МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЇ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ВПРОВАДЖЕННЯ ШТУЧНОГО ІНТЕЛЕКТУ В АЛГОРИТМИ ВЕБ-ПЛАТФОРМИ НА ПРИКЛАДІ ЗБОРІВ КОШТІВ НА ЗБРОЙНІ СИЛИ УКРАЇНИ

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

Виконав: студент групи МгІТ-23

Волошин В.М

Науковий керівник к.т.н., доц.

Резанова В.Г.

Рецензент

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЇ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

_____ Наталія ЧУПРИНКА

«_____» _____ 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Волошину Василю Миколайовичу

1. Тема кваліфікаційної роботи Впровадження штучного інтелекту в алгоритми веб-платформи на прикладі зборів коштів на Збройні Сили України

   Науковий керівник роботи к.т.н., доц. Резанова Вікторія Георгіївна

   Затверджені наказом КНУТД від 03.09.2024 року № 188-уч

2. Вихідні дані до кваліфікаційної роботи : Розробка кафедри комп'ютерних наук та технологій, рекомендована література, додатки.
3. Зміст кваліфікаційної роботи: Вступ, Розділ 1 Аналіз веб-застосунків та штучного інтелекту; Розділ 2 Архітектурні рішення веб-платформи; Розділ 3 Розробка веб-платформи; Висновки; Список літератури; Додаток А програмний код; Додаток Б тези доповіді; Додаток В презентація.
4. Дата видачі завдання 08.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапу кваліфікаційної роботи (проєкту)	Орієнтовний термін виконання	Примітка про виконання
1	Вступ	20.09.24	
2	Розділ 1. АНАЛІЗ ВЕБ-ЗАСТОСУНКІВ ТА ШТУЧНОГО ІНТЕЛЕКТУ	21.09.24	
3	Розділ 2. АРХІТЕКТУРНІ РІШЕННЯ ВЕБ-ПЛАТФОРМИ	22.09.24	
4	Розділ 3. РОЗРОБКА ВЕБ-ПЛАТФОРМИ	26.09.24	
5	ВИСНОВКИ	7.11.24	
6	Оформлення (чистовий варіант)	10.11.24	
7	Подача кваліфікаційної роботи (проєкту) науковому керівнику для відгуку (за 14 днів до захисту)	11.11.24	
8	Подача кваліфікаційної роботи (проєкту) для рецензування (за 12 днів до захисту)	13.11.24	
9	Перевірка кваліфікаційної роботи (проєкту) на наявність ознак плагіату (за 10 днів до захисту)	15.11.24	
10	Подання кваліфікаційної роботи (проєкту) завідувачу кафедри (за 7 днів до захисту)	18.11.24	

З завданням ознайомлений:

Студент _____ Василь ВОЛОШИН
(підпис)

Науковий керівник _____ Вікторія РЕЗАНОВА
(підпис)

АНОТАЦІЯ

Волошин В.М. Впровадження штучного інтелекту в алгоритми веб-платформи на прикладі збору коштів на Збройні Сили України.

Кваліфікаційна робота за спеціальністю 122 – «Комп’ютерні науки» – Київський національний університет технологій та дизайну, Київ, 2024 рік.

Кваліфікаційна робота присвячена розробці веб-платформи для збору коштів на ЗСУ із використанням штучного інтелекту для надання персоналізованих рекомендацій. У роботі проведено аналіз існуючих платформ для збору коштів, визначено їхні недоліки та сформульовано вимоги до нової платформи. Розглянуто сучасні технології штучного інтелекту, зокрема впровадження чат-ботів, інтегрованих у веб-додаток. Представлено архітектурні рішення системи, включаючи вибір технологій для фронтенду (React) і бекенду (Nest.js), а також інтеграцію з OpenAI для реалізації рекомендаційних функцій. Детально описано реалізацію системи, а також результати тестування, що підтверджують її працездатність, продуктивність і відповідність вимогам.

Ключові слова: штучний інтелект, веб-платформа, збір коштів, React, Nest.js, OpenAI, рекомендаційна система.

ANNOTATION

Voloshyn V.M. Implementation of artificial intelligence in the algorithms of a web platform to raise funds for the Armed Forces of Ukraine.

Qualification work in specialty 122 - "Computer Science" - Kyiv National University of Technology and Design, Kyiv, 2024.

The qualification work is dedicated to the development of a web platform for fundraising for the Armed Forces of Ukraine, utilizing artificial intelligence to provide personalized recommendations. The study includes an analysis of existing fundraising platforms, identification of their limitations, and formulation of requirements for a new platform. Modern artificial intelligence technologies, including chatbot integration into the web application, are examined. Architectural solutions for the system are presented, including the choice of technologies for the frontend (React) and backend (Nest.js), as well as integration with OpenAI for implementing recommendation functions. The implementation of the system is detailed, along with testing results that confirm its functionality, performance, and compliance with the requirements.

Keywords: artificial intelligence, web platform, fundraising, React, Nest.js, OpenAI, recommendation system.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі, де інформаційні технології швидко розвиваються, цифрові платформи стають незамінними інструментами у багатьох сферах життя. Веб-платформи для збору коштів є одними з ключових рішень, які дозволяють ефективно мобілізувати ресурси для різних потреб. Особливо важливим є створення таких платформ для підтримки критично важливих ініціатив, як-от допомога Збройним Силам України. Існуючі рішення, хоча й пропонують базовий функціонал, мають обмеження, такі як відсутність персоналізованих рекомендацій, низька інтерактивність і незручність у використанні.

Впровадження штучного інтелекту (ШІ) у веб-застосунки дозволяє значно розширити можливості цих платформ. ШІ забезпечує персоналізацію, автоматизацію рутинних процесів і покращує комунікацію з користувачами. Особливо актуальним є використання чат-ботів, які можуть надавати рекомендації на основі запитів користувачів. Такий підхід дозволяє підвищити ефективність зборів і задовольнити потреби донорів у зручному та швидкому доступі до актуальної інформації про збори.

Таким чином, створення платформи, яка використовує сучасні технології ШІ для автоматизації взаємодії з користувачами та персоналізації рекомендацій, є актуальною задачею. Це дозволяє зробити внесок у розвиток технологій збору коштів, а також забезпечити підтримку військових, які захищають Україну.

Мета дослідження. Метою дослідження є розробка веб-платформи для збору коштів на підтримку Збройних Сил України із впровадженням штучного інтелекту. Основна увага приділяється створенню алгоритмів, які забезпечують персоналізовані рекомендації для користувачів, автоматизацію комунікації через чат-бота, а також інтеграцію з базами даних і зовнішніми сервісами для покращення ефективності зборів.

Завдання роботи. Основним завданням роботи є розробка веб-платформи для збору коштів на підтримку Збройних Сил України, яка використовує штучний інтелект для персоналізованих рекомендацій та автоматизації взаємодії з користувачами. Для досягнення цього передбачено проведення аналізу існуючих платформ для збору коштів та їх недоліків, визначення вимог до нової системи, а також дослідження сучасних технологій ШІ, які можуть бути інтегровані у веб-застосунки. Завданням також є проєктування архітектури платформи із врахуванням взаємодії між клієнтською і серверною частинами, реалізація ключових компонентів, таких як чат-бот з елементами штучного інтелекту та алгоритми для формування рекомендацій,

інтеграція із зовнішніми API та забезпечення відповідності платформи вимогам щодо продуктивності, точності і зручності. Завершальним етапом роботи є тестування платформи з метою оцінки її ефективності та відповідності поставленим цілям.

Об’єкт і предмет дослідження. Об’єктом дослідження є веб-платформи для збору коштів на Збройні Сили України. Предметом дослідження є впровадження штучного інтелекту для забезпечення персоналізованих рекомендацій і автоматизації взаємодії з користувачами.

Методи та засоби дослідження. У роботі використовувалися такі методи дослідження:

- Метод аналізу існуючих платформ для збору коштів.
- Методи проектування програмних систем, зокрема клієнт-серверної архітектури.
- Алгоритми штучного інтелекту для побудови персоналізованих рекомендацій.
- Інструменти для розробки веб-застосунків, такі як React, Nest.js, Prisma, та інтеграція з API.

Наукова новизна та практичне значення. Наукова новизна роботи полягає у використанні штучного інтелекту для персоналізації рекомендацій у платформі збору коштів. Вперше на веб-платформі запропоновано інтеграцію чат-бота з алгоритмами ШІ для персоналізованих рекомендацій зборів на ЗСУ.

Практичне значення роботи полягає у створенні інструменту, який може використовуватися для ефективного збору коштів на підтримку Збройних Сил України. Платформа забезпечує зручний інтерфейс, високу продуктивність і персоналізовані рекомендації, що сприяє залученню більшої кількості донорів.

Розділ 1 АНАЛІЗ ВЕБ-ЗАСТОСУНКІВ ТА ШТУЧНОГО ІНТЕЛЕКТУ

Сучасні веб-застосунки займають ключову роль у багатьох сферах суспільного життя, зокрема у благодійності, що є надзвичайно важливою для залучення фінансування та підтримки соціально значущих ініціатив. У контексті підтримки Збройних Сил України (ЗСУ) веб-платформи для збору коштів є необхідним інструментом, який дозволяє швидко організувати кампанії, залучати донорів та забезпечувати прозорість використання зібраних ресурсів. Однак, існуючі рішення, які використовуються для цієї мети, мають низку обмежень, що ускладнюють їх ефективне використання.

Впровадження штучного інтелекту у веб-застосунки відкриває нові можливості для автоматизації взаємодії з користувачами, персоналізації контенту та покращення загального користувацького досвіду. Аналіз поточних платформ збору коштів та сучасних технологій штучного інтелекту є основою для розробки інноваційного рішення, яке зможе подолати існуючі проблеми та забезпечити високий рівень продуктивності й зручності для користувачів.

1.1 Аналіз вимог до веб-платформи збору коштів на ЗСУ

1.1.1 Огляд поточних платформ для збору коштів

На сьогоднішній день існує низка платформ, які використовуються для організації зборів коштів на підтримку Збройних Сил України. Серед них можна виділити такі відомі ресурси, як КОЛО, Donate1024, Fund24 та Peoples Project. Ці платформи відіграють важливу роль у забезпеченні фінансової підтримки військових підрозділів, закупівлі необхідного обладнання, медикаментів та іншого оснащення для потреб фронту. Вони стали невід'ємною частиною сучасного волонтерського руху та дозволяють ефективно комунікувати з потенційними донорами.

Платформа КОЛО спеціалізується на проведенні кампаній для адресної допомоги, пропонуючи користувачам підтримати конкретні ініціативи, пов’язані з певними потребами військових. Donate1024 зосереджується на організації швидких та прозорих зборів коштів, надаючи мінімалістичний інтерфейс для зручності користувачів. Fund24 забезпечує підтримку широкого спектра ініціатив, дозволяючи організовувати різні кампанії через один ресурс, а Peoples Project працює як фонд, який об'єднує зусилля різних груп волонтерів.

Попри те, що ці платформи пропонують основний набір функцій для збору коштів, їхній функціонал залишається обмеженим. Наприклад, багато платформ орієнтовані на централізовані кампанії, які адмініструються організаціями або фондами. Це обмежує користувачів у виборі зборів, які відповідають їхнім

особистим інтересам чи пріоритетам. Користувачам пропонується лише підтримати ініціативи, обрані адміністраторами платформи, що звужує можливості для залучення до процесу донорства.

Іншим важливим обмеженням є відсутність інструментів для пошуку та фільтрації зборів. Наприклад, жодна з розглянутих платформ не надає можливості швидкого пошуку за ключовими словами або категоріями. Це змушує користувачів переглядати всі доступні збори вручну, що займає багато часу і може бути незручним. Відсутність фільтрації за географічним розташуванням, терміновістю чи популярністю також створює труднощі для користувачів, які хочуть швидко знайти найбільш актуальну інформацію.

Крім того, більшість платформ не мають зручного інтерфейсу для роботи зі списками зборів. Наприклад, відсутність пагінації або можливості вибору кількості елементів на сторінці створює проблеми при роботі з великими списками. Користувачі змушені прокручувати сторінки вручну, що значно знижує зручність взаємодії з платформою, особливо для тих, хто використовує мобільні пристрої.

Таким чином, хоча платформи для збору коштів на ЗСУ є важливим інструментом для підтримки військових, їхній функціонал потребує суттєвого вдосконалення. Розробка нових рішень, які враховують потреби користувачів, інтеграцію сучасних технологій, таких як штучний інтелект, та підвищують рівень персоналізації, є важливим напрямом для забезпечення ефективності зборів коштів. Це дозволить створити платформу, яка не тільки відповідає сучасним вимогам, але й забезпечує користувачам максимально комфортний досвід взаємодії.

1.1.2 Основні проблеми та обмеження існуючих рішень

Хоча сучасні веб-платформи для збору коштів на підтримку Збройних Сил України забезпечують базовий функціонал, вони мають низку суттєвих обмежень, які знижують їхню ефективність та зручність. Однією з головних проблем є відсутність персоналізованого підходу до користувачів. Поточні рішення часто не враховують індивідуальні потреби або інтереси донорів, що призводить до того, що користувач змушений самостійно переглядати велику кількість зборів у пошуках того, що його цікавить. Це може відбити бажання зробити внесок, особливо у тих, хто не готовий витрачати багато часу на пошуки.

Ще одним важливим обмеженням є відсутність пошукової системи або ефективних механізмів фільтрації. На багатьох платформах користувач не може здійснити пошук зборів за ключовими словами, географічним розташуванням, категоріями чи іншими критеріями. Крім того, не передбачено функції сортування

зборів, наприклад, за новизною, популярністю або терміновістю потреби у фінансуванні. Це значно ускладнює процес взаємодії із платформою.

Багато платформ також не мають можливості пагінації — розбиття списку зборів на сторінки. У результаті користувачеві доводиться прокручувати довгі списки зборів, що знижує зручність використання. Відсутність вибору кількості відображених елементів на сторінці додає додаткових труднощів, особливо на мобільних пристроях.

Крім того, багато платформ є прив’язаними до конкретних фондів або цілей. Наприклад, деякі з них підтримують лише конкретні кампанії або збори, що унеможливлює вибір інших варіантів, які могли б більше відповідати інтересам донорів. Це звужує можливості користувача і знижує його зацікавленість у використанні таких платформ.

Ще однією проблемою є загальна низька інтерактивність веб-платформ. Відсутність інтеграції сучасних інструментів, таких як чат-боти, які могли б допомогти користувачеві знайти потрібний збір або надати інформацію про доступні кампанії, робить взаємодію менш ефективною.

Таким чином, поточні рішення потребують суттєвого вдосконалення, зокрема у частині підвищення зручності використання, персоналізації контенту та розширення функціоналу. Це дозволить зробити взаємодію з платформами більш комфортною для широкого кола користувачів.

1.1.3 Вимоги до нової платформи

Розробка нової платформи для збору коштів на Збройні Сили України повинна враховувати сучасні виклики і забезпечити рішення основних проблем, які властиві існуючим платформам. Одним із ключових аспектів є впровадження персоналізованих рекомендацій. Система повинна автоматично аналізувати вподобання користувачів, їхні попередні дії та запити, щоб надавати найбільш релевантні пропозиції. Наприклад, якщо користувач цікавиться збором для певного підрозділу або регіону, система повинна пропонувати саме такі збори.

Необхідно також інтегрувати пошуковий функціонал, який дозволить користувачам швидко знаходити збори за ключовими словами, категоріями, або іншими параметрами. Фільтрація за популярністю, терміновістю та іншими характеристиками повинна бути частиною базового функціоналу. Це дозволить користувачам отримувати доступ до інформації більш ефективно.

Функція пагінації повинна бути обов'язковою для забезпечення зручності перегляду довгих списків зборів. Користувачі повинні мати можливість самостійно обирати кількість відображуваних елементів на сторінці, наприклад

10, 25 чи 50 зборів. Це особливо важливо для тих, хто використовує платформу на пристроях з невеликим екраном, таких як смартфони або планшети.

Іншим важливим аспектом є незалежність платформи від конкретних фондів. Користувач повинен мати можливість обирати між різними зборами на основі своїх вподобань, а не лише підтримувати кампанії, визначені адміністраторами платформи.

Крім того, платформа повинна забезпечувати високий рівень інтерактивності. Наприклад, інтеграція чат-бота з елементами штучного інтелекту дозволить користувачам отримувати рекомендації, відповіді на запитання та іншу корисну інформацію у реальному часі.

Вимоги до продуктивності системи є не менш важливими. Швидкість відповіді сервера на запит зборів повинна бути максимальною та не перевищувати 150 мс , а час завантаження сторінок — не перевищувати 500 мс. Це забезпечить комфортну роботу навіть при значному навантаженні.

Таким чином, нова платформа повинна поєднувати сучасний функціонал, високу продуктивність та інноваційні рішення для забезпечення зручності та ефективності збору коштів.

1.2 Аналіз сучасних технологій штучного інтелекту у веб-застосунках

Штучний інтелект (ШІ) є одним із найважливіших досягнень сучасної науки, що стрімко змінює всі аспекти життя, зокрема сферу веб-застосунків. Завдяки своїм можливостям, таким як аналіз великих обсягів даних, автоматизація рутинних процесів та персоналізація взаємодії, ШІ став ключовим елементом для покращення користувацького досвіду. У цьому розділі розглянемо загальні відомості про ШІ, його використання у веб-застосунках, а також специфіку впровадження чат-ботів на основі ШІ у веб-платформи.

1.2.1 Загальні відомості про штучний інтелект

Штучний інтелект (ШІ) — це галузь інформатики, яка дозволяє створювати системи, здатні імітувати людський інтелект. ШІ охоплює такі напрямки, як машинне навчання, обробка природної мови (NLP), комп’ютерний зір, генерація контенту та багато інших. Завдяки швидкому розвитку, ШІ трансформував численні сфери життя, включаючи бізнес, медицину, освіту, розваги та веб-застосунки.

Однією з найпоширеніших технологій є алгоритми машинного навчання, які дозволяють обробляти величезні обсяги даних, знаходити в них закономірності та використовувати ці знання для прийняття рішень. Наприклад, такі алгоритми застосовуються в системах персоналізованих рекомендацій на веб-платформах,

пошукових системах, а також у додатках для прогнозування. На рисунку 1.1 зображено основні сфери використання ШІ у сучасному світі.

