ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Веб-додаток для управління персональними фінансами з використанням React і Tailwind CSS»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)

освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ _____
(підпис) (Ім'я, ПРІЗВИЩЕ здобувача)

Виконав: здобувач вищої освіти гр. ІСД-41
Микола УНКУ
Ім'я ПРІЗВИЩЕ

Керівник:
наукова ступінь, вчене звання
Каміла СТОРЧАК, д. т. н., професор
Ім'я ПРІЗВИЩЕ

Рецензент:
наукова ступінь, вчене звання
Ім'я ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних ситем та технологій

Ступінь вищої освіти бакалавр

Спеціальність 126 Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ

_____Каміла СТОРЧАК

«____» _____20___р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Унку Микола Вікторович

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Веб-додаток для управління персональними фінансами з використанням React і Tailwind CSS

керівник кваліфікаційної роботи Каміла СТОРЧАК, д. т. н., професор,

(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56

2. Строк подання кваліфікаційної роботи «30» травня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: офіційна технічна документація React JS, Tailwind CSS, відкриті результати досліджень впливу якості користувацього інтерфейсу на користувацький досвід, кейси програмної реалізації фінансово-технічних продуктів.

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

1. Аналіз існубчих рішень та вибір технологічного стеку для створення системи фінансового планування
2. Проєстування структури, архітектури та програмна реалізації веб-додатку
3. Тестування та оцінка ефективності розробленого рішення

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «24»лютого 2025 р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	01.03.2025 р.	
2	Дослідження існуючих систем особистого фінансового планування	15.03.2025 р.	
3	Формулювання технічного завдання, функціональних та нефункціональних вимог на розробку веб-додатку	17.03.2025 р.	
4	Вибір технологічного стеку та проєктування архітектури і структури веб-додатку	4.04.2025 р.	
5	Реалізація та тестування вебдодатку фінансового планування за допомогою ReactJS, Tailwind CSS	16.05.2025 р.	
6	Розробка демонстраційних матеріалів	26.05.2025 р.	
7	Попередній захист дипломної роботи	27.05.2025 р.	
8	Подання роботи в деканат	30.05.2025 р.	

Здобувач(ка) вищої освіти _____ Микола УНКУ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Каміла СТОРЧАК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 72 стор., 61 рис., 6 табл., 21 джерел

Мета роботи - розробка адаптивного веб-додатку для управління персональними фінансами з використанням технологій React та Tailwind CSS, що дозволяє користувачам зручно контролювати доходи, витрати та аналізувати фінансові потоки у режимі реального часу.

Об'єкто дослідження - процеси управління фінансами через веб-додаток із можливістю інтеграції банківських даних.

Предметом дослідження - реалізація адаптивного інтерфейсу, логіки обробки транзакцій та візуалізації фінансових показників з використанням React, Tailwind CSS та Monobank API.

Короткий зміст роботи - складається з трьох розділів. У першому розділі проведено аналіз існуючих веб-додатків для управління фінансами, обґрунтовано вибір технологій та розглянуто основні інструменти для розробки адаптивного інтерфейсу. У другому розділі розглянуто структуру веб-додатку, компоненти Dashboard.jsx, Transactions.jsx, Analytics.jsx та ChartByCategory.jsx, їхню інтеграцію з Monobank API та логіку обробки даних. У третьому розділі виконано тестування адаптивності додатку для різних пристроїв, аналіз продуктивності компонентів за допомогою DevTools та Lighthouse, а також оцінку користувацького досвіду.

КЛЮЧОВІ СЛОВА: АДАПТИВНИЙ ВЕБ-ДОДАТОК, REACT, TAILWIND CSS, MONOBANK API, ФІНАНСОВИЙ МЕНЕДЖМЕНТ, ТРАНЗАКЦІЇ, ВІЗУАЛІЗАЦІЯ ДАНИХ, АНАЛІЗ ПРОДУКТИВНОСТІ.

ABSTRACT

Text part of the master's qualification work: 72 pages, 61 pictures, 6 table, 21 sources.

The purpose of the work – the development of a responsive web application for personal finance management using React and Tailwind CSS technologies, allowing users to conveniently control income, expenses, and analyze financial flows in real-time.

The object of research – financial management processes through a web application with the possibility of bank data integration.

The subject of research – the implementation of a responsive interface, transaction processing logic, and financial data visualization using React, Tailwind CSS, and Monobank API.

The content of the work – consists of three sections. The first section analyzes existing web applications for financial management, justifies the choice of technologies, and examines the main tools for developing a responsive interface. The second section discusses the structure of the web application, the components Dashboard.jsx, Transactions.jsx, Analytics.jsx, and ChartByCategory.jsx, their integration with Monobank API, and data processing logic. The third section includes testing the application’s adaptability for different devices, analyzing the performance of components using DevTools and Lighthouse, and assessing the user experience.

KEYWORDS: RESPONSIVE WEB APPLICATION, REACT, TAILWIND CSS, MONOBANK API, FINANCIAL MANAGEMENT, TRANSACTIONS, DATA VISUALIZATION, PERFORMANCE ANALYSIS.

ЗМІСТ

ВСТУП

Актуальність теми: у сучасному цифровому середовищі ефективне управління персональними фінансами стає дедалі важливішим для користувачів, які прагнуть контролювати свої витрати, доходи та баланс у зручному, інтуїтивному форматі. Водночас актуальність створення адаптивних вебінтерфейсів полягає у необхідності забезпечення коректної роботи додатків на різних пристроях — від смартфонів до десктопів. З огляду на це, використання сучасних технологій, таких як React та Tailwind CSS, дозволяє поєднати високу продуктивність, масштабованість та гнучкість у реалізації адаптивного дизайну, що відповідає сучасним вимогам до веброзробки.

Додатково, інтеграція зовнішніх API, зокрема Monobank API, дозволяє створювати динамічні додатки з доступом до реальних фінансових даних у режимі реального часу. Це забезпечує не лише зручність для кінцевого користувача, а й розкриває потенціал для розширення функціоналу та підвищення аналітичних можливостей застосунку.

