use” дозволяє серверу зручно парсити “COOKIES”, “COOKIE” це невеликий набір даних, що надсилається веб-сервером і зберігається на комп'ютері користувача без змін та будь-якої обробки. Веб-клієнт щоразу при зверненні до відповідного сайту пересилає ці дані веб-серверу у складі HTTP-запиту. Зазвичай куки застосовують для: автентифікації користувача.

Наступним викликом use додаємо обробку URL-кодованих даних за допомогою ”express.urlencoded({ extended: true })”. Це розширення дозволяє у запитах обробляти складніші структури даних, наприклад, вкладені об'єкти, масиви, що може бути необхідним при побудові форм, які передають на сервер більше ніж просто плоский набір пар "ключ-значення". Така функціональність

особливо важлива у веб-застосунках, де передача складних структур у HTTP-запитах – цілком звична річ. Об'єктивно кажучи, нехтування такою можливістю могло б обмежити функціональність клієнтської частини додатку, а отже, й знизити гнучкість усього проєкту загалом.

Після того як налаштували всі необхідні middleware-функцій, підключаємо маршрути через шлях “/api” до контролерів, які були імпортовані з окремого модуля controllers. Це структурне рішення дозволяє централізовано управляти всіма запитами, які надходять на сервер та легко масштабувати додаток у майбутньому.

І нарешті, викликаємо метод listen, який запускає сервер на вказаному порту. У момент запуску на консоль виводиться повідомлення з адресою, за якою доступний даний сервер – це не лише зручно для розробника під час тестування, але й слугує підтвердженням того, що серверна частина додатку була успішно ініціалізована і готова до обробки вхідних запитів. Функція start – це точка входу у серверний застосунок, яка виконує ініціалізацію всіх основних компонентів.

Розберемо найцікавіший роут, це роут на аналіз та оптимізацію клієнтського коду.

```
import { Router } from 'express'

import { chatNewController } from '../../controllers/chat'

const router = Router()

router.post('/', chatNewController)

export { router as chatRoute }
```

По роуту “/api/chat” викликається функція контролер “chatNewController” в якому вже і відбувається виклик функції по взаємодії з “ChatGPT API”. Нижче код контролера “chatNewController”:

```
type ChatNewControllerRequestData = Request<
  {},
  {},
  {
    code: string
    options: {
      optimize: boolean
      explain: boolean
    }
  }
>
```

```
const chatNewController = async (
  req: ChatNewControllerRequestData,
  res: Response,
) => {
  try {
    const { code, options } = req.body

    const openai = new OpenAIClientService()

    const data = await openai.sendCodePrompt(code, options)

    response(res, {
      status: ResponseStatus.OK,
      message: 'Success response openai chat',
      data: data.choices[0].message.content,
      success: true,
    })
  } catch (error) {
    console.error(error)
    response(res, {
      status: ResponseStatus.INTERNAL_SERVER_ERROR,
      message: 'Login error',
      success: false,
      error,
      error_type: ResponseErrorType.CRITICAL,
    })
  }
}

export { chatNewController }
```

З фронта приходить об’єкт з кодом користувача та налаштуванням для аналізу, передаємо його в функцію “sendCodePrompt” код якої розберемо трохи пізніше. Чекаємо відповідь від ШІ та формуємо об’єкт відповіді, в якому повертаємо “status”, “message”, “data”, “success”, це все базові поля, вони повертаються у будь якому випадку, змінюється тільки їх значення.

Розберемо код класу “OpenAIClientService”

```
static API_KEY = env.OPEN_AI_API_KEY_MAIN

  private openAIClient

  constructor() {
    this.openAIClient = new OpenAI({
      apiKey: OpenAIClientService.API_KEY,
    })
  }

  private async sendPrompt(userMessage: string, systemPrompt?: string) {
    return await this.openAIClient.chat.completions.create({
      model: 'gpt-4o-mini',
      messages: systemPrompt
        ? [
            { role: 'system', content: systemPrompt },
            { role: 'user', content: userMessage },
          ]
        : [{ role: 'user', content: userMessage }],
```

```
        })
    }
```

в конструкторі створюємо екземпляр класу “OpenAI”, передаємо “apiKey”. Нижче створюємо приватну функцію “sendPrompt” в якій приймаємо “userMessage” та “systemPrompt”. В тілі функції викликаємо і одразу повертаємо проміс створення запиту до “ChatGPT API” передаємо “systemPrompt” та “userMessage”,. А тепер подивимось на код публічної функції яку і викликали в контролері:

```
    async sendCodePrompt(
        code: string,
        options: {
          optimize: boolean
          explain: boolean
        },
      ) {
        const messageSystem = `тут базовий меседж для того щоб чат зрозумів контекст
    своєї роботи`.trim()

        const messageUser = `
            # Вхідний код від юзера:
            \`\`\`
            ${code}
            \`\`\`

            # Опції:
            - optimize: ${options.optimize}
            - explain: ${options.explain}
            `.trim()

        return await this.sendPrompt(messageUser, messageSystem)
    }
```

Зупинимось на “systemPrompt”. Це системне повідомлення, яке передається в API першим і виконує важливу роль – воно визначає, як саме має поводитись модель. Тобто, по суті, встановлюється базовий контекст або інструкція для “ChatGPT”, щоб він розумів свою роль у цій конкретній задачі. Наприклад, у даному випадку можна вказати, що він повинен виступати як аналізатор коду, давати пояснення, шукати можливості для оптимізації, ігнорувати зайвий текст, відповідати структуровано тощо. Без такого повідомлення модель може інтерпретувати запит занадто загально або відповідати не в тому стилі, який нам потрібен. Тому “systemPrompt” – це ключ до стабільної та передбачуваної поведінки “ChatGPT” в рамках даного застосунку.

Що таке “messageUser” інтуїтивно зрозуміло, це той самий код користувача з налаштуванням, в якому просто додано контекст для чату, щоб було зрозуміло, де які дані.

В кінці просто повертаємо проміс виклику приватної функції, тіло якої обговорювали вже раніше.

Ось така основна логіка Back-End частини, звісно можна розповісти про ще багато що, реалізацію реєстрації, оновлення токенів, специфіку налаштування сервера.

1.5 Мануальне тестування

Виконаємо ручне тестування створеного веб-застосунку, перевіримо основні функції ключових веб-сторінок.

Ручне тестування є процесом перевірки програмного забезпечення вручну для виявлення різноманітних дефектів без використання автоматизованих інструментів.

Цей метод тестування є основним та критично важливим етапом життєвого циклу будь-якого програмного продукту, він дозволяє виявити помилки, які можуть бути пропущені автоматизованими тестами, а також помилки, що впливають на користувацький досвід. Також ручне тестування допомагає перевірити відповідність інтерфейсу очікуванням користувача та оцінити зручність взаємодії з веб-застосунком. Його результати дозволяють своєчасно внести необхідні зміни перед запуском продукту.

1.5.1 Функціональний огляд сторінки авторизації “Auth”

Перша сторінка на яку попадає не авторизований користувач, це сторінка авторизації та по сумісності сторінка показу функціонала ШІ. Поглянемо на рисунок 1.12 нижче. На цій сторінці користувач має можливість ввести свої облікові дані для отримання доступу до захищених розділів веб-застосунку.

Крім того, інтерфейс одразу демонструє базову функціональність ШІ, що дозволяє користувачу ознайомитися з можливостями асистента ще до повної авторизації.

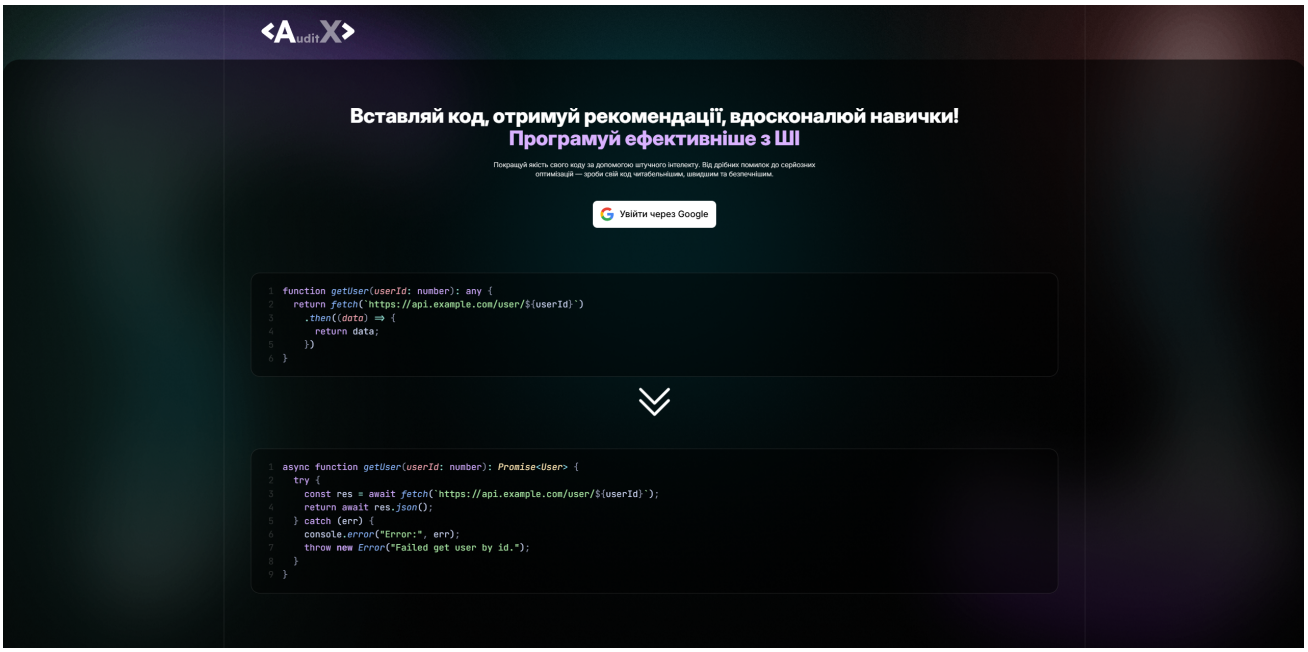


Рисунок 1.12. Сторінка авторизації “Auth”

На сторінці присутній хедер з логотипом “AuditX”, заголовок з описом головної функції, є змога увійти через “Google”, також присутній приклад роботи ШІ.

Сторінка повністю адаптована під різні пристрої. Було вирішено не перевантажувати сторінку авторизації, тому вона на стільки мінімалістична. Функції генерації UUID та HASH було вирішено на сторінку авторизації поки не виносити, можливо в майбутньому буде визначено, як зробити більш гарно та висвисти всі функції.

Після кліку на кнопку “Увійти через Google” користувача перенаправляє на сторінку вибору аккаунту, вибираємо потрібний аккаунт, або створюємо його через сервіси Google, потім запитуємо доступ до потрібних функцій, та після успішної авторизації відбувається перенаправлення на захищену сторінку, у даному випадку це головна сторінка.

1.5.2 Функціональний огляд головної сторінки “Home”

Одразу після реєстрації користувач потрапляє на головну сторінку, по сумісності сторінку відправки запиту до ШІ для аналізу та оптимізації користувацького коду. Поглянемо на рисунок 1.13 сторінки “Home” нижче. Сторінка має інтуїтивно зрозумілий інтерфейс, що дозволяє швидко сформувати

запит і отримати відповідь у зручному форматі.

