Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

Допущено до захисту
Завідувач кафедри
комп'ютерних наук, канд. тех. наук, доцент

_Ірина МАШКІНА
«_____» _____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
Спеціальність 122 Комп'ютерні науки
Освітня програма 122.00.01 Інформатика

Тема: Розробка програмного забезпечення для управління особистими фінансами з використанням засобів штучного інтелекту

Виконав
студент групи ІНб-2-21-4 0д
Арнаут Вадим Віталійович **Науковий керівник**
Кандидат технічних наук , доцент
Яскевич В О

_____

Київ – 2025

РЕФЕРАТ

Дипломна робота студента Арнаута Вадима Віталійовича на тему: *«Розробка програмного забезпечення для управління особистими фінансами з використанням засобів штучного інтелекту»* Напрям підготовки: 122 – Комп’ютерні науки

Ця робота присвячена створенню вебзастосунку FinScope - інструменту для ведення особистих фінансів, який не просто зберігає витрати, а допомагає краще їх зрозуміти

Мета проєкту - дати користувачеві просту, але розумну систему, яка аналізує фінансову поведінку і дає поради на основі реальних даних Для цього використано React на фронтенді, Node js і MongoDB на сервері, а також Python-модуль на Flask для реалізації елементарної аналітики

Виконано: розробку архітектури застосунку (фронтенд, бекенд, база, ШІ-модуль); створення інтерфейсу користувача з фокусом на простоту та UX; реалізацію API для керування транзакціями та категоріями; інтеграцію з Flask-сервером для аналізу витрат; побудову графіків та формування інсайтів у вигляді коротких порад; деплоймент на Vercel і Heroku, конфігурація CORS та авторизації

Проєкт включає авторизацію, захист маршрутів, динамічну візуалізацію фінансових даних, і приклад базового AI-модуля, що надає користувачеві поради типу “Зверніть увагу: витрати на їжу зросли на 42%”

Обсяг роботи: 58 сторінок, 24 рисунки, 8 додатків Кількість використаних джерел: 24

ВСТУП

У сучасному світі фінансова грамотність та здатність до ефективного управління особистими коштами є не просто бажаними якостями, а необхідною умовою економічної стабільності та незалежності кожної людини Швидкі темпи цифровізації, збільшення обсягу безготівкових операцій, зростання кількості доступних фінансових інструментів і розширення функціоналу банківських та фінансових застосунків створюють як нові можливості, так і нові виклики для користувачів Людина змушена не лише контролювати доходи і витрати, а й аналізувати свої фінансові звички, будувати прогнози та ухвалювати рішення в умовах обмежених ресурсів і великого обсягу інформації

Традиційні способи обліку витрат, як-от нотатки на папері, таблиці Excel чи банальні калькулятори, вже не відповідають потребам сучасного користувача Вони не забезпечують аналітичної глибини, не автоматизують процесів і не дозволяють адаптувати фінансове планування до змін у

поведінці користувача У цьому контексті на перший план виходять **інтелектуальні вебзастосунки**, які здатні не лише реєструвати транзакції, а й допомагати людині приймати зважені фінансові рішення на основі даних, рекомендацій і прогнозів

Особливої актуальності набуває впровадження елементів **штучного інтелекту** (ШІ) у такі системи Моделі машинного навчання, здатні аналізувати витрати, класифікувати транзакції, виявляти нетипові патерни або формувати персоналізовані поради, значно підвищують цінність подібних застосунків для кінцевого користувача Це дозволяє перейти від простої автоматизації до **інтелектуального асистування** у сфері особистих фінансів

Усе це зумовлює актуальність теми кваліфікаційної роботи - створення сучасного вебзастосунку з підтримкою ШІ для управління особистими фінансами, що відповідає запитам ринку та потребам цифрового покоління

Мета роботи:

Мета роботи полягає у розробці вебзастосунку, який дозволяє користувачеві не лише зручно фіксувати свої фінансові операції, а й отримувати корисну аналітику, автоматичну класифікацію витрат, а також персоналізовані рекомендації на основі аналізу фінансової поведінки з використанням інструментів штучного інтелекту

Об’єктом дослідження є процес управління особистими фінансами в умовах цифрового середовища

Предметом дослідження є технології та методи створення вебзастосунку з функціональністю збору, обробки, збереження й інтелектуального аналізу

фінансових даних користувача, зокрема - застосування інструментів штучного інтелекту в таких системах

Завдання кваліфікаційної роботи:

- Проаналізувати наявні програмні засоби для управління особистими фінансами, визначити їхні сильні та слабкі сторони
- Визначити функціональні та нефункціональні вимоги до майбутнього застосунку
- Обґрунтувати вибір архітектурних рішень, інструментів і стеку технологій
- Реалізувати клієнтську частину застосунку (frontend) із сучасним адаптивним інтерфейсом
- Реалізувати серверну частину (backend), API, базу даних і обробку транзакцій
- Розробити та інтегрувати модуль ШІ для класифікації, аналізу витрат і надання рекомендацій
- Провести тестування функціональності та забезпечити розгортання застосунку в хмарному середовищі з CI/CD

Методи дослідження:

- **аналіз предметної області** - вивчення існуючих аналогів, їхньої функціональності, недоліків та сильних сторін;
- **порівняльний аналіз інструментів розробки** - вибір стеку технологій з урахуванням вимог до продуктивності, масштабованості та зручності підтримки;

- **моделювання та проєктування** - побудова архітектури клієнт-серверної системи з урахуванням розподілу відповідальностей між модулями;
- **прототипування** - створення та тестування робочого макета інтерфейсу користувача;
- **реалізація програмного забезпечення** - написання коду з дотриманням принципів модульності та читабельності;
- **використання бібліотек штучного інтелекту** - застосування готових моделей та алгоритмів для класифікації витрат і генерації рекомендацій;
- **тестування** - перевірка функціональності, стабільності, коректності роботи кожного модуля та інтегрованої системи;
- **розгортання** - організація CI/CD-процесу та публікація застосунку в хмарному середовищі

Структура пояснювальної записки:

- Кваліфікаційна робота складається зі вступу, шести основних розділів, висновків, та списку використаних джерел
- **У першому розділі** розглянуто предметну область, проаналізовано існуючі рішення, окреслено актуальність проблеми
- **У другому розділі** наведено обґрунтування вибору архітектурних рішень та інструментів реалізації
- **У третьому розділі** детально описано процес реалізації фронтенду, бекенду, бази даних і AI-модуля
- **У четвертому розділі** викладено процедуру розгортання застосунку та налаштування CI/CD
- **П’ятий розділ** присвячено тестуванню системи, виявленню помилок і оптимізації продуктивності

- **У шостому розділі** сформульовано основні висновки щодо результатів розробки

Розділ 1: Аналіз проблемної області та огляд аналогів

1.1 Актуальність проблеми

У сучасному суспільстві, де фінансова інформація доступна майже миттєво, а способи витрат постійно змінюються, контроль над особистими коштами стає дедалі складнішим завданням Швидкий розвиток цифрових технологій привів до зростання кількості онлайн-платежів, мікротранзакцій, підписок та інших регулярних витрат, які складно відстежувати без відповідного програмного забезпечення

Більшість користувачів бажають не лише фіксувати витрати, а й отримувати розгорнуту аналітику, бачити тенденції та слабкі місця у фінансовій поведінці Водночас, існуючі сервіси часто обмежуються простим обліком транзакцій і не враховують індивідуальні потреби кожного користувача

Особливого значення набуває використання **інтелектуальних технологій** - таких, що дозволяють здійснювати персоналізовані фінансові рекомендації, автоматично категоризувати витрати та прогнозувати майбутні фінансові показники Саме це відкриває новий рівень взаємодії між користувачем і фінансовим інструментом

Таким чином, створення застосунку, який поєднує **зручність інтерфейсу, сучасну архітектуру та елементи штучного інтелекту**, є актуальним як з точки зору технологій, так і з практичної користі для широкого кола користувачів

1.2 Типові задачі управління фінансами

Управління особистими фінансами - це комплексна діяльність, що включає не лише фіксацію доходів і витрат, а й планування, аналіз, прийняття рішень та постійну оптимізацію Користувачі очікують, що застосунок для фінансового менеджменту допоможе вирішити низку практичних задач, серед яких найпоширенішими є:

1. **Ведення обліку доходів і витрат:** Автоматичне або ручне внесення фінансових операцій із зазначенням категорій, дати, суми, способу оплати
2. **Категоризація витрат** Автоматичне або напівавтоматичне розподілення витрат за категоріями (їжа, транспорт, розваги тощо) для подальшої аналітики
3. **Візуалізація бюджету** Побудова діаграм, графіків і таблиць, які показують, на що витрачаються кошти і як змінюється фінансова поведінка з часом
4. **Постановка фінансових цілей** Визначення коротко- та довгострокових цілей (накопичення, виплата боргів тощо) з можливістю моніторингу прогресу
5. **Аналіз регулярних платежів** Виявлення підписок і регулярних витрат, які можуть залишатись непоміченими або втрачати актуальність
6. **Прогнозування витрат** Побудова моделей майбутніх витрат на основі історичних даних та поведінкових патернів
7. **Фінансові поради** Надання персоналізованих рекомендацій, заснованих на даних користувача: «де можна зекономити», «що змінити у структурі витрат», тощо

Ці задачі створюють основу для побудови функціоналу застосунку FinScope Їх реалізація із використанням сучасних технологій дає змогу значно підвищити фінансову обізнаність користувачів та поліпшити якість їхніх фінансових рішень

1.3 Огляд існуючих сервісів

У світі існує велика кількість застосунків для управління особистими фінансами Вони відрізняються за функціональністю, платформою, рівнем автоматизації, дизайном та підтримкою користувачів Проте навіть найпопулярніші з них мають суттєві обмеження: недостатня персоналізація, відсутність ШІ-компонентів, складність інтерфейсу або надлишкова комерціалізація функцій

Нижче розглянуто найвідоміші та найбільш вживані фінансові застосунки, які стали точками відліку під час планування архітектури FinScope

CoinKeeper

Мобільний застосунок із візуальним бюджетуванням (монетки в гаманці), орієнтований на початкових користувачів Сильна сторона - простота, слабка - обмеженість аналітики

Wallet by BudgetBakers

Один із найбільш функціональних сервісів Має багаторівневу аналітику, підключення банків, планування цілей, експорт у таблиці Проте інтерфейс перевантажений, а ШІ-рішень немає

Spendee

Простий та привабливий інтерфейс, хороший візуальний стиль Надає основну аналітику, але не має просунутих функцій чи інтелектуальної допомоги

Toshl Finance

Один із найстаріших сервісів, орієнтований на фрілансерів Пропонує мультивалютність, планування, інтеграції Відсутня підтримка штучного інтелекту

YNAB (You Need A Budget)

Популярний сервіс із філософією «кожному долару - робота» Має складну, але гнучку систему бюджетування Не підходить для новачків і потребує звикання Платний