Мета дослідження: розробка адаптивного модульного вебдодатку для управління персональними фінансами з використанням React, Tailwind CSS та Monobank API, що забезпечує зручний інтерфейс для обліку доходів, витрат, балансу та фінансової аналітики у реальному часі.

Завдання дослідження включають:

- Проаналізувати особливості створення адаптивних вебінтерфейсів з використанням Tailwind CSS;
- Дослідити можливості React для побудови модульної архітектури інтерфейсу;
- Інтегрувати зовнішній API Monobank для отримання фінансових даних;
- Реалізувати функціонал обліку доходів, витрат та загального балансу;

- Побудувати графіки та таблиці для візуалізації транзакцій і аналітики;
- Перевірити адаптивність інтерфейсу для різних типів пристроїв;
- Здійснити тестування функціональності та продуктивності вебдодатку;
- Оцінити ефективність реалізованого рішення та надати рекомендації.
- *Об'єктом дослідження* є процес розробки адаптивного вебдодатку для управління фінансами, заснованого на використанні сучасних JavaScript-технологій.

Предметом дослідження є методи, інструменти та технології реалізації модульного адаптивного вебінтерфейсу для фінансового обліку з використанням React, Tailwind CSS та API Monobank.

Для досягнення мети використовуватимуться такі *методи дослідження:*

- Аналіз літературних джерел та документації;
- Компонентне проєктування інтерфейсу;
- Емпіричне тестування функціональності;
- Візуалізація даних за допомогою Recharts;
- Тестування продуктивності через DevTools та Lighthouse;
- Практична реалізація з використанням REST API.

Апробація дослідження здійснювалась у формі участі в ІІ всеукраїнській науково-технічній конференції «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу» з поданням тез на тему " Створення модульних адаптивних інтерфейсів на основі Tailwind CSS"**.**

1. АНАЛІЗ ІНСТРУМЕНТІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ АДАПТИВНИХ ВЕБ-ДОДАТКІВ

1.1 Аналіз сучасних веб-додатків для управління фінансами

У сучасному цифровому суспільстві технології виконують важливу функцію в усіх сферах життєдіяльності, зокрема у сфері управління фінансами. Веб-додатки, що призначені для фінансового менеджменту, набули широкого застосування серед приватних користувачів, родин, підприємців, а також представників малого і середнього бізнесу. Застосування таких систем дозволяє впорядковувати фінансові потоки, проводити аналіз витрат, формувати бюджети, відслідковувати доходи та досягати поставлених фінансових цілей без використання традиційних паперових інструментів та таблиць. Особливу привабливість веб-додатки демонструють завдяки своїй універсальній доступності, адаптивності та інтуїтивній зрозумілості інтерфейсу. Завдяки цьому забезпечується швидкий доступ до актуальної фінансової інформації з будь-якого пристрою, що сприяє автоматизації процесів управління.

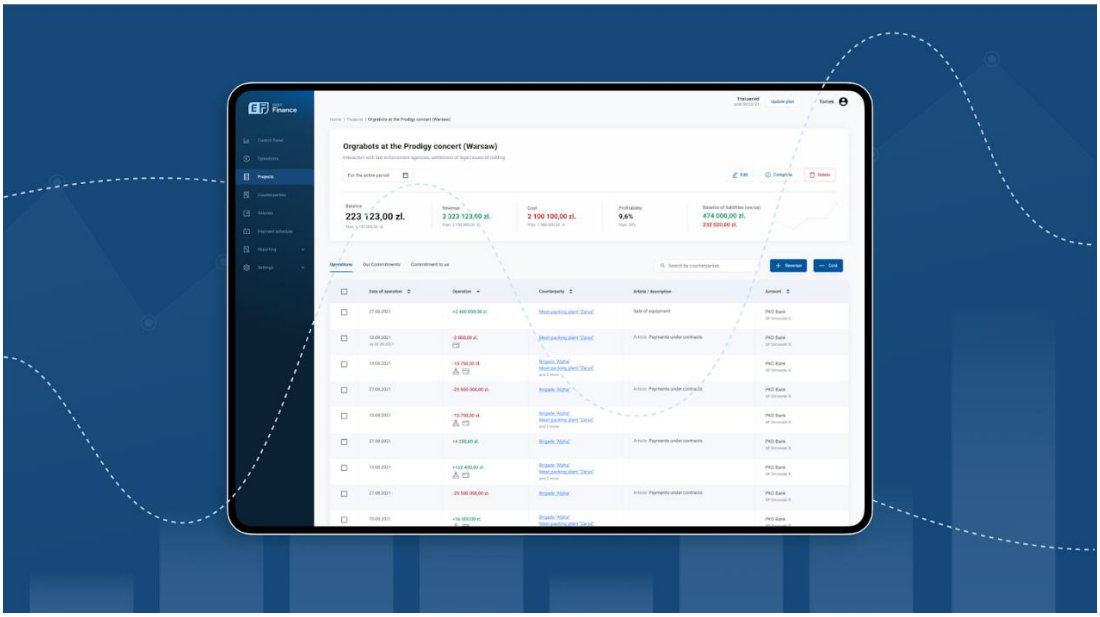


Рис. 1.1 Система управління фінансами в інтерфейсі мобільного додатку.[1]

Аналіз цього розділу зосереджено на дослідженні ключових можливостей, функціональних особливостей, сильних і слабких сторін сучасних веб-додатків, що забезпечують управління фінансами. Особливу увагу приділено огляду популярних платформ, зокрема Mint, YNAB (You Need A Budget), Wallet, які стали поширеними рішеннями у цій сфері.