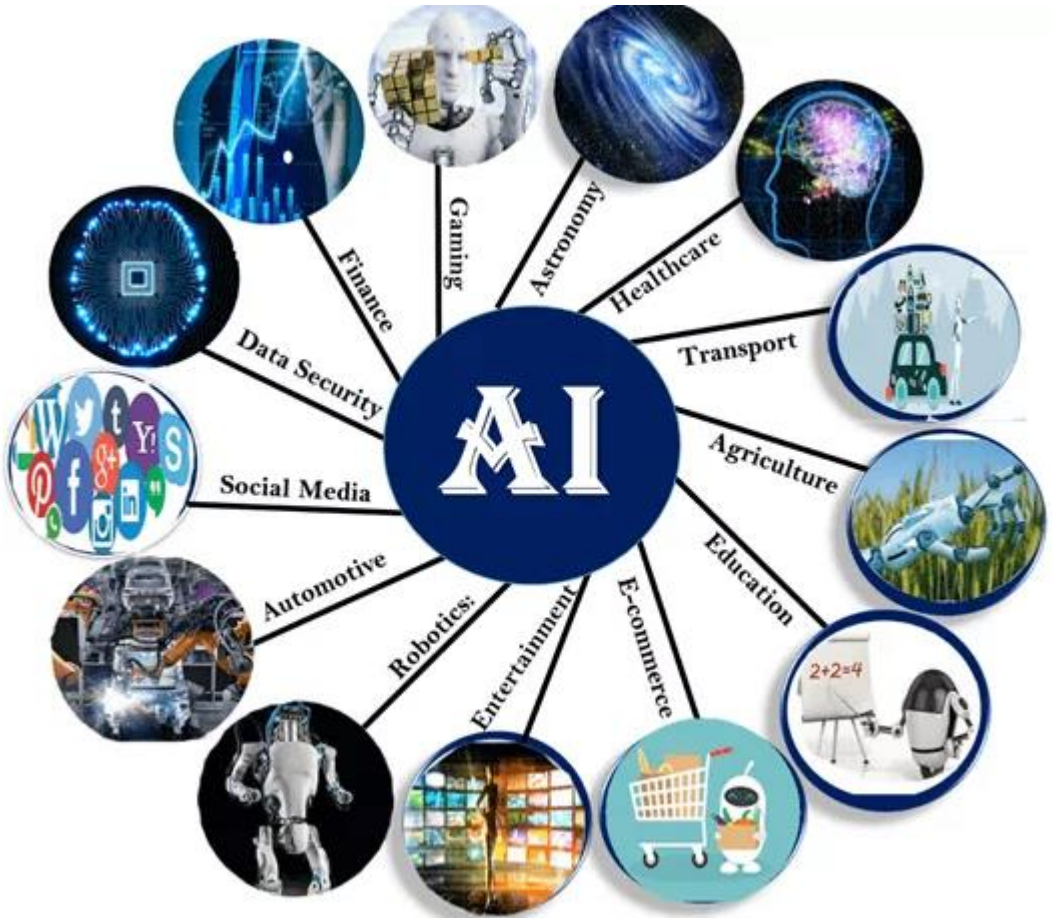


Рисунок 1.1. Основні сфери використання ШІ

Сучасні ШІ-інструменти здатні виконувати широкий спектр завдань, включаючи автоматизацію рутинних процесів, аналіз даних, генерування текстів, зображень і навіть відео. Наприклад, моделі, такі як ChatGPT від OpenAI, здатні вести діалог, відповідати на складні запитання, створювати технічну документацію, допомагати у написанні кодів і багато іншого.

Серед найбільш популярних ШІ-рішень — чат-боти, такі як ChatGPT, які здатні обробляти текстові запити користувачів у реальному часі. Завдяки обробці природної мови (NLP), такі боти не лише відповідають на питання, але й можуть проводити контекстний аналіз, пропонувати персоналізовані рішення, а також формувати детальні відповіді. Наприклад, у контексті платформи збору коштів, бот може аналізувати вподобання користувачів, допомагаючи їм знайти найбільш відповідні збори.

Іншою важливою функцією ШІ є генерація контенту. Наприклад, такі моделі, як DALL-E, також від OpenAI, дозволяють створювати зображення на

основі текстового опису. Це відкриває нові можливості для дизайну, реклами та творчих проектів. Аналогічно, інші інструменти дозволяють створювати відео або аудіо, адаптовані під конкретні потреби.

Серед популярних платформ, які активно використовують ШІ, можна виділити Google Cloud AI, Amazon AWS AI, Microsoft Azure AI та OpenAI. Кожна з цих платформ надає широкий спектр послуг, таких як аналіз даних, розробка моделей машинного навчання, автоматизація рутинних процесів і навіть розробка власних ШІ-додатків. Перевагою цих платформ є їхня масштабованість, інтеграція з іншими сервісами та наявність готових рішень для різних галузей.

Ще одним цікавим прикладом є MidJourney — ШІ-сервіс для генерації зображень, який здатний створювати високоякісні візуальні матеріали на основі текстових запитів. Його можливості активно використовуються в маркетингу, мистецтві та розробці контенту.

Окрім текстових та візуальних функцій, ШІ також застосовується для аналізу відео та аудіо. Наприклад, такі системи можуть розпізнавати обличчя, аналізувати рух або генерувати мову на основі тексту. Ці технології широко використовуються у веб-застосунках для автоматизації відеомоніторингу, створення відеороликів або розробки інтерактивних презентацій.

Відмінною рисою сучасного ШІ є його гнучкість і універсальність. Завдяки відкритим API та сервісам, будь-який розробник може інтегрувати ШІ у свій проект, отримуючи доступ до технологій, які раніше були доступні лише великим корпораціям. Це сприяє швидкому впровадженню інновацій у різних галузях, зокрема у веб-застосунках.

У сучасному світі ШІ виконує не тільки допоміжну, але й ключову роль у багатьох процесах. Наприклад, такі сервіси, як ChatGPT, стають незамінними для автоматизації роботи із запитами користувачів, генерації контенту та навіть навчання. Їх використання дозволяє значно скоротити витрати часу та ресурсів, підвищуючи продуктивність і ефективність.

Провідні розробники штучного інтелекту пропонують найсучасніші моделі ШІ через хмарні сервіси. OpenAI, наприклад, забезпечує доступ до десятків потужних мовних моделей, спеціалізованих на чаті, обробці природної мови (NLP), створенні зображень і програмного коду, які інтегровані в платформу Azure. Nvidia, у свою чергу, використовує підхід, орієнтований на хмарну інфраструктуру, пропонуючи базові моделі ШІ для роботи з текстами, зображеннями та медичними даними, доступні через різних хмарних провайдерів. Крім того, сотні інших компаній розробляють моделі, адаптовані для специфічних галузей та унікальних сценаріїв використання.

Таким чином, ШІ є одним із найперспективніших напрямків розвитку технологій, які впливають на всі аспекти суспільного життя. Його використання та впровадження у різні сфери забезпечує зручність, швидкість та інноваційність, що є важливими факторами для створення конкурентоспроможних продуктів.

1.2.2 Використання штучного інтелекту у веб-застосунках

Штучний інтелект уже активно використовується у багатьох веб-застосунках для вирішення різних задач, таких як персоналізація контенту, рекомендаційні системи, аналітика користувацької поведінки та автоматизація підтримки клієнтів.

Одним із найяскравіших прикладів є рекомендаційні системи, які дозволяють пропонувати користувачам релевантний контент на основі їхньої поведінки. Наприклад, такі системи широко використовуються у стримінгових сервісах, як-от Netflix чи Spotify, а також у платформах електронної комерції, таких як Amazon.

Важливою перевагою ШІ у веб-застосунках є можливість автоматизації рутинних завдань. Наприклад, чат-боти на основі ШІ здатні замінити операторів технічної підтримки, відповідаючи на запити користувачів у режимі реального часу. Це значно скорочує витрати компаній і покращує досвід користувачів.

Штучний інтелект (ШІ) стає невід'ємною частиною сучасних веб-застосунків, надаючи широкі можливості для автоматизації, персоналізації та аналітики. Завдяки технологіям ШІ розробники можуть створювати сервіси, які не лише полегшують взаємодію користувачів із системами, але й відкривають нові горизонти функціональності.

Одним із найважливіших аспектів використання ШІ є аналіз даних. Великі обсяги інформації, які генеруються користувачами, можуть бути оброблені й проаналізовані для виявлення закономірностей, прогнозування поведінки чи виявлення аномалій. Наприклад, у сфері електронної комерції ШІ допомагає створювати персоналізовані рекомендації, аналізуючи історію покупок користувача, його уподобання та поведінкові дані. У веб-платформах для збору коштів такі технології можуть використовуватися для визначення найбільш релевантних зборів для користувачів на основі їхніх попередніх внесків чи інтересів.

1.2.3 Впровадження чат-ботів на основі ШІ у веб-платформи

Одним із найпопулярніших напрямів використання ШІ у веб-застосунках є чат-боти. Вони забезпечують зручний і швидкий спосіб взаємодії користувачів із системами, автоматизуючи багато рутинних завдань. Чат-боти на базі ШІ

використовують обробку природної мови (NLP), що дозволяє їм розуміти запити користувачів, аналізувати їхній контекст і формувати відповідні відповіді.

На відміну від традиційних ботів із жорстко заданими сценаріями, чат-боти, побудовані на основі сучасних мовних моделей, таких як ChatGPT, здатні вести більш складний діалог, адаптуючись до потреб користувача. Вони можуть не лише відповідати на запитання, а й надавати поради, генерувати контент, допомагати у прийнятті рішень або навіть аналізувати емоційний стан співрозмовника.

Використання чат-ботів у веб-застосунках має численні переваги:

- **Цілодобова доступність:** Чат-боти працюють 24/7, забезпечуючи підтримку користувачів у будь-який час.
- **Швидкість відповіді:** Завдяки інтеграції з потужними мовними моделями, відповіді генеруються практично миттєво.
- **Персоналізація:** Аналізуючи дані про користувачів, чат-боти можуть пропонувати індивідуальні рішення або рекомендації.
- **Масштабованість:** Один чат-бот може обслуговувати необмежену кількість користувачів одночасно, що значно знижує витрати.

У контексті веб-платформ для збору коштів чат-боти можуть виконувати такі завдання:

1. **Допомога у виборі зборів:** Аналізуючи побажання користувачів, бот пропонує найбільш релевантні збори.
2. **Надання інформації:** Бот відповідає на запитання про кампанії, їхній прогрес або способи внесення коштів.
3. **Персоналізовані рекомендації:** На основі історії попередніх внесків бот формує індивідуальні пропозиції.
4. **Зворотний зв'язок:** Користувачі можуть залишати відгуки чи пропозиції через інтерактивні форми, оброблені ботом.

На рисунку 1.2 представлено загальну схему роботи чат-бота, інтегрованого у веб-платформу.

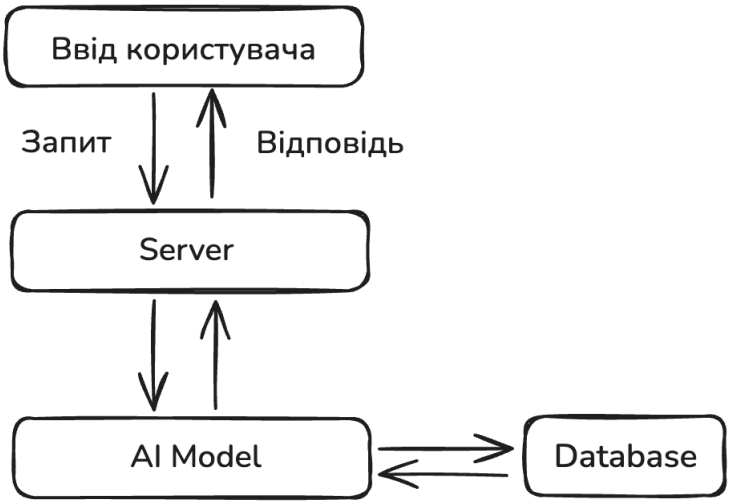


Рис. 1.2 Загальна схема роботи чат-бота

Таким чином, чат-боти на основі ШІ є важливим елементом сучасних веб-застосунків. Їх використання дозволяє значно покращити якість обслуговування, знизити витрати на підтримку користувачів та забезпечити високий рівень персоналізації. У поєднанні з іншими технологіями ШІ вони відкривають нові можливості для створення ефективних, інтерактивних і зручних веб-платформ.

1.3 Висновок до розділу

У цьому розділі проведено всебічний аналіз вимог до веб-платформи збору коштів для Збройних Сил України, а також розглянуто сучасні технології штучного інтелекту, які можуть бути використані у веб-застосунках.

Аналіз поточних платформ для збору коштів показав, що багато з них мають обмежену функціональність. Відсутність пошуку, сортування або персоналізації робить ці рішення менш зручними для користувачів. Більшість платформ зосереджені на підтримці конкретних зборів, що ускладнює вибір збору за індивідуальними вподобаннями. На основі цього аналізу було визначено основні проблеми: недостатня адаптивність інтерфейсу, низька швидкість обробки запитів і відсутність інтеграції з сучасними сервісами для покращення користувацького досвіду. Ці обмеження стали основою для формування вимог до нової платформи, яка має забезпечувати інтуїтивний інтерфейс, високий рівень продуктивності та можливість персоналізації зборів за допомогою інтегрованого штучного інтелекту.

Дослідження сучасних технологій штучного інтелекту підтвердило їхню важливість для реалізації персоналізованих рішень. Штучний інтелект сьогодні є ключовою складовою для аналізу даних, автоматизації процесів і взаємодії з

користувачами в режимі реального часу. Особлива увага була приділена використанню AI у вигляді чат-ботів, які дозволяють користувачам отримувати швидкі та релевантні відповіді на свої запити. Впровадження подібних технологій у платформу ZSU-JARS сприяє її конкурентоспроможності та значно покращує взаємодію з користувачем.

Таким чином, проведений аналіз підтвердив необхідність створення платформи, яка буде задовольняти сучасні вимоги до веб-застосунків і використовувати технології штучного інтелекту для надання користувачам персоналізованого досвіду. Результати аналізу дали чітке розуміння того, як повинна виглядати і функціонувати платформа, щоб забезпечити її ефективність, зручність та інноваційність.

Розділ 2 АРХІТЕКТУРНІ РІШЕННЯ ВЕБ-ПЛАТФОРМИ

Архітектура веб-платформи є основою для забезпечення її функціональності, продуктивності та масштабованості. У процесі розробки веб-застосунків важливо вибрати оптимальну архітектуру, яка задовольняє вимоги проекту, забезпечує зручність підтримки та відповідає сучасним тенденціям у сфері інформаційних технологій. У цьому розділі розглянуто популярні види архітектур, їхні переваги та недоліки, а також загальну архітектуру розроблюваної платформи.

2.1 Види архітектур

2.1.1 Архітектури цілого застосунку.

Ці архітектури визначають загальний підхід до проектування системи, включаючи взаємодію між клієнтською частиною (фронтенд), серверною частиною (бекенд) та іншими компонентами, такими як база даних чи зовнішні API. Вони охоплюють усю систему і забезпечують її структурну цілісність. До цієї категорії належать: монолітна архітектура, клієнт-серверна архітектура, мікросервісна архітектура, архітектура, орієнтована на події, serverless-архітектура.

Ці підходи визначають, як основні компоненти системи взаємодіють, як розподіляється навантаження, як забезпечується масштабованість і продуктивність. Вони використовуються для побудови інфраструктури та організації загальної логіки роботи застосунку.

Монолітна архітектура. Ця архітектура передбачає створення єдиного застосунку, який об'єднує всі компоненти системи: фронтенд, бекенд, базу даних і бізнес-логіку. Цей підхід є простим у розробці, тестуванні та розгортанні. Однак, монолітні застосунки мають низьку гнучкість і складність у масштабуванні. Вони підходять для невеликих проектів, де немає необхідності в розподілі навантаження.

Клієнт-серверна архітектура. Цей підхід є одним із найпопулярніших для розробки сучасних веб-застосунків. Система складається з двох основних частин: клієнта (фронтенд) та сервера (бекенд). Клієнт відповідає за відображення інтерфейсу користувача та взаємодію з ним, а сервер виконує всі бізнес-операції, обробляє запити та забезпечує доступ до бази даних.

Основними перевагами клієнт-серверної архітектури є чіткий розподіл відповідальності, можливість незалежної розробки клієнтської та серверної частин, а також легкість інтеграції з іншими системами. Такий підхід забезпечує

високу продуктивність і масштабованість, що робить його ідеальним для середніх і великих проектів.

Мікросервісна архітектура. Мікросервіси дозволяють розділити систему на незалежні модулі, кожен із яких виконує певну функцію. Цей підхід забезпечує високу гнучкість і простоту масштабування. Наприклад, кожен сервіс може бути розгорнутий і масштабований окремо, залежно від навантаження. Однак мікросервіси мають складнішу реалізацію, вимагають продуманого механізму взаємодії між сервісами (наприклад, REST API чи GraphQL) і значних ресурсів для підтримки. На рисунку 2.1 наведено різницю між мікросервісною та монолітною архітектурами.

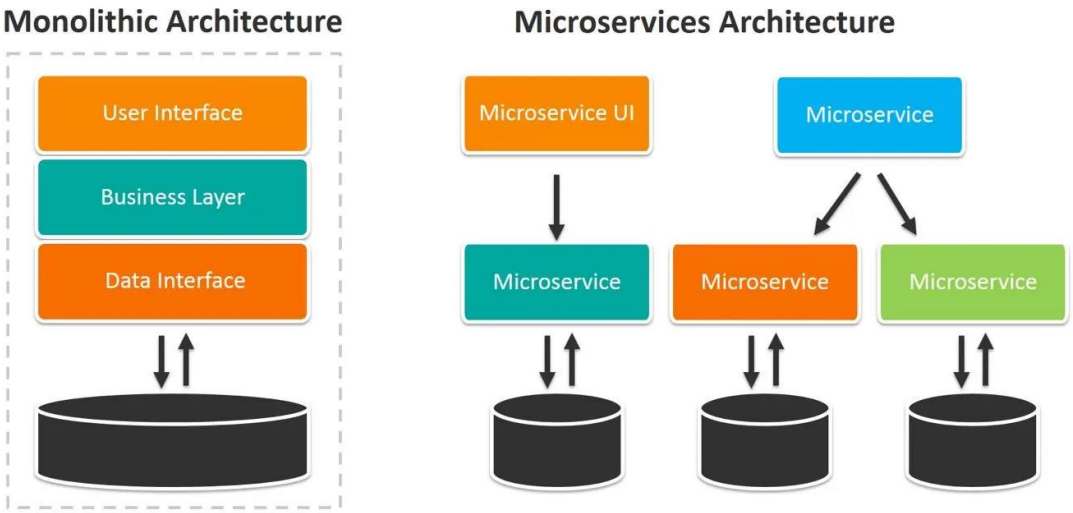


Рис.2.1 Вигляд монолітної та мікросервісної архітектури

Веб-платформи, які очікують високий рівень навантаження або потребують частого оновлення окремих компонентів, обирають мікросервісний підхід.

Архітектура, орієнтована на події (Event-Driven Architecture). У цій архітектурі основна увага приділяється обробці подій, які генеруються різними компонентами системи. Події обробляються асинхронно, що дозволяє досягти високої швидкодії та продуктивності. Така архітектура підходить для систем із великою кількістю одночасних запитів і подій, наприклад, у фінансових сервісах або системах реального часу.

Serverless-архітектура. Serverless (безсерверна) архітектура передбачає використання функцій як сервісу (FaaS). У цьому підході розробники пишуть код окремих функцій, а вся інфраструктура (сервери, масштабування, балансування навантаження) управляється хмарними провайдерами, такими як AWS Lambda чи Azure Functions. Цей підхід забезпечує низькі витрати на підтримку та високу масштабованість, але має обмеження у продуктивності для складних обчислень.

2.1.2 Архітектури організації коду

Ці архітектури зосереджені на тому, як структуровано код і папки у фронтенд чи бекенд частині системи. Вони забезпечують легкість навігації, підтримки та розширення проекту, а також знижують ризик появи "спагеті-коду". До цієї категорії належать:

- Feature-Slice Design (FSD). Підхід до організації коду за фічами (функціональними областями), де кожна фіча має власні сегменти: компоненти, моделі, сервіси тощо.
- Модульна (проста) архітектура. Традиційна організація коду, де компоненти розподіляються за технічними аспектами, такими як "components", "services", "utils".
- Atomic Design. Підхід для фронтенду, що організовує компоненти інтерфейсу за рівнями: атоми, молекули, організми, шаблони.
- Логічна структура за шарами. Використання шарів, таких як "data", "domain", "presentation", для розділення логіки та відповідальностей.

Ці архітектури спрямовані на внутрішню організацію проекту, знижують залежності між компонентами, полегшують роботу команди та прискорюють додавання нового функціоналу. Нижче наведений більш детальний опис кожної з архітектур.

Feature-sliced архітектура. Архітектура, побудована за принципами Feature-Sliced Design (FSD), набирає популярності серед фронтенд-розробників завдяки своїм численним перевагам. Вона визнана надійною, чудово масштабується і відмінно підходить для роботи великих команд, особливо якщо мова йде про розробку складних і масштабних проектів.