Параметр	Mint	YNAB	Personal Capital	PocketGuard	Goodbudget	Empower (раніше Personal Capital)
Основна функціональність	Управління витратами, бюджети, інтеграція з банківськими рахунками	Планування бюджету, звітність, контроль фінансів	Фінансовий моніторинг, управління інвестиціями	Контроль витрат, визначення доступного бюджету	Метод конвертів для управління витратами, ручне введення даних	Бюджетування, інвестиційний аналіз, фінансові рекомендації
Архітектура	Клієнт-сервер, REST API	Клієнт-сервер, REST API	Клієнт-сервер, інтеграція з банківськими платформами	Клієнт-сервер, REST API	Клієнт-сервер, REST API	Клієнт-сервер, інтеграція з банками та біржами
Використані технології	React, Node.js, PostgreSQL	Angular, Ruby on Rails, PostgreSQL	React, Node.js, MongoDB	React Native, Firebase	Java, Spring Boot, MySQL	React, Node.js, SQL Server
ШІ-функціональність	Класифікація транзакцій за категоріями	Немає	Інвестиційна аналітика, прогнозування портфелю	Автоматичне розпізнавання категорій витрат	Немає	Автоматичний аналіз інвестиційного портфелю
Інтерфейс	Сучасний, інтеграція з фінансовими інструментами	Чистий інтерфейс, фокус на простоті	Деталізований, оптимізований для професійних користувачів	Мобільний фокус, легкий доступ	Спрощений для бюджетування вручну	Професійний аналітичний інтерфейс
Переваги	Інтуїтивно зрозумілий інтерфейс, автоматизація витрат	Детальний контроль бюджету, освітні ресурси	Сильний інструмент для фінансових аналітиків	Зручність для щоденного використання, мобільна інтеграція	Методика конвертів, низькі вимоги до навчання	Розширена підтримка інвестицій, гнучкість інтеграцій
Недоліки	Немає функції прогнозування, працює здебільшого в США	Платна підписка, відсутність гнучкості для інтеграції	Ускладнений інтерфейс для звичайного користувача	Обмежені аналітичні можливості	Відсутність автоматизації, обмежені функції	Складний для нових користувачів, орієнтація на США

Figure 1.1 Порівняльна таблиця функціональності популярних застосунків для управління особистими фінансами

1.4 Проміжні висновки

У результаті аналізу проблемної області та наявних рішень було встановлено, що попри велику кількість сервісів для управління особистими фінансами, більшість із них:

- орієнтовані на базову реєстрацію витрат, без глибокої аналітики;
- не мають механізмів персоналізації фінансових порад;
- обмежені у використанні інструментів штучного інтелекту;
- мають складний або перевантажений інтерфейс;
- не надають користувачу цілісного уявлення про власні фінанси

Таким чином, постає завдання створити застосунок, який не просто копіює функціональність існуючих рішень, а дає **нову якість** користувацького досвіду - шляхом поєднання простоти інтерфейсу, гнучкої аналітики та **AI-підказок**, що справді допомагають користувачу ухвалювати обґрунтовані фінансові рішення

Отримані висновки стали основою для формування вимог до системи, вибору технологій та архітектурних рішень, що будуть розглянуті у наступному розділі

Розділ 2 Обґрунтування вибору засобів реалізації

2.1 Вимоги до системи

Перед розробкою будь-якого програмного забезпечення необхідно чітко сформулювати вимоги до функціональності, продуктивності, безпеки та користувацького досвіду Це дозволяє правильно підібрати інструменти, визначити архітектуру, а також уникнути проблем у процесі реалізації та масштабування

Усі вимоги умовно поділяються на **функціональні** та **нефункціональні**

Функціональні вимоги:

- Реєстрація та аутентифікація користувачів
- Створення, перегляд, редагування та видалення фінансових транзакцій
- Автоматична та ручна категоризація витрат
- Візуалізація витрат у вигляді графіків (пончики, стовпчики, лінії)
- Інтеграція з модулем ШІ для:
- аналізу витрат за категоріями;
- виявлення аномалій;
- надання персоналізованих порад
- Розділення транзакцій за користувачами (багатокористувацька система)
- Підтримка адаптивного інтерфейсу для мобільних пристроїв

Нефункціональні вимоги:

- Швидкий час відгуку (менше 300 мс для основних операцій)
- Надійне збереження даних (реплікація, регулярне резервне копіювання)
- Безпека даних (хешування паролів, захищене з'єднання, валідація введення)
- Масштабованість - можливість розширення системи без переробки архітектури
- Підтримка багатоплатформенності (десктоп/мобільні браузери)
- Простота у підтримці й розгортанні (контейнеризація, CI/CD)

Ці вимоги стали відправною точкою для вибору архітектурних підходів та інструментів реалізації, які розглядаються в наступних підрозділах

2.2 Вибір архітектурної моделі

Вибір архітектури програмного забезпечення відіграє ключову роль у забезпеченні надійності, масштабованості та зручності розробки застосунку У межах даної кваліфікаційної роботи було прийнято рішення реалізувати застосунок FinScope за **класичною клієнт-серверною моделлю**, яка є однією з найбільш поширених для вебзастосунків

Клієнт-серверна архітектура

Ця модель передбачає чіткий розподіл обов'язків між двома сторонами:

Клієнт (frontend) відповідає за взаємодію з користувачем, візуалізацію даних, збирання введеної інформації та надсилання запитів до сервера

Сервер (backend) обробляє запити клієнта, взаємодіє з базою даних, реалізує бізнес-логіку, обробляє та повертає відповідні відповіді

Архітектурні переваги такого підходу:

Гнучкість: окремий розвиток клієнта й сервера, можливість заміни або масштабування окремих компонентів

Безпека: можливість контролю доступу до даних через централізований сервер

Простота тестування: модулі можна перевіряти незалежно

Розширюваність: легко додавати нові функції (наприклад, нові API або модулі ШІ)

Інтеграція модуля ШІ

Для реалізації функціоналу штучного інтелекту (класифікація витрат, поради, виявлення аномалій) застосовується **допоміжний сервер на Python (Flask)**, який працює окремо від основного Node js-сервера Він взаємодіє

через API - тобто запити з основного backend надсилаються до Flask-серверу, який повертає оброблені результати

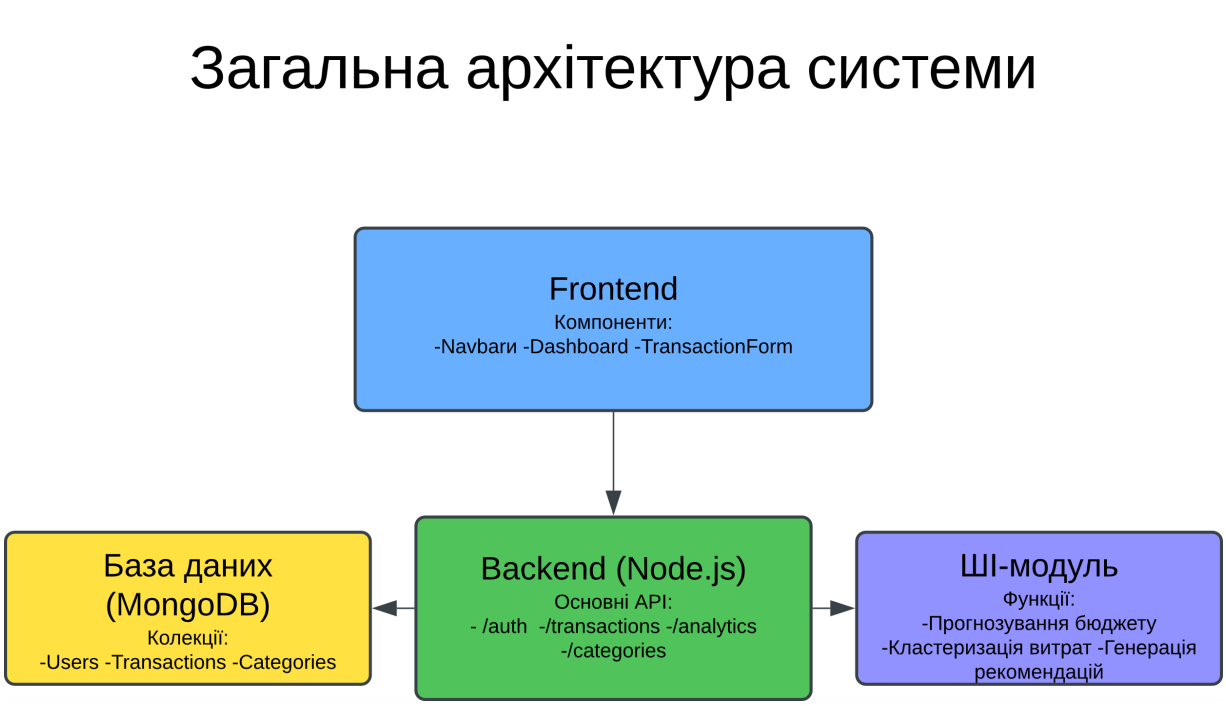


Figure 2.1 Загальна архітектура системи FinScope із поділом на фронтенд, бекенд, базу даних та ШІ-модуль

Обрана архітектура забезпечує модульність, надійність і простоту в подальшому масштабуванні системи, що є критично важливим для сучасного вебзастосунку

2.3 Вибір інструментів для фронтенду

Для реалізації клієнтської частини застосунку FinScope було обрано сучасний стек інструментів, що забезпечує високу продуктивність, зручність розробки та адаптивний інтерфейс для користувачів на різних пристроях Основу фронтенду становить **JavaScript-фреймворк React**, який доповнено бібліотеками Tailwind CSS та Chart js

React

React - це популярна бібліотека для побудови інтерфейсів користувача, розроблена компанією Meta Вона дозволяє створювати масштабовані й динамічні SPA (Single Page Applications) з компонентною архітектурою

Переваги використання React:

- Висока продуктивність завдяки Virtual DOM
- Компонентний підхід, що спрощує повторне використання коду
- Велика спільнота та підтримка екосистеми
- Добра інтеграція з іншими бібліотеками

Tailwind CSS

Tailwind CSS - утилітарна CSS-бібліотека, яка дозволяє створювати адаптивні інтерфейси без написання кастомних стилів Її використання суттєво зменшує обсяг CSS-коду та прискорює процес верстки

Чому Tailwind CSS:

- Швидка розробка інтерфейсу без перемикання між HTML і CSS
- Висока адаптивність
- Сучасний візуальний стиль без потреби в UI-фреймворках

Chart js

Для реалізації візуалізації фінансових даних (графіки витрат, динаміка по категоріях) використовується Chart js - потужна JS-бібліотека для створення інтерактивних графіків і діаграм

Переваги:

- Підтримка великої кількості типів графіків (лінійні, кругові, стовпчикові)

- Добра інтеграція з React через обгортки (react-chartjs-2)
- Простота налаштувань та кастомізації

Вибраний стек фронтенду дозволяє побудувати швидкий, гнучкий і сучасний інтерфейс користувача, який легко підтримується та масштабовується

2.4 Вибір інструментів для бекенду та бази даних

Серверна частина застосунку FinScope реалізована на платформі **Node js** із використанням фреймворку **Express js** Для зберігання структурованих фінансових даних обрано **MongoDB** - сучасну документно-орієнтовану NoSQL базу даних, що добре поєднується з JavaScript-стеком