Personal Finance Software	intuit mint	RECOMMENDED YNAB.
Review	Mint.com Review	YNAB Review
Rating	8.5/10	8.5/10
Cost	10/10	8/10
Customer Service	6/10	8.5/10
Ease of Use	9/10	9/10
Tools & Resources	8/10	8.5/10
Synchronization	7/10	9/10
Accessibility	8.5/10	9/10
Promotions	None	FREE First Two Months
Price	FREE	$6.99/month
Trial Period	-	34 days
Refund Policy	-	-
Budgeting	✓	✓
Bill Payment	✗	✗
Bill Management	✓	✓
Investment Tracking	✓	✓
Retirement Planning	✗	✗
Tax Reporting	✓	✗
Reconcile Transactions	✗	✓
Credit Score Monitoring	✓	✗
Zillow Tracking	✓	✗
Custom Categories	✓	✓
Manual Entries	✓	✓
Import QFX, QIF Files	✗	✓
Currency Support	US/Canada	US
Access	Website, iOS App, Apple Watch, Android App, SMS	Website, iOS App, Apple Watch, Android App
Two-Factor Authentication	✓	✗
Customer Service	Email	Email

Рис. 1.2 Порівняльна характеристика програмного забезпечення Mint та YNAB для управління фінансами. [2]

Веб-додатки, що спеціалізуються на фінансовому плануванні, пропонують широкий перелік функцій, які орієнтовані на покращення процесів управління витратами, доходами, а також прогнозування фінансових потоків на основі аналітичної інформації. Однією з найважливіших функцій виступає інтеграція з банківськими системами, яка надає змогу автоматично імпортувати дані про транзакції, зменшуючи необхідність ручного введення. Такий підхід дозволяє користувачам оперативно отримувати оновлену інформацію у зручному форматі. Як приклад, платформа Mint підтримує синхронізацію з більш ніж 20 000 банківських установ, що дає змогу забезпечити високий рівень точності й автоматизації при обробці даних.

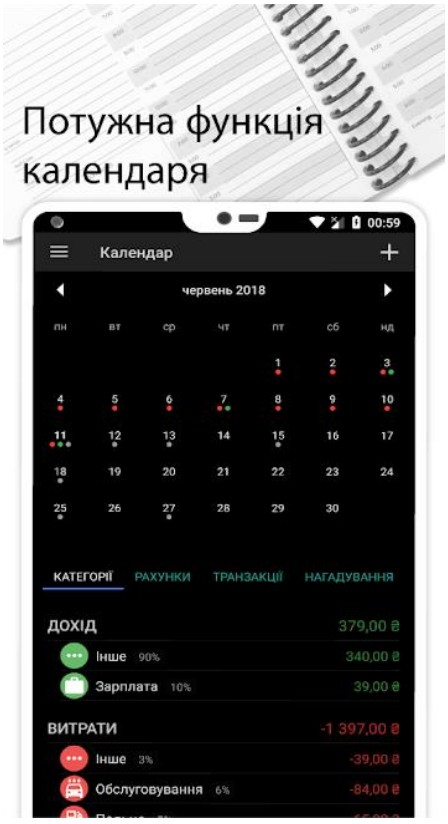


Рис 1.3 Інтерфейс календаря для відстеження доходів [3]

Одним із критичних елементів подібних систем є можливість категоризації витрат, що сприяє структуруванню фінансових потоків. Це реалізується шляхом розподілу транзакцій за визначеними категоріями, зокрема "житло", "транспорт", "продукти харчування" тощо. У додатку YNAB, наприклад, користувачі мають

змогу як використовувати попередньо задані категорії, так і формувати власні, що забезпечує індивідуальну адаптацію системи до конкретних потреб.

Функції візуалізації, які застосовуються у зазначених платформах, створюють можливість миттєвого аналізу стану фінансів. Зазвичай візуалізація реалізується у формі діаграм, графіків або динамічних таблиць, що дає змогу оперативно зчитувати інформацію про доходи, витрати та залишок коштів. Наприклад, у додатку Wallet впроваджено функціонал створення аналітичних звітів з гнучкими параметрами налаштування, що сприяє формуванню персоналізованих прогнозів та рішень щодо подальшого розподілу ресурсів.

Застосування алгоритмів машинного навчання у сучасних веб-додатках дає можливість не лише аналізувати минулі транзакції, але й прогнозувати майбутні витрати. Це стає особливо актуальним для тих, хто прагне уникати фінансових труднощів та своєчасно планувати свої витрати. У додатку PocketGuard така функціональність дозволяє користувачам отримувати персоналізовані рекомендації на основі аналізу попередніх фінансових даних.

Бюджетування виступає ще одним критично важливим компонентом для забезпечення ефективного управління фінансами. Воно охоплює як формування бюджетів для окремих категорій, так і контроль за дотриманням фінансових лімітів. Платформа YNAB орієнтується на детальне планування кожної витрати з метою збереження дисципліни та запобігання перевищенню бюджету.

Для родин або малих підприємств особливо актуальною є функція спільного доступу до бюджету. Завдяки такій функціональності декілька осіб можуть одночасно керувати загальним бюджетом, координувати витрати та забезпечувати прозорість у фінансовому плануванні. У Wallet реалізовано повноцінну підтримку спільного доступу до бюджету, що робить платформу придатною для колективного використання.

Подібні функції інтегруються і в сучасні вебдодатки, зокрема з використанням спільних акаунтів або синхронізації даних через хмарні сервіси, що дозволяє забезпечити актуальність даних і зручність доступу з різних пристроїв.

Таблиця 1.1

Основні функції сучасних веб-додатків для управління фінансами

Основна функція	Опис	Приклад реалізаці
Інтеграція з банками	Автоматичний імпорт транзакцій	Mint
Категоризація витрат	Розподіл витрат за категоріями	YNAB
Візуалізація даних	Інтерактивні графіки та звіти	Wallet
Фінансові прогнози	Прогнозування витрат на основі аналізу попередніх даних	PocketGuard
Бюджетування	Планування витрат із встановленням лімітів	YNAB
Спільний доступ	Колективне управління фінансами	Wallet

Платформа Wallet також забезпечує підтримку понад 150 валют, що дозволяє ефективно використовувати її для міжнародних транзакцій та роботи з різними фінансовими системами.

Порівняльний аналіз основних платформ, серед яких найбільш популярними залишаються Mint, YNAB і Wallet, дає змогу охарактеризувати їхні ключові особливості, функціональні можливості та рівень зручності для користувачів. Зокрема, Mint забезпечує ефективну інтеграцію з банками, автоматичний імпорт транзакцій, інтелектуальне сортування витрат за категоріями, побудову звітів і сповіщення про перевищення лімітів. Платформа надає можливість не лише відстежувати фінансову активність, а й формувати персоналізовані фінансові поради на основі звичок користувача. Такий підхід дозволяє підвищити фінансову

грамотність і сприяє кращому розумінню структури власних витрат. Інтерфейс системи є максимально спрощеним, інтуїтивно зрозумілим, що робить її особливо привабливою для нових користувачів, які лише розпочинають знайомство з фінансовими технологіями.