Однією з головних особливостей FSD, яка вирізняє її серед інших архітектурних підходів у фронтенді, є орієнтація на бізнес-завдання. Це робить архітектуру універсальною для різних проектів, забезпечуючи структурований підхід до розвитку функціональності. Ще однією важливою перевагою є можливість поступового впровадження FSD, що дозволяє інтегрувати її навіть у вже існуючі продукти. Проте слід пам’ятати, що цей підхід був спеціально розроблений для застосування у фронтенд-розробці.

У FSD-архітектурі структура проекту базується на трьох основних рівнях організації: рівні (layers), зрізи (slices) та сегменти (segments). Кожен рівень складається із функціональних зрізів, а ті, своєю чергою, поділяються на сегменти, які виконують певні завдання. Перед тим як глибше розглянути

концепції шарів, зрізів і сегментів, корисно оцінити, як ця архітектура виглядає загалом , це показано на рисунку 2.2

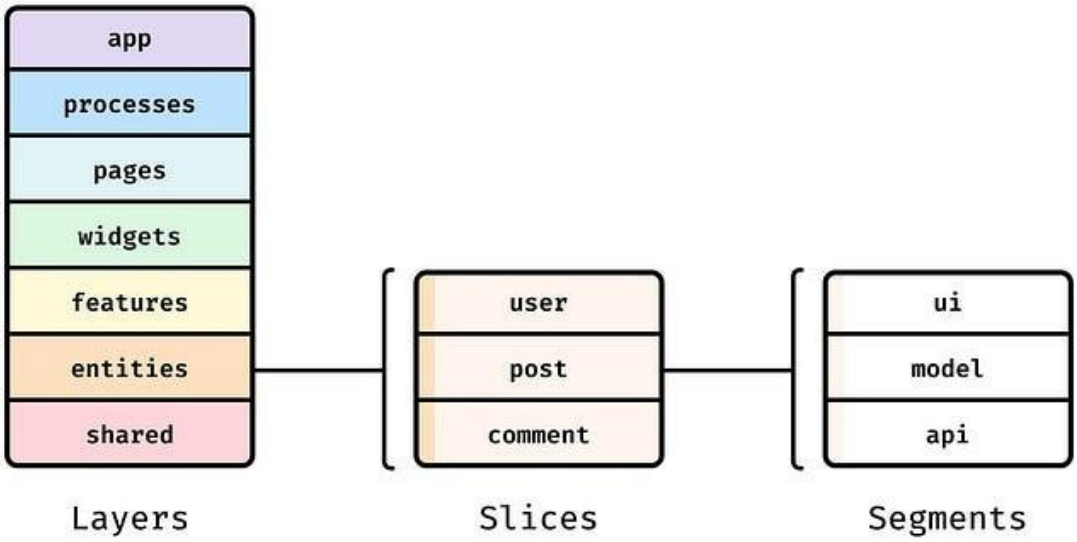


Рис.2.2 FSD архітектура

Структура FSD виглядає наступним чином

Сегменти: Відповідають за конкретні аспекти реалізації модуля, такі як:

- api/ — робота з API.
- config/ — конфігурація модуля.
- lib/ — утилітарні функції та допоміжні бібліотеки.
- model/ — бізнес-логіка: store, actions, effects, reducers тощо.
- ui/ — відображення: компоненти інтерфейсу.

Слайси. Групують файли, що реалізують певну функціональність бізнес-логіки. Це можуть бути папки, які об'єднують модулі за їх призначенням.

Шари. Визначають рівень відповідальності та впливу змін. Чим вище шар, тим більша його відповідальність та знання про інші шари. Нижчі шари більш абстрактні та використовуються у верхніх шарах, тому зміни в них можуть мати ширший вплив.

Переваги FSD:

- Логіка кожного модуля ізольована, що полегшує навігацію та розуміння коду.
- Мінімізація зв'язків між різними частинами коду зменшує ризик виникнення помилок.
- Різні модулі можуть розроблятися паралельно без конфліктів, що підвищує ефективність команди.

- Завдяки ізоляції логіки, зміни в одному модулі не впливають на інші, що спрощує процес рефакторингу.

Недоліки FSD:

- Новим розробникам може знадобитися більше часу для розуміння структури проекту.
- У деяких випадках можуть виникати схожі функції або утиліти в різних модулях, що потребує перенесення їх у спільну область.
- Для невеликих застосунків така структура може бути надмірною.

Впровадження Feature-Sliced Design може значно покращити якість коду, підвищити ефективність розробки та спростити підтримку проекту, особливо у великих та складних системах.

Модульна (проста) архітектура. Це підхід до розробки програмного забезпечення, при якому застосунок розбивається на незалежні компоненти — модулі. Кожен модуль виконує певну функцію та має власну логіку та інтерфейс для взаємодії з іншими модулями. Таким чином, розробляючи проєкт за цим підходом, ви "будуєте" свій продукт з окремих "цеглинок". Звичайно, модульна архітектура — це термін, який використовується не лише в розробці програмного забезпечення. Продовжуючи аналогію з будівництвом: прикладами модульних систем можуть служити модульні будівлі, сонячні батареї, вітряні турбіни. Модульні конструкції поєднують у собі переваги стандартизації з перевагами налаштування, так що модульний підхід до архітектури дозволяє розробляти застосунки в більш масштабованому та модульному стилі.

Інтерфейс користувача (UI). Інтерфейс користувача складається з компонентів, які використовуються найчастіше, і є своєрідним проектним UI-китом. Він служить основою для створення інших шарів проекту, проте не повинен містити бізнес-логіку. Це дозволяє уникнути складнощів і помилок під час повторного використання компонентів з цього модуля, зберігаючи їх простоту та стабільність.

Компоненти. Компоненти, що знаходяться в цьому каталозі, є окремими елементами інтерфейсу користувача. Хоча вони можуть включати базову бізнес-логіку, бажано зберігати їх якомога простішими. Ці компоненти виконують роль вищого рівня абстракції інтерфейсу, забезпечуючи його функціональність і естетичність.

Модулі. Кожен модуль повинен бути максимально автономним і мати чітко визначену відповідальність. Усі необхідні функції, включаючи утиліти, сховища даних, константи та API для взаємодії, мають бути оголошені й використовуватися лише в межах модуля. Це дозволяє уникнути дублювання

коду, забезпечує простоту підтримки проекту та знижує ризик функціональної плутанини. Таким чином, модулі створюють добре структуровану систему з високою узгодженістю всередині себе і мінімальною залежністю від інших частин проекту.

Сторінки. Вони є комбінацією модулів і слугують найвищим рівнем організації проекту. Вони можуть бути поділені на підкаталоги для полегшення навігації. Основна різниця між сторінками та модулями полягає в обсязі бізнес-логіки: сторінки мають бути якомога більш спрощеними, а вся бізнес-логіка повинна бути розміщена в модулях. Це дозволяє використовувати ту саму логіку на різних сторінках, забезпечуючи структурованість і зменшуючи складність проекту.

Як і будь-який інший архітектурний підхід, модульний має свої сильні сторони та обмеження.

Переваги:

- Завдяки концепції "публічного API" модулі чітко ізольовані, що забезпечує простоту взаємодії.
- Інформація рухається від сторінок до модулів, далі до компонентів, і, врешті, до інтерфейсу користувача.
- Чіткий розподіл шарів дозволяє легко використовувати елементи проекту повторно.
- Чітка структура модулів спрощує внесення змін і оновлення проекту.
- Модулі працюють автономно, зберігаючи мінімальну залежність один від одного.

Недоліки:

- Іноді складно зрозуміти, чи варто розміщувати певну функціональність у модулі чи в компоненті
- Можливі випадки, коли один модуль потребує іншого, що може викликати труднощі в організації.
- У деяких випадках бізнес-логіка може виявитися "зайвою" для модулів або компонентів.
- Зв’язок з глобальними змінними та помічниками може бути складним.

Хоча цей підхід має деякі очевидні переваги, він все ще не підходить для великих і складних проектів. Коли з'являється багато функцій та бізнес-логіки, стає досить складно підтримувати її за допомогою цієї архітектури.

Atomic Design. Це методологія організації інтерфейсних компонентів у фронтенді, що ґрунтується на принципах композиції. Ідея полягає у поділі компонентів на п'ять рівнів: атоми, молекули, організми, шаблони та сторінки. Рівні Atomic Design:

- Атоми (Atoms): Найменші одиниці інтерфейсу, такі як кнопки, текстові поля, іконки. Вони не мають складної логіки й є базовими будівельними блоками. Приклад: <Button />, <Input />, <Icon />.
- Молекули (Molecules): Комбінації атомів, які створюють простіші елементи інтерфейсу. Наприклад, текстове поле з кнопкою. Приклад: <SearchBar />, яке складається з <Input /> і <Button />.
- Організми (Organisms): Складні компоненти, що об'єднують молекули й атоми для створення більш функціональних блоків. Приклад: <Header />, яке містить навігацію, логотип і поле пошуку.
- Шаблони (Templates): Макети, які організують організми в готові розташування на сторінці. Це може бути шаблон сторінки із заголовком, контентною частиною та футером.
- Сторінки (Pages): Повноцінні сторінки, які включають в себе шаблони з реальними даними.

Переваги:

- Систематичність і зрозумілість організації компонентів.
- Простота у створенні дизайн-систем.
- Легкість повторного використання компонентів.

Недоліки:

- Може бути зайвим для малих проектів.
- Вимагає детального планування на початкових етапах.
- Логічна структура за шарами

Цей підхід особливо популярний у розробці дизайн-систем і забезпечує систематичність підходу до UI.

Логічна структура за шарами. Цей підхід передбачає поділ проекту на логічні шари, які виконують різні функції. Основна мета — розмежування відповідальностей і зменшення зв'язків між частинами системи.

Шари у логічній структурі:

1. Presentation (Презентаційний шар): Відповідає за інтерфейс користувача та взаємодію з ним. У фронтенді це UI-компоненти, які відповідають за відображення та введення даних. Приклад: React-компоненти, CSS-стилі.

2. Domain (Шар бізнес-логіки): Містить бізнес-правила та основну логіку роботи програми. Цей шар не залежить від UI чи джерел даних. Приклад: Сервіси, моделі.

3. Data (Шар даних): Відповідає за зберігання, обробку та отримання даних. Включає запити до бази даних, API або сторонніх сервісів. Приклад: API-запити, функції для роботи з базою даних.

Переваги:

- Чітке розмежування відповідальностей.
- Легкість внесення змін у певний шар без впливу на інші.
- Легкість у тестуванні.

Недоліки:

- Може вимагати більше часу для налаштування.
- Підвищує складність невеликих проектів.

2.2 Загальна архітектура системи

Для розробки веб-платформи ZSU-JARS було обрано клієнт-серверну архітектуру через її численні переваги та відповідність вимогам проекту. Цей підхід дозволяє чітко розмежувати функціональні зони між клієнтською та серверною частинами, що підвищує зручність розробки, масштабованість і підтримуваність системи.

Клієнт-серверна архітектура передбачає розподіл функцій між клієнтом (інтерфейс користувача) і сервером (логіка та управління даними). Клієнт забезпечує зручний і адаптивний інтерфейс, тоді як сервер обробляє запити, виконує бізнес-логіку і надає доступ до бази даних. Такий підхід дозволяє розробляти, тестувати і масштабувати кожну частину системи незалежно, що є важливим у великих проектах із динамічними вимогами. На рисунку 2.3 представлена архітектурна схема веб-платформи з елементами штучного інтелекту. Двійними стрілками на схемі показано рух інформаційних потоків між клієнтом (користувачем), сервером, базою даних та системою OpenAI.

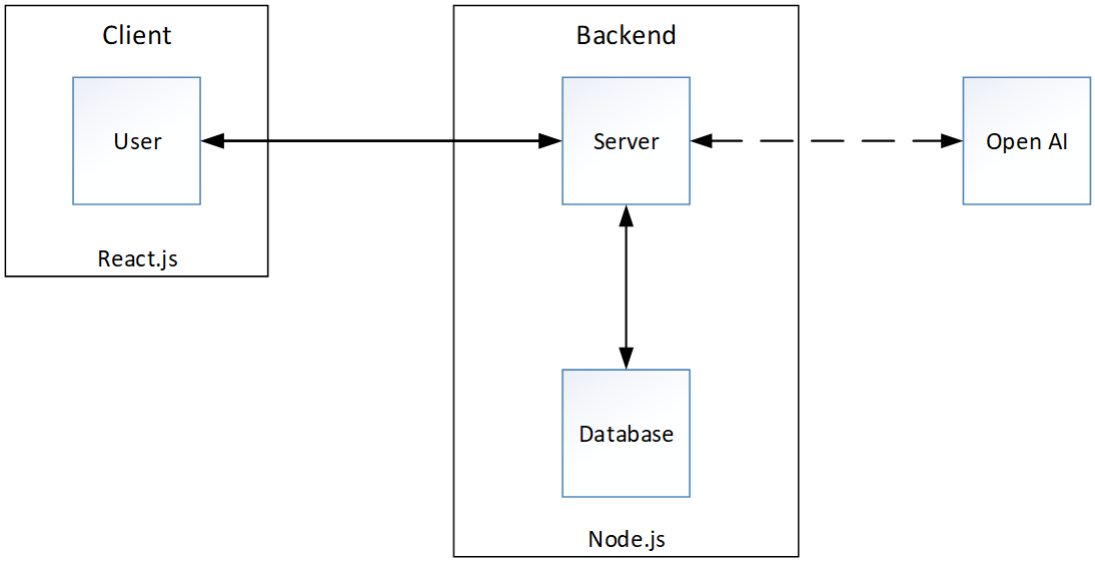


Рис. 2.3 Клієнт-серверна архітектура застосунку

Основними причинами вибору клієнт-серверної архітектури для ZSU-JARS є:

- Чіткий розподіл обов'язків. Клієнт відповідає за взаємодію з користувачем, а сервер — за обробку запитів і зберігання даних.
- Масштабованість. Клієнт і сервер можна оптимізувати або масштабувати окремо, залежно від навантаження.
- Можливість інтеграції: Легка інтеграція з іншими сервісами та API, такими як Monobank API, для отримання інформації про збір.
- Гнучкість у розробці. Різні команди можуть працювати паралельно над клієнтською та серверною частинами.

2.2.1 Архітектура бекенд-частини платформи

У бекенд-частині платформи використано просту архітектуру організації коду. Цей підхід забезпечує інтуїтивну організацію коду, що спрощує розробку, підтримку та розширення системи. Основними причинами вибору цієї архітектури є:

- Прозорість структури. Розробники легко знаходять потрібний код, оскільки він організований за зрозумілими категоріями, такими як "services", "controllers", "models".
- Легкість навігації. Простота організації папок дозволяє швидко орієнтуватися навіть новим членам команди.
- Гнучкість. Додавання нових функцій чи модулів не порушує загальної структури.
- Мінімізація дублювання коду. Кожен компонент має чітке місце розташування, що запобігає дублюванню.

На рисунку 2.4 зображено структуру папок бекенд-частини платформи ZSU-JARS.

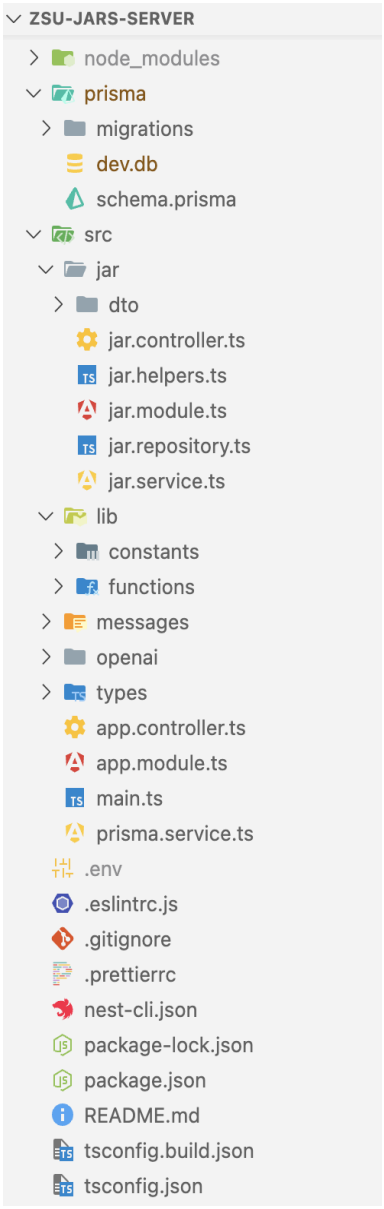


Рисунок 2.4. Структура папок бекенд-частини платформи

Основні папки:

Prisma/ - містить файли, пов'язані з базою даних, зокрема:

schema.prisma — описує схему бази даних, включаючи моделі та зв’язки між ними.

migrations/ — зберігає історію змін у базі даних для підтримки узгодженості структури даних.

src/ - основна папка з кодом застосунку. Вона містить такі підпапки:

jar/ - модуль для роботи із сутністю "Jar".

messages/ - модуль для роботи із сутністю "ChatMessage".

openai/ - модуль для роботи з AI

lib/

constants/ — константи, які використовуються у проекті.

functions/ — загальні функції.

types/ — визначення типів TypeScript, які використовуються в проекті.

2.2.2 Архітектура фронтенд-частини платформи

У фронтенд-частині платформи також використано просту архітектуру організації коду. Ця структура була обрана через її адаптивність до потреб проекту, зручність у розробці та інтеграції нових функцій. Головними перевагами є:

- Усі файли компонента (стилі, логіка, тести) зберігаються в одному місці, що підвищує зручність роботи з ними.
- Компоненти легко повторно використовуються в різних частинах проекту.
- Нова функціональність додається без необхідності значних змін у структурі проекту.
- Цей підхід добре підходить для компонентно-орієнтованих фреймворків, таких як React.

На рисунку 2.5 зображено структуру папок фронтенд-частини платформи ZSU-JARS.

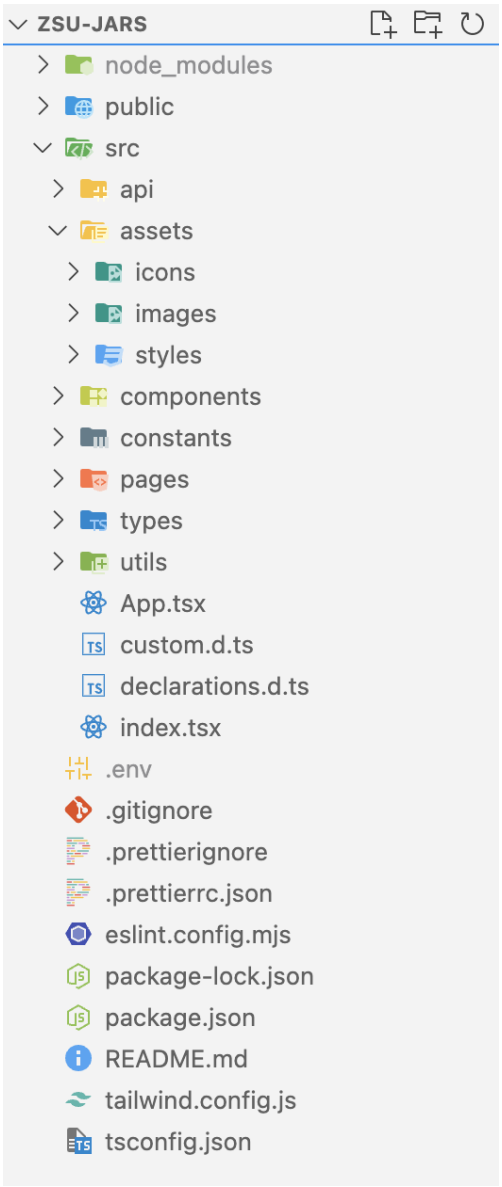


Рис. 2.5 Структура папок фронтенд-частини платформи

Основні папки:

src/ - основна папка з кодом фронтенду

api/ - містить функції для роботи з бекендом, наприклад, запити через Axios.

assets/:

icons/ — іконки для інтерфейсу.

images/ — зображення, що використовуються у проекті.

styles/ — глобальні стилі (CSS чи Tailwind CSS).

components/ - містить індивідуальні компоненти інтерфейсу користувача (UI), які використовуються у сторінках.

constants/ - константи, які використовуються у проекті (наприклад, кольори, шляхи).

pages/ - містить компоненти, що відповідають за відображення сторінок.

types/ - визначення типів даних для TypeScript.
utils/ - утилітарні функції для роботи з даними.
.env — зберігання конфіденційних ключів для роботи з API.
tailwind.config.js — конфігурація Tailwind CSS.