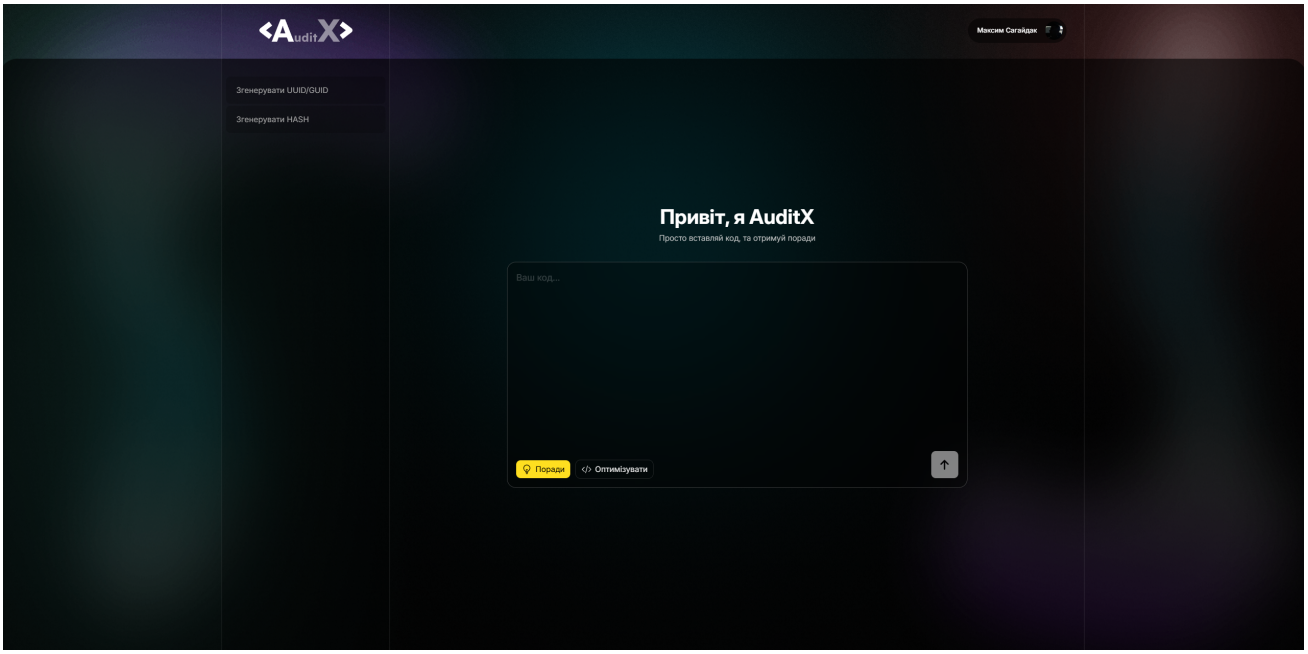


Рисунок 1.13. Головна сторінка “Home”

Після реєстрації в хедер додається панель користувача по кліку на яку відкривається список з посиланням на сторінку “Info” про яку поговоримо пізніше та кнопка по кліку на яку відбувається вихід з аккаунту.

Ліворуч бачимо navbar з двома пунктами “Генерація UUID/GUID” та “Генерація HASH”, по кліку подаємо на відповідну сторінку, також розберемо пізніше.

По середині сторінки в блоці main бачимо форму для відправки коду на аналіз та оптимізацію. Є змога вибрати додатковий пункт “Оптимізувати”, тоді ШІ зробить аналіз та дасть свій варіант коду, який буде оптимізованою версією користувацького коду. Ну і праворуч знизу кнопка для відправки коду, вона буде не активною якщо кількість символів в коді буде менше за дозволену.

Після відправки ШІ аналізує код та дає відповідь, після відправляє користувача на сторінку результату, яку розберемо нижче.

1.5.3 Функціональний огляд сторінки “Result”

Після успішного аналізу та оптимізації коду на сторінці “Home”, користувач потрапляє на сторінку “Result”. Нижче рисунок 1.14. сторінки “Result”. На цій сторінці користувач бачить результати опрацювання запиту, включаючи

оптимізований код, рекомендації та можливість зберегти або повторно відправити запит.

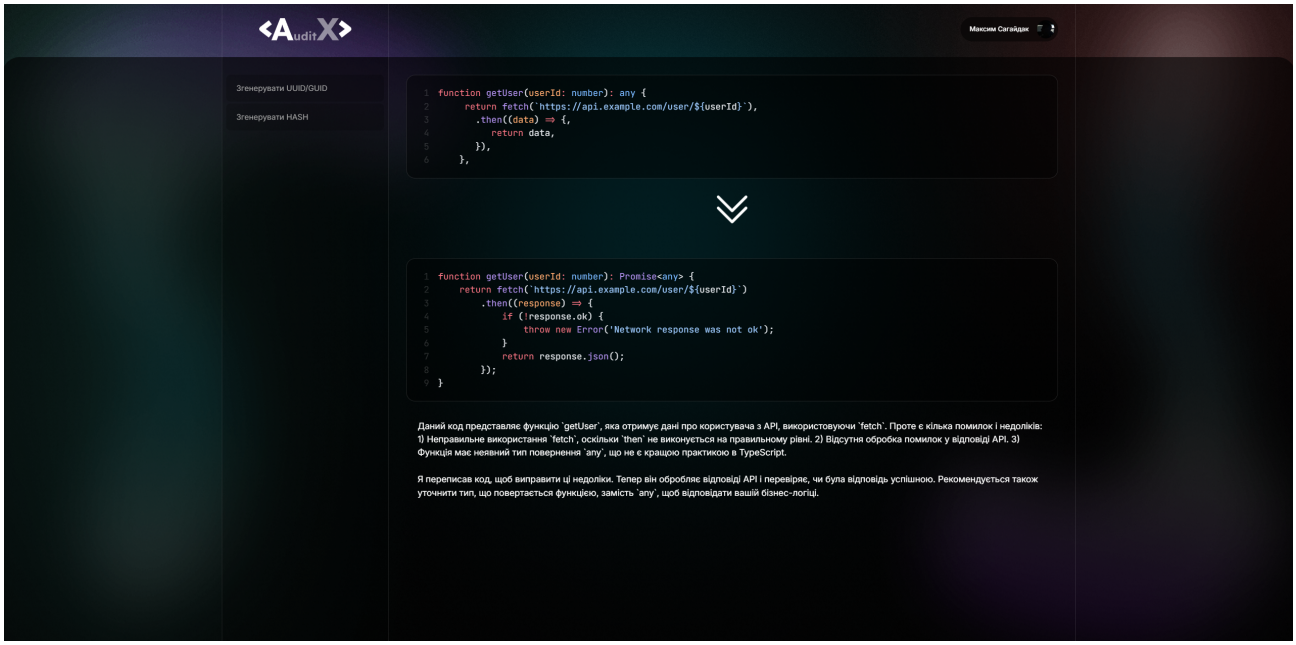


Рисунок 1.14. Сторінка результату аналізу та оптимізації коду “Result”

Перший блок коду це код який користувач відправив на аналіз, а другий це вже відповідь від ШІ, оптимізований та аналізований код. В низу опис того як працює цей код, зручно якщо не розумієш чужого коду.

1.5.4 Функціональний огляд сторінки “UUID”

Потрапити на сторінку генерації UUID/GUID можна з будь якої сторінки, але обов’язково бути авторизованим. Поглянемо на рисунок 1.15, де зображено сторінку для генерації UUID/GUID.

Ця сторінка призначена для швидкої генерації унікальних ідентифікаторів, які можуть бути використані в різних програмних сценаріях, зокрема для маркування об’єктів або забезпечення унікальності записів у базі даних. Інтерфейс сторінки мінімалістичний, що дозволяє зосередитися виключно на головній функції — генерації UUID. Користувач може обрати тип ідентифікатора, а також кількість необхідних значень. Згенеровані UUID миттєво відображаються у спеціальному полі з можливістю копіювання. Також реалізовано механізм історії — користувач може переглянути раніше згенеровані ідентифікатори. Це особливо корисно під час роботи з великими масивами даних або в процесі тестування.

Завдяки доступності цієї сторінки з будь-якого розділу застосунку, користувач завжди має швидкий доступ до потрібного інструменту.

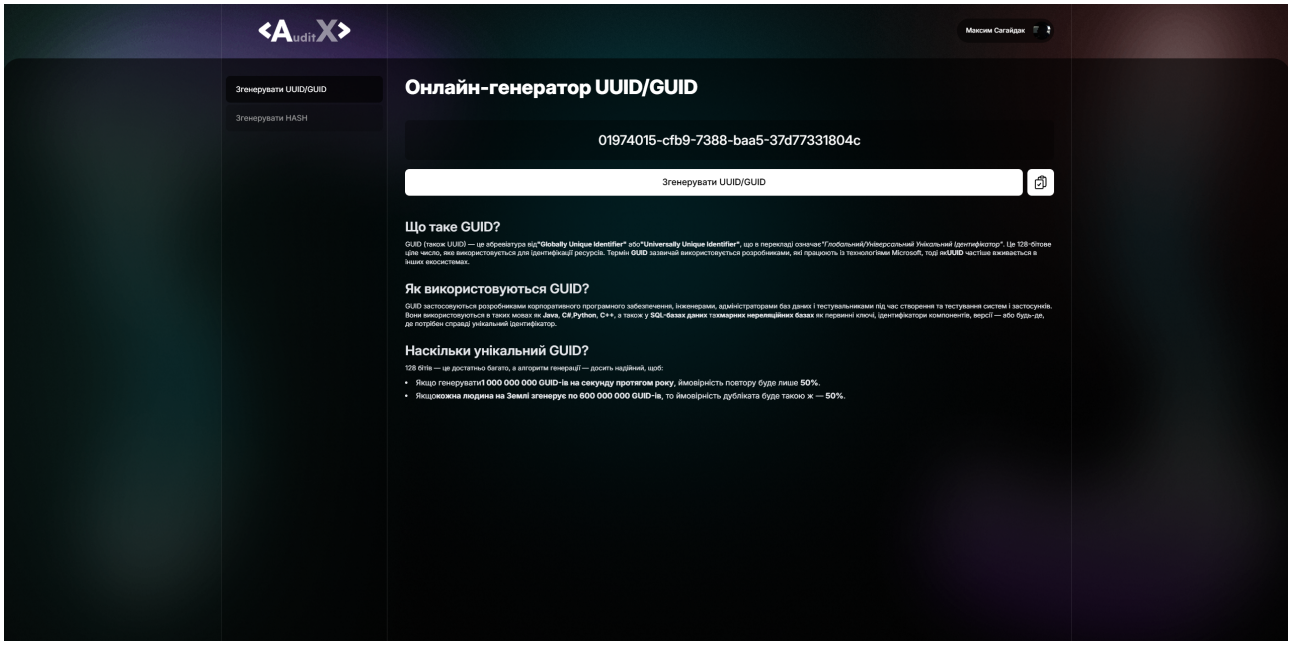


Рисунок 1.15. Сторінка генерації UUID/GUID “UUID”

На цій сторінці бачимо:

- кнопку для генерації GUID;
- кнопку для копіювання GUID;
- опис того, що таке GUID та як його використовують.

Всі кнопки працюють як потрібно.

1.5.5 Функціональний огляд сторінки “HASH”

Ця сторінка дуже схожа на сторінку генерації UUID/GUID. Також, щоб, потрапити на сторінку генерації HASH можна з будь якої сторінки, але також обов’язково бути авторизованим та достатньо натиснути на вже другий пункт navbar. Поглянемо на рисунок 1.16, де зображено сторінку для генерації HASH.

Сторінка генерації HASH надає можливість створити хеш-значення для введених користувачем даних за допомогою популярних алгоритмів, таких як SHA-256. Користувач вводить текст у відповідне поле, і система миттєво генерує хеш. Цей функціонал є корисним для перевірки цілісності даних або зберігання конфіденційної інформації в захищеному вигляді. Інтерфейс зручний та зрозумілий, що забезпечує швидку роботу навіть для новачків. Історія

згенерованих хешів зберігається локально в межах сесії, що дозволяє при потребі повернутися до попередніх результатів.