Wallet, зі свого боку, орієнтований на забезпечення функціональності спільного управління фінансами. Можливість одночасного доступу до бюджету декількома користувачами, підтримка великої кількості валют, а також зручний інтерфейс роблять платформу придатною як для домашнього використання, так і для потреб малого бізнесу. Завдяки адаптивному дизайну, Wallet легко використовувати на різних пристроях.

Таблиця 1.2

Порівняння платформ для управління фінансами

Платформа	Кількість користувачів	Основні функції
Mint	30 млн	Інтеграція з банками, автоматизація витрат
YNAB	10 млн	Методика 4 правил, дисципліноване бюджетування
Wallet	5 млн	Мультивалютність, спільний доступ

1.2 Порівняння популярних технологій розробки адаптивних веб-додатків

Розробка адаптивних веб-додатків становить одну з ключових складових сучасного програмування, оскільки забезпечує створення рішень, здатних функціонувати коректно на різних пристроях і екранах. Цей процес передбачає використання ефективних засобів розробки, до яких належать JavaScript-фреймворки та CSS-фреймворки. Застосування таких інструментів сприяє досягненню високого рівня продуктивності, забезпечує гнучкість у реалізації функціоналу й створює комфортні умови для роботи розробника.

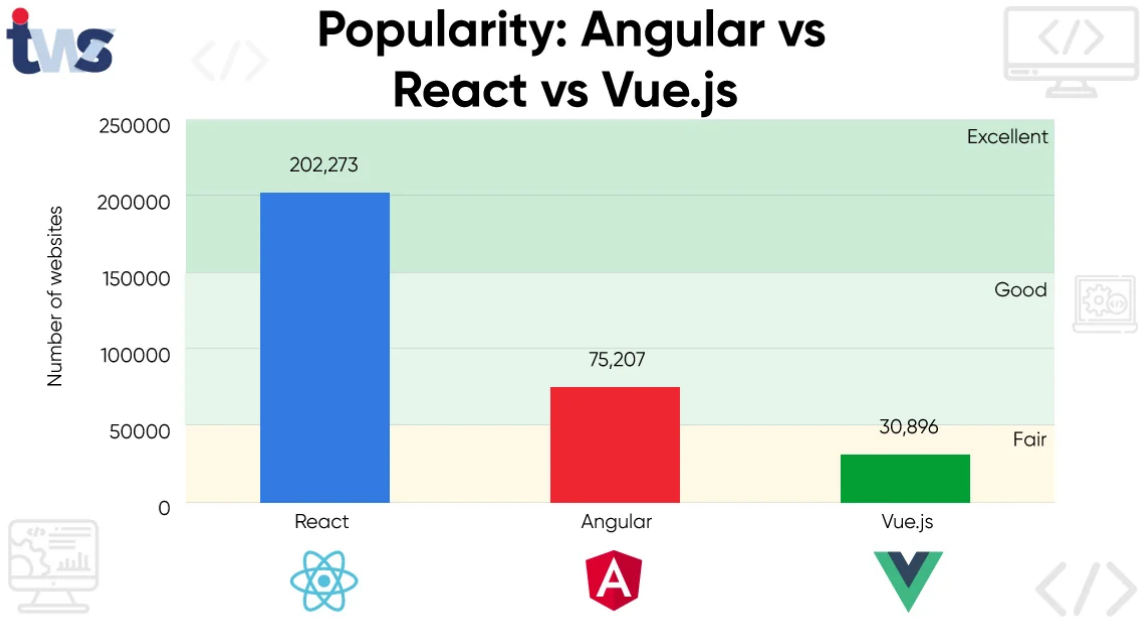


Рис. 1.4 Популярність бібліотек React, Angular та Vue.js [4]

До сучасних JavaScript-технологій, орієнтованих на адаптивну розробку, належать потужні інструменти й фреймворки, здатні забезпечити масштабованість, гнучку архітектуру та стабільну продуктивність веб-додатків. Одним із найпоширеніших інструментів виступає бібліотека React, що була створена компанією Facebook для розробки інтерактивних користувацьких інтерфейсів. Реактивний принцип програмування, який реалізовано в цій бібліотеці, дозволяє формувати багатокомпонентні системи з високим ступенем адаптивності. Упровадження концепції Virtual DOM надає можливість ефективно працювати з великими обсягами даних, знижуючи навантаження на браузер і зменшуючи кількість взаємодій з реальним DOM.

Ще одним ефективним інструментом виступає Angular — повноцінний фреймворк, який розроблений компанією Google. Цей фреймворк побудовано на принципі двостороннього зв’язку між даними, завдяки чому модель та представлення залишаються синхронізованими без потреби у написанні додаткового коду. Серед особливостей Angular виокремлюється підтримка бібліотеки RxJS, що дозволяє ефективно обробляти асинхронні потоки даних.

До числа прогресивних рішень належить і Vue.js — фреймворк, що вирізняється простотою інтеграції, зрозумілою документацією та мінімальним порогом входу. Серед ключових переваг Vue.js варто зазначити його здатність до поступового впровадження в існуючі проєкти без кардинальних змін структури. Такий підхід робить цей фреймворк зручним як для невеликих команд, так і для повноцінних середовищ розробки з великою кількістю учасників. Vue.js підтримує компонентну архітектуру, що забезпечує повторне використання коду та спрощує тестування окремих частин додатку.

Для реалізації адаптивного дизайну веб-додатків активно застосовуються CSS-фреймворки, що забезпечують швидке й зручне оформлення інтерфейсу. Одним із найсучасніших рішень виступає Tailwind CSS — утилітарний CSS-фреймворк, який надає набір готових класів для розробки інтерфейсів з нуля. Упровадження утилітарного підходу забезпечує повний контроль над візуальними стилями без потреби створення додаткових CSS-файлів. Це спрощує роботу розробника та дозволяє зосередитися на логіці застосунку, не витрачаючи час на структурування стилів.