Node js + Express js

Node js - це середовище виконання JavaScript на стороні сервера, що дозволяє створювати масштабовані та високопродуктивні мережеві застосунки Express js - мінімалістичний вебфреймворк, що працює поверх Node js і спрощує створення API та обробку запитів

Переваги цього підходу:

- Один стек JavaScript на фронті й на бекенді → спрощення розробки
- Висока швидкість обробки запитів завдяки неблокуючій моделі
- Велика екосистема npm-бібліотек
- Легка інтеграція з MongoDB, JWT, і сторонніми сервісами

MongoDB

MongoDB - база даних, яка зберігає дані у вигляді JSON-подібних документів Це ідеальне рішення для проєктів, де дані мають гнучку структуру й швидко змінюються, як у випадку з транзакціями користувачів

Переваги MongoDB:

- Гнучка структура - можна зберігати різні типи даних без зміни схеми
- Добра масштабованість і висока швидкість читання/запису
- Простий синтаксис запитів, сумісний із JavaScript
- Можливість використання хмарного сервісу MongoDB Atlas із безкоштовним тарифом

Бонусом є використання бібліотеки **Mongoose**, яка дозволяє описувати моделі документів, валідувати дані та полегшує роботу з MongoDB через об'єктно-документне відображення (ODM)

Цей стек забезпечує стабільну та надійну основу для зберігання та обробки даних у застосунку FinScope, а також легко масштабується у майбутньому при зростанні кількості користувачів

2.5 Вибір технологій для штучного інтелекту

Однією з ключових відмінностей застосунку FinScope є наявність інтегрованого модуля штучного інтелекту, який забезпечує інтелектуальну обробку транзакцій, виявлення аномальних витрат і формування персоналізованих рекомендацій Для реалізації цього функціоналу було обрано стек технологій на основі **Python** та **Flask**, а також базові алгоритми машинного навчання

Flask (Python)

Flask - це легкий вебфреймворк для Python, що ідеально підходить для створення API та мікросервісів Він дозволяє швидко розгорнути сервер, який буде отримувати дані від основного бекенду Node js, обробляти їх і повертати результат у форматі JSON

Переваги Flask для ШІ-модуля:

- Простота у розгортанні та інтеграції;
- Добра сумісність з бібліотеками Python;
- Мінімалістичний підхід - лише потрібне;
- Підтримка форматів REST API

Python-бібліотеки машинного навчання

Для реалізації інтелектуального аналізу транзакцій були використані базові інструменти Python:

- **scikit-learn** - для кластеризації та класифікації витрат;
- **NumPy, Pandas** - для обробки, групування та агрегації фінансових даних;
- **joblib** - для серіалізації моделей, якщо потрібно зберігати попередньо навчені шаблони

Алгоритми, що використовуються:

- **K-Means** - для виявлення кластерів витрат за схожими патернами;
- **Лінійна регресія** - для прогнозування витрат на основі історичних даних;
- **Дерева рішень або евристики** - для генерації текстових порад;

Інтеграція з основною системою

Flask-сервер працює як окремий мікросервіс, до якого Node js-бекенд надсилає запит /analyze, передаючи масив транзакцій у форматі JSON Після обробки Python-сервер повертає результат - список висновків, категорії ризику або рекомендацій

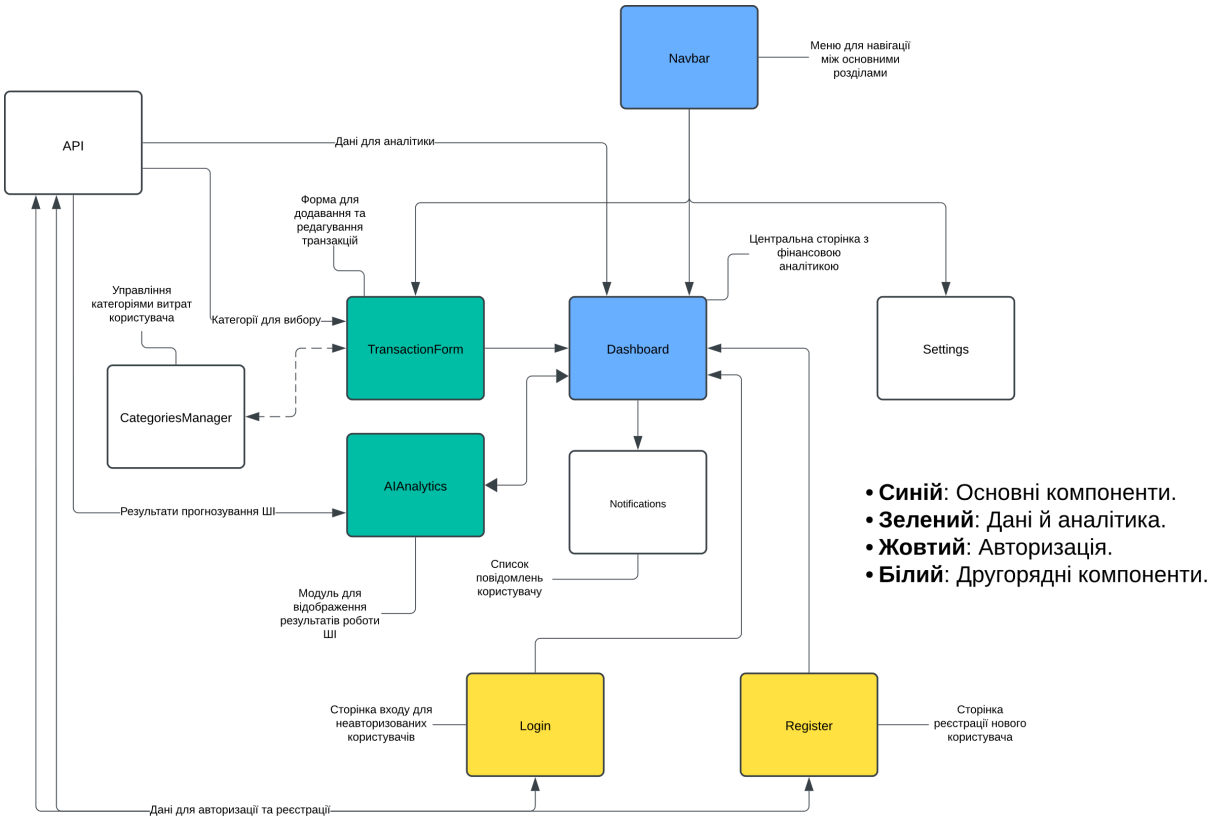


Figure 2 2 Компонентна структура фронтенду FinScope та взаємозв'язки між модулями інтерфейсу

Обрані технології забезпечують просту, надійну й ефективну інтеграцію інтелектуального аналізу у фінансовий застосунок з мінімальними витратами на підтримку

2.6 Висновки

У цьому розділі було обґрунтовано вибір архітектури, інструментів та технологій, використаних для реалізації застосунку FinScope Основою рішення стала потреба у створенні гнучкої, масштабованої та сучасної системи, яка дозволяє не лише обліковувати фінансові операції, а й надавати користувачу інтелектуальну підтримку у прийнятті фінансових рішень

Вибір **клієнт-серверної архітектури** з чітким поділом на фронтенд, бекенд і окремий модуль штучного інтелекту забезпечив модульність,

простоту тестування, можливість незалежного масштабування кожного з компонентів

Фронтенд реалізовано з використанням **React** і **Tailwind CSS**, що дозволило створити адаптивний, зручний та продуктивний інтерфейс Бекенд на **Node js + Express** забезпечує надійний обробник запитів і логіку взаємодії з базою даних **MongoDB**, яка, своєю чергою, гарантує гнучкість у зберіганні фінансових даних користувачів

Окрему увагу приділено вибору **інструментів для реалізації ШІ-модуля**, де застосовано **Python, Flask** і базові алгоритми машинного навчання Така інтеграція дозволяє підняти функціональність фінансового застосунку на новий рівень, надаючи користувачеві не лише статистику, а й аналітичні висновки

Таким чином, обрані засоби реалізації відповідають вимогам, поставленим на етапі аналізу системи, і повною мірою дозволяють реалізувати поставлену мету роботи

Розділ 3 Реалізація застосунку

3.1 Реалізація клієнтської частини (frontend)

Клієнтська частина застосунку FinScope реалізована з використанням бібліотеки **React** у поєднанні з **Tailwind CSS** для стилізації та **Axios** для взаємодії з API Структура фронтенду базується на компонентному підході, що дозволяє розділити інтерфейс на незалежні блоки з чіткими зонами відповідальності

Основні компоненти інтерфейсу:

<App /> - головний контейнер, який ініціалізує маршрутизацію та глобальні провайдери

<Navbar /> - верхня панель навігації з адаптивною поведінкою

<Login />, <Register /> - форми автентифікації

<Dashboard /> - основна панель з графіками, підсумками та швидкими діями

<TransactionForm /> - компонент для додавання/редагування витрат

<TransactionList /> - список останніх транзакцій із можливістю фільтрації

<Analytics /> - секція для візуалізації динаміки витрат

<AIInsights /> - компонент для відображення порад, згенерованих ШІ

<Settings /> - зміна персональних параметрів та категорій

Маршрутизація

Для навігації використано **React Router v6** Застосунок поділено на приватні (доступні після входу) та публічні маршрути

Figure 3 1 Налаштування маршрутизації в React-додатку із захищеним доступом до сторінки Dashboard

Стан застосунку

Керування глобальним станом реалізовано за допомогою React Context API для аутентифікації та Zustand для зберігання транзакцій та теми інтерфейсу

Візуальна складова

За основу стилізації використано **Tailwind CSS**, що дало змогу:

швидко реалізувати адаптивну верстку, використовувати темну/світлу теми, забезпечити консистентність UI без сторонніх UI-фреймворків

Зв'язок з API

Для відправки запитів застосовується **Axios**, у якому налаштовано інтерцептор для автоматичного додавання токена авторизації до всіх запитів:

Figure 3 2 Перехоплення запитів у Axios для автоматичної передачі токена авторизації

Завдяки вибору React та Tailwind було створено легкий, гнучкий та візуально приємний інтерфейс, який забезпечує користувачу інтуїтивний доступ до всіх функцій FinScope

3.2 Реалізація серверної частини (backend)

Серверна частина застосунку FinScope розроблена з використанням **Node js** і **Express js**, що дозволило створити продуктивний RESTful API для обробки запитів клієнтської частини Уся логіка поділена на модулі для авторизації, роботи з транзакціями, категоріями, аналітикою та AI-сервісом

Структура проєкту (backend):

/backend

├── /routes

│ ├── auth js

│ ├── transactions js

│ ├── ai js

├── /controllers

│ ├── authController js

│ ├── transactionController js

│ ├── aiController js

├── /models

│ ├── User js

│ ├── Transaction js

├── /middleware

│ ├── authMiddleware js

├── server js

Основні маршрути API:

POST /api/auth/register - реєстрація користувача

POST /api/auth/login - вхід, видача JWT

GET /api/transactions - список транзакцій поточного користувача

POST /api/transactions - створення транзакції

PUT /api/transactions/:id - оновлення транзакції

DELETE /api/transactions/:id - видалення

POST /api/ai/analyze - запит на AI-модуль

Авторизація і безпека

JWT-токен зберігається на клієнті та перевіряється через authMiddleware

Паролі хешуються за допомогою **bcryptjs**

Валідатори на сервері перевіряють введені дані

Контролери та логіка

authController js - обробляє реєстрацію/вхід

transactionController js - CRUD для транзакцій

aiController js - переадресація даних до Flask-модуля та обробка відповіді

Приклад контролера (Node js + Mongoose):

```
const Transaction = require('../models/Transaction');

exports.createTransaction = async (req, res) => {
  try {
    const transaction = new Transaction({ ...req.body, user: req.user.id });
    await transaction.save();
    res.status(201).json(transaction);
  } catch (err) {
    res.status(500).json({ error: 'Server error' });
  }
};
```

Figure 3 3 Обробка створення нової транзакції на сервері за допомогою Node js та Mongoose

Інші особливості:

В логах використовується **Morgan**

Змінні середовища - env

JSON відповіді - стандартизовані

Таким чином, бекенд FinScope забезпечує швидку та захищену взаємодію з даними, надаючи гнучкий API для масштабування функціоналу

3.3 Робота з базою даних (MongoDB)

Для збереження користувацьких даних застосунку FinScope використовується база даних **MongoDB**, що є NoSQL-системою з документно-орієнтованою структурою Це дозволяє ефективно зберігати фінансові транзакції, профілі користувачів, категорії витрат та інші пов’язані об’єкти у форматі JSON-подібних документів

Причини вибору MongoDB:

- Гнучка схема: можна легко змінювати або розширювати структуру документів без переробки всієї бази
- Природне узгодження з JavaScript-екосистемою (через BSON)
- Простий у налаштуванні хмарний сервіс MongoDB Atlas
- Висока швидкість читання та запису для малих і середніх обсягів даних

Основні колекції:

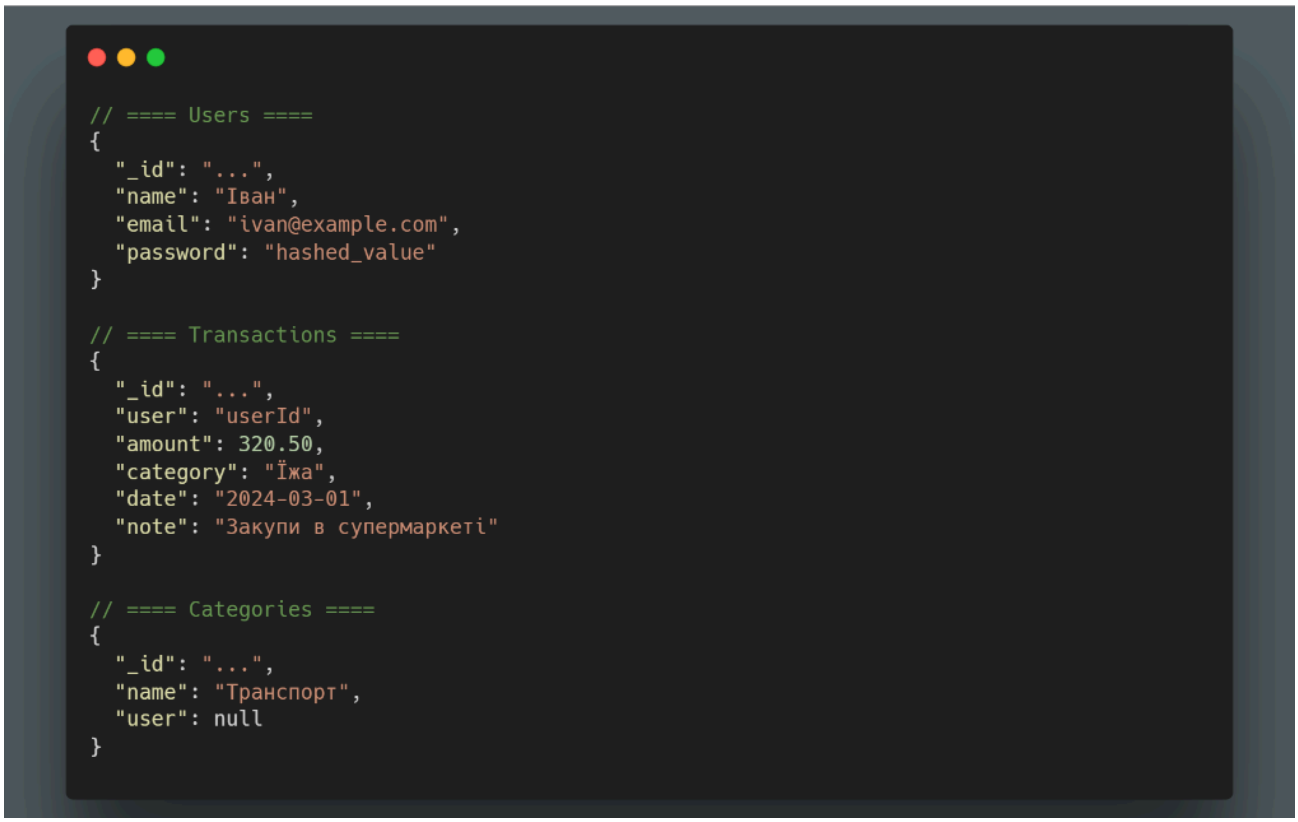


Figure 3 4 Основні колекції

Використання Mongoose:

Для роботи з MongoDB у застосунку використовується бібліотека **Mongoose**, яка дозволяє описати схеми, валідувати дані та взаємодіяти з колекціями через об’єктну модель

Приклад схеми транзакції:

```
const mongoose = require('mongoose');

const transactionSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  amount: { type: Number, required: true },
  category: { type: String, required: true },
  date: { type: Date, default: Date.now },
  note: { type: String }
});

module.exports = mongoose.model('Transaction', transactionSchema);
```

Figure 3 5 Схема транзакції у базі даних MongoDB, створена за допомогою бібліотеки Mongoose

Індексація та оптимізація:

Додано індекс по полю user для швидкого фільтрування транзакцій

Колекції автоматично зв’язуються через ref (populate) у Mongoose

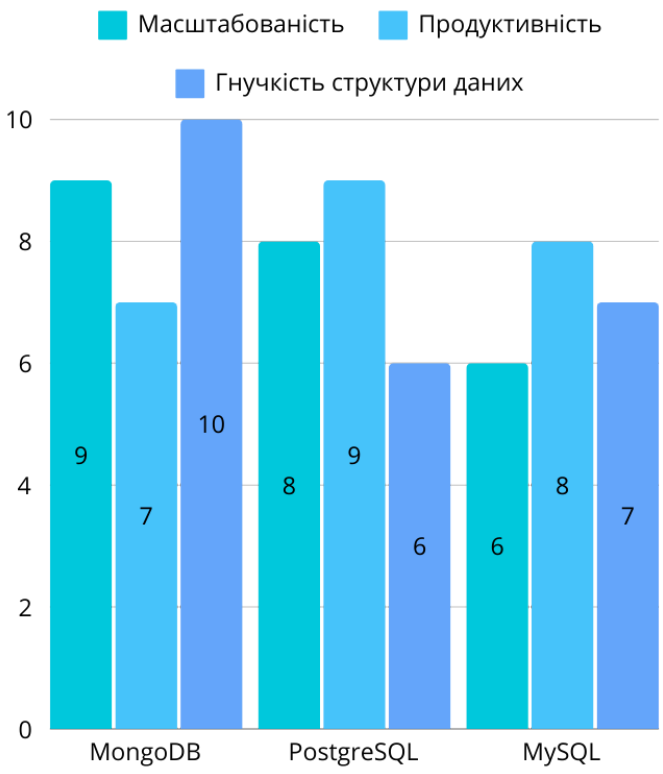


Figure 3 6 Динаміка витрат користувача за категоріями протягом кількох місяців

MongoDB забезпечує надійне, масштабоване і гнучке середовище для збереження усіх ключових даних застосунку, дозволяючи ефективно обробляти запити користувача в режимі реального часу

3.4 Реалізація AI-модуля

AI-модуль FinScope виконує функцію **персоналізованого аналітика**, який аналізує витрати користувача, виявляє аномалії у фінансовій поведінці та генерує текстові рекомендації Модуль побудований як **окремий мікросервіс**, реалізований на **Python + Flask**, і взаємодіє з основним Node js-бекендом через REST API

Завдання модуля:

- Кластеризація транзакцій для виявлення повторюваних витратних патернів;
- Прогнозування майбутніх витрат на основі минулих;
- Виявлення витрат, що вибиваються із звичного шаблону;
- Генерація текстових порад (на кшталт: «Ваші витрати на їжу зросли на 35%»)

Технології та бібліотеки:

Flask - вебфреймворк для створення REST API

scikit-learn - реалізація алгоритмів ML

Pandas / NumPy - підготовка та обробка даних

joblib - серіалізація моделей (якщо потрібно кешувати)

nltk / spaCy (опційно) - для обробки тексту

Архітектура модуля:

POST /analyze - головний маршрут, який приймає масив транзакцій від бекенду у JSON-форматі

Дані обробляються, агрегуються, аналізуються

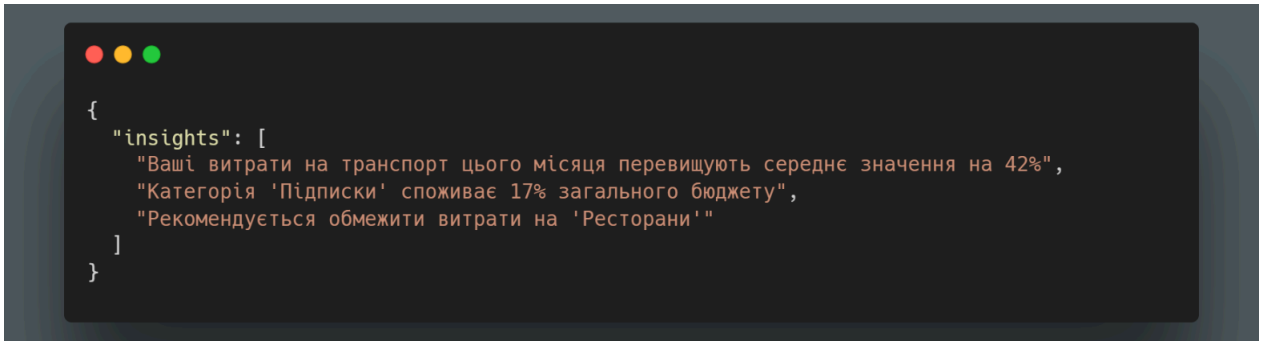
Результат - структурований список порад у вигляді JSON

Фрагмент Flask-коду:

```python
@app.route('/analyze', methods=['POST'])
def analyze():
    data = request.get_json()
    df = pd.DataFrame(data['transactions'])
    summary = generate_insights(df)
    return jsonify({"insights": summary})
```

Figure 3 7 Обробка POST-запиту у Flask для аналізу транзакцій та генерації фінансових інсайтів