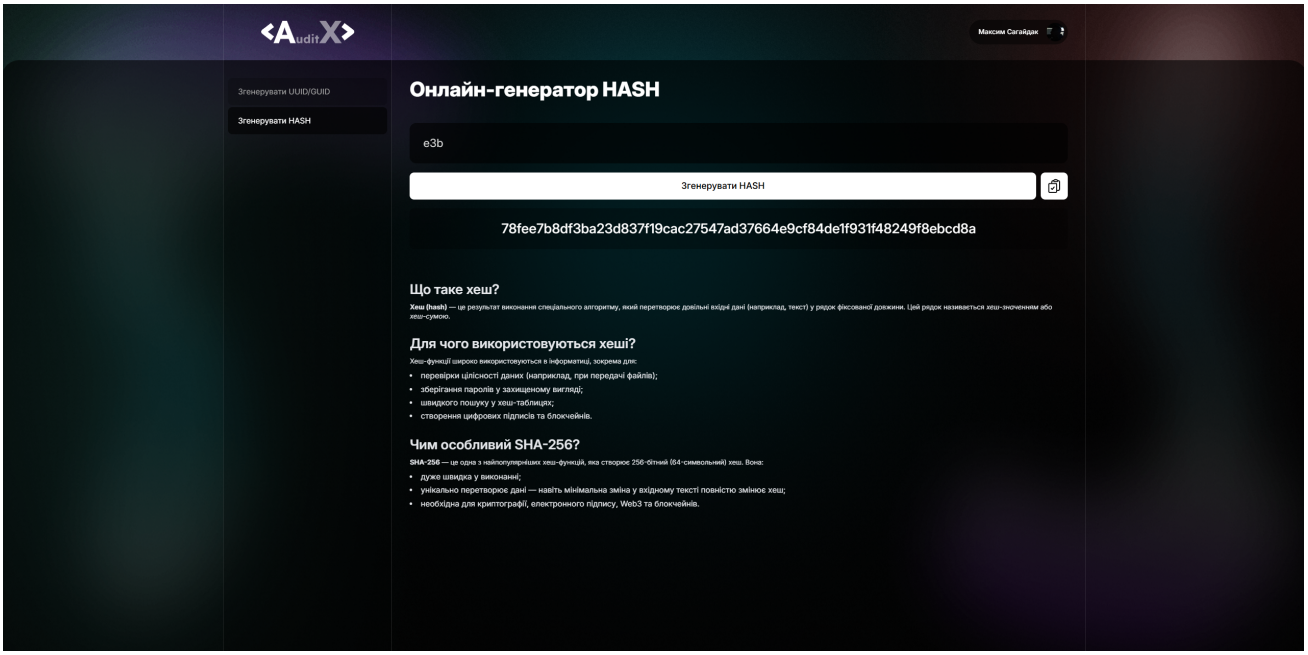


Рисунок 1.16. Сторінка генерації HASH “HASH”

На цій сторінці бачимо поле для вводу значення на основі якого і буде генеруватися HASH, нижче є кнопка відповідно для самої генерації HASH, навпроти кнопка для копіювання HASH, опис того, що таке HASH як його використовують.

Всі кнопки працюють як потрібно, нарікань немає.

1.5.6 Функціональний огляд сторінки “Info”

Сторінка “Info” виконує суто інформаційну функцію. Вона створена з метою ознайомлення користувача з автором проєкту, яким у даному випадку є безпосередньо розробник системи – я.

На сторінці подано загальні відомості, що можуть бути цікавими або корисними для користувача, зокрема відповіді на найпоширеніші запитання, які можуть виникнути під час роботи з веб-застосунком, також корисно якщо користувачу стане цікаво, хто є розробником.

Такий підхід дозволяє не лише забезпечити прозорість у відношенні до кінцевого користувача, але й підвищує рівень довіри до самого продукту. На рисунку 1.17 зображено вигляд сторінки “Info”, яка є важливою складовою

частиною інтерфейсу веб застосунку.

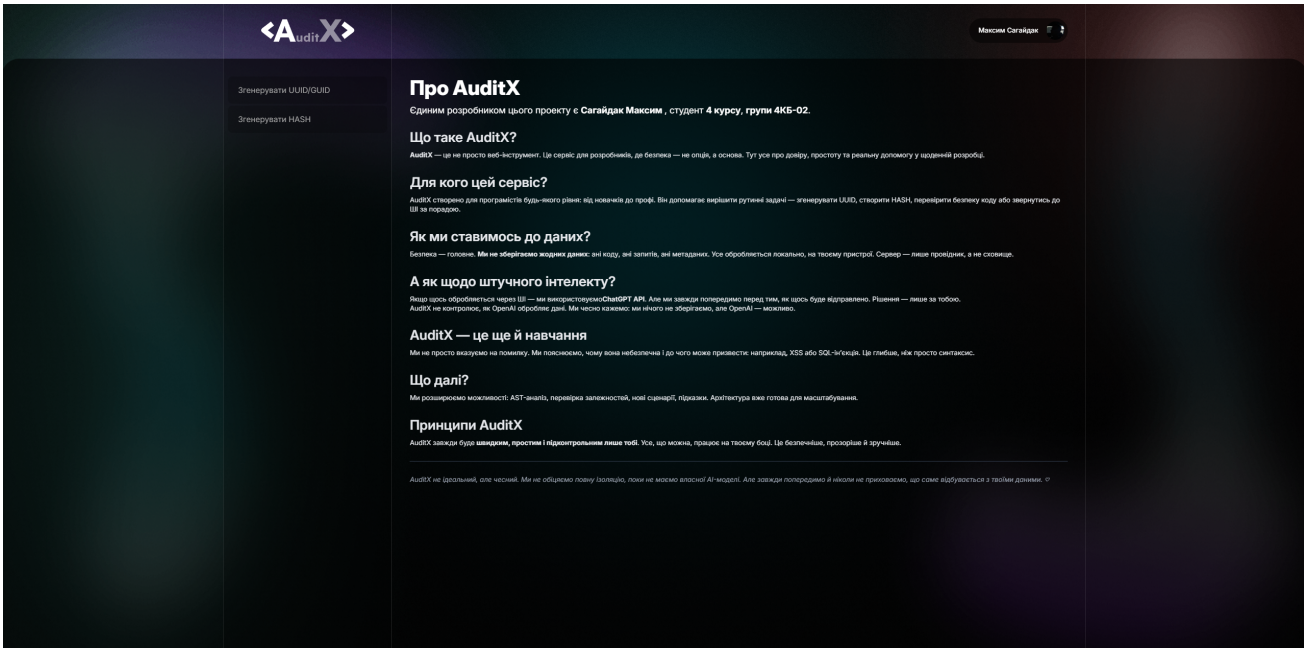


Рисунок 1.17. Сторінка інформації про проєкт та його розробника “Info”

Дане мануальне (ручне) тестування довело працездатність створеної системи, що підкреслює успішне отримання результатів та доцільність використаних технологій та архітектурних рішень.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

В дипломному проекті розроблений веб-застосунок захищеної системи аналізу та оптимізації програмного коду.

Веб-застосунок розроблений з використанням найновітніших технологій сучасного світу веб-розробки. Основна функція веб-застосунку полягає у взаємодії з програмним кодом за допомогою штучного інтелекту для виявлення помилок, підвищення якості та пришвидшення розробки. Система особливо корисна для стартапів та розробників-початківців.

Оскільки даний веб-застосунок не є комерційним тому економічну ефективність не розраховуємо, визначаймо тільки витрати на розробку.

Загальні витрати ($В_з$) на створення сайту складаються з декількох параметрів:

$$B_з = B_р + B_в + B_е \ , \qquad (2.1)$$

де

- $В_р$ – витрати на розробку сайту;
- $В_в$ – витрати на впровадження сайту;
- $В_е$ – витрати на експлуатацію сайту.

Витрати на розробку сайту ($В_р$) є одноразовими та складаються з вартості наступних видів робіт зі створення сайту:

1. Розробка дизайну сайту: розробка макетів дизайну всіх сторінок, розробка фірмового стилю, розробка логотипу
2. Підготовка сторінок-шаблонів: підготовка сторінок-шаблонів включає створення базової структури веб-сторінок із урахуванням дизайну та функціональних вимог, що дозволяє забезпечити єдину основу для подальшої розробки та наповнення контентом.
3. Програмна розробка Front-End сайту: написання програмного коду Front-End частини сайту, програмування динамічних елементів, анімаційних елементів
4. Програмна розробка Back-End сайту: написання програмного коду Back-End частини сайту
5. Наповнення сайту інформацією: наповнення веб-сторінок, обробка

зображень, оптимізація

Розпочинаємо з розрахунку вартості розробки веб-застосунку ($В_р$). Для цього необхідно визначити загальну оплату праці всіх учасників процесу, які були безпосередньо залучені до створення програмного продукту. У даному випадку, реалізацію веб-застосунку здійснював дипломник. Саме мої трудові витрати лягли в основу розрахунку загальної вартості розробки.

Для більш точного обчислення трудомісткості ($В_р$), склали детальний план-графік виконання робіт, який охоплює всі основні етапи створення веб-застосунку. У цьому графіку враховано тривалість кожного виду робіт, а також їх розподіл по відповідних етапах розробки. Усі етапи та відповідні їм виконавці структуровано та представлені в таблиці 2.1, що дозволяє наочно оцінити обсяг та складність виконаних завдань.

Таблиця 2.1. План-графік по розробці Web-сайту

№	**Назва етапу**	**Час виконання (годин)**	**Посада виконавця**
1	Формування технічного завдання та структури майбутнього веб-застосунку	28 годин	Дипломник
2	Створення макетів інтерфейсу веб-застосунку та визначення логіки взаємодії	30 годин	Дипломник
3	Розробка Front-End частини із використанням бібліотеки React та TypeScript	46 годин	Дипломник
4	Розробка Back-End частини із використанням бібліотеки Express та TypeScript	30 годин	Дипломник
5	Проведення ручного тестування веб-застосунку для виявлення та виправлення помилок	12 годин	Дипломник
ВСЬОГО:		146 годин	

Для оцінки складності розробки веб-застосунку ($В_р$) був складений план-графік робіт з розробки веб-системи та визначення тривалості виконання завдань. Розподіл завдань за етапами та видами виконавців наведено у таблиці 2.2.

Таблиця 2.2. Витрати на заробітну плату

№	Персонал	Етапи розробки	Кількість робочих годин (або днів)	Погодинна ставка (або денна ставка), грн.	Заробітна плата, грн.
1	Дипломник	Формулювання технічного завдання, побудова загальної структури проєкту	28 годин	160грн./год.	4480 грн.
2	Дипломник	Розробка макетів інтерфейсу, розробка логіки взаємодії та навігації	30 годин	160грн./год.	4800 грн.
3	Дипломник	Реалізація клієнтської частини веб-застосунку з використанням React і TypeScrip та основної логіки	46 годин	160грн./год.	7360 грн.
4	Дипломник	Побудова серверної частини на базі Express і TypeScript, налаштування маршрутів та обробників запитів	30 годин	160грн./год.	4800 грн.
5	Дипломник	Проведення ручного тестування функціоналу, виявлення помилок, виправлень та оптимізація	12 годин	160грн./год.	1920грн.
ВСЬОГО:					$В_{зп}$ = 23 360 грн.

До складу витрат на оплату праці також включаються податки, збори і інші обов'язкові платежі, встановлені системою оподаткування, що діє. Розмір єдиного соціального внеску складає 22% від заробітної плати, розраховується за наступною формулою:

$$В_{єсв} = В_{зп} \times 0,22 = 23\ 360 \times 0,22 = 5139\ грн. \quad (2.2)$$

Загальні витрати ($В_р$) на розробку веб-сайту розраховуються як сума витрат

на заробітну плату праці персоналу ($В_{зп}$) та єдиного соціального внеску ($В_{єсв}$):

$$В_р = В_{зп} + В_{єсв} = 23\ 360 + 5139 = 28\ 499\ грн. \quad (2.3)$$

Витрати на впровадження сайту ($В_в$) складаються з двох складових:

1. Витрати на реєстрацію доменного імені .store становить 2 000 грн. ($В_{в1}$).
2. Додавання сайту до пошукових систем, наприклад Google через Google Search Console чи Bing через Bing Webmaster Tools, здійснюється безкоштовно і не потребує фінансових витрат ($В_{в2}$).

$$В_в = В_{в1} + В_{в2} = 2\ 000\ грн. \quad (2.4)$$

Витрати на експлуатацію сайту ($В_е$) включають вартість робіт з підтримки сайту в робочому стані і вартість послуг по продовженню доменного імені на один рік. Також рахуємо і вартість серверів на один рік.

Важливо зазначити, що веб-застосунок потребую постійного аналізу статистики та постійного оновлення таких даних як застарілі токени для взаємодії з штучним інтелектом. Також потрібно оновлювати інформацію стосовно технологій у відповідних розділах сайту.

У таблиці 2.3 визначаються постійні витрати як сума витрат на впровадження та експлуатацію сайту протягом року. Дані взяті з відповідних прайс-листів та технічних специфікацій провайдерів послуг.