До найпоширеніших CSS-фреймворків також належить Bootstrap — інструмент, що вже тривалий час активно застосовується для створення адаптивних веб-інтерфейсів. Bootstrap містить велику кількість попередньо створених компонентів, серед яких кнопки, форми, модальні вікна, навігаційні панелі та багато іншого. Ці елементи адаптуються під різні розміри екранів автоматично, що дозволяє розробнику оперативно реалізовувати задумане.

Ще одним ефективним рішенням для реалізації адаптивного дизайну є фреймворк Foundation, який орієнтований на модульність, доступність та семантичну структуру коду. В основі архітектури цього фреймворку закладено принцип mobile-first, що передбачає початкову орієнтацію на мобільні пристрої. Завдяки цьому забезпечується висока продуктивність і стабільна робота додатку на всіх типах пристроїв.

Підсумовуючи, можна зазначити, що вибір фреймворків залежить від особливостей проєкту, вимог до продуктивності та масштабування. React

забезпечує швидкодію й гнучкість, Angular — повний набір засобів для складних рішень, Vue.js — простоту інтеграції. Tailwind CSS дозволяє створювати унікальні дизайни без зайвого коду, Bootstrap — швидко прототипувати інтерфейси, а Foundation орієнтується на мобільну адаптацію. Раціональне поєднання цих технологій сприяє ефективній реалізації адаптивних веб-додатків..

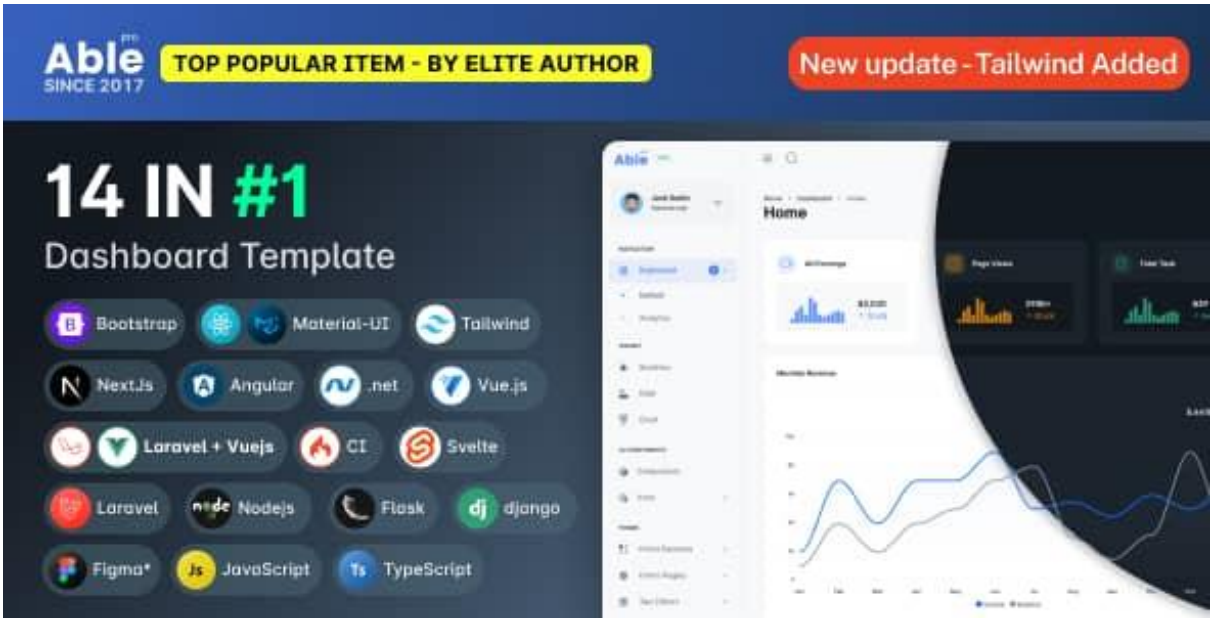


Рис. 1.5 Приклад використання Tailwind CSS [5]

Таблиця 1.3

Перевірка навігації між екранами: працюємо з кнопками «Назад» та «Головна»

Технологія	Тип	Складність освоєння	Основні особливості
React	Фреймворк	Середня	Компонентний підхід, Virtual DOM
Angular	Фреймворк	Висока	Двостороннє прив’язування даних
Vue.js	Фреймворк	Низька	Простота інтеграції
Tailwind CSS	CSS-фреймворк	Середня	Утилітарний підхід
Bootstrap	CSS-фреймворк	Низька	Готові компоненти, сітка
Foundation	CSS-фреймворк	Середня	Модульність, семантичність

Порівняльний аналіз популярних бібліотек та фреймворків для веб-розробки охоплює різні аспекти, що визначають доцільність їх використання в проєктах різного масштабу. Одним із важливих чинників виступає рівень підтримки та активність спільноти, що має істотне значення для розробників, особливо в контексті інтеграції інструментів та пошуку рішень у разі виникнення проблем.

Унаслідок широкої підтримки з боку ком’юніті та наявності великої кількості супровідної документації, React створює сприятливі умови для оперативного вирішення типових задач і сумісності з іншими технологіями. Фреймворки Angular та Bootstrap, завдяки тривалому періоду розвитку, демонструють стабільну

підтримку, що сприяє їхньому широкому застосуванню в корпоративному середовищі та великих програмних системах.

Продуктивність належить до ключових параметрів порівняння сучасних вебфреймворків і бібліотек. Angular орієнтовано на розробку складних та масштабованих рішень, завдяки чому фреймворк ефективно реалізується у великих додатках корпоративного рівня. Його архітектура забезпечує чіткий розподіл відповідальностей між модулями, а двостороннє зв'язування даних і використання бібліотеки RxJS формують високий рівень швидкодії при обробці потокової інформації та динамічному оновленні інтерфейсу. Такий підхід сприяє підтримці складної логіки взаємодії між компонентами без втрати продуктивності. У свою чергу, Vue.js демонструє відмінну легкість, простоту налаштування та швидкість роботи навіть у браузерах з обмеженими ресурсами, що робить його ефективним вибором для реалізації невеликих рішень, MVP-проєктів та додатків з високими вимогами до часу розробки.