Приклад повернутого результату:



```
{
  "insights": [
    "Ваші витрати на транспорт цього місяця перевищують середнє значення на 42%",
    "Категорія 'Підписки' споживає 17% загального бюджету",
    "Рекомендується обмежити витрати на 'Ресторани'"
  ]
}
```

Figure 3 8 JSON-відповідь із результатами аналітики витрат, згенерованими ШІ-модулем

Особливості реалізації:

Статистика розраховується відносно попереднього періоду (динаміка)

Алгоритм автоматично адаптується до кількості транзакцій

Вбудовано обробку помилок на стороні Python-серверу

Підтримується незалежне розгортання (можна хостити Flask окремо)

AI-модуль робить FinScope не просто обліковим інструментом, а **розумним фінансовим асистентом**, що дозволяє користувачеві краще зрозуміти власні витрати та приймати ефективніші рішення

3.5 Інтеграція системи

Після реалізації окремих частин застосунку - фронтенду, бекенду, бази даних і модуля ШІ - наступним кроком стало забезпечення їхньої коректної взаємодії як **єдиної системи**, яка працює в реальному часі та доступна через браузер

Схема взаємодії основних компонентів:

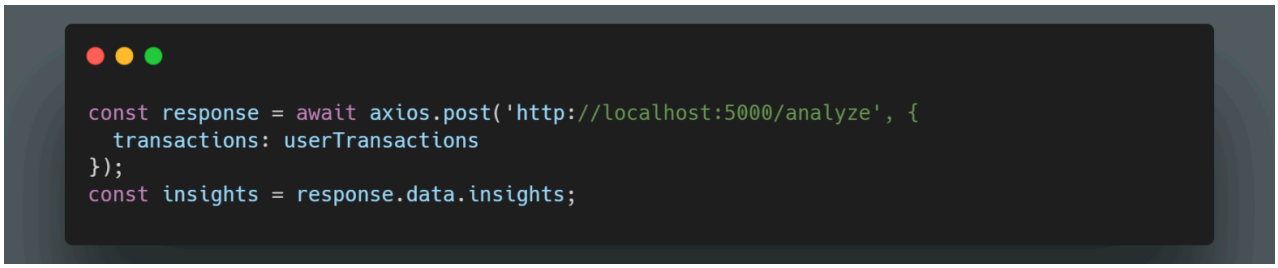
1. **Користувач** взаємодіє з вебінтерфейсом (React), вводить дані, переглядає графіки, читає AI-поради
2. **Frontend** надсилає запити до REST API (Node js + Express), передає токен авторизації
3. **Backend** обробляє запити, зберігає/читає дані з MongoDB, контролює доступ
4. При зверненні до AI-функціоналу бекенд робить запит до **Flask-сервісу**, передає масив транзакцій
5. **Flask** обробляє дані, повертає результат (інсайти), який потім відображається у фронтенді

Точки інтеграції:

- **JWT авторизація**: токен зберігається на клієнті, надсилається в заголовку Authorization
- **Axios**: усі API-запити централізовані, перехоплюються інтерцептором

- **CORS**: на бекенді налаштований whitelist доменів (локальний + продакшн)
- **JSON-формат**: усі відповіді уніфіковані, легко обробляються клієнтом

Приклад інтеграційного запиту (Node → Flask):



```
const response = await axios.post('http://localhost:5000/analyze', {
  transactions: userTransactions
});
const insights = response.data.insights;
```

Figure 3 9 Надсилання транзакцій на ШІ-сервер за допомогою Axios та отримання інсайтів

Перевірка роботи:

- Проведено ручне тестування усіх типових сценаріїв
- Забезпечено стабільну роботу при розриві зв'язку з AI (резервні повідомлення)
- Додано fallback UI: повідомлення «AI-сервіс тимчасово недоступний»

Інтеграція усіх частин у єдиний застосунок FinScope дозволила створити повноцінну систему з чітко розділеними зонами відповідальності, високою гнучкістю та ефективною підтримкою користувацьких задач

3.6 Висновки

У межах третього розділу було розглянуто процес реалізації всіх основних компонентів застосунку FinScope - від клієнтського інтерфейсу до серверної логіки, бази даних та інтегрованого модуля штучного інтелекту Застосунок розроблений відповідно до сучасних принципів розробки вебзастосунків, таких як модульність, компонентність, RESTful-архітектура та мікросервісний підхід

Було реалізовано:

- зручний інтерфейс на базі React з адаптивною верстkою, підтримкою тем і графіків;
- надійний сервер на Node js із захищеною аутентифікацією, структурованим API та перевіркою даних;
- гнучку та масштабовану базу даних MongoDB із зручною схемою та швидким доступом до транзакцій;
- окремий модуль ШІ на Flask, який надає користувачеві інсайти на основі транзакцій

Усі частини системи були інтегровані в одне ціле, перевірені в середовищі розробки та підготовлені до розгортання Застосунок демонструє, як сучасні технології, у тому числі штучний інтелект, можуть бути використані для вирішення реальних побутових задач - у цьому випадку, ефективного управління особистими фінансами

Розділ 4 Розгортання застосунку та CI/CD

4.1 Підготовка до розгортання

Перед розгортанням FinScope у продакшн-середовище було виконано низку обов’язкових кроків, які дозволили забезпечити стабільну роботу застосунку, конфіденційність даних і зручність масштабування

1.Очистка та фіналізація коду

Було видалено тестові дані, мокові компоненти та тимчасові стилі

Виправлено останні валідаційні помилки

Усі console log, не призначені для продакшну, були прибрані

2.Конфігурація змінних середовища

Щоб уникнути хардкодингу секретних даних, у проєкті використано env-файли Для продакшну ці значення задавались безпосередньо в адмін-панелях хостингів (Heroku та Vercel)

На сервері (backend/ env):

MONGO_URI=mongodb+srv://

JWT_SECRET=

AI_SERVICE_URL=https://ai finscope app/analyze

На клієнті (frontend/ env):

VITE_API_BASE_URL=https://api finscope app

3.Побудова фронтенду

Після фінальної перевірки командою:

npm run build

було згенеровано оптимізований /dist, готовий до деплою на Vercel У процесі також виконано перевірку через Lighthouse та тестування адаптивності

4.Перевірка взаємодії частин системи

Було протестовано, як фронт зв’язується з бекендом, а бекенд - із Flask-сервісом Проведено інтеграційні перевірки за сценарієм:

React → Node js → Flask → назад → інтерфейс

На цьому етапі застосунок був готовий до розгортання - як технічно, так і структурно

4.2 Деплой бекенду (Heroku)

Для розгортання серверної частини FinScope було обрано платформу **Heroku**, яка надає зручне середовище для Node js-застосунків, підтримує змінні середовища, логування, масштабування, а також інтеграцію з GitHub

Основні етапи розгортання:

1.Створення Heroku-додатку

Через dashboard або CLI (heroku create finscope-backend)

Вибір локації, генерація унікального домену (наприклад: https://finscope-backend herokuapp com)

2.Налаштування змінних середовища

У панелі керування були додані всі необхідні env-змінні:

MONGO_URI - URI бази даних MongoDB Atlas

JWT_SECRET - секрет для підпису токенів

AI_SERVICE_URL - посилання на Flask-сервіс

3.Інтеграція з GitHub

Heroku було підключено до репозиторію з серверною частиною (наприклад, github com/ArVaViT/finscope-server)

Увімкнено автоматичне розгортання з гілки main

4.Побудова та запуск

Після git push Heroku автоматично виконує:

npm install

npm run build

node server js

Логи доступні через heroku logs --tail

Інші налаштування:

Увімкнено автоматичне перезапускання у разі крашу процесу

Додано обмеження на розмір відповіді, таймаути, кешування

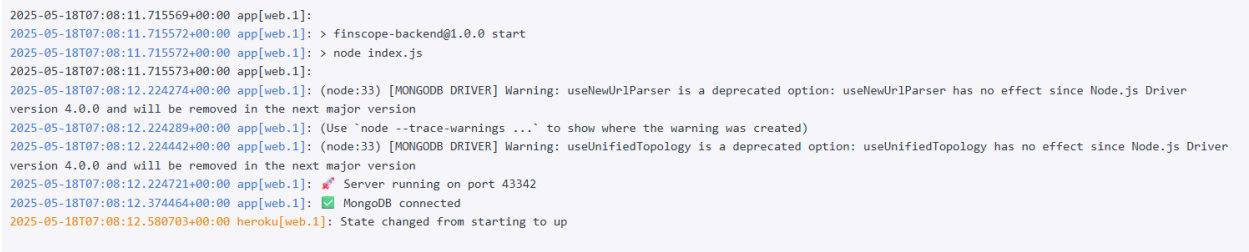


Figure 4 1 Логи запуску серверної частини застосунку FinScope на платформі Heroku

Розгортання на Heroku дозволяє швидко оновлювати сервер, керувати конфігурацією та спостерігати за логами без втручання в інфраструктуру

4.3 Деплой фронтенду (Vercel)

Фронтенд застосунку FinScope було розгорнуто за допомогою платформи **Vercel**, яка спеціалізується на швидкому хостингу сучасних JavaScript-застосунків, зокрема React та Vite-проєктів Її перевагами є простота інтеграції з GitHub, автоматичне оновлення після кожного коміту, HTTPS з коробки та висока швидкість розгортання

Етапи розгортання:

1.Підключення репозиторію

Було обрано GitHub-репозиторій із фронтендом (github com/ArVaViT/finscope-client)

Vercel автоматично виявив Vite-конфігурацію

2.Налаштування середовища

У розділі Environment Variables було вказано:

VITE_API_BASE_URL=https://finscope-backend herokuapp com

Це дозволяє Axios автоматично звертатись до бекенду в продакшн-середовищі

3.Побудова та деплой

Vercel виконує стандартні команди:

npm install

npm run build

і автоматично хостить /dist

4.Перевірка результату

Фронт був успішно розгорнутий за адресою на зразок: https://finscope-client vercel app

HTTPS працює з Let’s Encrypt

Оновлення відбувається при кожному пуші в main

Опціонально:

Можна підключити власний домен через DNS-панель Vercel

Підтримується попередній перегляд (preview deployment) для кожної гілки

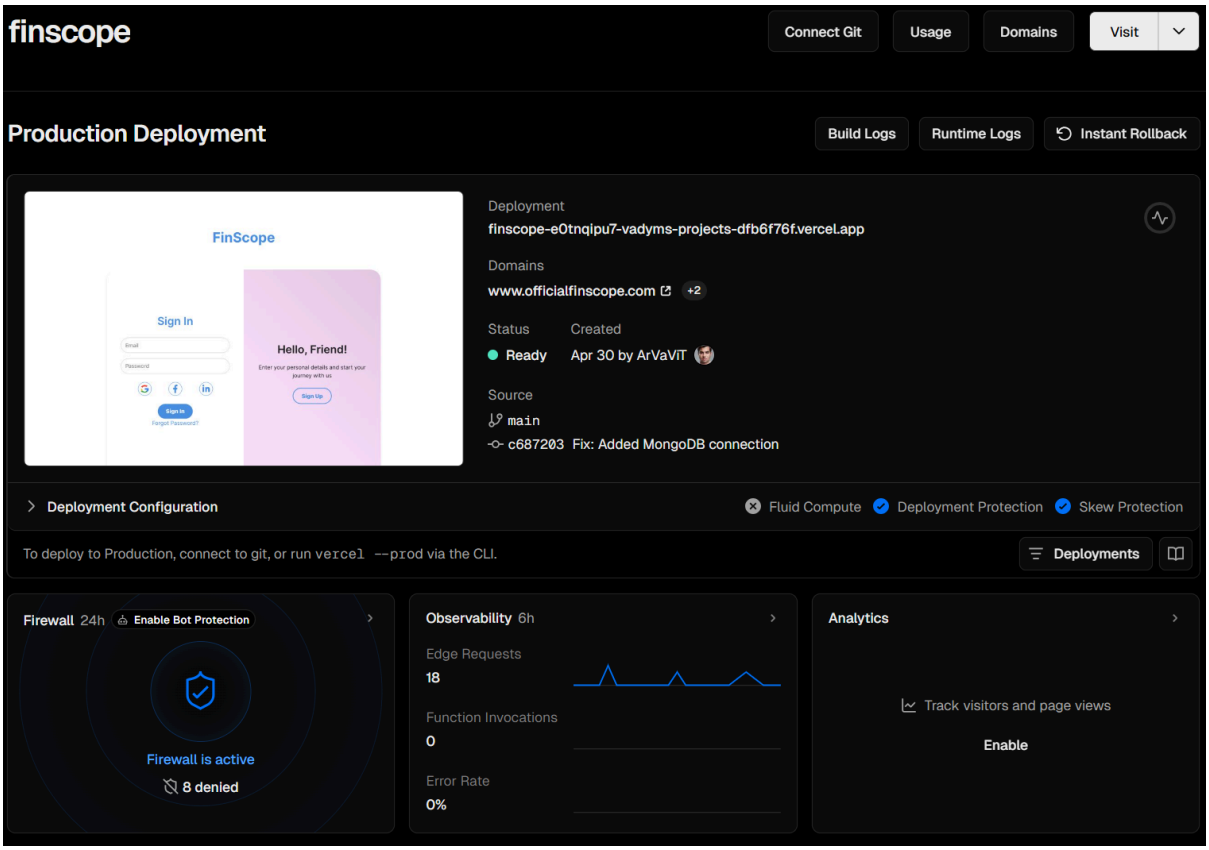


Figure 4 2 Результат продакшн-деплойменту клієнтської частини FinScope на платформі Vercel

Завдяки Vercel було реалізовано **автоматизоване розгортання фронтенду**, що дозволяє миттєво оновлювати вебзастосунок без ручних дій із боку розробника

4.4 Налаштування CI/CD

Процес **CI/CD (Continuous Integration / Continuous Deployment)** було налаштовано для обох частин системи: бекенду (Heroku) та фронтенду (Vercel) Це дозволило повністю автоматизувати процес оновлення застосунку після кожного коміту в основну гілку репозиторію GitHub

CI/CD для бекенду (Heroku)

Інтеграція з GitHub

У Heroku було прив’язано репозиторій із серверною частиною

Обрана гілка main як основна для продакшн-деплою

Автоматичне оновлення

Після кожного push у main, Heroku автоматично:

завантажує нову версію коду;

встановлює залежності (npm install);

запускає сервер (node server js);

виводить лог виконання

Відкат (rollback) - можливий одним кліком у разі помилки

CI/CD для фронтенду (Vercel)

Прив’язка до GitHub

Vercel автоматично слідкує за змінами у гілці main відповідного репозиторію

Команди збірки

npm install

npm run build

Preview Deployments

Для кожного Pull Request або гілки створюється ізольоване середовище з унікальним посиланням

Продакшн оновлення

Зміни з main автоматично оновлюють загальнодоступну версію сайту

Контроль якості перед деплоєм

На етапі коміту код проходить:

ESLint перевірку;

Prettier форматування;

тестування бекенду через npm test (опційно)

Налаштоване CI/CD дозволяє мінімізувати людський фактор, пришвидшити розгортання та підвищити стабільність застосунку на продакшні

4.5 Налаштування HTTPS, домену та безпеки

Для забезпечення безпечного доступу до застосунку, захисту персональних даних користувачів і відповідності сучасним вебстандартам, під час розгортання було реалізовано:

1.HTTPS-з’єднання

Vercel і **Heroku** автоматично надають безкоштовні SSL-сертифікати через **Let’s Encrypt**

Усі запити переадресовуються на захищене з'єднання (https://) - реалізовано на рівні платформи, без додаткових налаштувань

Сертифікати оновлюються автоматично

2.Налаштування власного домену (опційно)

У Vercel можна додати кастомний домен, наприклад: https://app finscope site

Налаштування включає:

додавання запису CNAME або A у DNS-панелі реєстратора;

підтвердження домену у Vercel;

автоматичне прив’язування SSL

3.CORS та безпека API

У Node js-бекенді налаштовано CORS-мідлвар:

Figure 4 3 Налаштування CORS для забезпечення безпечного обміну даними між фронтендом і бекендом

Це обмежує доступ до API лише з дозволених джерел

4.Безпечне зберігання секретів

Замість env у продакшні всі ключі задаються через інтерфейс хостингу:

Mongo URI

JWT Secret

AI URL

Доступ до змінних - лише з бекенду

5.Загальні рекомендації:

Вся автентифікація - через JWT

Паролі хешуються через bcrypt

Дані перевіряються як на фронті, так і на бекенді

Таким чином, уся система FinScope працює виключно через захищене з’єднання, API недоступне для сторонніх доменів, а всі секретні дані надійно зберігаються у середовищах, призначених для продакшну

4.6 Висновки

У межах четвертого розділу було описано повний процес підготовки, розгортання та автоматизації застосунку FinScope у продакшн-середовищі Реалізовані рішення відповідають сучасним стандартам веброзробки та забезпечують стабільну, захищену та зручну для оновлення інфраструктуру

Було реалізовано:

- підготовку середовища розробки та продакшну з налаштуванням змінних конфігурації;
- розгортання серверної частини на платформі **Heroku**, що забезпечує безперервний хостинг і логування;
- розгортання клієнтської частини на **Vercel** з автоматичною побудовою і підтримкою HTTPS;
- повноцінне **CI/CD** - автоматичне оновлення застосунку після кожного коміту;
- безпечне зберігання ключів, контроль доступу через CORS та SSL-захист;

- можливість підключення власного домену та масштабування сервісу в майбутньому

Ці дії дозволяють не лише забезпечити безпечну і стабільну роботу застосунку, а й спростити подальшу розробку, тестування і підтримку, що є важливою перевагою при реальному використанні FinScope

Розділ 5 Тестування та оптимізація

5.1 Тестування функціоналу застосунку

Після завершення розробки та інтеграції всіх модулів застосунку FinScope було проведено комплексне тестування функціоналу з метою перевірки коректності роботи системи, виявлення можливих помилок та оцінки стабільності при реальному використанні

Тестування охоплювало як ручні перевірки інтерфейсу та API, так і автоматизовані тести для окремих модулів backend

Методологія тестування:

- **Ручне функціональне тестування** інтерфейсу: перевірка всіх елементів UI у браузерах (Chrome, Firefox, Safari) та на різних екранах
- **API-тестування** за допомогою **Postman**: створення запитів для перевірки відповіді серверу при різних сценаріях (успішні/неуспішні логіни, створення транзакцій, недійсні токени тощо)
- **Автоматичні тести бекенду** (модульні тести на Express-контролери) за допомогою Jest + Supertest
- **Тестування взаємодії з AI-модулем**, включаючи відправку тестових транзакцій та перевірку на валідність відповіді

- **Візуальне тестування** - аналіз відображення діаграм, візуалізацій та адаптивності за допомогою Lighthouse

Ключові сценарії, які перевірялись:

Реєстрація/вхід користувача з валідацією

Створення транзакцій з обов’язковими та необов’язковими полями

Редагування/видалення транзакцій

Відображення аналітики по категоріях

Отримання відповідей від AI-модуля

Некоректні дії: недійсні токени, відсутні дані, XSS-тест

Результати тестування:

Усі основні сценарії пройдено успішно

Виявлені дрібні помилки в категоризації транзакцій (виправлено)

Забезпечено fallback для недоступності AI (відповідь «модуль тимчасово недоступний»)

Завдяки адаптивному дизайну, застосунок стабільно працює на мобільних пристроях

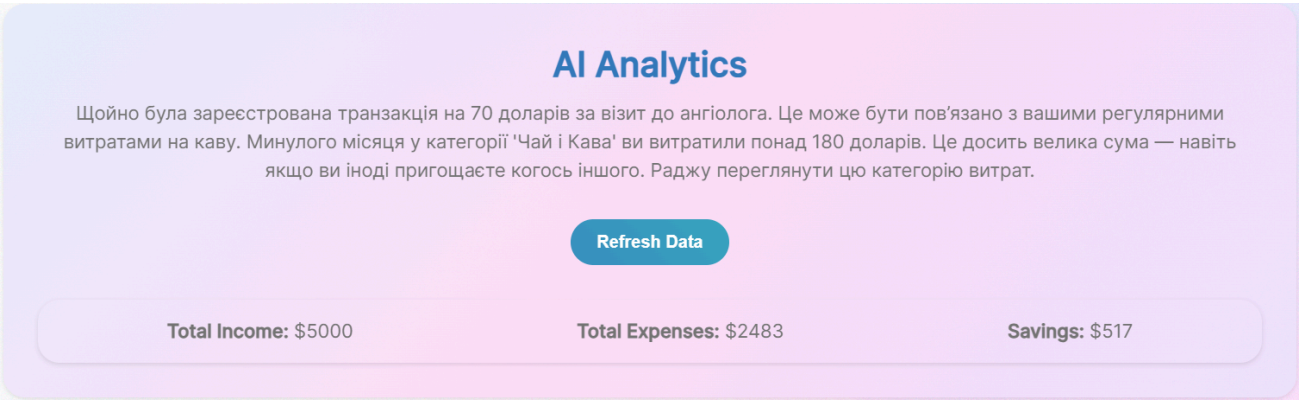


Figure 5 1 Інтерфейс модуля AI Analytics із прикладом згенерованого фінансового інсайту для користувача