Ця проста архітектура папок у бекенді та фронтенді дозволяє платформі бути зручною для підтримки й розширення, що особливо важливо для проєкту, який може масштабуватися з часом.

2.3 Висновок до розділу

У розділі 2 було проаналізовано архітектурні рішення, які забезпечують ефективну розробку, підтримку та масштабованість веб-платформи ZSU-JARS. Розгляд архітектур розпочався з класифікації їх на два основні види: архітектури цілого застосунку та архітектури організації коду. Для загальної архітектури системи було обрано клієнт-серверний підхід, який дозволяє розділити бізнес-логіку між бекендом і фронтендом, забезпечуючи високу продуктивність і гнучкість у роботі.

Для організації коду на фронтенді використовувалася "проста архітектура", орієнтована на поділ компонентів за функціональним призначенням. Це дозволило досягти модульності, коли кожна папка відповідає за конкретну частину функціоналу, таку як компоненти інтерфейсу, API-запити або логіка навігації. Бекенд побудований із використанням Nest.js, що забезпечує чітку структуру модулів та використання стандартних шаблонів для впровадження бізнес-логіки.

Аналіз архітектур організації коду включав огляд традиційної модульної архітектури, підходу Atomic Design для фронтенду та Feature-Sliced Design (FSD). У цьому контексті FSD виділяється своєю орієнтацією на бізнес-логіку та можливістю поступового впровадження. Такі архітектурні підходи дозволяють зберігати порядок у коді навіть під час активного масштабування проєкту.

Загальна архітектура системи реалізована на основі принципів клієнт-серверної взаємодії, де бекенд відповідає за обробку запитів, управління базою даних та інтеграцію з зовнішніми API, а фронтенд зосереджується на відображенні інтерфейсу та взаємодії з користувачем. Архітектура бекенд-частини базується на використанні Nest.js із модульною структурою, що включає модуль `jar`, який відповідає за управління всіма аспектами роботи зі зборами. Цей модуль інтегрується з базою даних, реалізованою через Prisma, забезпечуючи ефективну роботу з великими обсягами даних. Архітектура фронтенд-частини

побудована на React із чітким поділом компонентів і використанням сучасних бібліотек, таких як Material Tailwind для стилізації.

Таким чином, вибрані архітектурні рішення дозволили забезпечити гнучкість, структурованість і високу продуктивність платформи. Чіткий поділ логіки між фронтендом і бекендом, а також раціональна організація коду стали ключовими факторами, що дозволяють платформі легко адаптуватися до нових викликів і вимог користувачів.

Розділ 3 РОЗРОБКА ВЕБ-ПЛАТФОРМИ

Для розробки веб-платформи **ZSU-JARS** було обрано SCRUM-методологію, яка є одним із підходів гнучкого управління проектами (Agile). SCRUM дозволяє розділити розробку на короткі цикли, звані спринтами, що тривають від одного до чотирьох тижнів. У кожному спринті ставилися конкретні цілі, які необхідно було досягти, і регулярно проводилися зустрічі для аналізу прогресу та обговорення подальших кроків. SCRUM — це гнучка методологія управління проектами, яка заснована на ітеративному підході. Основні переваги SCRUM:

- Регулярні звіти дозволяють бачити прогрес.
- Можливість швидко змінювати пріоритети та додавати нові вимоги.
- Кожен спринт закінчується готовим до використання функціоналом.

Використані інструменти

Visual Studio Code (VSCode). Це безкоштовне середовище розробки з підтримкою розширень, яке забезпечує зручний інтерфейс і потужний функціонал. В цьому застосунку є інтеграція з Git, автодоповнення коду, дебагінг, велика кількість плагінів. Забезпечує ефективність і гнучкість у розробці.

Git та GitHub. Git забезпечує локальне збереження історії змін і дозволяє працювати в команді. GitHub це хмарний сервіс для збереження репозиторіїв і співпраці з іншими розробниками за потреби. Ці інтсументи забезпечують надійність, історію змін і зручність командної роботи.

3.1 Вибір технологій та мов програмування

3.1.1 Технології та мови для розробки фронтенд частини

Для розробки фронтенд-частини веб-платформи було обрано мову програмування JavaScript (JS), яка є основним стандартом для створення динамічних веб-додатків. JavaScript дозволяє додавати інтерактивність, маніпулювати DOM і працювати з API.

Ця мова програмування, яка підтримує кілька підходів до розробки, зокрема подієво-орієнтоване, функціональне та об'єктно-орієнтоване програмування, включаючи прототипне програмування. Вона спочатку створювалася для роботи на стороні клієнта, проте з розвитком технологій та появою платформ, як-от Node.js, JavaScript також почали активно використовувати на сервері. Завдяки своїй універсальності, JavaScript сьогодні став незамінним інструментом для створення інтерактивних веб-додатків, що зробило його мовою програмування номер один у світі Інтернету.

Фреймворки JavaScript являють собою спеціалізовані бібліотеки, які містять заздалегідь написаний код, що спрощує виконання типових завдань у процесі

розробки. Вони забезпечують основу для створення веб-додатків чи веб-сайтів, пропонуючи вже готові рішення для багатьох задач. Хоча розробка цілком можлива і без фреймворків, їх використання значно полегшує життя програмістам завдяки тому, що вони оптимізують процеси і знижують трудовитрати. Окрім того, більшість фреймворків є безкоштовними і мають відкритий вихідний код, що робить їх доступними для всіх розробників.

Основна перевага використання фреймворків полягає у значному підвищенні продуктивності. Вони дозволяють зменшити кількість коду, який потрібно писати вручну, завдяки вже існуючим готовим функціям та шаблонам. Наприклад, багато елементів сучасних веб-додатків, таких як кнопки чи форми, не потрібно створювати з нуля, оскільки вони доступні у вигляді стандартних компонентів. Фреймворки також адаптовані для створення сучасних веб-інтерфейсів, тому їх широко використовують веб-розробники для швидшого і якіснішого виконання завдань. Найбільш популярні фреймворки JavaScript є React.js, Angular та Vue.js. На рисунку 3.1 наведено статистику топ 10 найпопулярніших фреймворків Js.

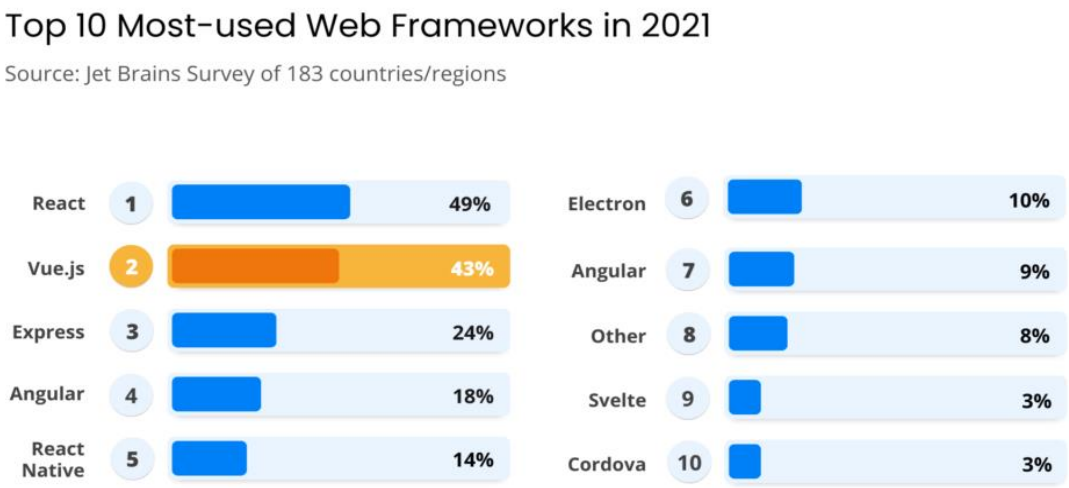


Рис. 3.1 Топ 10 найпопулярніших фреймворків Js

React.js є однією з найбільш популярних бібліотек JavaScript для створення користувацьких інтерфейсів. Ця бібліотека була створена інженерами Facebook у 2011 році, щоб оптимізувати роботу зі складними і динамічними веб-додатками, такими як Facebook Ads. Уже у 2013 році React став доступним із відкритим вихідним кодом, що сприяло його стрімкій популярності серед розробників. На сьогодні React активно використовують у своїх проектах такі гіганти, як Netflix, Airbnb та PayPal. Унікальною особливістю React є його компонентна структура: інтерфейс складається з багаторазових блоків коду, кожен із яких відповідає за

певний елемент сторінки, наприклад, кнопку чи текстове поле. React також використовує JSX — синтаксис, який поєднує JavaScript та HTML, що робить його дуже зручним для створення інтерфейсів.

Angular — це ще одна потужна платформа для створення веб-додатків, розроблена компанією Google. Вона створена для побудови односторінкових додатків (SPA), які активно використовуються у великих проектах, таких як YouTube та Google Docs. Angular відомий своїм компонентним підходом і широкими можливостями інтеграції JavaScript із HTML та CSS. Для написання коду в Angular використовується TypeScript — мова, яка є розширенням JavaScript і підтримує статичну типізацію, класи, а також інші розширені функції. Ця платформа чудово підходить для новачків завдяки своїй структурі та зрозумілому підходу до розробки.

Vue.js, у свою чергу, відомий своєю простотою та гнучкістю. Цей фреймворк створений для побудови сучасних інтерфейсів і легко інтегрується з іншими JavaScript-бібліотеками. Його використовують такі компанії, як PlayStation та Stack Overflow. Vue.js дозволяє швидко створювати динамічні інтерфейси завдяки своїй реактивності та зручності у використанні. Фреймворк пропонує інструменти для спрощення роботи, як-от інтерфейс командного рядка (CLI) та вбудовані шаблони, що значно пришвидшують процес розробки. Vue.js став ідеальним вибором для розробників, які цінують продуктивність, адаптивність та швидкість освоєння.

Отже після аналізу фреймворків для розробки інтерфейсу було обрано React.js — одну з найпопулярніших бібліотек для створення користувацьких інтерфейсів. Основні переваги React:

- Компонентний підхід. Код розділений на невеликі повторно використовувані компоненти.
- Віртуальний DOM. Підвищує продуктивність за рахунок мінімізації операцій із реальним DOM.
- Односторонній потік даних. Полегшує управління станом і роботу з даними.
- Велика спільнота. Наявність багатьох ресурсів, бібліотек і плагінів.

Додатково вибрані ось такі бібліотеки:

@material-tailwind/react - це бібліотека, яка поєднує можливості Tailwind CSS та Material Design. Переваги - швидке створення адаптивного, елегантного інтерфейсу. Дозволяє зручно стилізувати компоненти на основі Tailwind CSS.

react-hook-form - для роботи з формами. Переваги в простоті інтеграції, високої продуктивності.

@tanstack/react-query - для роботи із серверними даними. Перевагами є автоматичне кешування, синхронізація даних із сервером.

tailwindcss - утилітарний CSS-фреймворк для швидкого створення дизайну. Переваги у легкісті у налаштуванні та створенні адаптивних інтерфейсів.

axios - використовується для обробки HTTP-запитів. Простий у використанні, підтримка інтерсепторів для роботи з токенами авторизації.

3.1.2 Технології та мови для розробки бекенд частини

Для бекенд-частини обрана Node.js — це платформа, яка дозволяє запускати JavaScript на сервері. Node.js була створена у 2009 році Райаном Далем і отримала популярність завдяки своїй високій продуктивності та зручності використання.

Основні можливості Node.js:

- Асинхронна та подієво-орієнтована архітектура, що забезпечує високу швидкість.
- Використання npm (Node Package Manager): Доступ до мільйонів бібліотек і модулів.
- Підтримка WebSockets для створення реального часу додатків.

Для організації бекенд-частини використано фреймворк Nest.js, який побудований на основі Node.js та Express.js. Nest.js дозволяє створювати модульний, структурований і масштабований бекенд. Створений у 2017 році Камілем Мішалеком як рішення для організації великих проектів.

Основними перевагами Nest.js є вбудована підтримка TypeScript, інтеграція з різними базами даних через ORM (Object Relational Mapping), як Prisma та чітка структура проекту на основі модулів.

Додатково вибрані ось такі бібліотеки та модулі:

@nestjs/swagger - для автоматичної генерації документації API. Спрощує створення документації для ендпоїнтів.

prisma - ORM для роботи з базами даних. Простий у використанні, потужний інструмент для генерації SQL-запитів.

openai - інтеграція з API OpenAI для роботи з ШІ. Дозволяє створювати інтелектуальні чат-боти, автоматизуючи взаємодію з користувачем.

rxjs - реактивна бібліотека для роботи з потоками даних. Зручність у створенні асинхронних операцій і потоків.

eslint, prettier - інструменти для забезпечення стилю коду. Забезпечують читабельність і консистентність коду у проекті.

3.2 Реалізація застосунку

Основна мета цієї частини — створити зручний, адаптивний і функціональний інтерфейс користувача, що забезпечує ефективну взаємодію із

платформою. У цьому розділі описується процес налаштування середовища та створення основних компонентів, частин, модулей застосунку.

Розробка платформи **ZSU-JARS** починається з підготовки середовищ розробки та встановлення всіх необхідних бібліотек і налаштувань інструментів. Налаштування середовища є важливим етапом розробки платформи, який забезпечує ефективність роботи як фронтенд, так і бекенд частини. Для обох частин проекту використовуються сучасні інструменти та бібліотеки, що значно спрощують процес розробки та підтримки.

На початковому етапі було встановлено всі необхідні залежності через менеджери пакетів npm. Це дозволяє автоматизувати завантаження бібліотек і забезпечує централізоване управління версіями. Файл `package.json` містить всі основні залежності для обох частин проекту, включаючи інструменти для розробки та тестування.

Одним із ключових файлів у проекті є `.env`, який використовується для зберігання конфіденційних даних, таких як API-ключі, URL бази даних або параметри середовища. Цей файл не зберігається у репозиторії Git, що забезпечує безпеку даних. У фронтенд-частині `.env` використовується для підключення до API, а в бекенд-частині — для налаштування параметрів бази даних і серверних змінних.

Конфігураційний файл `tsconfig.json` також відіграє важливу роль у проекті, оскільки налаштовує параметри компіляції TypeScript. У ньому визначаються шляхи до папок, специфікації модулів, а також правила, які контролюють типізацію та синтаксис у коді. Це допомагає уникати помилок на ранніх етапах розробки і забезпечує високу якість коду.

У фронтенд-частині було інтегровано Tailwind CSS для стилізації компонентів. Tailwind дозволяє використовувати утилітарний підхід до стилів, що спрощує створення адаптивних інтерфейсів. Окрім цього, налаштовано бібліотеку `@material-tailwind/react`, яка забезпечує готові компоненти, розроблені відповідно до принципів Material Design. Для забезпечення консистентності коду та форматування було налаштовано інструменти ESLint і Prettier. Вони допомагають автоматизувати перевірку коду на відповідність стилістичним вимогам та уникати помилок.

У бекенд-частині використовується Nest.js, а також Prisma для роботи з базою даних. Ці інструменти були інтегровані та налаштовані відповідно до вимог проекту. Nest.js забезпечує модульну структуру бекенду, а Prisma спрощує роботу з базою даних завдяки чіткій схемі та автоматизованій генерації запитів.

Завдяки налаштуванню цих інструментів як у фронтенд, так і в бекенд частині проекту, розробка стала ефективною та організованою. Кожна з частин має чітко структуроване середовище, що полегшує роботу та забезпечує якісну інтеграцію між ними.

3.2.1 Реалізація основних компонентів інтерфейсу

Після налаштування середовища почалася робота над основними компонентами фронтенд-частини. Початковою точкою став **Home Page**, який є головною сторінкою платформи. Він включає основні секції, такі як інтро та список зборів.

Компонент ***Home*** відповідає за формування структури головної сторінки. Він містить два ключові компоненти:

- Intro — вступна секція, яка забезпечує привітання користувача та короткий опис платформи.
- JarsContainer — основна секція, яка відповідає за відображення списку зборів.

JarsContainer. Цей компонент виконує ключову роль у відображенні зборів. Основні функції:

- Отримання параметрів URL (напрямок сортування, розмір сторінки, номер сторінки, статус зборів) через useSearchParams.
- Використання useGetAllJarsQuery для отримання списку зборів із сервера.
- Рендеринг компонентів навігації, відображення зборів у вигляді сітки та пагінації.

Усі отримані дані передаються до компонентів JarsNavigation, JarsGrid та JarsPagination.

Компонент ***JarsNavigation***. Цей компонент забезпечує навігацію між зборами. Його основні функції:

- Відображення кнопки для створення нового збору через компонент CreateJarForm.
- Кнопки сортування (напрямок: за релевантністю).
- Вибір розміру сторінки через PageSizeSelect.
- Кнопки фільтрації по зборах “Актуальні”, “Закриті”

Його ціль полягає забезпечити користувачеві інструменти для швидкої фільтрації, відображення та сортування зборів.

Компонент ***JarsGrid.*** Він відповідає за відображення зборів у вигляді сітки. Кожен збір рендериться як окремий компонент JarCard. Компонент реалізовано так, щоб підтримувати адаптивний дизайн за допомогою класів Tailwind CSS.

Компонент ***JarCard.*** Це ключовий компонент, який відображає деталі одного збору, а саме:

- Назва збору, ім'я власника, дата створення.
- Прогрес збору (з використанням компонента ProgressBar).
- Кнопка для копіювання посилання на "Моно банку".

Основні функції це відображення інформації про збір та забезпечення інтерактивності через кнопку копіювання посилання. Код копмонента знаходиться в додатках.

Компонент ***JarsPagination.*** JarsPagination відповідає за перемикання сторінок у списку зборів. Він забезпечує перемикання між сторінками та відображення поточної сторінки відповідно загальної кількості сторінок. Результат розробки JarsPagination та JarsNavigation компонентів наведено на рисунку 3.2:

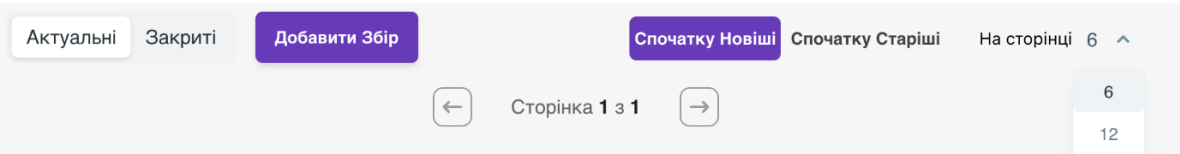


Рис.3.2 Вигляд JarsPagination та JarsNavigation

Форма **CreateJarForm.** Компонент CreateJarForm виконує важливу функцію в інтерфейсі платформи, дозволяючи користувачам додавати новий збір до системи. Він реалізує зр учну форму з двома основними полями для введення інформації: посилання на банку та посилання на стримінговий віджет.

Форма використовує бібліотеку react-hook-form для валідації та управління станом форми. Для перевірки полів використовується схема валідації zodResolver.