Модульність виступає ще одним важливим критерієм оцінювання веб-інструментів, оскільки надає можливість гнучкого розширення функціоналу, повторного використання коду та спрощує технічне обслуговування проєкту в довгостроковій перспективі. Фреймворки Foundation та Tailwind CSS побудовані на принципах модульності, що надає змогу оперативно додавати нові елементи або модифікувати існуючі без необхідності повного перегляду структури. Tailwind CSS, зокрема, пропонує утилітарний підхід до стилізації, що дозволяє швидко адаптувати інтерфейс під потреби користувача й уникати надлишкової кастомізації стилів.

Аналіз інструментів показує, що кожен має свої переваги й обмеження, які варто враховувати при виборі. React вирізняється підтримкою спільноти, готовими рішеннями та простотою інтеграції. Angular підходить для масштабних проєктів із високими вимогами до структури й типізації. Vue.js зручний для швидкого створення інтерфейсів і прототипів. Tailwind CSS і Foundation забезпечують модульну стилізацію, адаптивність та зручність без зайвих залежностей.

1.3 Технічні аспекти застосування бібліотеки React для створення інтерактивних інтерфейсів

React — це одна з найпопулярніших бібліотек JavaScript, що активно використовується для створення динамічних і інтерактивних веб-додатків. Її популярність пояснюється високою продуктивністю, адаптивністю та багатою екосистемою інструментів. В основі роботи React лежить кілька ключових концепцій, які роблять її унікальною серед інших технологій.

Однією з головних інновацій React це Virtual DOM. Це легка копія реального DOM, яка дозволяє зменшити кількість змін у браузері до мінімуму. Коли змінюється стан додатка, React порівнює нову версію Virtual DOM зі старою, знаходить відмінності та оновлює лише необхідні елементи, що забезпечує високу продуктивність та плавність роботи додатка.

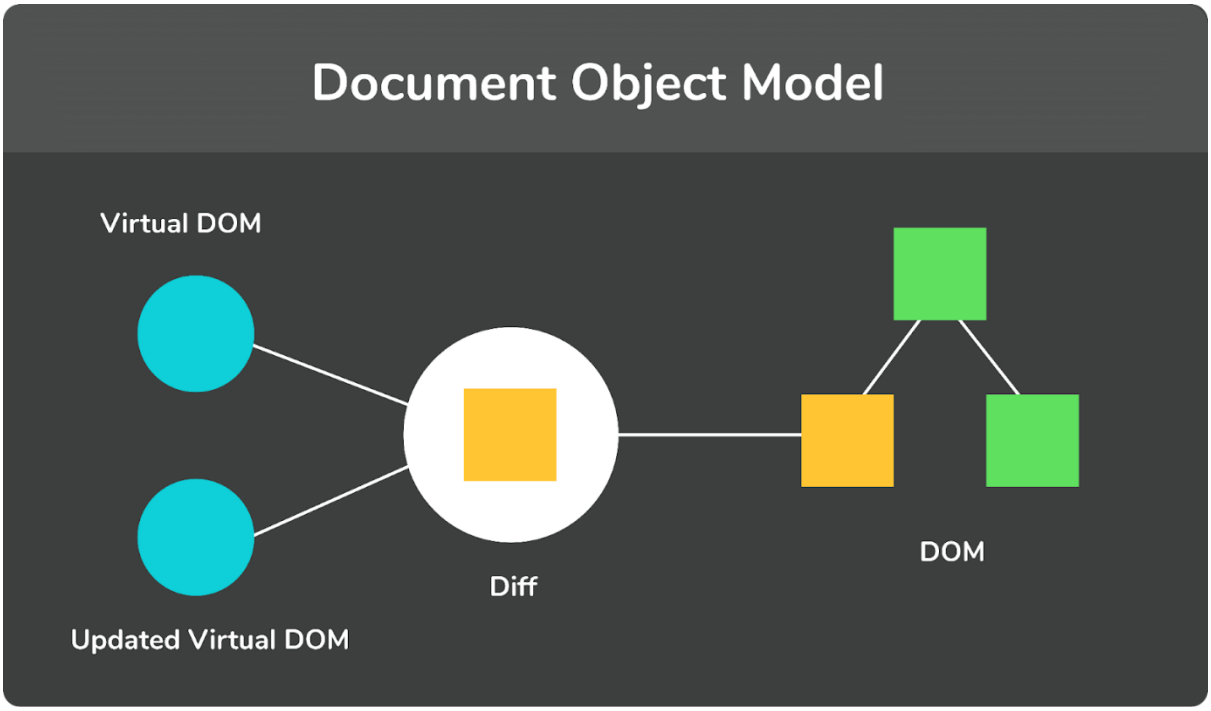


Рис. 1.6 Структура Document Object Model [6]

React використовує компонентний підхід, що дозволяє створювати багаторазові та незалежні частини інтерфейсу. Кожен компонент відповідає за певну функціональність і може легко використовуватися в різних частинах

програми. Це сприяє кращій організації коду та спрощує підтримку і масштабування додатка. JSX, синтаксичне розширення JavaScript, значно спрощує розробку інтерфейсів, дозволяючи описувати структуру додатка у формі, подібній до HTML, що підвищує читабельність коду та знижує ймовірність помилок.

Також фреймворк надає інструменти для управління станом додатків, такі як useState для роботи зі змінними, що зберігають стан, та Context API для передачі стану між компонентами. Для більш складних додатків React часто інтегрується з Redux або MobX, які пропонують розширені можливості управління даними.

Рендеринг на стороні сервера (SSR) становить одну з важливих і потужних функціональних можливостей, реалізованих у React., яка дозволяє значно підвищити швидкість завантаження сторінок, зменшуючи час очікування для користувачів. Використання SSR не лише сприяє швидкому відображенню контенту, але й покращує пошукову оптимізацію (SEO), адже попередньо відрендерені сторінки краще індексуються пошуковими системами. Це забезпечує більш високу видимість сайту та сприяє залученню органічного трафіку. Крім того, SSR полегшує інтеграцію з різними аналітичними платформами, що дозволяє точніше відстежувати дії користувачів та збирати статистичні дані.