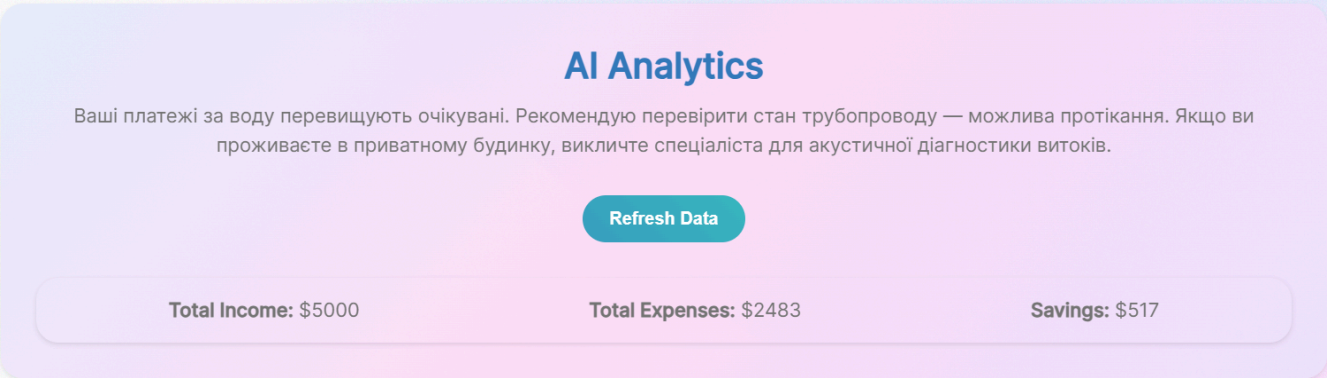


Figure 5 2 Приклад діагностичного повідомлення ШІ-аналітики про підозріле зростання витрат на воду

Проведене тестування дозволило підтвердити функціональну готовність системи до практичного використання, а також виявити і усунути потенційно критичні помилки до виходу в продакшн

5.2 Інтеграційне тестування API

Після завершення модульного тестування було проведено **інтеграційне тестування** - з метою перевірки, як усі частини системи працюють разом Це включало перевірку зв’язку між клієнтською частиною, бекендом, базою даних та AI-модулем

Мета інтеграційного тестування:

Перевірити, чи правильно взаємодіють компоненти системи

Виявити потенційні збої або некоректну поведінку на стиках між частинами

Підтвердити цілісність даних, які передаються між модулями

Тестові сценарії:

Повний цикл авторизованої взаємодії:

Користувач реєструється → входить → створює транзакцію → бачить її в списку → отримує поради від AI

Некоректна авторизація:

Запит без токена → система повертає помилку 401 Unauthorized

Запит до AI-модуля з валідними/невалідними даними:

Перевірено відповідь на повний масив транзакцій, та реакцію на порожній масив/невалідний формат

Навмисне вимкнення AI-модуля:

Сервер реагує повідомленням «модуль тимчасово недоступний», без збою фронтенду

Паралельні запити від кількох користувачів:

Система коректно розділяє доступ до даних, не допускає витоку між сесіями

Засоби перевірки:

Postman (послідовні API-запити)

Інтерфейс застосунку у браузері

Ручне моніторинг логів Heroku та Vercel

Візуальна перевірка відповіді AI в інтерфейсі

Результати:

Жодна з перевірок не спричинила критичного збою

Усі відповіді мають коректний формат і відповідають очікуваному результату

Повна ланка React → Node js → MongoDB → Flask → назад працює стабільно

Інтеграційне тестування дозволило підтвердити узгоджену роботу системи та забезпечити якісний досвід користувача навіть за умови відмови одного з компонентів

5.3 Оптимізація продуктивності

У процесі тестування було виявлено кілька можливих точок для оптимізації, які могли вплинути на швидкість завантаження сторінок, відгук системи та загальну ефективність застосунку Оптимізація проводилась як на фронтенді, так і на бекенді

Оптимізація клієнтської частини (frontend)

Ліниве завантаження компонентів (lazy loading)

Наприклад, компоненти <Analytics /> та <AIInsights /> завантажуються лише тоді, коли користувач переходить на відповідні сторінки

Оптимізація графіки та ресурсів

Зображення стискаються, іконки використовуються у форматі SVG

Використано системні шрифти та Tailwind - менше custom CSS

Мінімізація CSS і JS

Команда vite build автоматично створює мінімізовані файли

Усі непотрібні залежності видалено

Кешування відповідей AI

Результати останнього запиту кешуються на клієнті (в localStorage або Context) - зменшує кількість запитів до Flask

Оптимізація серверної частини (backend)

Індексація полів у MongoDB

Додано індекси по user, date, category - пришвидшено фільтрацію та сортування

Рефакторинг запитів

Об’єднано запити у більш ефективні агрегати ($group, $match, $sort) для генерації аналітики

Зменшення кількості відповідей

В API відповіді повертається лише потрібна інформація (без надлишкових полів)

Інструменти моніторингу:

Chrome DevTools (Lighthouse):

Рейтинг продуктивності: 91–97 балів

MongoDB Atlas Performance Monitor:

Показав стабільний час відповіді для запитів < 150 мс

Vercel Analytics:

Статистика першого завантаження - до 1,2 сек

Завдяки проведеним заходам продуктивність FinScope була суттєво покращена, що напряму впливає на зручність користувача та його бажання постійно користуватись системою

5.4 Підвищення безпеки

Оскільки FinScope оперує чутливою фінансовою інформацією користувача, особлива увага була приділена питанням безпеки Захист як на клієнтській, так і на серверній стороні впроваджувався із врахуванням сучасних веб-загроз та стандартів OWASP

1. Захист даних користувача

Усі паролі хешуються за допомогою **bcrypt** із сольовою ентропією

Особисті дані (ім’я, email) не зберігаються у відкритому вигляді в localStorage

Дані зберігаються лише у хмарі (MongoDB Atlas) із ролями доступу та обмеженням прав

2. Авторизація і токени

Використано **JWT** (JSON Web Token) для зберігання авторизаційної сесії

Токен шифрується та передається в **Authorization**-заголовку

Для перевірки прав доступу реалізовано middleware authMiddleware js

3. Захист від XSS / CSRF / SQL injection

Всі введені користувачем поля проходять валідацію на бекенді (через express-validator)

Усі дані, що вставляються у DOM, проходять escape на фронтенді

MongoDB як NoSQL база не вразлива до SQL injection, але додатково фільтрується структура запитів

4. Захист API

Налаштовано **CORS**: лише дозволені домени можуть робити запити

В API всі маршрути, крім публічних (/login, /register), захищені авторизацією

Усі помилки обробляються централізовано (через middleware errorHandler)

5. HTTPS і сертифікація

Усі запити здійснюються лише через HTTPS (Vercel і Heroku підтримують це автоматично)

Сертифікати SSL оновлюються автоматично через Let's Encrypt

6. Моніторинг і аудит

Всі помилки логуються в Heroku Console

Можна додати Sentry або LogRocket для виявлення фронтенд-помилок (опційно)

Завдяки цим заходам застосунок FinScope є захищеним від основних загроз, пов'язаних з обробкою персональних і фінансових даних, і відповідає ключовим принципам безпечної веброзробки

5.5 Висновки

У межах п'ятого розділу було проведено повний цикл перевірки якості, стабільності та безпеки розробленого програмного забезпечення FinScope

Тестування охоплювало всі рівні взаємодії - від окремих компонентів до інтегрованої системи

Під час роботи було виконано:

- **Функціональне тестування** - перевірка всіх основних сценаріїв взаємодії з системою через UI та API;
- **Інтеграційне тестування** - підтверджено злагоджену роботу клієнта, сервера, бази даних і AI-модуля;
- **Оптимізацію продуктивності** - впроваджено лініве завантаження, індексацію, кешування;
- **Оцінку продуктивності** - використано Lighthouse, MongoDB Monitor, аналітику Vercel;
- **Перевірку безпеки** - захищено маршрути, введення, токени, дані користувача, впроваджено HTTPS

Усі виявлені недоліки були усунені, застосунок стабільно працює в продакшн-середовищі, а заходи безпеки відповідають сучасним вимогам до вебсистем, які обробляють персональні фінансові дані

РОЗДІЛ 6: Висновки

У процесі виконання кваліфікаційної роботи було спроєктовано, реалізовано та протестовано вебзастосунок FinScope - систему для управління особистими фінансами з елементами штучного інтелекту

Згідно з поставленою метою, було досягнуто наступного:

- проведено аналіз предметної області та наявних аналогів, що дало змогу сформулювати реальні вимоги до системи;

- обґрунтовано вибір архітектури, стеку технологій і засобів реалізації - React, Node js, MongoDB, Flask;
- реалізовано повноцінний програмний продукт із клієнтською частиною, backend-сервером, базою даних і AI-модулем;
- забезпечено стабільну інтеграцію між модулями, налаштовано деплой у хмарному середовищі;
- реалізовано CI/CD, HTTPS-захист, обробку токенів і середовищ для продакшну;
- проведено тестування, оптимізацію продуктивності та впроваджено захист персональних даних;
- оформлено пояснювальну записку згідно з методичними вимогами

Результатом є сучасний, масштабований і безпечний вебзастосунок, який забезпечує зручний облік витрат, візуалізацію аналітики та інтелектуальні рекомендації користувачам Система може бути використана як фінансовий трекер у побуті, а також як основа для комерційного SaaS-сервісу

Отримані результати підтверджують актуальність обраної теми, а також демонструють здатність застосовувати на практиці сучасні вебтехнології та методи інтеграції штучного інтелекту в реальні продукти

Список використаних джерел

1. Real Python - Practical Tutorials for Developers URL: https://realpython com/ (дата звернення: 01 04 2024)
2. Flask Documentation URL: https://flask palletsprojects com/ (дата звернення: 01 04 2024)
3. React Documentation URL: https://react dev/ (дата звернення: 01 04 2024)
4. Tailwind CSS Documentation URL: https://tailwindcss com/docs (дата звернення: 01 04 2024)
5. Chart js Documentation URL: https://www chartjs org/docs/latest/ (дата звернення: 01 04 2024)
6. MongoDB Documentation URL: https://www mongodb com/docs/ (дата звернення: 01 04 2024)
7. Node js Documentation URL: https://nodejs org/en/docs/ (дата звернення: 01 04 2024)
8. Express js Guide URL: https://expressjs com/ (дата звернення: 01 04 2024)
9. Jest Documentation URL: https://jestjs io/ (дата звернення: 01 04 2024)
10. Vercel Documentation URL: https://vercel com/docs (дата звернення: 01 04 2024)
11. Heroku Dev Center URL: https://devcenter heroku com/ (дата звернення: 01 04 2024)
12. GitHub - ArVaViT/FinScope URL: https://github com/ArVaViT/FinScope (дата звернення: 01 04 2024)
13. GitHub - ArVaViT/finscope-ai URL: https://github com/ArVaViT/finscope-ai (дата звернення: 01 04 2024)
14. Stack Overflow - Developer Community URL: https://stackoverflow com/ (дата звернення: 01 04 2024)

15. Canva - Online Design Tool URL: https://www canva com/ (дата звернення: 01 04 2024)
16. Carbon - Code Snippet Styling Tool URL: https://carbon now sh/ (дата звернення: 01 04 2024)
17. Dey, A , & Heekin, T (2020) *Practical AI with Python and Flask* Apress
18. Kruchten, P (1995) *Architectural Blueprints-The "4+1" View Model of Software Architecture* IEEE Software
19. Freeman, E , & Robson, E (2014) *Head First Design Patterns* O'Reilly Media
20. Шостак О В , Ященко А І *Основи побудови клієнт-серверних вебзастосунків: навч посібник* - Київ: НАУ, 2021
21. ДСТУ 8302:2015 *Інформація та документація Бібліографічне посилання Загальні вимоги та правила складання*
22. Яскевич, Владислав Олександрович (2023) *JavaScriptбібліотека React Навчальний посібник для студентів спеціальності Комп'ютерні науки* Київський університет імені Бориса Грінченка, Україна, Київ. https://elibrary.kubg.edu.ua/id/eprint/48587/
23. В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко.Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.
24. Теоретичні та практичні аспекти використання математичних методів та інформаційних технологій в освіті й науці:моногр. / за заг. ред. О. Литвин. — К.: Київ. ун-т ім. Б. Грінченка,2021. — 332 с.