Таблиця 2.3. Постійні витрати

№	Стаття витрат	Вартість за рік, грн.
1	Сервери на 1 рік	4 500 грн.
2	Продовження доменного імені	2 000 грн.
3	Підтримка, аналіз статистики, оновлення даних	10 000 грн.
ВСЬОГО:		$В_е$ = 16 500 грн.

Загальні витрати ($В_з$) на розробку, впровадження та експлуатацію веб-сайту розраховуються за наступною формулою:

$$В_з = В_р + (В_в + В_е) = 28\ 499 + (2000 + 16\ 500) = 46\ 999\ грн. \quad (2.5)$$

Функціональна ефективність веб-застосунку “AuditX” полягає у здатності суттєво підвищити швидкість та якість написання коду шляхом інтеграції штучного інтелекту у робочий процес розробника. Завдяки зручному веб-

інтерфейсу, користувачі можуть без зайвих технічних знань отримати допомогу у написанні, рефакторингу чи аналізі програмного коду. Це особливо корисно для молодих спеціалістів, які не завжди мають достатній рівень досвіду, але прагнуть розвиватись у сфері ІТ.

Система розроблена з акцентом на швидкодію, тому реалізована з використанням ефективних веб-технологій (React, Tailwind CSS, []).

“AuditX” також пропонує адаптивний інтерфейс, зручний для користування як на комп’ютерах, так і на мобільних пристроях. Це дозволяє працювати з проєктом у будь-який час і з будь-якої локації, підвищуючи загальну гнучкість робочого процесу. Додатково реалізовані темна тема, що зменшує навантаження на зір під час довготривалої роботи.

Соціальна ефективність AuditX полягає в тому, що проєкт має значний потенціал у сфері соціального впливу – зокрема у сприянні розвитку молодих фахівців та підтримці нових технологічних ініціатив. Продукт дозволяє початківцям працювати з сучасними технологіями на практиці, отримуючи підтримку від ШІ, що імітує досвідченого наставника. Такий підхід зменшує бар’єр входу в професію, мотивує до навчання та сприяє професійному зростанню.

Доступність AuditX через веб-інтерфейс робить його зручним і відкритим для широкого кола користувачів. Сервіс не потребує встановлення, складного налаштування чи оплати за базову функціональність, що робить його придатним навіть для студентів або ентузіастів, які працюють над власними проєктами у вільний час.

Окрім цього, проєкт формує позитивний імідж організації або ініціативи, що його підтримує, демонструючи орієнтацію на майбутнє, підтримку молоді та впровадження інновацій. Це може стати хорошою основою для подальшої співпраці з навчальними закладами, технічними хабами чи ІТ-компаніями.

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Основні положення

У цьому розділі дипломного проєкту особлива увага приділяється важливим аспектам охорони праці, техніки безпеки, а також санітарно-гігієнічним умовам, що повинні бути дотримані під час реалізації програмного продукту “AuditX”. Ця система, що призначена для аналізу та оптимізації програмного коду з точки зору кібербезпеки, розроблялась у типових умовах, характерних для програмістської діяльності. Головна мета даного розділу – забезпечення комфортного, безпечного, фізично та психоемоційно здорового середовища для працівника, який виконує завдання з високою інтелектуальною напругою. Адже навіть за наявності найсучаснішого обладнання, без дотримання норм охорони праці жодна система не може бути створена ефективно та безпечно.

3.2 Умови праці та безпечне виконання робіт

Під час роботи над проєктом всі розробницькі дії, включаючи написання коду, його тестування, аналіз на вразливості, оптимізацію та візуалізацію результатів, проводилися в умовах офісного середовища з використанням персонального комп’ютера. Середовище було спеціально адаптоване для діяльності програміста, забезпечуючи належні технічні та побутові умови.

3.2.2 Небезпечні та шкідливі фактори

У ході програмної діяльності мають місце певні шкідливі або потенційно небезпечні фактори, які можуть впливати на фізичне та психоемоційне здоров’я виконавця. До них відносяться:

1. Зорове навантаження – викликане тривалою фокусованою роботою за монітором, що може спричинити втому очей, головний біль, сухість слизової оболонки очей.
2. Статичне навантаження на опорно-руховий апарат – тривале сидіння у статичній позі призводить до порушень кровообігу, напруження м’язів шиї, спини, кистей.
3. Психоемоційне виснаження – зумовлене дедлайнами, необхідністю

постійного аналізу великих обсягів інформації, пошуком помилок у коді тощо.

4. Неналежне освітлення – як занадто яскраве, так і тьмяне світло може викликати швидку втому очей і погіршення концентрації.
5. Вплив електромагнітного випромінювання – яке генерується комп'ютерною технікою, хоч і на безпечному рівні, але при тривалому впливі може мати негативний ефект.
6. Недотримання ергономіки – неправильне розташування робочих елементів, незручне крісло, невдале положення рук або ніг під час роботи негативно впливають на опорно-руховий апарат.

3.2.3 Заходи щодо забезпечення безпеки

Щоб уникнути або суттєво зменшити вплив зазначених факторів, необхідно дотримуватись низки профілактичних заходів:

1. Використання ергономічних меблів – крісло з регулюванням висоти, підлокітниками, підтримкою спини; стіл з достатньою глибиною та шириною.
2. Правильне розташування монітора – на рівні очей або трохи нижче, з відстанню до 70 см від користувача.
3. Регулярні перерви – не рідше ніж щогодини рекомендується робити 5–10-хвилинні перерви з розминкою для очей і тіла.
4. Застосування захисних окулярів або фільтрів – з метою зменшення впливу синього світла та відблисків.
5. Оптимальне освітлення – поєднання природного та штучного світла, з джерелом світла, розташованим зліва (для правші).
6. Психоемоційна гігієна – робочий графік із чергуванням завдань різної складності, регулярна фізична активність, вчасний відпочинок.
7. Інструктаж з охорони праці – кожен співробітник повинен бути ознайомлений з основами безпечної роботи за ПК, відповідним обладнанням та електроприладами.

3.3 Вимоги до робочого середовища

3.3.1 Приміщення

Для розміщення одного працівника необхідно забезпечити не менше 6 м² площі та 20 м³ об’єму повітря. Приміщення має бути достатньо просторим, добре освітленим, із якісною вентиляцією. Поверхні мають бути чистими, відсутні шкідливі запахи чи пил. Стелі не нижче 2,5 м. Уникається наявність джерел підвищеного шуму чи вібрації.

3.3.2 Мікроклімат

Температурний режим має бути в межах 18–22 °C, при відносній вологості 40–60%. Повітря не повинно бути пересушеним або надмірно вологим. Підтримання сталої температури забезпечується за допомогою систем опалення або кондиціювання.

3.3.3 Освітлення

Робоче місце має бути забезпечене природним та рівномірним штучним освітленням. Освітленість поверхні столу повинна становити не менше 500 лк. Перевагу слід надавати лампам теплого спектру, без мерехтіння. Використання світильників із можливістю регулювання інтенсивності є бажаним.

3.3.4 Шум

Рівень шуму не повинен перевищувати 50 дБ. Робоче місце має бути розташоване подалі від джерел постійного фонових шумів, таких як кондиціонери, офісні принтери чи розмови в офісі. При потребі допускається застосування шумоізоляційних перегородок.

3.4 Організація робочого місця

Крісло має підтримувати поперековий відділ спини, бути м’яким і регулюватись по висоті. Відстань від очей до екрана – 60–80 см. Монітор слід розташовувати фронтально, з невеликим нахилом. Клавіатура та миша повинні знаходитися на одному рівні, що дозволяє тримати руки у природному положенні.

3.4.1 Електробезпека

Комп’ютерна техніка та інше офісне обладнання мають бути підключені до мережі через сертифіковані пристрої із заземленням. Проводиться регулярна перевірка справності кабелів, розеток і фільтрів. Застосування подовжувачів із вбудованими запобіжниками є обов’язковим.

Співробітники мають бути навчені правилам користування електрообладнанням, зокрема вимиканням пристроїв у разі виявлення несправності.

3.4.2 Пожежна безпека

В офісі повинні бути встановлені вогнегасники відповідного класу. Усі працівники повинні знати місця їх розташування і порядок використання. Електрощити повинні бути вільно доступні, не загромаджені сторонніми предметами.

Кабелі мають бути справними, не пошкодженими та прокладеними в спеціальних кабель-каналах. Інструктаж з протипожежної безпеки є обов’язковим.

3.5 Підведення підсумків

Загалом, розробка програмного продукту “AuditX” вимагала не лише глибоких технічних знань і навичок програмування, а й суворого дотримання вимог охорони праці, техніки безпеки та ергономіки робочого місця.

Праця програміста передбачає тривале перебування за комп’ютером, тому створення комфортного, безпечного і добре організованого середовища є надзвичайно важливим для збереження здоров’я, підвищення продуктивності та забезпечення стабільного психоемоційного стану. Під час роботи над проєктом було враховано низку критично важливих факторів: правильне розташування робочого місця, оптимальне освітлення без зайвого навантаження на зір, забезпечення достатньої вентиляції та підтримка комфортної температури в приміщенні.

Важливу роль відіграє і ергономіка – правильна висота стільця та столу, положення монітора на рівні очей, зручна клавіатура й мишка. Ці чинники

сприяють зменшенню фізичного навантаження та запобігають професійним захворюванням, які часто зустрічаються серед фахівців ІТ-сфери.

Окрім фізичного комфорту, було дотримано й основних вимог щодо техніки безпеки при роботі з комп’ютерним обладнанням, зокрема: перевірка справності електропроводки, використання стабілізаторів напруги, правильна організація кабелів та запобігання випадковому контакту з розетками чи іншими небезпечними елементами. Також було дотримано режиму праці та відпочинку, що включає регулярні перерви для відпочинку очей та зміни положення тіла.

Такі заходи позитивно вплинули на загальний стан здоров’я, дозволили уникнути перенавантаження та сприяли кращій концентрації уваги на складних завданнях.

Отже, можна зробити висновок, що належна організація умов праці програміста безпосередньо впливає на якість розробленого програмного забезпечення. Виконання вимог охорони праці, дотримання технічних норм і створення комфортного робочого середовища є необхідною умовою для успішної, безпечної та ефективної діяльності в галузі інформаційних технологій.

ВИСНОВКИ

В дипломному проєкті було реалізовано веб-застосунок “AuditX” – інтелектуальний інструмент, що використовує можливості штучного інтелекту для взаємодії з програмним кодом через зручний, інтуїтивно зрозумілий веб-інтерфейс.

Основна мета проєкту полягала у підвищенні ефективності роботи розробників-початківців, а також у підтримці стартапів на ранніх стадіях розробки шляхом пришвидшення створення MVP-продуктів при мінімальних витратах.

У процесі реалізації проєкту були використані сучасні, надійні технології: “React” і “Tailwind CSS” для реалізації клієнтської частини, “Express.js” для серверної логіки. Завдяки інтеграції з моделлю “ChatGPT”, адаптованою під завдання, пов’язані з програмуванням, система дозволяє користувачам отримувати зрозумілі відповіді, рекомендації, пояснення, а також фрагменти коду, що допомагають швидко розв’язувати типові та нетипові задачі під час розробки.

У пояснювальній записці детально проаналізовано проблему, яку вирішує проєкт, наведено огляд існуючих рішень і доведено актуальність створення такого продукту. Проведено аналіз і обґрунтування вибору технологічного стеку, описано архітектуру застосунку, а також наведено поетапний процес розробки та тестування функціоналу. Особливу увагу було приділено практичному застосуванню продукту в реальних умовах стартапу та його перевагам для цільової аудиторії.