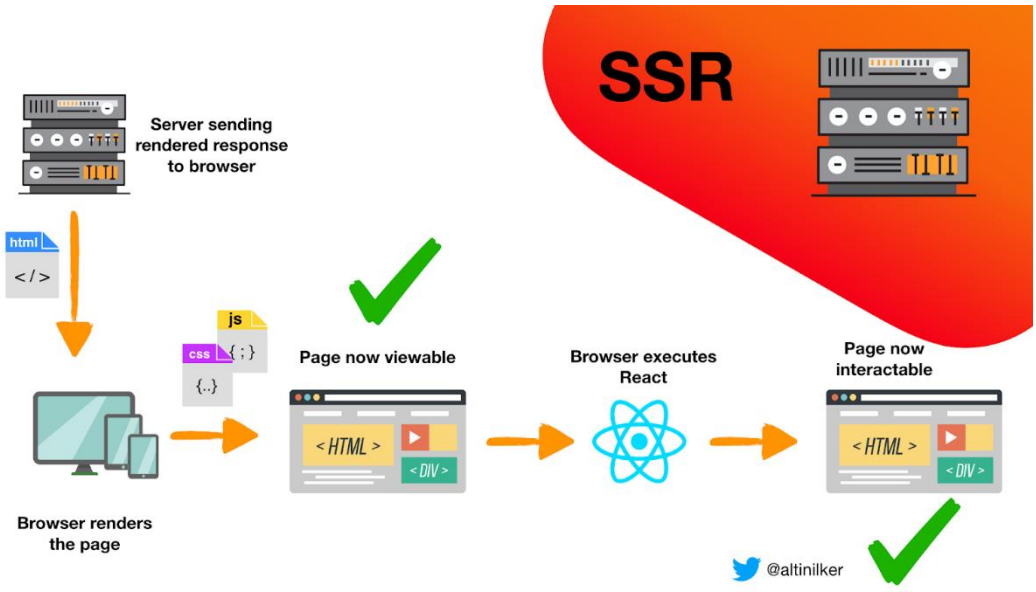


Рис. 1.7 Схема роботи серверного рендерингу [7]

Таблиця 1.4

Ключові особливості React

Особливість	Переваги
Virtual DOM	Зменшення навантаження на браузер, висока продуктивність
Компонентна архітектура	Масштабованість, повторне використання компонентів
JSX	Простота та зручність опису інтерфейсу
Управління станом	Гнучкість у роботі зі складними даними
SSR	Підтримка SEO, швидкість завантаження

React має розвинену екосистему, яка включає такі інструменти, як Redux для управління складним станом додатка, React Router для роботи з маршрутизацією, Styled Components для створення стилів безпосередньо в компонентах, Next.js для побудови серверних рендерингів та статичних сайтів і Gatsby для швидкого створення статичних сайтів з високою продуктивністю. Redux забезпечує централізоване управління станом, що особливо корисно у великих додатках зі складною логікою. React Router дозволяє реалізувати навігацію між різними сторінками додатка, забезпечуючи користувачам зручний та інтуїтивно зрозумілий інтерфейс. Styled Components дозволяє створювати стилі безпосередньо в компонентах, що полегшує підтримку та модифікацію стилів. Крім того, використання Next.js дозволяє реалізувати серверний рендеринг, що позитивно впливає на швидкість завантаження сторінок та покращує пошукову оптимізацію.