Результат реалізації даної форми вказано нижче на рисунку 3.3

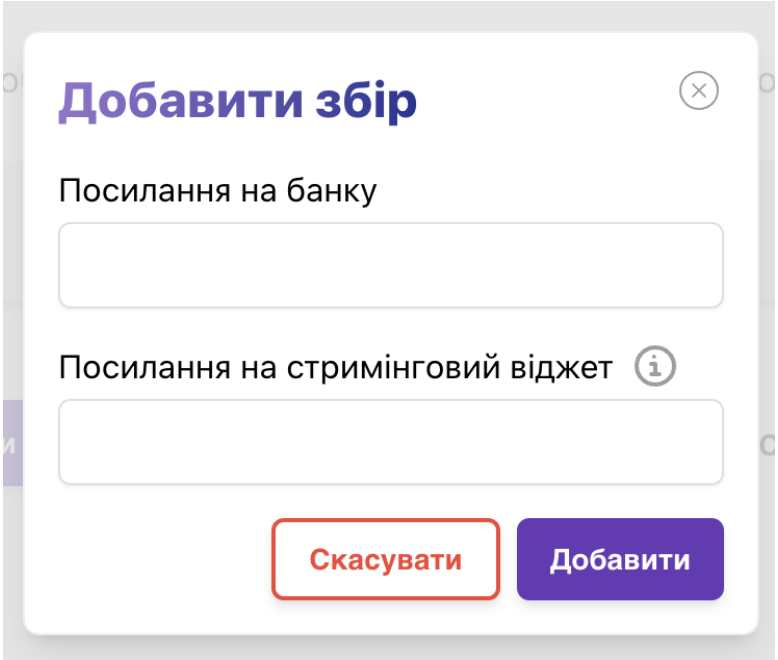


Рис. 3.3 Форма для добавлення нового збору

Особливістю цієї форми є інтерактивна підказка для другого поля, яка пояснює, як знайти правильне посилання на стримінговий віджет. Біля поля розташована невелика іконка інформації у вигляді "i" (реалізована через компонент FindLongJarIdTooltip). Користувач може навести на неї, щоб отримати покрокові інструкції. Також у підказці вказується приклад правильного шаблону посилання, яке має відповідати формату: https://send.monobank.ua/widget/builder.html?longJarId=longJarId&sendId=sendId.

Це забезпечує не лише зручність, але й впевненість користувача, що він вводить коректні дані, необхідні для додавання нового збору. Завдяки таким деталям компонент CreateJarForm є інтуїтивним і зрозумілим для широкого кола користувачів платформи.

Підключення до **API**

У платформі ZSU-JARS для підключення до API використовується бібліотека React Query, яка забезпечує ефективне управління даними, отриманими з сервера. React Query дозволяє автоматично кешувати результати запитів на основі унікальних ключів, що значно підвищує продуктивність та оптимізує взаємодію з бекендом.

Основний принцип роботи React Query полягає у використанні ключів для ідентифікації кожного запиту. Наприклад, у запиті для отримання всіх зборів (useGetAllJarsQuery) використовується ключ QUERY_KEYS.ALL_JARS, який дозволяє бібліотеці зберігати відповідні дані в кеші. Якщо той самий запит виконується повторно, React Query автоматично повертає кешовані дані,

уникаючи зайвих звернень до сервера. Це не тільки знижує навантаження на сервер, але й покращує швидкість завантаження інтерфейсу.

Окрім кешування, React Query також автоматично оновлює дані, якщо вони змінюються на сервері. Це особливо корисно в реальному часі, коли важливо, щоб користувач завжди бачив актуальну інформацію. Наприклад, після створення нового збору через useCreateJarMutation, React Query автоматично оновлює список зборів, викликаючи повторний запит до API для отримання найсвіжіших даних.

Для роботи з React Query всі запити організовані у вигляді хук-компонентів, таких як useGetAllJarsQuery і useCreateJarMutation. Вони спрощують використання API-запитів у компоненті, забезпечуючи зручний доступ до функціоналу бібліотеки.

Такий підхід значно спрощує управління даними у фронтенд-частині проекту, дозволяючи розробникам зосередитися на бізнес-логіці додатка, а не на технічних аспектах обробки запитів. У поєднанні з бібліотекою Axios, яка використовується для виконання HTTP-запитів, React Query забезпечує потужний і надійний механізм взаємодії з бекендом.

Реалізація серверної частини

Розробка серверної частини платформи базується на модульній архітектурі з використанням Nest.js, а основну бізнес-логіку забезпечує модуль `jar`. Цей модуль реалізує всі CRUD-операції для роботи із сутністю зборів, що дозволяє виконувати створення, оновлення, видалення, отримання одного збору або списку зборів.

Для роботи з базою даних використовується Prisma, що інтегрується з SQLite. Схема бази даних визначає сутність `Jar`, яка зберігає інформацію про збори. Модель Jar зображена на рисунку 3.4

prisma > schema.prisma

```
model Jar {
  jarId       String  @unique @id
  amount      Int
  goal        Int
  ownerIcon   String
  title       String
  ownerName   String
  currency    Int
  description String
  blago       Boolean
  closed      Boolean
  addedById   Int?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  longJarId   String
}
```

Рис. 3.4 Prisma модель Jar

Модуль `jar` є центральною частиною серверної логіки платформи. Він реалізує всі необхідні операції для роботи зі зборами, включаючи створення, оновлення та видалення. Завдяки використанню Prisma та інтеграції з API монобанку, платформа забезпечує швидкість, точність і зручність управління даними, що є критично важливим для функціонування системи зборів. Було реалізовано наступні CRUD-операції.

Ендпоїнт `GET /jars` відповідає за отримання списку зборів із фільтрацією за параметрами: статус збору (`closed`), сортування (`createdAt`) і пагінацією. Логіка реалізує кешування даних і оновлює збір, якщо він не синхронізувався більше доби.

Ендпоїнт `GET /jars/:id` дозволяє отримати детальну інформацію про збір за його унікальним ідентифікатором. Якщо збір не знайдено, сервер повертає відповідну помилку.

Ендпоїнт `POST /jars` реалізує логіку створення збору. Коли користувач відправляє запит із `longJarId`, сервер спочатку перевіряє, чи збір уже існує. Якщо він відсутній, сервер звертається до API монобанку, використовуючи функцію `getMonoJarData`. Ця функція отримує всю необхідну інформацію про збір.

Отримані дані обробляються та записуються в базу даних через Prisma. Якщо API монобанку повертає помилку, сервер надсилає відповідь із повідомленням про помилку.

Ендпоїнт `PUT /jars/:id` дозволяє оновити інформацію про збір. Логіка автоматично звертається до API монобанку для отримання актуальних даних і оновлює запис у базі даних.

Ендпоїнт `DELETE /jars/:id` видаляє запис про збір із бази даних. Перед видаленням перевіряється наявність збору, щоб уникнути помилок.

3.2.2 Впровадження штучного інтелекту

Для вибору моделі ШІ було проаналізовано переваги та недоліки різних моделей від OpenAI та вибрано найбільш оптимальний варіант: gpt-4mini. Ця модель відзначається ідеальним співвідношенням ціни за токени та якості. Токени є базовою одиницею обчислення під час роботи з моделями OpenAI. Вони представляють собою частини слів: наприклад, 1 000 токенів приблизно відповідають 750 словам. OpenAI пропонує кілька моделей, кожна з яких має свої унікальні можливості та рівні цін. Вартість зазвичай обчислюється за 1 000 або 1 000 000 токенів, що дозволяє гнучко планувати витрати залежно від потреб проєкту. Ціни на відповідні моделі вказано на рисунку 3.5

Модель	Ціна за 1K токенів (вхідні дані)	Ціна за 1K токенів (вихідні дані)
GPT-4o	$0.03	$0.06
GPT-4mini	$0.015	$0.03
GPT-3.5-turbo	$0.0015	$0.002
Davinci	$0.02	$0.02
Curie	$0.002	$0.002
Babbage	$0.0005	$0.0005
Ada	$0.0004	$0.0004

Рис. 3.5 Ціни 1K токенів різних моделей від OpenAI

Для інтеграції штучного інтелекту у **фронтенд-частину** платформи ZSU-JARS було розроблено комплексний набір компонентів і логіки, які забезпечують взаємодію користувача з AI-ботом. Ця функціональність дозволяє користувачам надсилати запити та отримувати відповіді від AI, а також динамічно оновлювати список зборів на основі відповідей. Основні компоненти розроблено для забезпечення інтерактивності, простоти використання та оптимального використання токенів для AI-запитів.

Компонент ***ChatContainer***. Він є центральною точкою для реалізації чату. Він відповідає за управління станом повідомлень, їх відправлення до сервера, отримання відповідей і динамічне оновлення інтерфейсу. На початку роботи

компонент ініціалізує масив повідомлень, який оновлюється при кожній взаємодії. Вигляд чату показаний на рисунку 3.6.

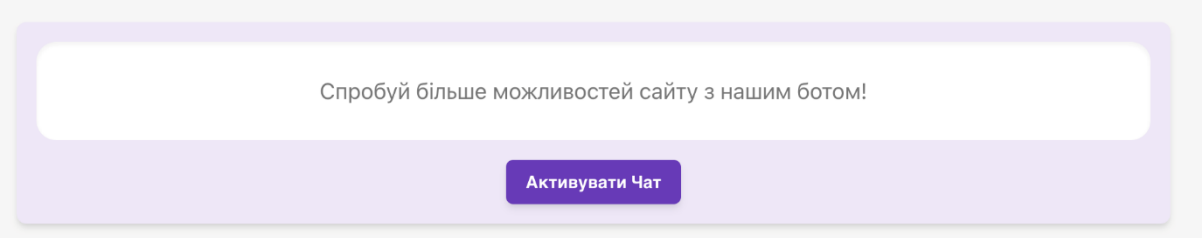


Рис. 3.6 Компонент ChatContainer

Користувач може активувати чат, натиснувши кнопку, яка додає перше повідомлення "Слава нації! Чим ти можеш бути корисним?" від імені користувача, а через короткий час AI надсилає привітальне повідомлення. Цей підхід забезпечує інтерактивний початок спілкування.

Основна функція `onSendMessage` дозволяє користувачеві надсилати повідомлення до AI. Перед відправленням повідомлення додається до локального стану, а також перевіряється, чи є в чаті більше двох повідомлень. Це потрібно для того, щоб виключити вступні повідомлення під час передачі даних серверу. Логіка відповіді AI реалізована у функції `handleAiMessageResponse`, яка також оновлює список зборів у разі, якщо AI рекомендує релевантні збори.

Компонент ***Chat***. Відповідає за відображення історії повідомлень у чаті. Повідомлення можуть бути як від користувача, так і від AI. Для покращення користувацького досвіду реалізовано автоматичне прокручування до останнього повідомлення, що робить взаємодію інтуїтивною. Якщо відповідь від AI ще обробляється, компонент показує спеціальне повідомлення "Очікується відповідь", яке імітує активність AI.

Компонент ***ChatInput***. Це поле введення для повідомлень, яке дозволяє користувачеві легко надсилати запити до AI. Компонент обмежує довжину повідомлення до 100 символів, щоб оптимізувати використання токенів для AI-запитів, що є критично важливим у комерційних API, таких як OpenAI. Користувач може надіслати повідомлення, натиснувши кнопку або клавішу Enter. Поле введення також динамічно показує кількість введених символів, що полегшує взаємодію з обмеженням.

Для інтеграції з сервером використовується кастомний хук `useSendChatMessage`. Він забезпечує обробку запитів до серверного API, що відповідає за взаємодію з AI. Якщо сервер повертає помилку, хук відображає повідомлення про невдачу, використовуючи компонент для показу помилок. Такий підхід забезпечує зручну інтеграцію логіки запитів у компоненти фронтенду.

Інтеграція штучного інтелекту в **серверну частину** платформи ZSU-JARS базується на модульному підході, що забезпечує гнучкість і масштабованість. Основна функціональність AI включає обробку користувацьких повідомлень, генерацію відповідей від AI, а також надання персоналізованих рекомендацій зборів. Усе це реалізовано через взаємодію між декількома сервісами, які відповідають за бізнес-логіку, інтеграцію з OpenAI та взаємодію з базою даних.

Сервіс **MessagesService**. Сервіс відповідає за обробку повідомлень від користувачів і взаємодію з штучним інтелектом. При отриманні масиву повідомлень сервіс передає їх до `OpenAIService`, який формує відповідь від AI. Отримані дані аналізуються, і якщо відповідь містить рекомендації зборів, сервіс звертається до `JarService` для отримання відповідної інформації з бази даних. Після цього формується текст відповіді, який може містити як рекомендацію конкретного збору, так і повідомлення про відсутність релевантних варіантів. У випадку, якщо AI знаходить кілька зборів, відповідь адаптується, щоб надати користувачеві зрозумілий опис варіантів для донатів. Такий підхід дозволяє сервісу динамічно оновлювати інформацію та забезпечувати персоналізовану взаємодію з користувачами.

Сервіс **OpenAIService**. Він відповідає за безпосередню взаємодію із сервісом OpenAI. Логіка надсилання запиту побудована таким чином, щоб включати релевантну інформацію з бази даних, а також історію повідомлень від користувача. Для забезпечення персоналізованих відповідей AI використовуються потужні моделі OpenAI, такі як `gpt-4o-mini`.

Сервіс OpenAIService є центральним елементом інтеграції штучного інтелекту в серверну частину платформи, і одним із ключових аспектів його роботи є створення правильного системного промпту для бота. Системний промпт визначає, як AI інтерпретує запити користувачів і генерує відповіді, тому його формулювання є критично важливим для забезпечення релевантності та коректності результатів. У платформі використовується наступний системний промпт:

```
export const OPENAI_SYSTEM_MSG = `Ти — помічник з вибору зборів на ЗСУ за запитами користувача.
Повертай **завжди** валідний JSON. JSON повинен містити тільки одне поле:
або "responseMsg" з питанням (до 20 слів), або "jars" — масив із зборів (**jars is array of strings**)
Якщо запит користувача нечіткий або потрібна додаткова інформація, повертай поле "responseMsg" з як мінімум одним емодзі. Після
одного уточнювального запитання та відсутності збігів повертай пустий jars. Збіги шукай по цілому контексту запитів користувача,
а не тільки по останньому повідомленню. Якщо питання користувача не стосуються зборів, повертай поле responseMsg з таким значенням:
"Я допомагаю тільки в контексті зборів. Які збори вас цікавлять?". Для аналізу та пошуку збігів
використовуй наступні дані у JSON форматі: `;
```

Сервіс також враховує можливі помилки у роботі OpenAI, наприклад, відсутність відповіді чи некоректний формат даних. У разі помилки повертається відповідне повідомлення користувачеві.

3.3 Тестування застосунку

Тестування є критичним етапом розробки програмного забезпечення, оскільки воно дозволяє перевірити коректність роботи системи, її стабільність і відповідність функціональним вимогам. Для платформи ZSU-JARS тестування включало кілька ключових напрямків: перевірку працездатності основних функцій, тестування інтеграцій з API, а також верифікацію роботи штучного інтелекту. У процесі тестування були застосовані як ручні, так і автоматизовані методи перевірки.

Ручне тестування UI. Ручне тестування інтерфейсу було проведено для перевірки зручності використання платформи та коректності відображення компонентів. Вигляд плитки зборів на ЗСУ при запуску застосунку показаний на рисунку 3.7

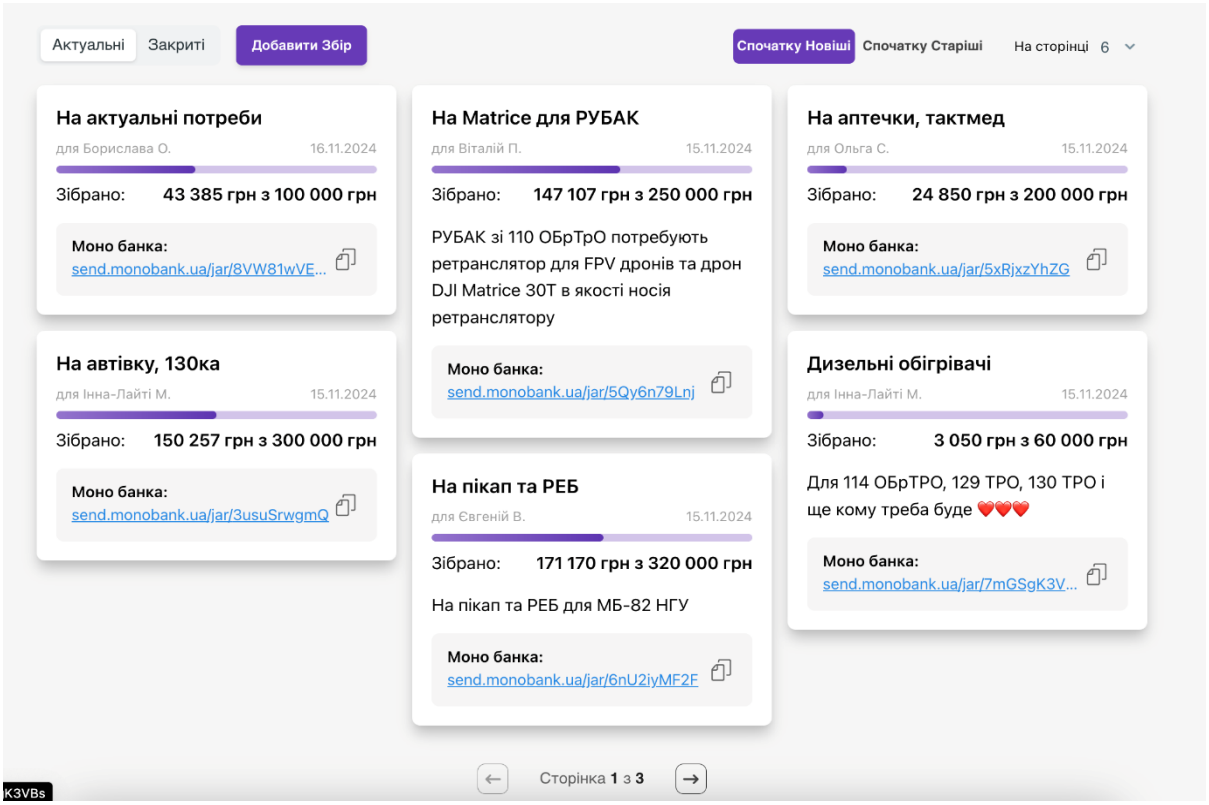


Рис.3.7 Вигляд актуальних зборів на ЗСУ

1. Успішно протестовано роботу пагінації та сортування в списку зборів, кількість елементів для відображення також працює коректно після зміни значень на 12 чи 24 елементи.