Поставлену мету проєкту досягнуто повною мірою: створено прототип інструменту з реальним практичним застосуванням, який може значно пришвидшити процес розробки програмного забезпечення, підвищити продуктивність початківців у команді та зменшити технічні бар’єри для нових технологічних ініціатив.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Соммервілл, І. Інженерія програмного забезпечення. 10-е вид. – К. : ВНТЛ-Класика, 2021. – 832 с.
2. Крамаренко, С. В.; Ковальчук, Ю. П. Основи безпеки інформаційних систем. – Київ: КНУ імені Тараса Шевченка, 2020. – 248 с.
3. Хільчевський, В. М.; Барановська, І. Ю. Web-програмування: навчальний посібник. – Київ: КНЕУ, 2021. – 336 с.
4. ДСТУ ISO/IEC 27002:2015. Інформаційні технології. Методи захисту. Практичні правила управління інформаційною безпекою. – Київ: ДП «УкрНДНЦ», 2015. – 96 с.
5. RFC 7519: JSON Web Token (JWT). Internet Engineering Task Force (IETF), 2015. – 33 p. Режим доступу: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 15.05.2025).
6. Open Web Application Security Project (OWASP). Top 10 Web Application Security Risks. – 2023. [Веб-сайт]. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 20.05.2025).
7. ISO/IEC 27001:2017. Information technology – Security techniques – Information security management systems – Requirements. – Geneva: ISO, 2017. – 30 p.
8. McGraw. Software Security: Building Security. – Addison-Wesley, 2006. – 448 p.
9. Welling, Luke; Thomson, Laura. PHP and MySQL Web Development. 5th Edition. – Addison-Wesley Professional, 2016. – 1008 p.
10. Flanagan, David. JavaScript: The Definitive Guide. 7th Edition. – O'Reilly Media, 2020. – 706 p.
11. Duckett, Jon. JavaScript and JQuery: Interactive Front-End Web Development. – Wiley, 2014. – 640 p.
12. OWASP Authentication Cheat Sheet. – 2023. [Веб-сайт]. URL:https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (дата звернення: 22.05.2025).
13. MDN. HTTP Auth – 2024. [Веб-сайт]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication (дата звернення: 10.06.2025).

ДОДАТОК А. Фрагмент програмного коду React-компонентів

```
// MainPage.tsx

const MainPage = () => {
  return (
    <div className="flex h-full w-full">
      <div className="flex flex-col gap-y-2 text-center">
        <h1 className="font-bold">Привіт, я AuditX</h1>
        <p className="text-sm font-light text-neutral-300">
          Просто вставляй код та отримуй поради
        </p>
      </div>
      <CodeForm />
    </div>
  )
}

export { MainPage }

// UuidPage.tsx

const UuidPage: FC = () => {
  const [uuid, setUuid] = useState<string>(uuidv7())

  const handleButtonClick = () => {
    setUuid(uuidv7())
  }

  return (
    <div className="flex flex-col">
      <h1>Онлайн-генератор UUID/GUID</h1>
      <div className="mt-10 flex flex-col gap-4">
        <TextBlock>{uuid}</TextBlock>
        <div className="flex gap-2">
          <ButtonPrimary className="w-full" onClick={handleButtonClick}>
            Згенерувати UUID/GUID
          </ButtonPrimary>
          <CopyButtonPrimary value={uuid} />
        </div>
      </div>
      <div className="mt-10">
        <section className="space-y-6 text-sm text-gray-700 dark:text-gray-200">
          <div>
            <h2 className="mb-2  font-semibold">Що таке GUID?</h2>
            <p>
              GUID (також UUID) — це абревіатура від
              <strong>"Globally Unique Identifier"</strong> або
              <strong>"Universally Unique Identifier"</strong>, що в перекладі
              означає
              <em>"Глобальний/Універсальний Унікальний Ідентифікатор"</em>. Це
              128-бітове ціле число, яке використовується для ідентифікації
              ресурсів. Термін <strong>GUID</strong> зазвичай використовується
              розробниками, які працюють із технологіями Microsoft, тоді як
              <strong>UUID</strong> частіше вживається в інших екосистемах.
            </p>
          </div>
          <div>
            <h2 className="mb-2  font-semibold">Як використовуються GUID?</h2>
            <p>
```

```
              GUID застосовуються розробниками корпоративного програмного
              забезпечення, інженерами, адміністраторами баз даних і
              тестувальниками під час створення та тестування систем і
              застосунків. Вони використовуються в таких мовах як
            </p>
          </div>
          <div>
            <h2 className="mb-2  font-semibold">Наскільки унікальний GUID?</h2>
            <p>
              128 бітів — це достатньо багато, а алгоритм генерації — досить
              надійний, щоб:
            </p>
            <ul className="mt-2 list-inside list-disc space-y-1">
              <li>
                Якщо
                <strong>
                  кожна людина на Землі згенерує по 600 000 000 GUID-ів
                </strong>
                , то ймовірність дубліката буде такою ж — <strong>50%</strong>.
              </li>
            </ul>
          </div>
        </section>
      </div>
    </div>
  )
}

export { UuidPage }

// HashPage.tsx

const HashPage: FC = () => {
  const [value, setValue] = useState<string>('')
  const [hash, setHash] = useState<string>('')

  const handleInput: FormEventHandler<HTMLInputElement> = (e) => {
    setValue(e.currentTarget.value)
  }

  const handleGenerateHashButtonClick = async () => {
    const hash = await generateHash(value)
    setHash(hash)
  }

  return (
    <div className="grid grid-cols-1">
      <h1>Онлайн-генератор HASH</h1>
      <div className="mt-10 flex flex-col gap-4">
        <TextInput
          size="xl"
          onInput={handleInput}
          value={value}
          placeholder="Що ми будемо хешувати?"
        />
        <div className="flex gap-2">
          <ButtonPrimary
            className="w-full"
            onClick={handleGenerateHashButtonClick}
          >
            Згенерувати HASH
          </ButtonPrimary>
          <CopyButtonPrimary disabled={!hash} value={hash} />
        </div>
```

```
        <TextBlock className={cn('opacity-0', { 'animate-opacity': hash })}>
          {hash}
        </TextBlock>
      </div>
      <div className="mt-10">
        <section className="space-y-6 text-sm text-gray-700 dark:text-gray-200">
          <div>
            <h2 className="mb-2 font-semibold">Що таке хеш?</h2>
          </div>

          <div>
            <h2 className="mb-2 font-semibold">
              Для чого використовуються хеші?
            </h2>
            <ul className="mt-2 list-inside list-disc space-y-1">
              <li>
                перевірки цілісності даних (наприклад, при передачі файлів);
              </li>
              <li>зберігання паролів у захищеному вигляді;</li>
              <li>швидкого пошуку у хеш-таблицях;</li>
              <li>створення цифрових підписів та блокчейнів.</li>
            </ul>
          </div>
          <div>
            <h2 className="mb-2 font-semibold">Чим особливий SHA-256?</h2>
            <p>
              <strong>SHA-256</strong> — це одна з найпопулярніших хеш-функцій,
              яка створює 256-бітний (64-символьний) хеш. Вона:
            </p>
            <ul className="mt-2 list-inside list-disc space-y-1">
              <li>дуже швидка у виконанні;</li>
              <li>
                унікально перетворює дані — навіть мінімальна зміна у вхідному
                тексті повністю змінює хеш;
              </li>
              <li>
                необхідна для криптографії, електронного підпису, Web3 та
                блокчейнів.
              </li>
            </ul>
          </div>
        </section>
      </div>
    </div>
  )
}

export { HashPage }

// AuthGoogleService.ts

type VerifyIdTokenCallback = (err: Error | null, login?: LoginTicket) => void
class AuthGoogleService {
  static CLIENT_ID = env.AUTH_GOOGLE_CLIENT_ID
  static CLIENT_SECRET = env.AUTH_GOOGLE_CLIENT_SECRET
  static REDIRECT_URL = env.AUTH_GOOGLE_REDIRECT_URL
  static SCOPES = [
    'https://www.googleapis.com/auth/userinfo.profile',
    'https://www.googleapis.com/auth/userinfo.email',
  ]

  private oauth2Client

  constructor() {
```

```
    this.oauth2Client = new google.auth.OAuth2(
      AuthGoogleService.CLIENT_ID,
      AuthGoogleService.CLIENT_SECRET,
      AuthGoogleService.REDIRECT_URL,
    )
  }

  generateAuthUrl() {
    return this.oauth2Client.generateAuthUrl({
      access_type: 'offline',
      scope: AuthGoogleService.SCOPES,
      include_granted_scopes: true,
      prompt: 'consent',
    })
  }

  async getTokens(code: string) {
    const { tokens } = await this.oauth2Client.getToken(code)

    return {
      ...tokens,
    }
  }

  async generateIdToken(refreshToken: string) {
    this.oauth2Client.setCredentials({ refresh_token: refreshToken })
    const tokens = await this.oauth2Client.refreshAccessToken()
    return tokens.credentials.id_token
  }

  async verifyIdToken(idToken: string, callback?: VerifyIdTokenCallback) {
    if (callback) {
      return this.oauth2Client.verifyIdToken({ idToken }, callback)
    }

    return await this.oauth2Client.verifyIdToken({ idToken })
  }

  async revokeRefreshToken(refreshToken: string) {
    await this.oauth2Client.revokeToken(refreshToken)
  }
}
export { AuthGoogleService }

// OpenAIClientService.ts

class OpenAIClientService {
  static API_KEY = env.OPEN_AI_API_KEY_MAIN
  private openAIClient

  constructor() {
    this.openAIClient = new OpenAI({
      apiKey: OpenAIClientService.API_KEY,
    })
  }

  private async sendPrompt(userMessage: string, systemPrompt?: string) {
    return await this.openAIClient.chat.completions.create({
      model: 'gpt-4o-mini',
      messages: systemPrompt
        ? [
            { role: 'system', content: systemPrompt },
            { role: 'user', content: userMessage },
          ]
```

```
        : [{ role: 'user', content: userMessage }],
    })
  }
  async sendCodePrompt(
    code: string,
    options: {
      optimize: boolean
      explain: boolean
    },
  ) {
    const messageSystem = `
        Ти — II-помічник для аналізу та покращення коду. Отримуєш від користувача довільний фрагмент коду.
        Твої завдання:
        1. Визнач мову програмування та дай мені відповідь лише таким словом яке підходить як ключ для бібліотеки shiki, це npm бібліотека для відображення коду, якщо такої мови немає, дай найбільш схожу.
        2. Якщо вказано \`optimize = true\`, перепиши код з покращеннями.
        3. Якщо вказано \`explain = true\`, поясни, що робить код і надай поради щодо його вдосконалення. У такому разі змінювати код не потрібно, а поле \`after\` має містити \`null\`.
        5. Якщо вказано \`optimize = true\` і \`explain = true\` то перепиши код з покращеннями та надай поради щодо його вдосконалення а поле \`after\` має містити оновлений код.
        4. Завжди повертай відповідь точно у **форматі JSON** — **без жодної обгортки типу \`\`\`json.**:

        {
          "code": {
            "language": "мова_програмування",
            "before": "тут_код_який_тобі_передав_юзер",
            "after": "тут_код_після_оптимізації_або_null"
          },
          "text": "пояснення, поради або інша релевантна інформація, якщо потрібно"
        }
        Якщо якесь поле не потрібне — постав \`null\`.
        **Не додавай жодного тексту поза JSON.** Код у полі \`after\` має бути без обгортки \`\`\`.
        `.trim()
    const messageUser = `
        # Вхідний код від юзера:
        \`\`\`
        ${code}
        \`\`\`

        # Опції:
        - optimize: ${options.optimize}
        - explain: ${options.explain}
        `.trim()
    return await this.sendPrompt(messageUser, messageSystem)
  }
}

export { OpenAIClientService }
```

ДОДАТОК Б. Слайди мультимедійної презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

КБ.02.19.000.00 ДП ПЗ

ДИПЛОМНИЙ ПРОЄКТ

На тему:

Розробка захищеної системи аналізу та оптимізації програмного коду

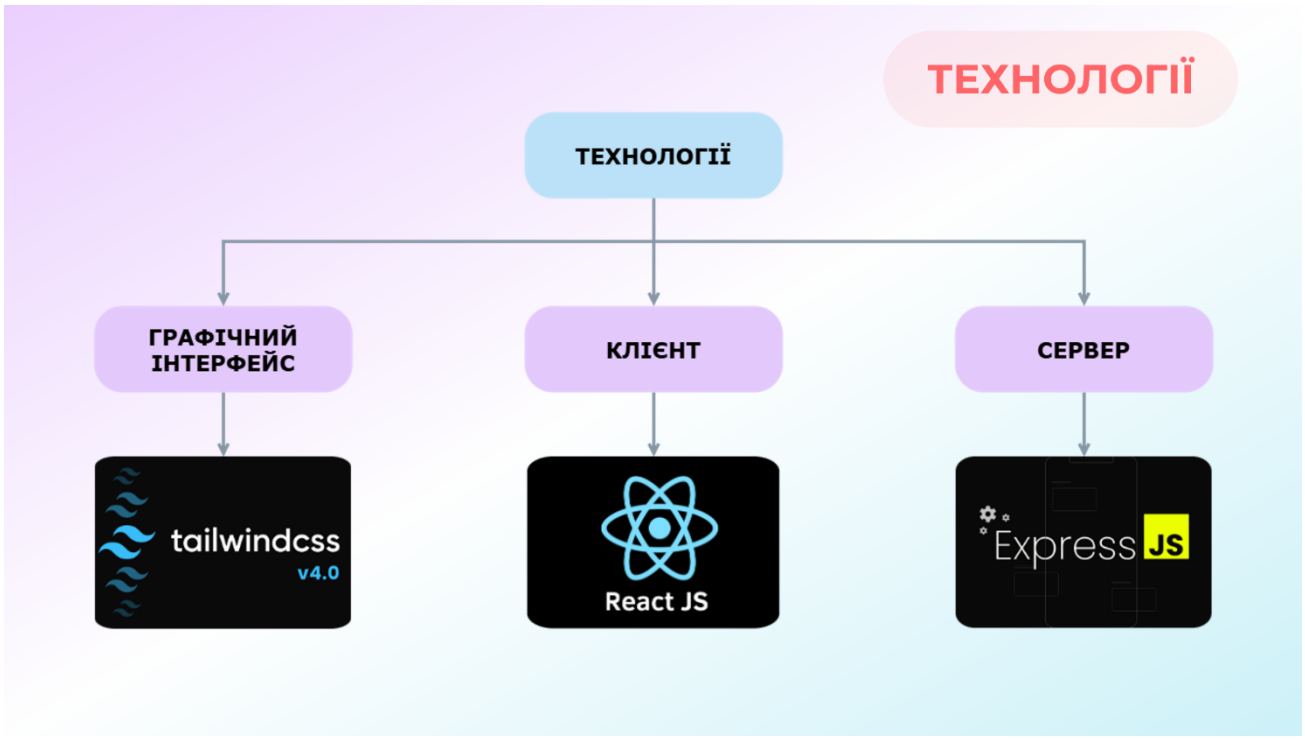
Сагайдак Максим Іванович
Спеціальність: 123 - «Комп'ютерна інженерія»
Освітньо-професійна програма: «Безпека комп'ютерних систем і мереж»

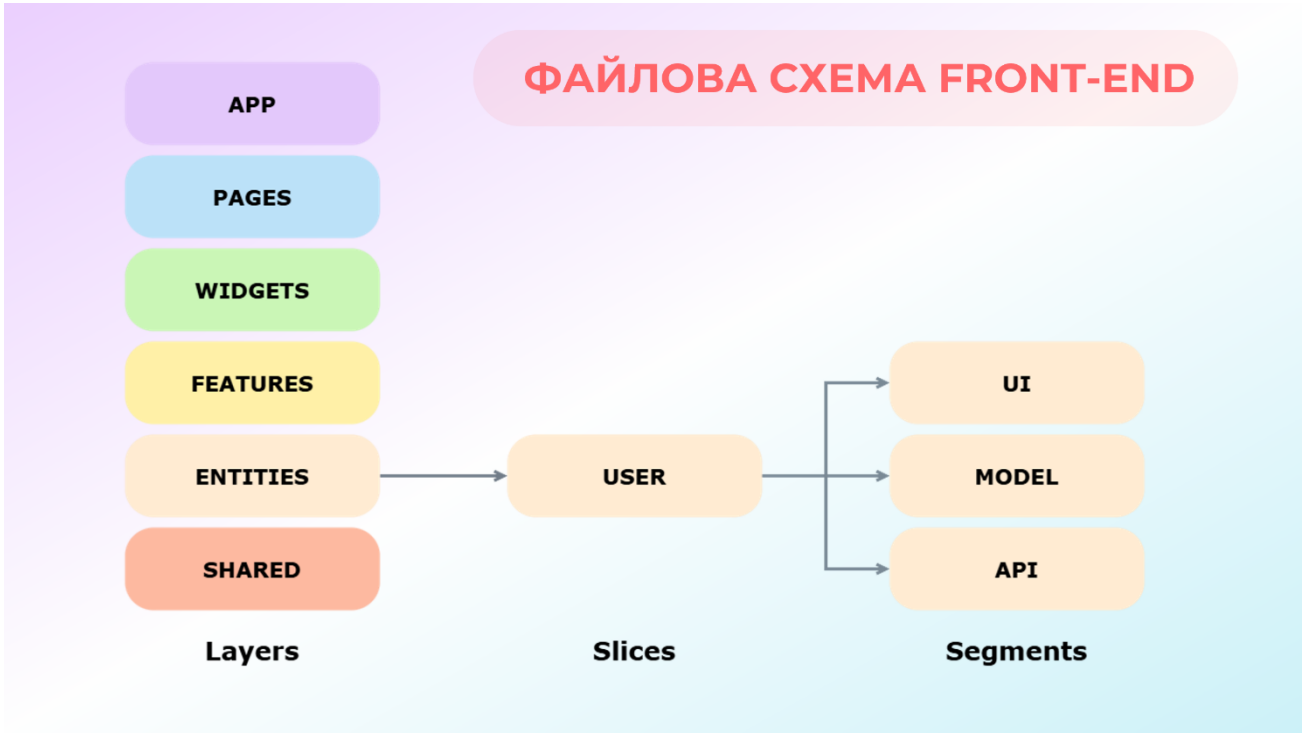
ОСНОВНА МЕТА ПРОЄКТУ “AUDITX”

Більшість стартапів не доходять навіть до стадії MVP через обмежений бюджет і брак досвідчених розробників. Швидкість та вартість розробки – критично важливі фактори. Саме тому інтеграція ШІ у процес розробки може значно підвищити ефективність, особливо для Junior/Middle-розробників.

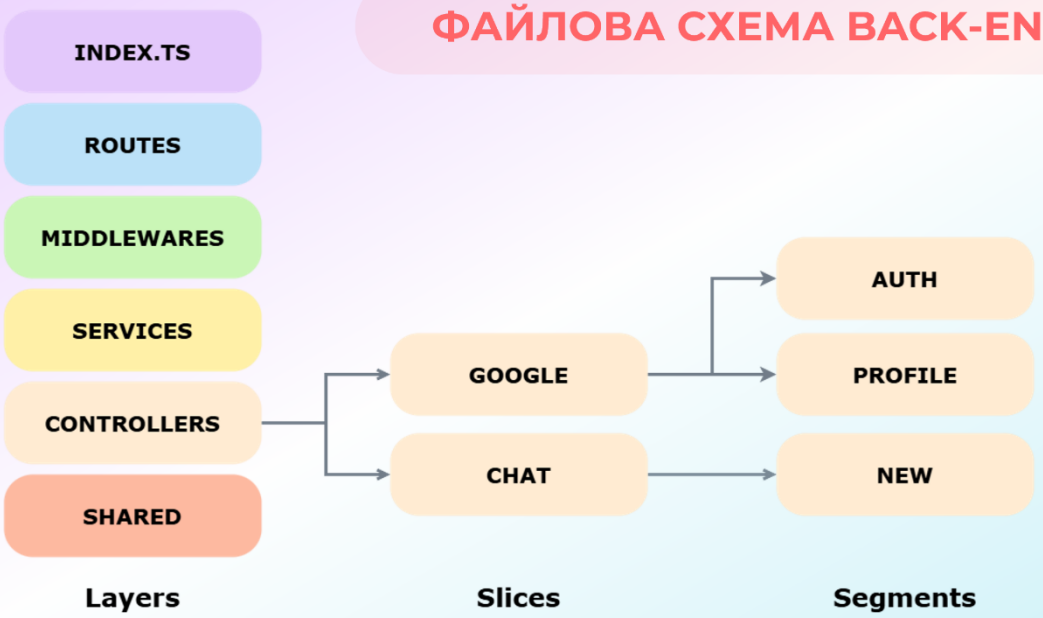
Мій проєкт AuditX – це інтерфейс на базі ChatGPT, створений для взаємодії з кодом, що допомагає пришвидшити розробку та знизити поріг входу для новачків.

Окрім основної взаємодії з ШІ, у проєкті реалізовано додаткові інструменти, зокрема генерацію UUID та HASH, які спрощують типові задачі під час розробки.

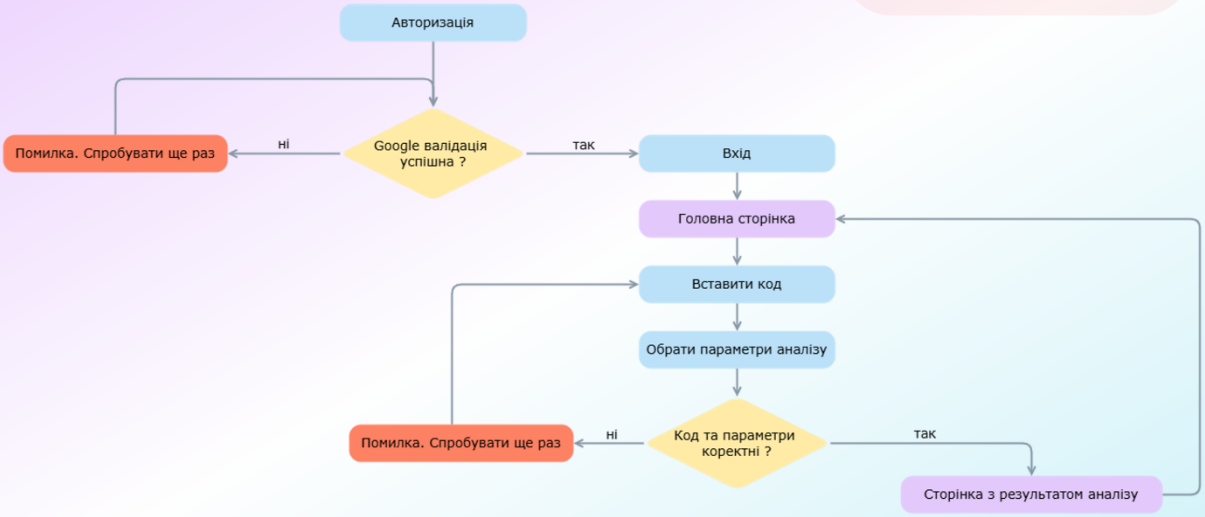

ТЕХНОЛОГІЇ
ТЕХНОЛОГІЇ
ГРАФІЧНИЙ ІНТЕРФЕЙС
КЛІЄНТ
СЕРВЕР
tailwindcss
v4.0
React JS
Express JS


ФАЙЛОВА СХЕМА FRONT-END
APP
PAGES
WIDGETS
FEATURES
ENTITIES
SHARED
USER
UI
MODEL
API
Layers
Slices
Segments

ФАЙЛОВА СХЕМА BACK-END



USER FLOW



СТВОРЕННЯ ДИЗАЙНУ

У проєкті дизайн виконаний у стилі, що імітує прозорі скляні панелі – це надає інтерфейсу сучасного, легкого та візуально привабливого вигляду. Такий підхід дозволяє акцентувати увагу на важливому контенті, зберігаючи загальну гармонію та легкість сприйняття.

Використання прозорих ефектів та плавних переходів допомагає створити відчуття глибини та простору, що покращує користувацький досвід.

Для стилізації застосувався Tailwind CSS, що дозволило швидко та гнучко реалізувати потрібні ефекти без зайвої складності. Такий дизайн підкреслює інноваційність проєкту і робить взаємодію максимально комфортною та приємною для користувача.

РОЗРОБКА FRONT-END ЧАСТИНИ

Фронтенд частина проєкту побудована з використанням FSD-структури, що забезпечує зручну організацію коду та полегшує масштабування. Для розробки інтерфейсу використаний React з TypeScript – це поєднання гарантує високу продуктивність, типобезпечність та гнучкість коду.

Стилізація виконана за допомогою Tailwind CSS, що дозволяє швидко створювати адаптивний і привабливий дизайн без зайвих складнощів. Такий підхід поєднує зручність розробки із якісним користувацьким досвідом, що є ключовим для швидкого впровадження MVP.

Завдяки цьому стеку і структурі, фронтенд проєкту залишається підтримуваним і гнучким, що важливо для подальшого розвитку та додавання нових функцій.