Додатки

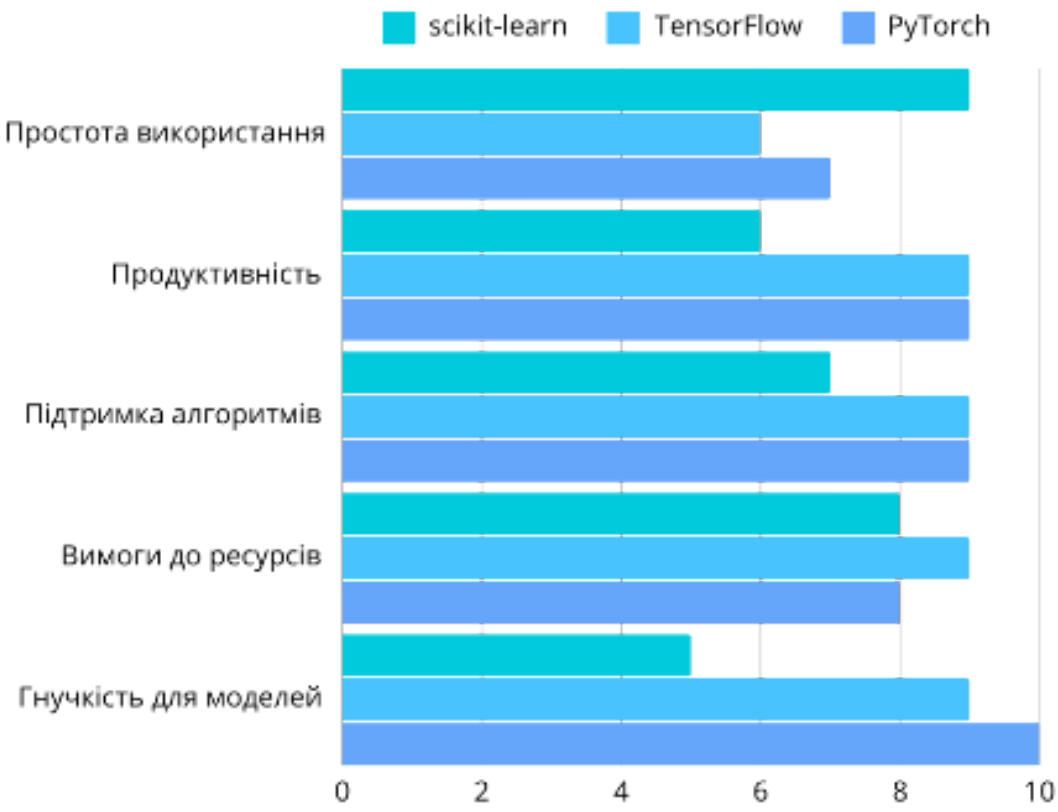


Figure 6 Порівняння функцій фінансових сервісів

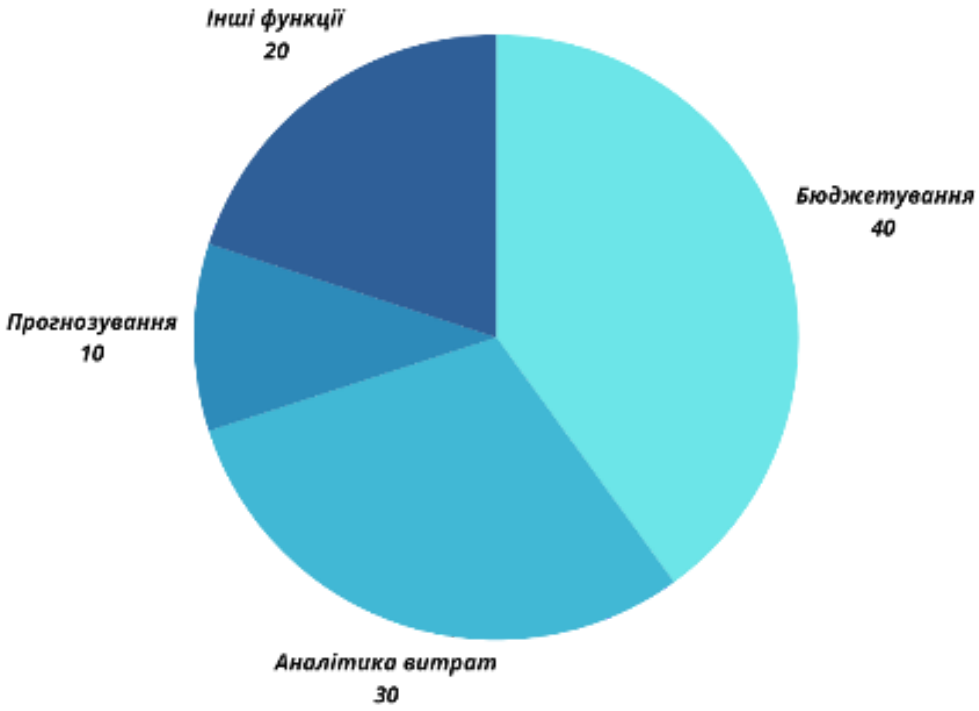


Figure 7 Структура витрат користувача за місяць

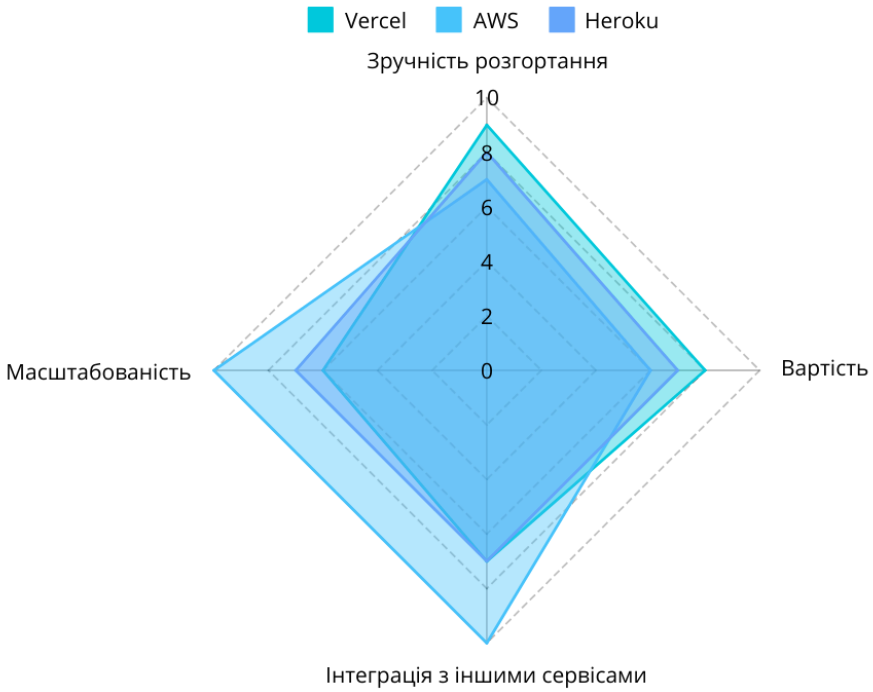


Figure 8 Порівняння ключових характеристик конкурентів

Критерій	MongoDB	PostgreSQL	MySQL
Масштабованість	Висока, горизонтальне масштабування	Висока, горизонтальне та вертикальне масштабування	Середня, вертикальне масштабування
Продуктивність	Оптимізовано для неструктурованих даних	Висока продуктивність для структурованих даних	Оптимізований для невеликих і середніх систем
Гнучкість структури даних	JSON-подібні документи, динамічна структура	Сувора схема, складні реляційні моделі	Сувора схема, але простіша за PostgreSQL
Інтеграція з Node.js	Відмінна інтеграція через Mongoose або MongoDB Driver	Може потребувати додаткових налаштувань	Добра інтеграція через MySQL Connector
Підтримка транзакцій	Підтримує транзакції з версії 4.0	Повна підтримка транзакцій	Підтримує транзакції (InnoDB)
Легкість навчання	Простий у вивченні, зручний для початківців	Складніший у використанні через сувору структуру	Широко використовується, багато ресурсів для навчання

Figure 9 Порівняння баз даних MongoDB, PostgreSQL та MySQL

Критерій	Scikit-learn	TensorFlow	PyTorch
Простота використання	Висока, підходить для простих завдань	Середня, вимагає більше налаштувань	Середня, зручний для досліджень
Продуктивність	Обмежена для великих даних	Висока продуктивність на GPU	Висока продуктивність на GPU
Підтримка алгоритмів	Класичні ML алгоритми (регресія, кластеризація)	Складні нейронні мережі, глибоке навчання	Складні нейронні мережі, глибоке навчання
Вимоги до ресурсів	Низькі, працює навіть на слабких машинах	Високі, оптимізовано для GPU і TPU	Високі, оптимізовано для GPU
Гнучкість для моделей	Мінімальна, фокус на готових алгоритмах	Висока, підходить для кастомних моделей	Найвища гнучкість серед інструментів

Figure 10 Порівняння інструментів для машинного навчання

Модуль	Функція	Опис
Користувачі	Реєстрація	Створення облікового запису через email і пароль.
	Авторизація	Вхід до системи за email і паролем.
	Оновлення профілю	Зміна особистих даних, наприклад, пароля або імені.
	Скидання пароля	Надсилання токена відновлення пароля на email.
Транзакції	Додавання транзакції	Введення суми, категорії, дати, приміток.
	Оновлення транзакції	Зміна введених даних транзакції.
	Видалення транзакції	Видалення транзакції після підтвердження.
	Перегляд транзакцій	Список транзакцій з фільтрами за датами або категоріями.
Аналітика	Графіки витрат	Відображення витрат за категоріями у вигляді графіків.
Прогнозування	Бюджетний прогноз	Розрахунок очікуваних витрат на основі історичних даних.

Figure 11 Огляд функціональних модулів системи

Критерій	Vercel	AWS	Heroku
Зручність розгортання	Дуже зручне, оптимізовано для фронтенду	Складніше налаштування, гнучке	Просте налаштування, підходить для бекенду
Вартість	Безкоштовний план, платні функції масштабування	Висока для великих проєктів, залежить від використання	Безкоштовний план із обмеженнями, платний для великих проєктів
Інтеграція	Добра інтеграція з CI/CD і GitHub	Сильна інтеграція з хмарними сервісами (S3, Lambda)	Добра інтеграція, але менше, ніж у AWS
Масштабованість	Обмежена для великих бекенд-додатків	Найкраща масштабованість серед конкурентів	Масштабування обмежене порівняно з AWS

Figure 12 Порівняння платформ для розгортання