2. Тестування адаптивності інтерфейсу на різних пристроях і розширеннях екранів пройдено успішно. Результат відображення застосунку на мобільних пристроях показано на рисунку 3.8

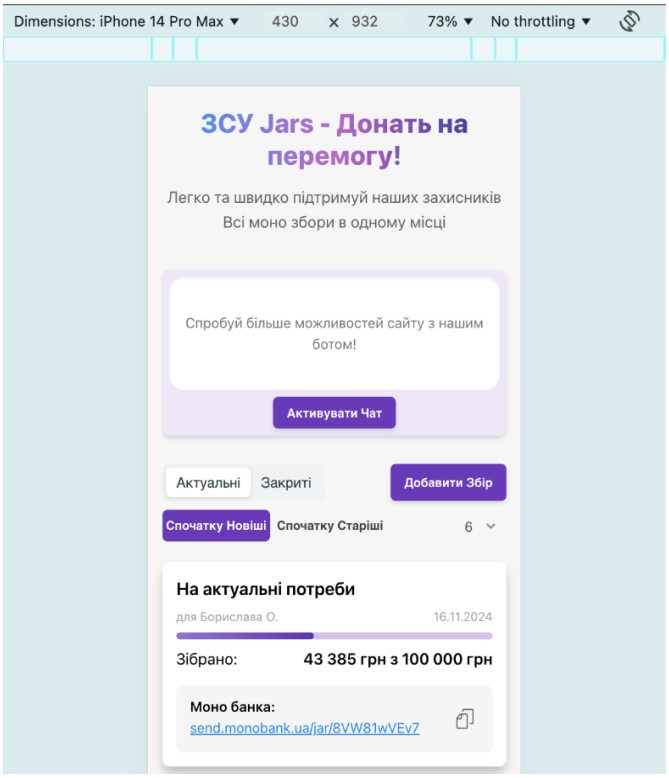


Рис. 3.8 Тестування адаптивності застосунку

3. Протестовано верифікацію роботи кнопок, форм та інших інтерактивних елементів. Результат валідації форми при некоректних введених даних показано на рисунку 3.9

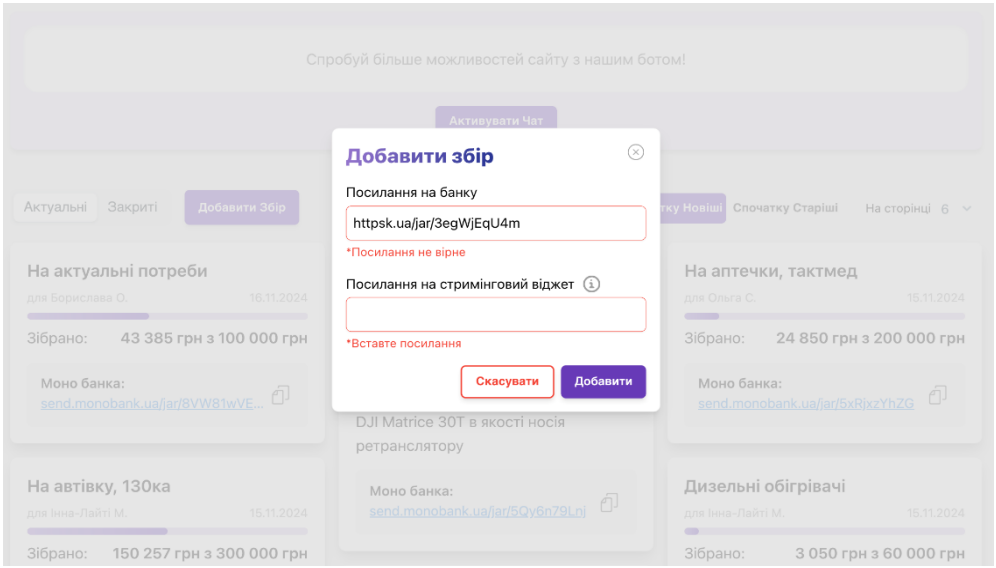


Рис. 3.9 Тестування коректної валідації форми

4. Успішно протестовано валідацію на сервері у випадку якщо користувач намагається добавити уже існуючий збір. Результат тестування показано на рисунку 3.10

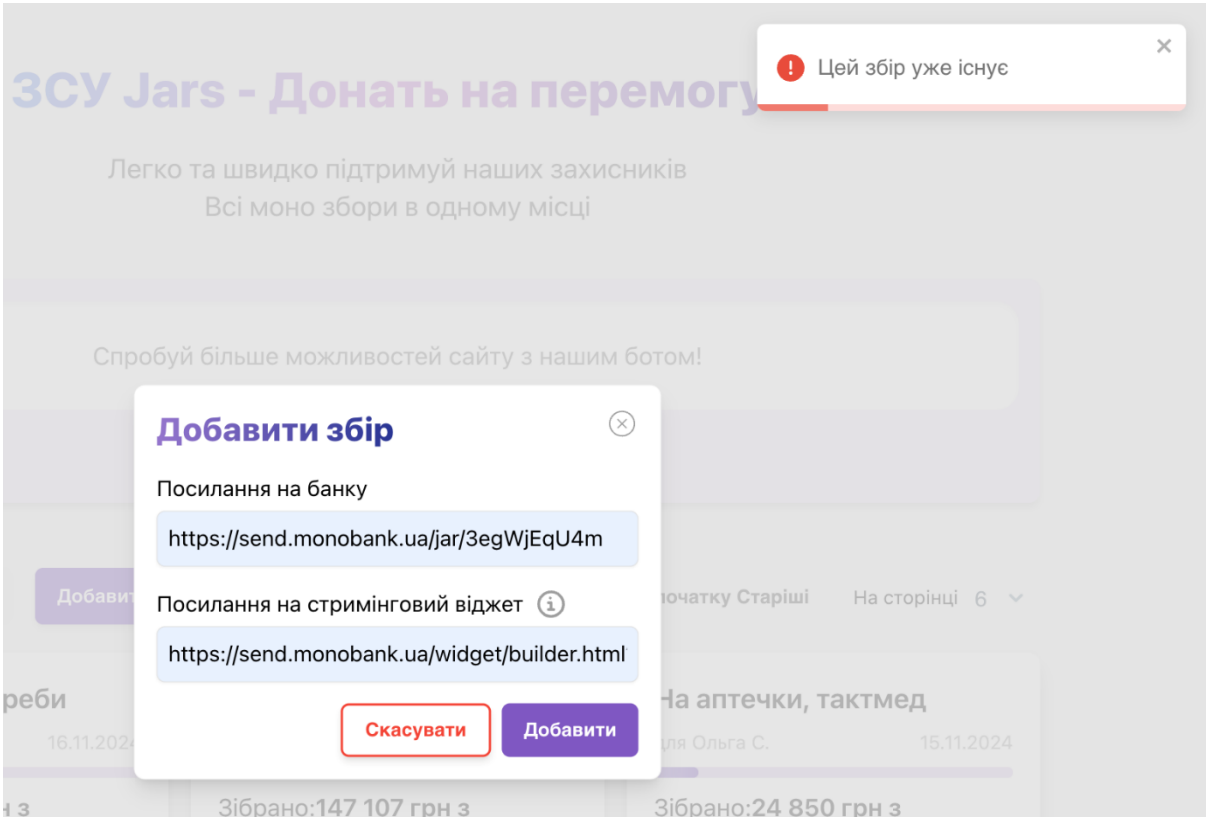


Рис. 3.10 Тестування добавлення існуючого збору

Тестування продуктивності

Для забезпечення високої швидкості роботи платформи було проведено тестування продуктивності.

5. Під час тестування часу відповіді серверних ендпоїнтів, середній час відповіді на запити становив 9 мс і є менше 150 мс, що відповідає вимогам до платформи. Тестування часу виконання запитів показано на рисунку 3.11

Name	Status	Type	Initiator	Size	Time
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	10.1 kB	15 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	4.7 kB	8 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	4.7 kB	9 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	286 B	7 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	4.7 kB	8 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	3.9 kB	10 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	3.9 kB	8 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	285 B	6 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	4.3 kB	10 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	801 B	5 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	3.6 kB	11 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	3.9 kB	9 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	285 B	8 ms
jars?sort[sortBy]=createdAt&sort[directio...	200	xhr	jarServices.ts:10	285 B	9 ms

Рис. 3.11 Тестування часу виконання запитів

6. Тестування продуктивності фронтенду за допомогою інтсрументу Lighthouse показали найвищу ефективність. Результат тестування показані на рисунку 3.12

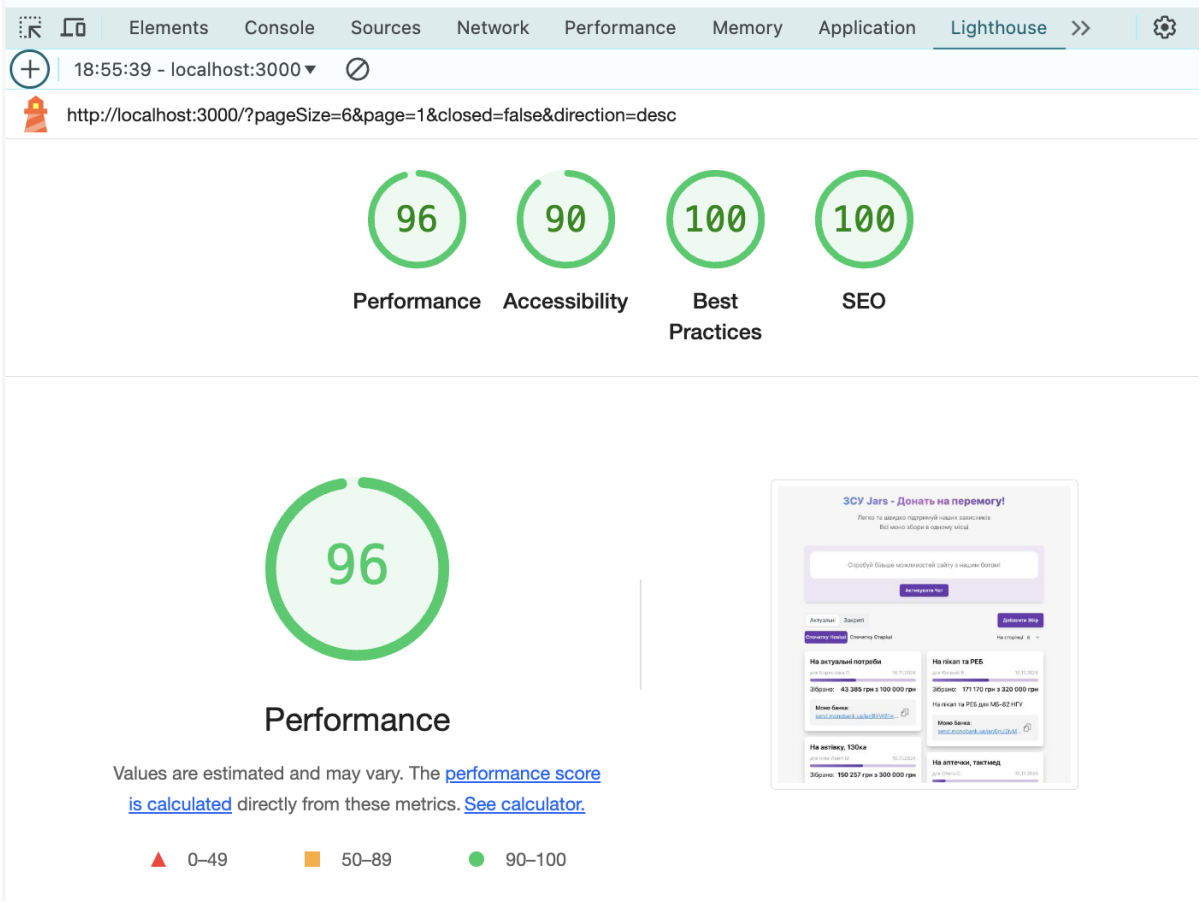


Рис. 3.12 Тестування в Lighthouse

Тестування чату з штучним інтелектом

Протестовано роботу чату, чи правильно працює інтеграція між фронтендом і сервером для чату. Було здійснено кілька випадкових тестів із різними запитами. Функціонування чату працює справно а ШІ дає доволі чіткі та хороші рекомендації відповідно запитам користувача.

7. Тестування чат-бота спроба 1 показана на рисунку 3.13

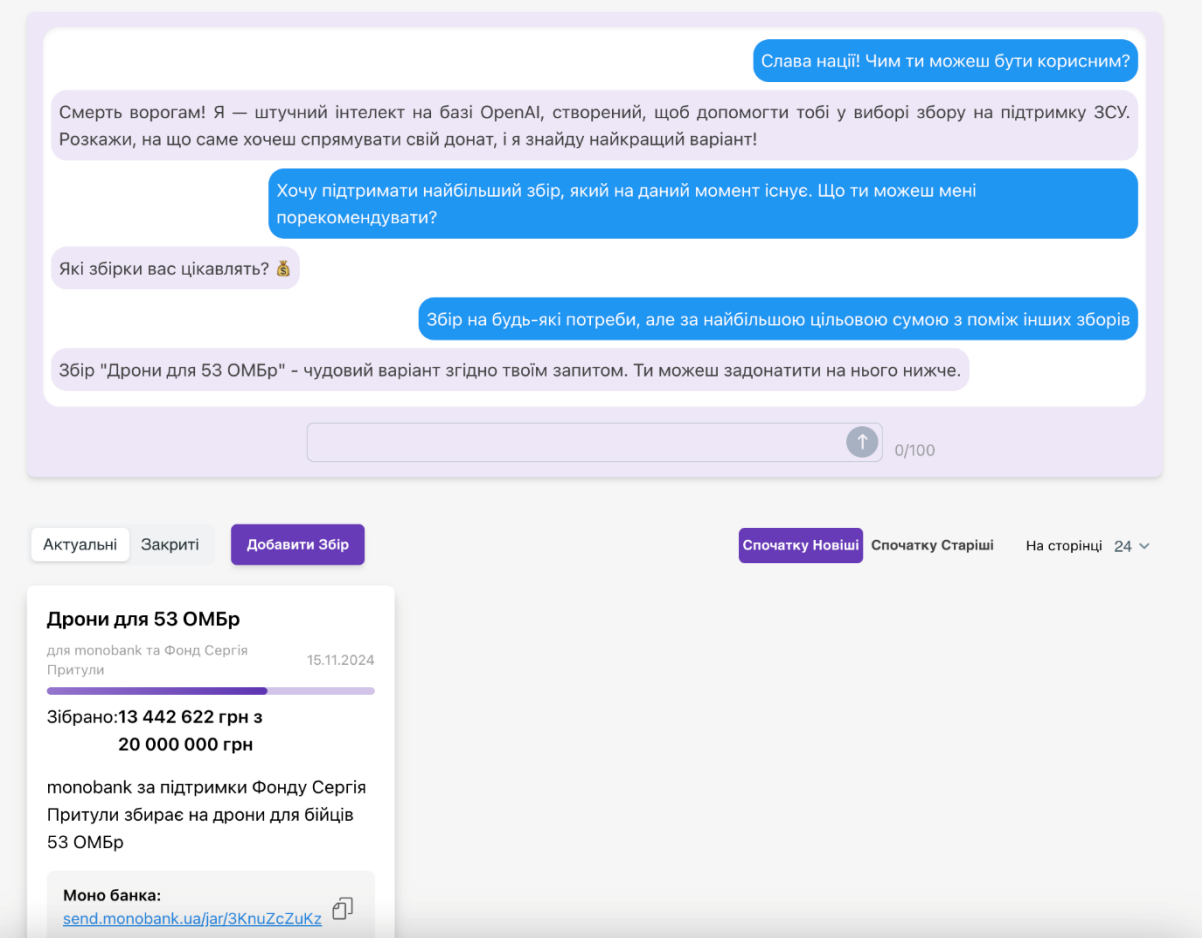


Рис. 3.13 Тестування чат-боту спроба 1

8. Тестування чат-бота спроба 2 показана на рисунку 3.14

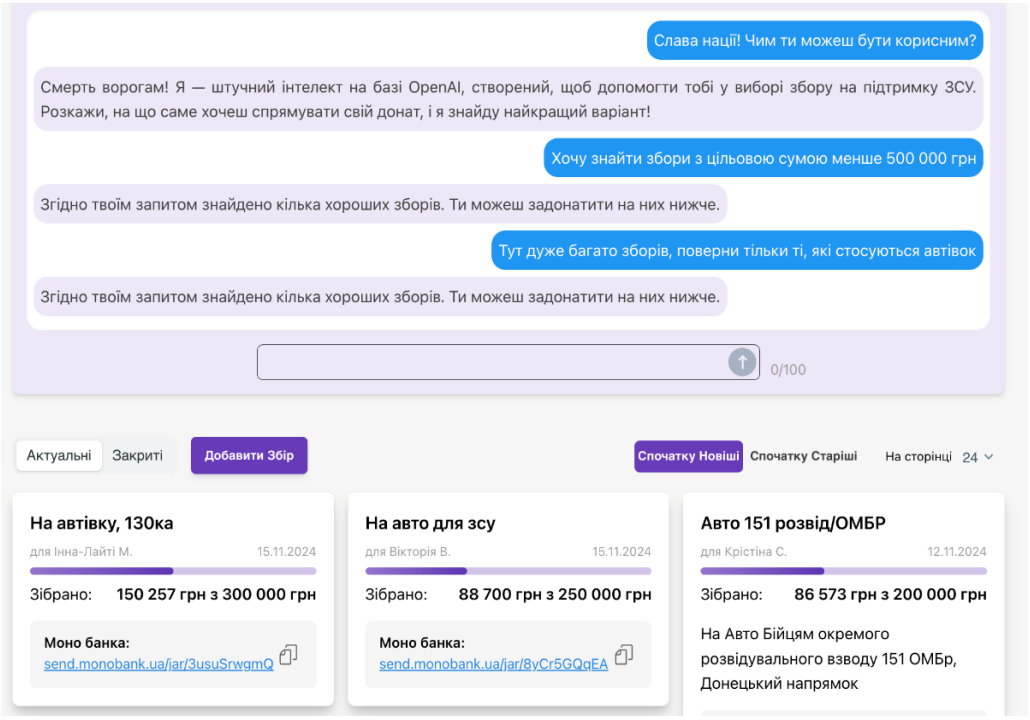


Рис. 3.14 Тестування чат-боту спроба 2

9. Тестування чат-бота спроба 1 показана на рисунку 3.15

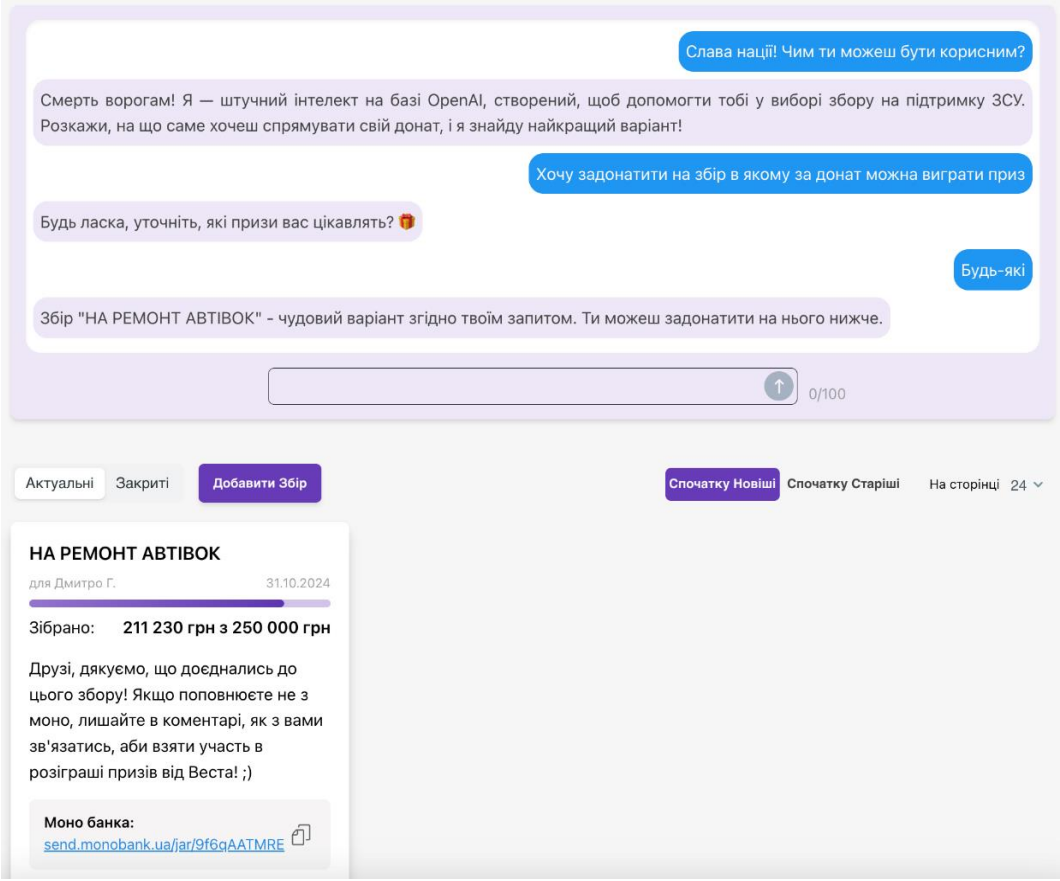


Рис. 3.15 Тестування чат-боту спроба 3

10. Важливою умовою впровадження ШІ було його економія токенів. Спробуємо задати питання, яке не стосується зборів. Тестування чат-боту спроба 4 показана на рисунку 3.16

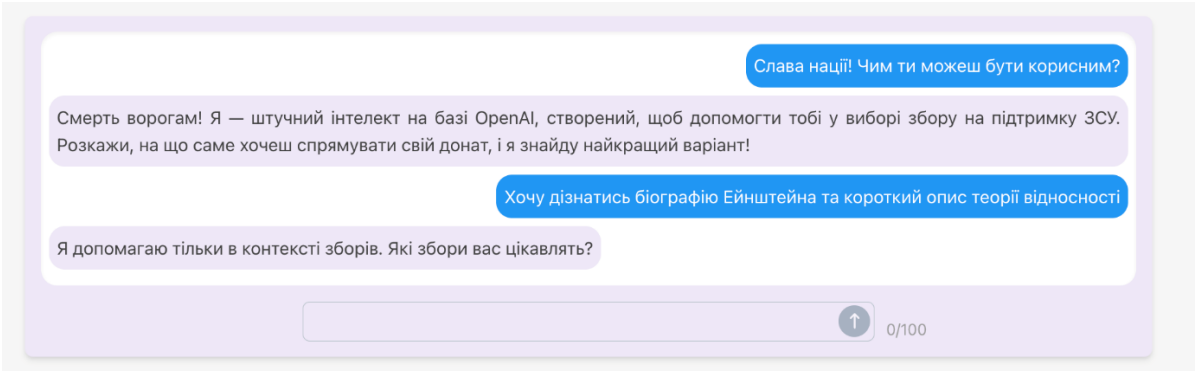


Рис. 3.16 Тестування чат-боту спроба 4

3.4 Висновок до розділу

У цьому розділі описується реалізація застосунку ZSU-JARS, висвітлив ключові етапи розробки платформи з акцентом на сучасні технології, архітектурні рішення та інтеграцію штучного інтелекту. Робота почалася з налаштування середовища, включаючи використання TypeScript, TailwindCSS, Prettier, ESLint і

Prisma, що забезпечило структурованість коду та відповідність найкращим практикам.