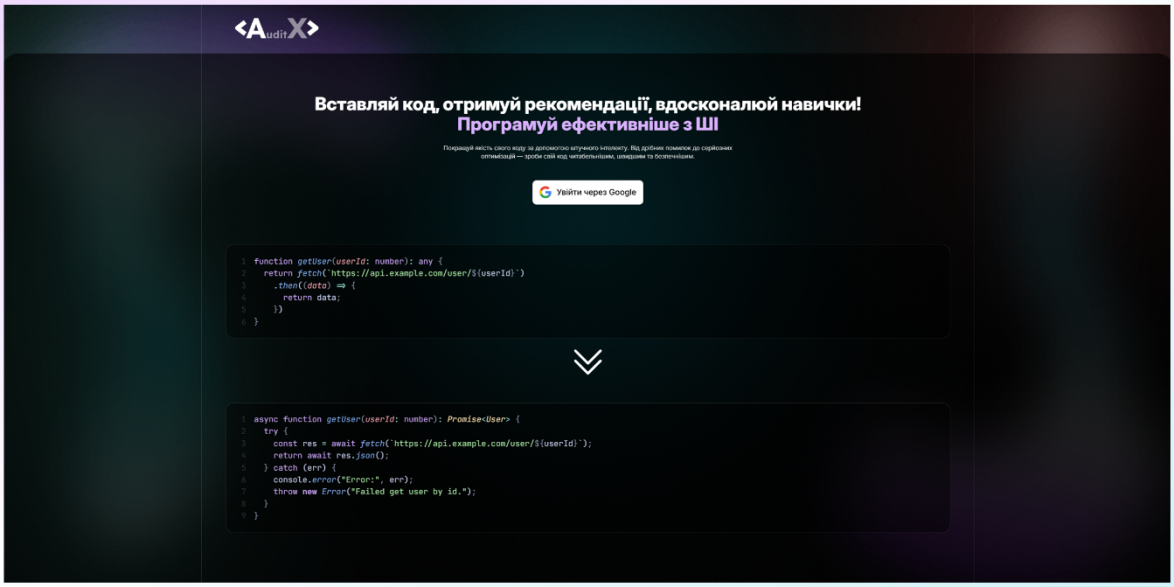
РОЗРОБКА BACK-END ЧАСТИНИ

Бекенд частина проєкту розроблена з використанням Express та TypeScript, що забезпечує надійність, високу продуктивність та типобезпечність коду. Для аутентифікації користувачів реалізована авторизація через Google, що спрощує вхід у систему та покращує користувацький досвід.

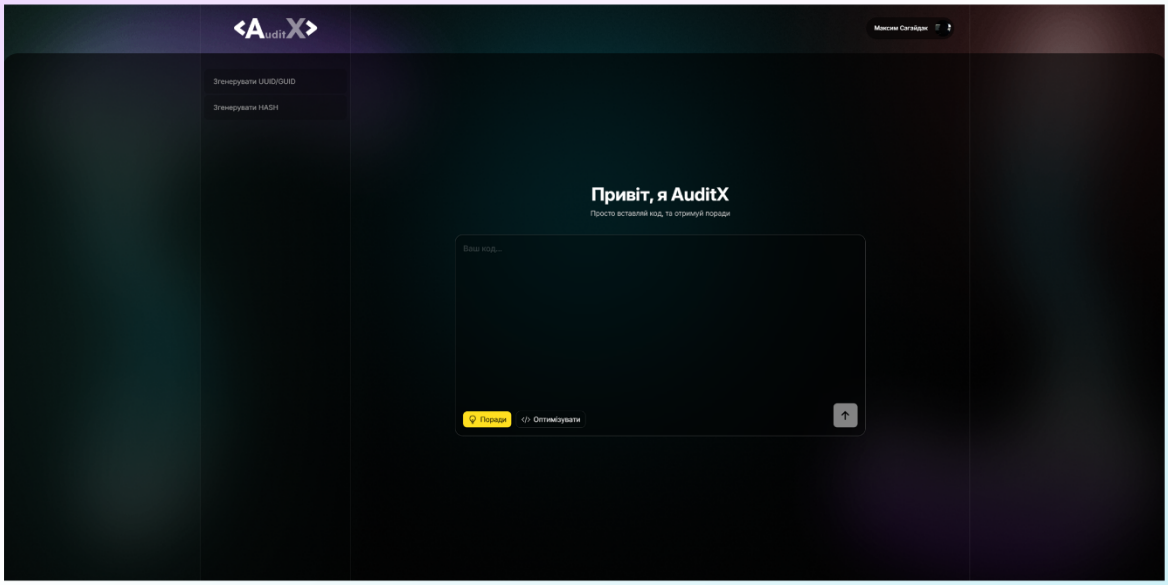
У проєкті свідомо не використовується база даних – це рішення було прийнято з метою підвищення безпеки, оскільки сервер не зберігає жодних клієнтських даних. Відсутність постійного зберігання інформації знижує ризики витоку.

Такий підхід дозволяє зосередитись на швидкому запуску MVP, мінімізуючи складність бекенд-частини і полегшуючи її подальше масштабування.

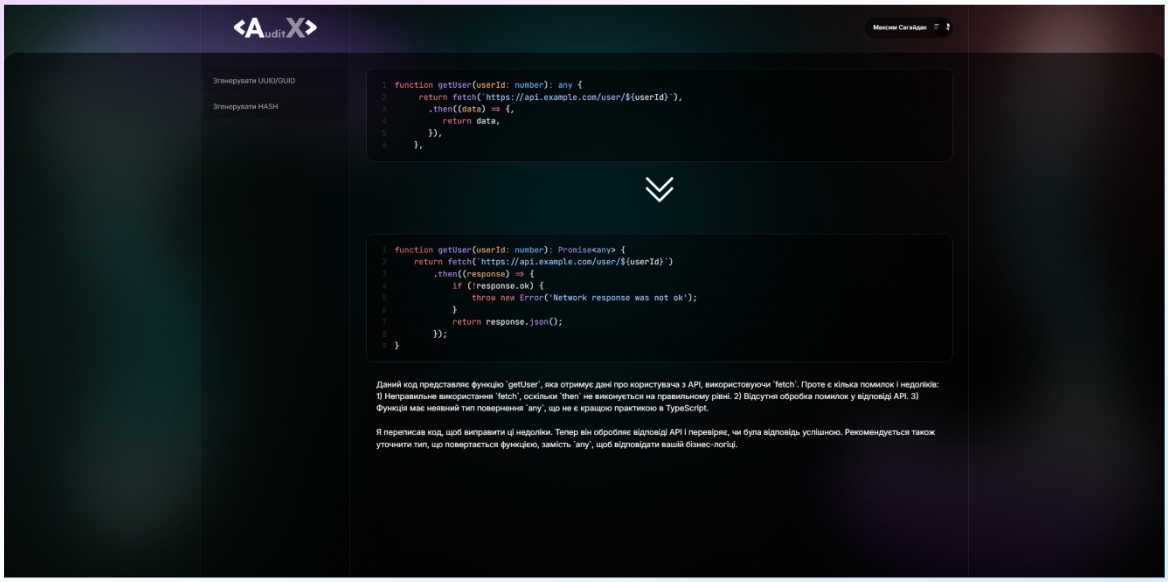
СТОРІНКА АВТОРИЗАЦІЯ У ВЕБ-ЗАСТОСУНКУ



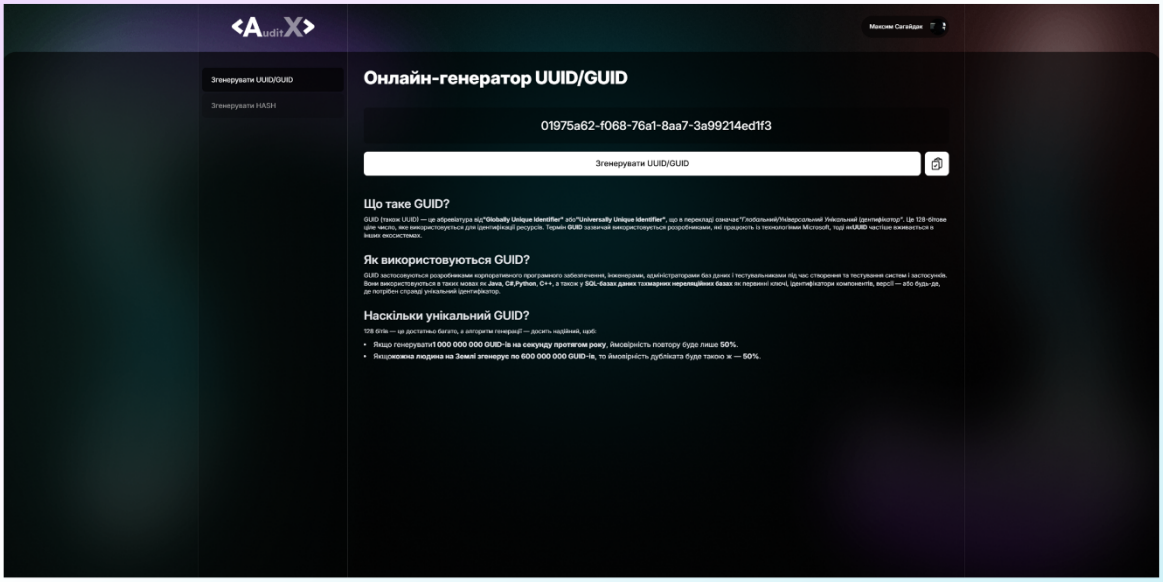
ГОЛОВНА СТОРІНКА



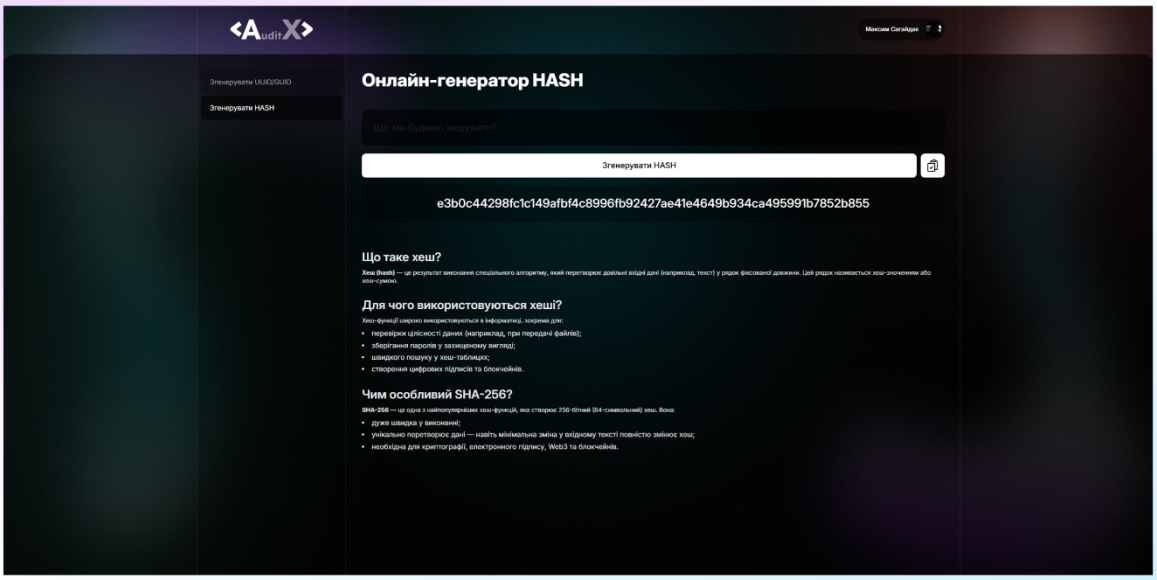
СТОРІНКА З РЕЗУЛЬТАТОМ АНАЛІЗУ КОДУ



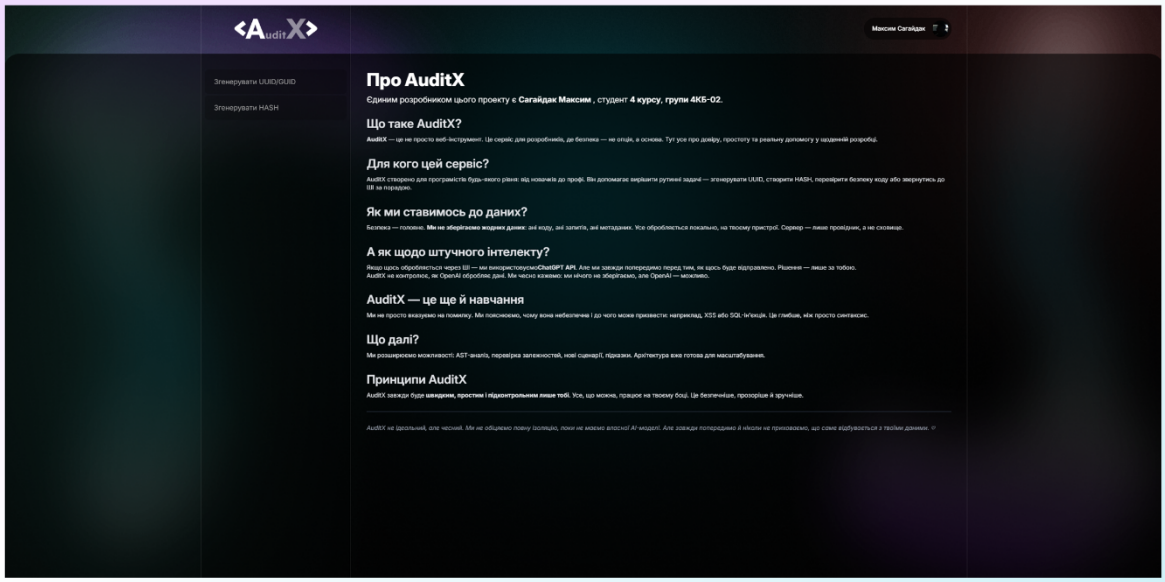
СТОРІНКА ГЕНЕРАЦІЇ UUID/GUID



СТОРІНКА ГЕНЕРАЦІЇ HASH



СТОРІНКА З ІНФОРМАЦІЄЮ



ВИВАНТАЖЕННЯ НА СЕРВЕРИ

Клієнтська частина проєкту розміщена на платформі **Cloudflare**, а серверна – на **Render**.

Для проєкту було придбано домен **auditx.store** через сервіс **NIC.UA**.

Всі налаштування, включно з деплоєм, конфігурацією серверів та DNS, виконані мною самостійно, що забезпечило повний контроль над процесом і стабільну роботу застосунку.

auditx.store

ВИСНОВКИ

У дипломному проєкті створено веб-застосунок AuditX – інтелектуальний інструмент на базі ChatGPT для роботи з кодом через зручний веб-інтерфейс.

Мета – підвищити продуктивність початківців і підтримати стартапи, пришвидшуючи створення MVP при мінімальних витратах. Використано React + TypeScript, Tailwind CSS, Express.js + TypeScript, а також реалізовано додаткові інструменти для генерації UUID та HASH, що спрощують розробку.

Проєкт досяг поставленої мети – створено робочий прототип, який допомагає швидше розв'язувати задачі та знижує технічні бар'єри для нових команд, та розробників-початківців.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Сагайдака Максима Івановича

(прізвище, ім`я та по батькові)

Спеціальність *123 «Комп'ютерна інженерія»*

Освітня програма *«Безпека комп'ютерних систем та мереж»*

Керівник дипломного проекту (роботи) *Закроєв Юрій Михайлович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка захищеної системи аналізу та оптимізації програмного коду*

Обсяг розрахунково-пояснювальної записки *81* сторінок

Обсяг графічної (презентаційної) частини *17* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений на рецензію дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячений проблемі аналізу програмного коду та складається з пояснювальної записки, додатку з програмним кодом та мультимедійної презентації, що містить приклади роботи програми.

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (аналізу предметної області, проектування застосунку, реалізації застосунку, тестування застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора КТ. Економічний розділ проекту містить розрахунок витрат на НДР та реалізацію проекту.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 17 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, скріншоти роботи програмного застосунку, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки добра, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту *Належним чином організовані таблиці, зв'язки та запити, що відповідає вимогам до сучасних ПЗ.*

GUI містить графіки та пояснення для користувача, покращуючи досвід користувача.

д) основні недоліки дипломного проекту

Графічний інтерфейс веб-застосунку занадто темний, що може вплинити на аспект доступності (Accessibility). Незначні недоліки оформлення пояснювальної записки

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис:

« *20* » *червня* 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Сагайдака Максима Івановича

(прізвище, ім`я та по батькові)

Спеціальність: *123 «Комп'ютерна інженерія»*

Освітня програма: *«Безпека комп'ютерних систем і мереж»*

Тема дипломного проекту: *Розробка захищеної системи аналізу та оптимізації коду*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка до дипломного проекту містить 81 сторінки. У пояснювальній записці описано етапи розробки захищеної системи аналізу та оптимізації коду засобами прогресивного фреймворку Next.js та технологіями штучного інтелекту. Графічна частина складається з окремих 17 слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра.*

б) самостійність роботи над проектом: *Протягом виконання дипломного проекту здобувач освіти Сагайдак Максим поступово та послідовно виконував всі етапи, проявляв ініціативу в створенні загальної концепції та реалізації роботи. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Сагайдак Максим під час роботи над дипломним проектом вивчив достатньо багато літературних та інтернет-джерел за даною тематикою. Вважаю, що теоретична підготовка дипломника достатня і він готовий до захисту проекту.*

г) вміння розв'язувати виробничі та конструкторські питання *Під час виконання дипломного проекту здобувач освіти Сагайдак Максим показав вміння організовано працювати над поставленим завданням, застосовувати знання у галузі програмування та математики, розробляти, встановлювати та налаштовувати спеціалізоване програмне забезпечення, оформлювати слайди та складати презентації, користуючись сучасними комп'ютерними програмними засобами, такими як MS VS Code та Next.js.*

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проекту *Жадан Артур Сергійович*

Місце роботи і посада керівника дипломного проекту *ВСП «Одеський технічний фаховий коледж ОНТУ», викладач спецдисциплін циклової комісії комп'ютерної техніки та програмної інженерії*

Підпис

«16» 06 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Сагайдак Максим Іванович,
здобувач освіти гр. 4КБ-02, та

Закроєв Юрій Михайлович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка захищеної системи аналізу та оптимізації програмного коду» (автор роботи – Сагайдак Ю.М., керівник роботи – Закроєв Ю.М.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Сагайдак М.І. /

Керівник _____ / Закроєв Ю.М. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4КБ-02

Сагайдака Максима Івановича

на тему *Розробка захищеної системи аналізу та оптимізації програмного коду*

Висновок відповідальної особи за проведення нормоконтролю:

пояснювальна записка до дипломного проєкту виконана з деякими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 14.06.2025 р. значення коефіцієнту подібності в роботі становить 10,50%, коефіцієнт цитування – 3,45%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Сагайдака М.І.* (П.І.Б.)

проведена «*16*» *червня* 20*25* р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/14/2025
Дата редагування 6/14/2025

Документ приймається

Звіт подібності

метадані

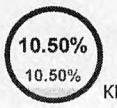
Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка захищеної системи аналізу та оптимізації програмного коду

Автор
Сагайдак Максим Іванович
Науковий керівник / Експерт
Закроєв Юрій Михайлович

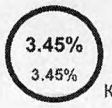
підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КЦ

25	15504	123926
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		0
Мікропробіли		2
Білі знаки		0
Парафрази (SmartMarks)	a	47

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	45 0.29 %
2	https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download	45 0.29 %
3	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	37 0.24 %
4	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	36 0.23 %
5	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	35 0.23 %

7	https://card-file.ontu.edu.ua/bitstreams/538ada8a-2c79-4b1e-b7d2-b0c97f68bc1c/download	71 (6) 0.46 %
8	**https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download**	**62 (3) 0.40 %**
9	https://card-file.ontu.edu.ua/server/api/core/bitstreams/8da72e29-656f-4ee4-9b22-716dedf53ff5/content	40 (3) 0.26 %
10	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	38 (4) 0.25 %
11	**https://www.linkedin.com/posts/uptargetco_architects-mindset-activity-6867388669741821952-R_Rf**	**36 (2) 0.23 %**
12	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	33 (2) 0.21 %
13	**https://card-file.ontu.edu.ua/bitstreams/9908b7a9-6b3e-46f5-a46e-84d83787cfd4/download**	**30 (2) 0.19 %**
14	https://mfikri.com/en/blog/react-express-mysql-authentication	29 (3) 0.19 %
15	https://aircconline.com/ijait/V15N2/15225ijait01.pdf	19 (2) 0.12 %
16	https://anila.me/posts/redux-thunk/	17 (1) 0.11 %
17	https://icr-cert.com.ua/iso-27001-2013-information-security-management-systems-isms/	12 (1) 0.08 %
18	https://www.cs.ubbcluj.ro/files/curricula/2020/syllabus/IR_sem4_MLR5015_ro_bufny_2020_5156.pdf	11 (1) 0.07 %
19	https://blog.logrocket.com/handling-user-authentication-redux-toolkit/	11 (1) 0.07 %
20	https://repository.lnup.edu.ua/jspui/bitstream/123456789/631/1/%D0%9B%D0%B8%D1%81%D0%B5%D0%B9%D0%BA%D0%BE_%D0%86%D1%82-22%D1%81%D0%BF.pdf	7 (1) 0.05 %
21	https://card-file.ontu.edu.ua/bitstreams/12d5c0ab-e979-48f2-a8ec-d5fc31f71fd5/download	5 (1) 0.03 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»_
Освітньо-професійна програма: **«Безпека комп'ютерних систем і мереж» Група:** 4КБ- 02

Дипломний **проект здобувача освіти денної форми навчання КБ.** 02.19.000.ДП

САГАЙДАКА
МАКСИМА ІВАНОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»_
Освітньо-професійна програма: «Безпека комп'ютерних систем і мереж»
Група: 4 КБ-02

ПОЯСНЮВАЛЬНА ЗАПИСКА
до дипломного проекту на тему:

_____

_____

Проектний матеріал складається з пояснювальної записки на _____ сторінках та графічного (презентаційного) матеріалу на _____ аркушах (слайдах)

6	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	34 0.22 %
7	https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download	32 0.21 %
8	Удосконалення засобів захисту веб-застосунків від несанкціонованого впливу [redacted] 3/15/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	31 0.20 %
9	https://www.linkedin.com/posts/uptargetco_architects-mindset-activity-6867388669741821952-R_Rf	27 0.17 %
10	https://card-file.ontu.edu.ua/bitstreams/9908b7a9-6b3e-46f5-a46e-84d83787cfd4/download	25 0.16 %

з домашньої бази даних (0.28 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розробка 3D-гри у жанрі survival-horror з налаштуваннями рівнів складності [redacted] 6/12/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	43 (5) 0.28 %

з програми обміну базами даних (0.39 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Удосконалення засобів захисту веб-застосунків від несанкціонованого впливу [redacted] 3/15/2025 **National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)**	**31 (1) 0.20 %**
2	bitstream_6e6c8e1b-41fc-4c17-82dd-baf0b96438af [redacted] 12/8/2024 National Technical University "Kharkiv Polytechnic Institute" students papers (National Technical University "Kharkiv Polytechnic Institute" students papers)	12 (1) 0.08 %
3	Lb_1_Nasheba_A.docx [redacted] 4/22/2023 Yuriy Fedkovych Chernivtsi National University(CNU) course papers (Deanery)	10 (1) 0.06 %
4	Програмне забезпечення для зберігання та керування паролями користувача [redacted] 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	8 (1) 0.05 %

з Інтернету (9.83 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/server/api/core/bitstreams/44c16132-5f53-48e2-b6c0-61e9a2f0fd75/content	617 (51) 3.98 %
2	**https://card-file.ontu.edu.ua/bitstreams/55e2b8f2-7d3c-4235-99fc-2be51199b96d/download**	**123 (7) 0.79 %**
3	**https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download**	**111 (8) 0.72 %**
4	**https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download**	**103 (9) 0.66 %**
5	**https://card-file.ontu.edu.ua/bitstreams/6cf43324-8f08-4031-ba42-f80b18efbbc8/download**	**77 (2) 0.50 %**
6	**https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download**	**72 (3) 0.46 %**