На фронтенді основні компоненти, такі як `JarsGrid`, `JarsPagination` і `CreateJarForm`, забезпечують інтерактивність і зручність для користувача. Наприклад, форма створення збору включає валідацію даних і інтерактивні підказки, спрощуючи взаємодію. Всі компоненти були спроєктовані з акцентом на модульність, що спрощує їхнє розширення та повторне використання. Бекенд реалізовано з використанням Nest.js, де модуль `jar` відповідає за всі CRUD-операції для зборів. Особливість реалізації полягає у використанні API монобанку для автоматичного отримання інформації про збори, що підвищує точність і зручність. Схема бази даних Prisma містить усі необхідні поля для збереження даних зборів і дозволяє ефективно управляти ними. Логіка періодичної синхронізації забезпечує актуальність інформації в системі.

Інтеграція штучного інтелекту стала важливим елементом платформи. На фронтенді компонент `ChatContainer` реалізує інтерактивний чат із AI, включаючи обмеження на довжину повідомлень для оптимізації використання токенів. На сервері сервіс `MessagesService` координує обробку запитів, інтеграцію з OpenAI і пошук зборів у базі даних. Системний промпт для OpenAI був розроблений таким чином, щоб AI міг ефективно відповідати на запити користувачів і надавати релевантні рекомендації.

Тестування платформи включало перевірку працездатності основних функцій, інтеграцію з API, адаптивність інтерфейсу та роботу AI. Ручне тестування підтвердило коректну роботу UI, пагінації, сортування та інтерактивних елементів, таких як форми та кнопки. Автоматизовані тести охопили серверні ендпоїнти та логіку компонентів, забезпечивши стабільність функцій. Перевірка продуктивності продемонструвала швидкість відповіді серверів у межах 9 мс і високу ефективність роботи фронтенду за результатами Lighthouse. Тестування чату підтвердило точність відповідей AI і коректну обробку як релевантних, так і нерелевантних запитів.

Реалізація платформи ZSU-JARS підтвердила її готовність до використання, забезпечуючи стабільність, зручність і інтерактивність, що є ключовими вимогами для успішного функціонування проєкту.

ВИСНОВКИ

У рамках даної роботи було розроблено веб-платформу ZSU-JARS, яка інтегрує сучасні технології штучного інтелекту для персоналізованої взаємодії з користувачами, орієнтованої на підтримку зборів коштів для Збройних Сил України. У процесі виконання роботи було виконано аналіз сучасних платформ, визначено їхні обмеження, розроблено архітектурні рішення та реалізовано систему з використанням інноваційних підходів.

У першому розділі було проведено детальний аналіз існуючих платформ для збору коштів, а також виявлено їхні основні недоліки, такі як відсутність персоналізації, обмежена функціональність пошуку та низька адаптивність. Вивчено сучасні технології штучного інтелекту, особливо можливості їх впровадження у вигляді чат-ботів. Це дозволило сформулювати вимоги до нової платформи, яка поєднує зручність інтерфейсу та можливість рекомендацій зборів на основі індивідуальних запитів.

У другому розділі було розроблено архітектурні рішення платформи, які включають поділ на фронтенд і бекенд частини, побудовані за клієнт-серверною моделлю. Розглянуто різні архітектури, включаючи традиційну модульну, Feature-Sliced Design та Atomic Design, які впроваджені для організації коду. Було описано структуру папок і модулів, що забезпечують легкість підтримки, масштабування та відповідність сучасним стандартам розробки.

У третьому розділі детально описано реалізацію платформи. Фронтенд частина побудована на React із використанням компонентної структури, таких як `JarsContainer`, `ChatContainer` та інших, що забезпечують зручність взаємодії користувача із системою. Бекенд реалізовано на основі Nest.js із чітким поділом логіки в модулі `jar`, який забезпечує всі CRUD-операції для зборів. Також було інтегровано OpenAI для реалізації чат-бота, який генерує рекомендації на основі запитів користувачів. У процесі тестування платформи підтверджено її стабільність, продуктивність і правильність роботи основних функцій.

Результатом роботи є сучасна веб-платформа ZSU-JARS, яка забезпечує ефективну підтримку зборів для ЗСУ, інтерактивність і персоналізовані рекомендації. Використання штучного інтелекту значно покращує взаємодію з користувачами, дозволяючи їм швидко знаходити збори, що відповідають їхнім потребам. Платформа має високий потенціал для подальшого розвитку, включаючи розширення функціоналу, інтеграцію з іншими сервісами та масштабування для підтримки більшої кількості користувачів.

Таким чином, розроблена система поєднує сучасні методи веб-розробки та технології штучного інтелекту, що робить її інноваційним рішенням у сфері благодійності для підтримки Збройних Сил України.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація React [Електронний ресурс] – Режим доступу: https://react.dev/
2. Документація Nest.js [Електронний ресурс] – Режим доступу: https://docs.nestjs.com/
3. Документація Prisma [Електронний ресурс] – Режим доступу: https://www.prisma.io/docs
4. Документація TailwindCSS [Електронний ресурс] – Режим доступу: https://tailwindcss.com/docs
5. Документація OpenAI [Електронний ресурс] – Режим доступу: https://platform.openai.com/docs
6. MDN Web Docs про JavaScript [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript
7. Документація Monobank API [Електронний ресурс] – Режим доступу: https://api.monobank.ua/docs
8. Learning React: Functional Web Development with React and Redux. Alex Banks and Eve Porcello. O'Reilly Media, 2020.
9. Node.js Design Patterns. Mario Casciaro and Luciano Mammino. Packt Publishing, 2020.
10. Ось додаткові електронні джерела, які ще не були включені до вашого списку:
11. Документація Material Tailwind [Електронний ресурс] – Режим доступу: https://www.material-tailwind.com/
12. Документація Axios [Електронний ресурс] – Режим доступу: https://axios-http.com/
13. Документація React Query [Електронний ресурс] – Режим доступу: https://tanstack.com/query/v4/docs
14. Документація ESLint [Електронний ресурс] – Режим доступу: https://eslint.org/
15. Документація TypeScript [Електронний ресурс] – Режим доступу: https://www.typescriptlang.org/docs/

16. Документація Prettier [Електронний ресурс] – Режим доступу: https://prettier.io/docs/
17. Документація Node.js [Електронний ресурс] – Режим доступу: https://nodejs.org/en/docs/
18. Pro JavaScript Design Patterns. Ross Harmes and Dustin Diaz. Apress, 2008.
19. Fullstack Vue: The Complete Guide to Vue.js. Hassan Djirdeh, Nate Murray, and Ari Lerner. Fullstack.io, 2020.
20. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 2020.
21. Garcia-Molina, H., Ullman, J. D., Widom, J. Database Systems: The Complete Book. Pearson, 2008.
22. Forta, B. SQL in 10 Minutes, Sams Teach Yourself. Sams Publishing, 2011.
23. Freeman, E., & Freeman, E. Head First Design Patterns. O'Reilly Media, 2004.
24. Legrand, T., & Fernandes, A. Building Social Web Applications: A New Era of Web-based Social Networking. O'Reilly Media, 2013.
25. Winand, M. SQL Performance Explained. self-published, 2012.
26. Social Media and Networking: A Guide for Beginners. Springer, 2020.
27. The Social Network Business Plan: 18 Steps to Building Your Company’s Billion-Dollar Strategy. Entrepreneur Press, 2011.
28. JavaScript: The Good Parts. Douglas Crockford. O'Reilly Media, 2008.
29. Eloquent JavaScript: A Modern Introduction to Programming. Marijn Haverbeke. No Starch Press, 2018.
30. You Don’t Know JS: Scope & Closures. Kyle Simpson. O'Reilly Media, 2014.
31. Professional JavaScript for Web Developers. Nicholas C. Zakas. Wrox Press, 2012.
32. Building Microservices with Node.js. David Gonzalez. Packt Publishing, 2016.
33. Web Development with Node and Express: Leveraging the JavaScript Stack. Ethan Brown. O'Reilly Media, 2019.

Додаток А

Програмний код

Фронтенд частина

Jar.ts

```
export interface TimestampData {
createdAt: string;
updatedAt: string;
}


export interface Jar extends TimestampData {
amount: number;
goal: number;
ownerIcon: string;
title: string;
ownerName: string;
currency: number;
description: string;
jarId: string;
longJarId: string;
blago: boolean;
closed: boolean;
}
```

ChatContainer.tsx

```
const ChatContainer: React.FC = () => {
const [messages, setMessages] = useState<ChatMessage[]>([]);
const inputRef = useRef<HTMLInputElement>(null);
const { mutateAsync, isPending } = useSendChatMessage();
const queryClient = useQueryClient();


const handleAiMessageResponse = ({ aiMessage, jars }: SendChatMessageResponse) => {
setMessages((prevState) => [...prevState, { content: aiMessage, role: "assistant" }]);


if (jars.length) {
queryClient.setQueriesData({ queryKey: ["jars"] }, { jars, pageCount: 1 });
}
};


const onSendMessage = async (message: string) => {
const userMessage: ChatMessage = {
content: message.trim(),
role: "user",
};
if (userMessage.content) {
setMessages((prev) => [...prev, userMessage]);


if (messages.length >= 2) {
try {
const messagesWithoutIntro = messages.slice(2);
const response = await mutateAsync([...messagesWithoutIntro, userMessage]);
handleAiMessageResponse(response);
```

```
} catch (error) {}
}
}
};


const onStartChattingClick = () => {
setMessages([
{
content: "Слава нації! Чим ти можеш бути корисним?",
role: "user",
},
]);
setTimeout(() => {
setMessages((prev) => [...prev, { content: INTRO_AI_MSG, role: "assistant" }]);
}, 500);
};


return (
<div className="flex flex-col h-full p-2 md:p-4 bg-deep-purple-50 rounded-lg shadow-md mb-8 lg:mb-12 relative">
<Chat
messages={messages}
isPandingAnswer={isPending}
/>
<div className="flex items-center mt-2 md:mt-4 space-x-2 justify-center">
{messages.length ? (
<ChatInput
ref={inputRef}
onSend={onSendMessage}
disabled={isPending}
/>
) : (
<Button
size="lg"
onClick={onStartChattingClick}
>
Активувати чат
</Button>
)}
</div>
</div>
);
};
```

CreateJarForm.tsx

```
const CreateJarForm: React.FC<CreateJarFormProps> = ({ className, ...rest }) => {
const createJarMutation = useCreateJarMutation();
const queryClient = useQueryClient();


const {
register,
handleSubmit,
formState: { errors },
reset,
} = useForm<CreateJarSchema>({
resolver: zodResolver(createJarSchema),
});


useEffect(() => {
```

```
reset();
}, [rest.isOpen, reset]);


const onSubmit: SubmitHandler<CreateJarSchema> = async (data) => {
const jarId = data.jarLink.split("/").at(-1);
const longJarId = new URLSearchParams(new URL(data.streamLink).search).get("longJarId");


const postData = {
jarId: jarId!,
longJarId: longJarId!,
};
try {
await createJarMutation.mutateAsync(postData);
rest.handleClose();
queryClient.invalidateQueries({ queryKey: [QUERY_KEYS.ALL_JARS] });
} catch (error) {
console.error(error);
}
};


return (
<ModalWindow
{...rest}
title="Добавити збір"
className=" w-[calc(100%-20px)] md:w-96"
>
<form
onSubmit={handleSubmit(onSubmit)}
className="flex flex-col gap-y-4"
>
<FormInput
label="Посилання на банку"
{...register("jarLink")}
error={errors.jarLink}
/>
<FormInput
label={
<div className="flex">
Посилання на стрімінговий віджет {<FindLongJarIdTooltip />}
</div>
}
{...register("streamLink")}
error={errors.streamLink}
/>
<div className="flex justify-end gap-2">
<Button
type="button"
onClick={rest.handleClose}
className="border-red-500 hover:bg-red-400 hover:text-white text-red-500 font-semibold"
variant="outlined"
>
Скасувати
</Button>
<Button
type="submit"
className="font-semibold"
>
Добавити
```

```
</Button>
</div>
</form>
</ModalWindow>
);
};
export default CreateJarForm;
```

JarsContainer.tsx

```
const JarsContainer: React.FC = () => {
const [searchParams, setSearchParams] = useSearchParams();
const direction = (searchParams.get("direction") || "desc") as SortOrder;
const pageSize = searchParams.get("pageSize") || DEFAULT_PAGE_SIZE;
const activePage = searchParams.get("page") || "1";
const isClosed = searchParams.get("closed") === "true";


const [page, setPage] = useState(+activePage);


useEffect(() => {
if (+activePage !== page) setPage(+activePage);
}, [activePage, page]);


const { data } = useGetAllJarsQuery({
sort: {
sortBy: "createdAt",
direction,
},
pagination: {
pageSize: +pageSize,
page: +page,
},
isClosed,
});


const { jars, pageCount = 1 } = data || {};


const onPageChange = (newPage: number) => {
setPage(newPage);
setSearchParams((prevState) => ({
...Object.fromEntries(prevState.entries()),
page: newPage.toString(),
}));
};


return (
<section id="jars">
<JarsNavigation />
{jars?.length && <JarsGrid jars={jars} />}
<JarsPagination
page={page}
setPage={onPageChange}
pageCount={pageCount}
/>
</section>
);
};
```

Api folder

```
export const useGetAllJarsQuery = (params: Parameters<typeof getJars>[0]) =>
useQuery<GetAllJarsData, QueryError>({
queryKey: [QUERY_KEYS.ALL_JARS, params],
queryFn: () => getJars(params),
refetchOnWindowFocus: false,
});


export const useCreateJarMutation = () =>
useMutation<Jar, QueryError, Parameters<typeof createJar>[0]>({
mutationFn: createJar,
onSuccess: () => {
showSuccessToast("Збір додано успішно!");
},
onError: (error) => {
console.error(error);
showErrorToast(error?.response?.data.message || "Добавлення збору не вдалось");
},
});


export const useSendChatMessage = () =>
useMutation<SendChatMessageResponse, QueryError, Parameters<typeof sendChatMessage>[0]>({
mutationFn: sendChatMessage,
onError: (error) => {
console.error(error);
showErrorToast(error?.response?.data.message || "Не вдалось відправити запит");
},
});


export const sendChatMessage = async (messages: ChatMessage[]) => {
const response = await axiosInstance.post<SendChatMessageResponse>("/messages", { messages });
return response.data;
};


export const getJars = async (params: {
sort: SortParams<Pick<Jar, "createdAt">>;
pagination: PaginationParams;
isClosed: boolean;
}): Promise<GetAllJarsData> => {
const response = await axiosInstance.get("/jars", { params });
return response.data;
};


export const createJar = async (newJar: Pick<Jar, "jarId" | "longJarId">): Promise<Jar> => {
const response = await axiosInstance.post<Jar>("/jars", newJar);
return response.data;
};


export const updateJar = async (updatedJar: Partial<Jar>): Promise<Jar> => {
const response = await axiosInstance.put<Jar>(`/jars/${updatedJar.jarId}`, updatedJar);
return response.data;
};
```

Серверна частина

Jar.service.ts

```
@Injectable()
export class JarService {
private jarRepository = createJarsRepository(this.prisma);

constructor(private prisma: PrismaService) {}

async getJarById(id: string): Promise<Jar | null> {
try {
return await this.jarRepository.findUniqueOrThrow({
where: { jarId: id },
select: jarSelect.default,
});
} catch {
throw new NotFoundException('Збір не знайдено');
}
}

async getJarsByIds(ids: string[]): Promise<Jar[]> {
return await this.jarRepository.findMany({
where: { jarId: { in: ids } },
orderBy: { createdAt: 'desc' },
select: jarSelect.default,
});
}
async getJars(
query: GetAllJarsParams,
): Promise<{ jars: Jar[]; pageCount: number }> {
const { isClosed, pagination, sort } = query;
const direction = sort?.direction || 'desc';
const { pageSize = '6', page = '1' } = pagination || {};
const closed = isClosed === 'true';

const total =
(
await this.jarRepository.findMany({
where: { closed },
})
)?.length || 0;
const pageCount = Math.max(1, Math.ceil(+total / +pageSize));

const jars = await this.jarRepository.findMany({
where: { closed },
select: jarSelect.default,
orderBy: { createdAt: direction as SortOrder },
take: parseInt(pageSize),
skip: (parseInt(page) - 1) * parseInt(pageSize),
});

jars.forEach((jar, index) => {
setTimeout(() => this.updateJarIfNeeded(jar), index * 1000);
});

return {
jars,
pageCount,
};
```

```
}

async createJar({ longJarId }: { longJarId: string }): Promise<Jar> {
const existingJar = await this.prisma.jar.findFirst({
where: { longJarId },
});
if (existingJar) {
throw new ConflictException('Цей збір уже існує');
}

const monoJar = await getMonoJarData(longJarId);
if (!monoJar) {
throw new NotFoundException('Збір не знайдено');
}
return this.jarRepository.create({
data: { ...monoJar, longJarId },
select: jarSelect.default,
});
}

async updateJarById(jarId: string): Promise<Jar> {
const existingJar = await this.getJarById(jarId);
if (!existingJar) {
throw new NotFoundException('Зібр не знайдено');
}

const monoJar = await getMonoJarData(existingJar.longJarId);
return this.jarRepository.update({
where: { jarId },
data: monoJar,
});
}

async deleteJar(jarId: string): Promise<Jar> {
await this.getJarById(jarId);
return this.jarRepository.delete({
where: { jarId },
select: jarSelect.default,
});
}

async getJarsDataForAI(): Promise<string> {
const jars = await this.jarRepository.findMany({
where: { closed: false },
select: jarSelect.aiSelect,
});

const data = {
jars: jars.map((jar) => ({
jarId: jar.jarId,
description: `${jar.title} ${jar.description} goalPrice:${divideBy100(jar.goal)}`,
})),
};
return JSON.stringify(data);
}
```

```
private async updateJarIfNeeded(jar: Jar): Promise<void> {
const oneDayInMillis = 24 * 60 * 60 * 1000;
const lastUpdated = new Date(jar.updatedAt);
const now = new Date();

if (now.getTime() - lastUpdated.getTime() > oneDayInMillis) {
await this.updateJarById(jar.jarId);
}
}
}
```

Message.service.ts

```
@Injectable()
export class MessagesService {
constructor(
private openAIService: OpenAIService,
private jarService: JarService,
) {}

async handleUserMessage(
messages: ChatMessage[],
): Promise<SendMessageResponse> {
const { responseMsg, jars } =
await this.openAIService.sendRequestToAI(messages);

let jarsData: Jar[] = [];
let aiMessageResponse = responseMsg || '';
if (jars?.length > 0) {
jarsData = await this.jarService.getJarsByIds(jars);

aiMessageResponse =
jars.length === 1
? `Збір "${jarsData[0].title}" - чудовий варіант згідно твоїм запитом. Ти можеш задонатити на нього нижче.`
: `Згідно твоїм запитом знайдено кілька хороших зборів. Ти можеш задонатити на них нижче.`;
} else if (jars?.length === 0) {
aiMessageResponse =
'На жаль, жодного збору в базі даних знайти не вдалось. Спробуй змінити свій запит.';
}

return {
aiMessage: aiMessageResponse,
jars: jarsData,
};
}
}
```

OpenAi.service.ts

```
@Injectable()
export class OpenAIService {
private openai: OpenAI;

constructor(private jarService: JarService) {
this.openai = new OpenAI();
}
```

```
async sendRequestToAI(messages: ChatMessage[]): Promise<AIResponse> {
const jarsInfoJSON = await this.jarService.getJarsDataForAI();
const systemMsg = `${OPENAI_SYSTEM_MSG} ${jarsInfoJSON}`;


const requestMessages: ChatCompletionMessageParam[] = [
{
role: 'system',
content: systemMsg,
},
...messages,
];


try {
const completion = await this.openai.chat.completions.create({
model: 'gpt-4o-mini',
messages: requestMessages,
});


const aiResult = completion.choices[0].message.content;
return JSON.parse(aiResult);
} catch (error: any) {
console.error(error);
throw new InternalServerErrorException(
'AI не відповідає. Спробуйте пізніше!',
);
}
}
}
```

Додаток Б

Тези доповіді

В.М. Волошин, В.Г. Резанова Впровадження штучного інтелекту в алгоритми веб-платформи збору коштів на ЗСУ // Мехатронні системи: інновації та інжиніринг : тези доповідей VIII Міжнародної науково-практичної конференції, 7 листопада 2024 р., м. Київ : КНУТД, 2024, с. 139-140

УДК 004.8

ВПРОВАДЖЕННЯ ШТУЧНОГО ІНТЕЛЕКТУ В АЛГОРИТМИ ВЕБ-ПЛАТФОРМИ ЗБОРУ КОШТІВ НА ЗСУ

В.М. Волошин, студент
Київський національний університет технологій та дизайну
В.Г. Резанова, кандидат технічних наук, доцент
Київський національний університет технологій та дизайну

Ключові слова: веб-платформа, веб-технології, штучний інтелект, алгоритми пошуку, збір коштів.

Веб-платформи для збору коштів на підтримку Збройних Сил України стають важливим інструментом залучення фінансових ресурсів на допомогу армії. Проте важливим завданням є ефективне надання рекомендацій користувачам, які шукають конкретні збори. Впровадження штучного інтелекту у веб-технології дозволяє забезпечити більш гнучке і персоналізоване спілкування з користувачами платформи та підвищити ефективність роботи сервісу.

Одним із перших рішень зі штучним інтелектом, які можна впровадити на платформу збору коштів, є чат-боти. Їх здатність одночасно спілкуватися з багатьма користувачами, надаючи миттєві відповіді 24/7, має трансформаційний вплив на обслуговування користувачів. Це зменшує час очікування та забезпечує вирішення основних питань без участі людини.

Чат-боти також відіграють важливу роль як перша точка контакту для користувачів із простими та часто запитуваними питаннями, що становлять більшість запитів. За даними Com100, у деяких випадках чат-боти здатні вирішувати до 91% запитів. Більше того, 45% клієнтів віддають перевагу використанню чат-ботів як основного каналу комунікації з службою підтримки.

Ще одним важливим елементом впровадження штучного інтелекту у веб-платформи є можливість автоматизованого аналізу великих обсягів даних про користувачів та їхню активність. Це дозволяє створювати більш точні прогнози щодо майбутніх дій користувачів, а також покращувати рекомендаційні алгоритми. Наприклад, система може відслідковувати, які збори отримують найбільше пожертвувань, і на основі цього пропонувати аналогічні збори іншим користувачам. Таким чином, штучний інтелект стає ключовим інструментом для оптимізації процесу збору коштів та підвищення залученості користувачів.

На рисунку нижче представлена структурна схема алгоритму веб-платформи з елементами штучного інтелекту. Двійними стрілками на схемі показано рух інформаційних потоків між клієнтом (користувачем), сервером, базою даних та системою OpenAI.

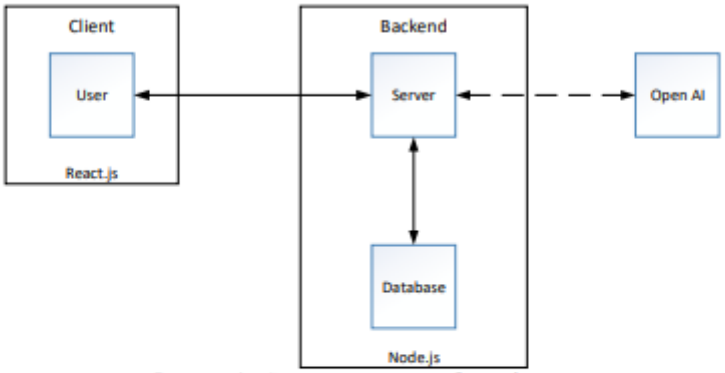


Рисунок 1 - Структурна схема веб-платформи

Процес взаємодії починається з того, що користувач відправляє свій запит (промт) на сервер. Сервер отримує запит і здійснює початкову обробку. Далі сервер має можливість взаємодіяти з базою даних для отримання необхідної інформації, яка може бути корисною для формування відповіді користувачу. У випадку необхідності, сервер передає запит до OpenAI для подальшого аналізу з використанням алгоритмів штучного інтелекту. Цей етап включає обробку природної мови, аналіз контексту та формування рекомендацій або відповідей на основі вхідного запиту користувача. Взаємодія між сервером і системою OpenAI здійснюється через хмару, що дозволяє використовувати потужні обчислювальні ресурси для швидкого аналізу даних. Отримавши відповідь від OpenAI, сервер знову звертається до бази даних (у разі потреби), щоб доповнити інформацію, якщо вона є частиною відповіді. Після цього сервер передає остаточну відповідь користувачу.

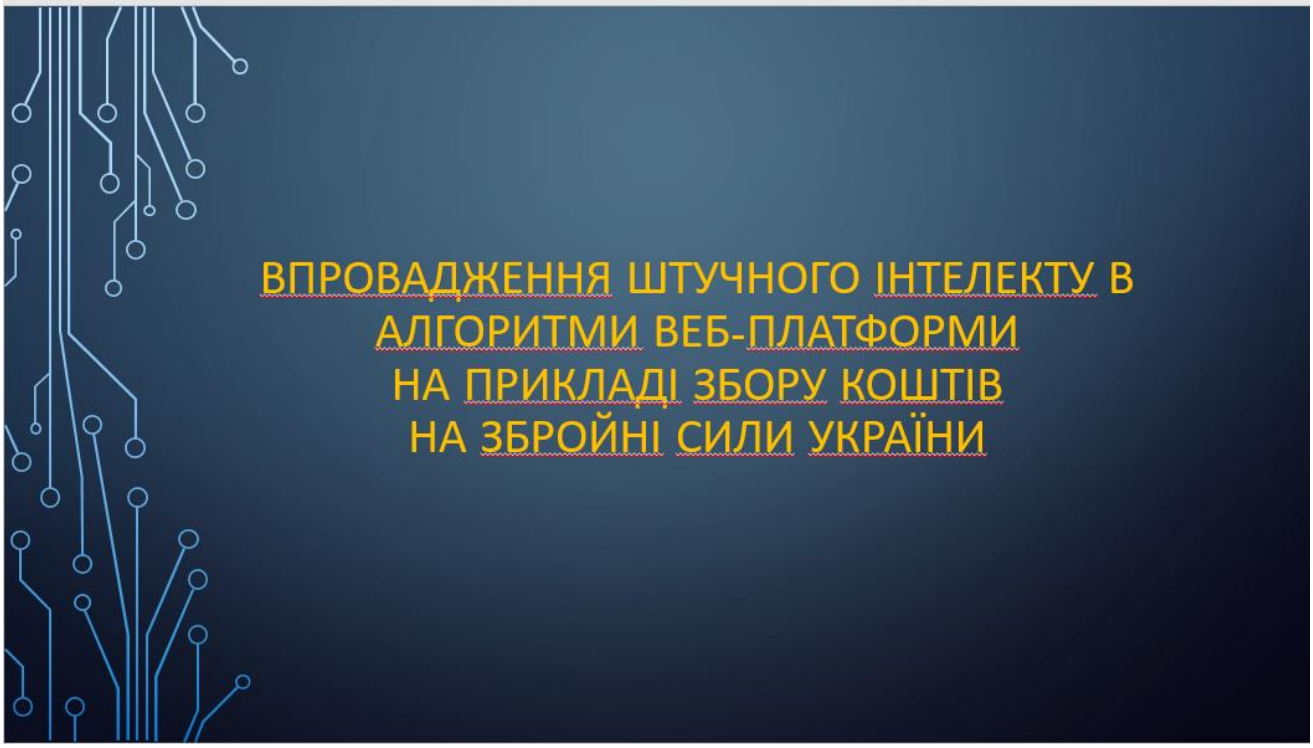
Результатом розробки передбачається, що ця архітектура з використанням штучного інтелекту в якості чат-бота дозволить створювати персоналізовані рекомендації для користувачів за їхніми запитами. Це значно підвищує ефективність зборів для ЗСУ на веб-платформі та її зручність для користувачів.

Список використаних джерел

1. Батареєв В.В. Методи та системи штучного інтелекту // Вісник Хмельницького національного університету. – 2021. – №1. – С. 17-21.
2. Шинкарук О.М. Управління якістю програмних веб-систем засобами розробки / О.М. Яшина, О.Г. Онишко // Вісник Хмельницького національного університету. – 2020. - №6. – С. 39-44.
3. Марусенко О. М., Мєтєльов В. О., Сенько А. В., Стрілець Ю. В. Розробка веб-застосунку для автоматизації формування проєктних команд // Вісник Національного технічного університету «ХПІ». – 2024. - №4. – С. 19-26.

Додаток В

Презентація



МЕТА РОБОТИ

Основною метою даної роботи є розробка веб-платформи ZSU-JARS, яка використовує штучний інтелект для оптимізації процесу збору коштів. Платформа створена з метою надання персоналізованих рекомендацій користувачам, покращення досвіду взаємодії з системою та збільшення ефективності зборів. ШІ інтегрується для автоматичного аналізу запитів, генерування відповідей та надання пропозицій відповідно до запитів користувача.

ТЕХНОЛОГІЇ ТА МОВИ ПРОГРАМУВАННЯ

Платформа ZSU-JARS розроблена із застосуванням сучасних технологій, які забезпечують продуктивність, зручність і масштабованість.

Мови програмування:

- JavaScript
- TypeScript

OpenAI

nest

TS

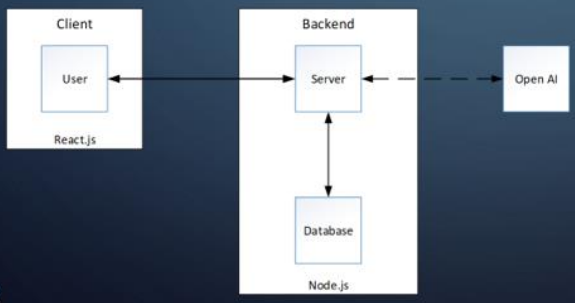
Основні фреймворки та бібліотеки:

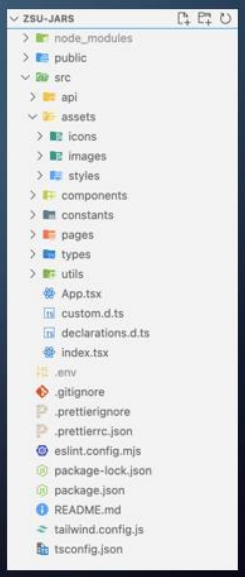
- React
- Nest.js
- Prisma
- TailwindCSS
- OpenAI GPT
- React Query

React JS

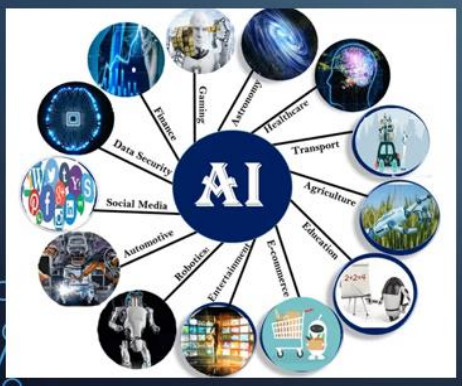
ЗАГАЛЬНА АРХІТЕКТУРА

Використано клієнт-серверну архітектуру застосунку, фронтенд на React, бекенд на Nest.js, база даних SQLite та інтеграція OpenAI. Впроваджено просту архітектуру організацію коду





ШТУЧНИЙ ІНТЕЛЕКТ: МОЖЛИВОСТІ ТА ЗАСТОСУВАННЯ



Штучний інтелект (ШІ) — це галузь комп'ютерних наук, що займається створенням систем, здатних виконувати завдання, які зазвичай вимагають людського інтелекту. Це включає розпізнавання мови, навчання, прийняття рішень і розуміння природної мови.

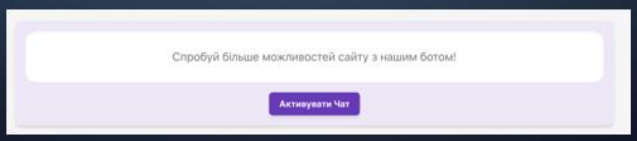
ШІ надає широкі можливості, такі як аналіз великих обсягів даних для виявлення закономірностей, створення контенту (текстів, зображень, відео), персоналізація взаємодії з користувачами та автоматизація рутинних завдань. Його застосування охоплює електронну комерцію, охорону здоров'я, освіту, веб-платформи (чат-боти, автоматизовані процеси) та багато інших.

ІНТЕГРАЦІЯ ШІ

Штучний інтелект є основою персоналізованих рекомендацій на платформі.

- ШІ аналізує запити користувачів і порівнює їх із даними про збори, які зберігаються в базі.
- Генерує відповіді, адаптовані до контексту запитів. Наприклад, бот може надати список зборів, які відповідають конкретним вимогам користувача.
- Чат-бот працює на основі OpenAI GPT-4mini, що забезпечує швидку обробку запитів і економію токенів для зниження витрат.

Завдяки цьому платформа стає більш зручною, продуктивною та корисною для користувачів.



ТОКЕНИ ТА ЇХ ОПТИМІЗАЦІЯ

Токени використовуються для взаємодії з моделями OpenAI. Кожен запит до ШІ розраховується за кількістю токенів, які є частинами слів (1000 токенів ≈ 750 слів). Оптимізація промптів дозволила зменшити витрати і підтримувати швидкість обробки запитів. Наприклад, ліміти символів у чаті запобігають надмірному використанню токенів.

Модель	Ціна за 1K токенів (вхідні дані)	Ціна за 1K токенів (вихідні дані)
GPT-4o	$0.03	$0.06
GPT-4mini	$0.015	$0.03
GPT-3.5-turbo	$0.0015	$0.002
Davinci	$0.02	$0.02
Curie	$0.002	$0.002
Babbage	$0.0005	$0.0005
Ada	$0.0004	$0.0004

ВИКОРИСТАННЯ ТОКЕНІВ

$$\text{Середнє} = \frac{24,023 \text{ токенів}}{32 \text{ запити}} = 750.72 \text{ токенів на запит.}$$

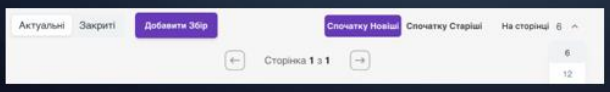
На графіку представлено статистику використання токенів для моделі **GPT-4o-mini**. Загалом було здійснено **32 запити**, під час яких витрачено **24,023 токени**

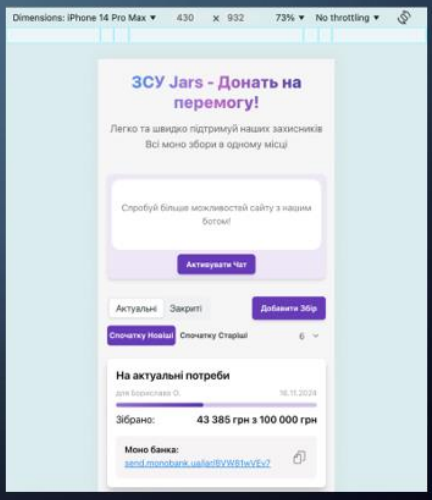
Використання штучного інтелекту, зокрема OpenAI API, є ефективним для роботи з даними та рекомендаціями, однак поточна вартість запитів може бути доволі високою (1 запит = 50 копійок). Це вимагає більшої оптимізації використання токенів і кешування для зменшення витрат.

ФУНКЦІОНАЛЬНІСТЬ ПЛАТФОРМИ

Платформа ZSU-JARS забезпечує:

- Пошук і сортування зборів за різними критеріями.
- Пагінацію для управління великими списками зборів.
- Інтеграцію з Monobank API для автоматичного оновлення даних про збори.
- Реалізацію адаптивного інтерфейсу, який підтримує різні розширення екранів.

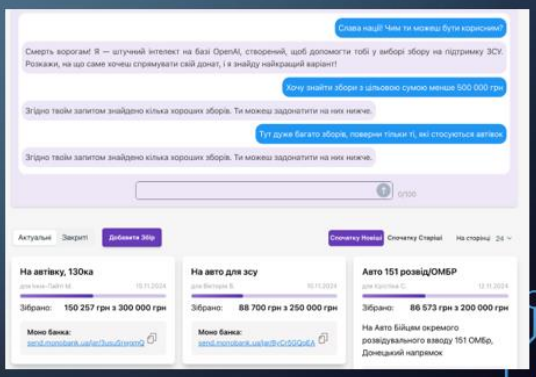




ПЕРЕВАГИ ПЛАТФОРМИ

Платформа ZSU-JARS має такі переваги:

- Використання передових технологій, таких як React, Nest.js і OpenAI.
- Зручність для користувачів завдяки адаптивному дизайну і персоналізованим рекомендаціям.
- Висока швидкість обробки запитів і генерації відповідей.



ВИСНОВКИ

Результатом роботи є сучасна веб-платформа ZSU-JARS, яка забезпечує ефективну підтримку зборів для ЗСУ, інтерактивність і персоналізовані рекомендації. Використання штучного інтелекту значно покращує взаємодію з користувачами, дозволяючи їм швидко знаходити збори, що відповідають їхнім потребам. Платформа має високий потенціал для подальшого розвитку, включаючи розширення функціоналу, інтеграцію з іншими сервісами та масштабування для підтримки більшої кількості користувачів.