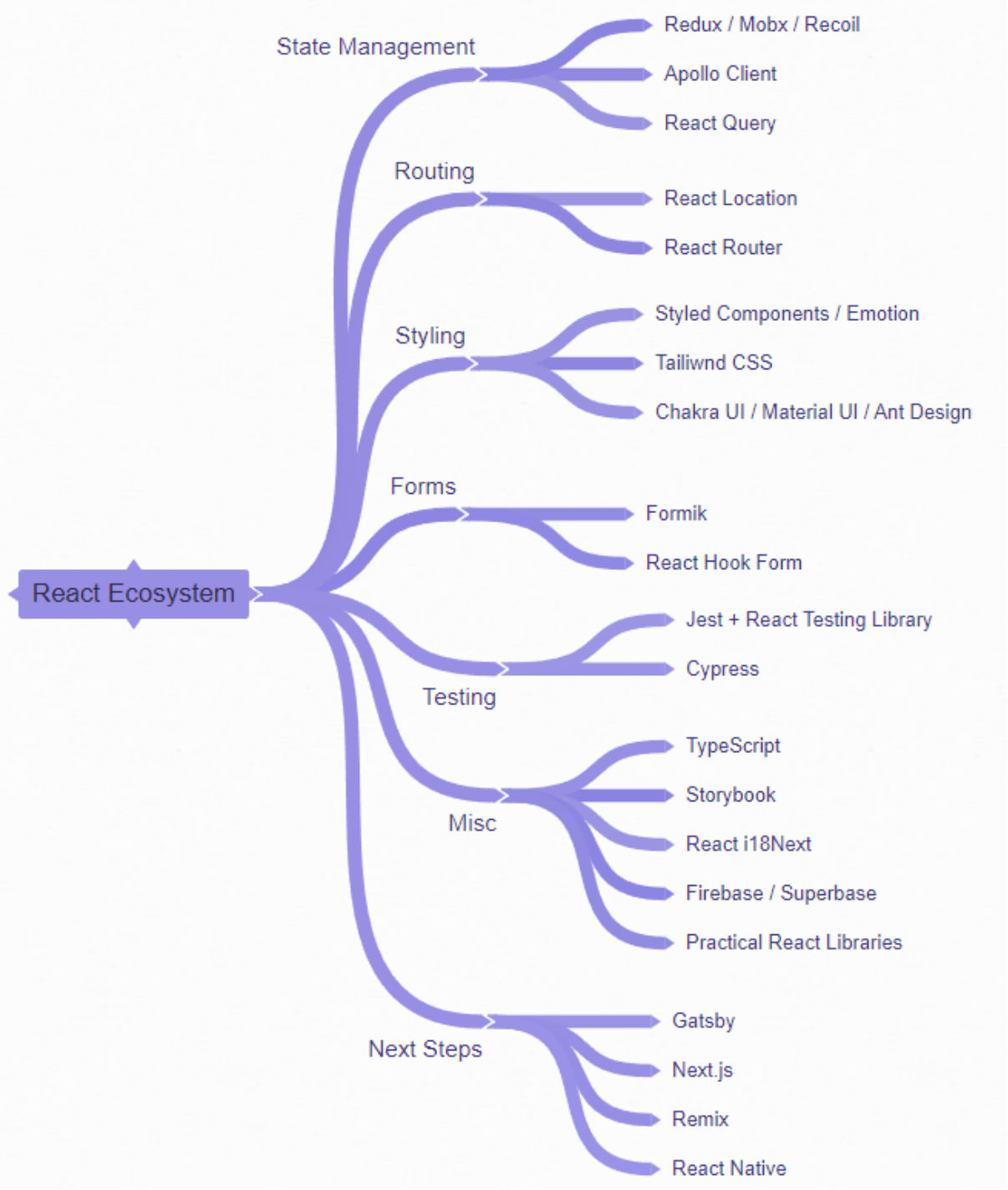


Рис. 1.8 Екосистема React основні бібліотеки та інструменти [8]

Крім перелічених переваг, React відрізняється ще й своєю гнучкістю та адаптивністю до різних завдань. Завдяки відкритій архітектурі та великій спільноті розробників, з'являються численні плагіни та бібліотеки, які розширюють можливості фреймворку. Наприклад, такі інструменти, як Material-UI або Ant

Design, надають готові компоненти з сучасним дизайном, що значно скорочує час розробки інтерфейсів.

Також варто згадати про можливості тестування в React. Фреймворк надає інструменти для створення модульних та інтеграційних тестів, які допомагають підтримувати високу якість коду. Інструменти, такі як Jest та React Testing Library, забезпечують легке та зручне тестування компонентів, що підвищує надійність та стабільність додатків.

Розробники можуть використовувати такі платформи, як Next.js, для створення додатків з серверним рендерингом та статичним генеруванням сторінок. Це надає додаткові можливості для покращення SEO та продуктивності додатків. Next.jstакож забезпечує спрощену налаштування роутингу та автоматичне розділення коду, що сприяє зменшенню часу завантаження сторінок. До переваг React також належить підтримка мобільної розробки, що реалізується за допомогою технології React Native. Ця бібліотека дозволяє створювати нативні мобільні додатки для iOS та Android, використовуючи той самий підхід та принципи, що і у веб-розробці з React. Це забезпечує високу ефективність та зручність для розробників, які можуть використовувати свої знання та досвід з веб-розробки для створення мобільних додатків.

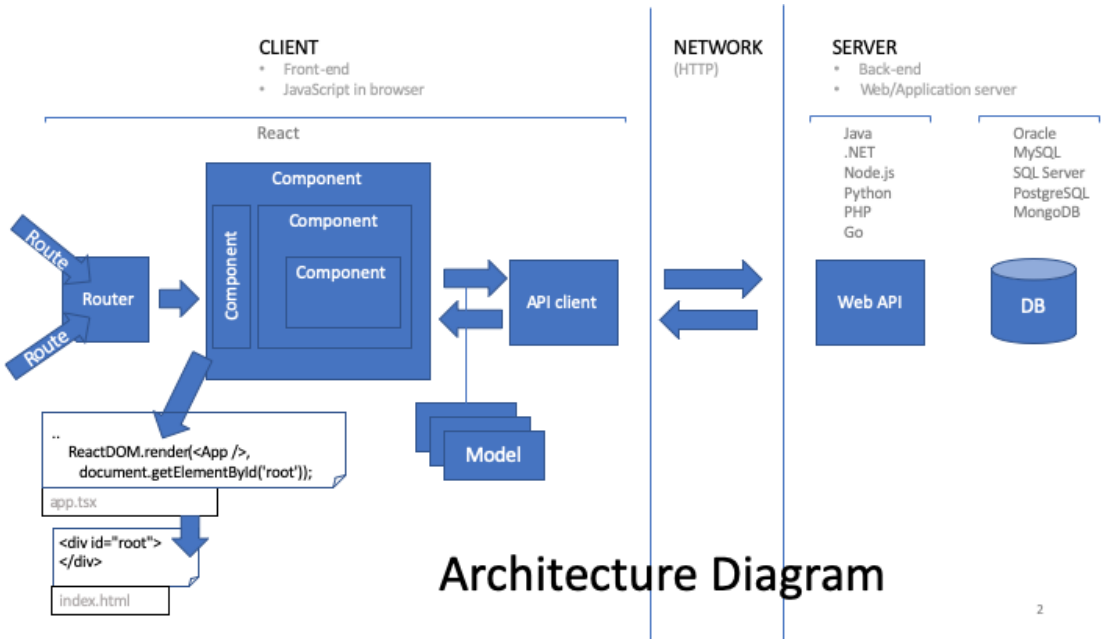


Рис. 1.9 Архітектура взаємодії клієнтської та серверної частин вебдодатку [9]

React має структуровану архітектуру, яка забезпечує гнучкість та простоту у створенні динамічних інтерфейсів користувача. В її основі лежить компонентний підхід, що дозволяє розділяти додаток на незалежні елементи, які можна використовувати повторно у різних частинах проєкту. Компоненти можуть бути як простими, так і складеними, а їхня взаємодія реалізується через передачу властивостей (props) та управління станом (state), що полегшує розподіл функцій між елементами.

Virtual DOM становить важливу складову архітектури React, що забезпечує ефективне оновлення інтерфейсу користувача — полегшена версія реального DOM, яка оновлюється у пам'яті, а не безпосередньо у браузері. Це рішення дозволяє мінімізувати кількість операцій з реальним DOM, що сприяє підвищенню продуктивності додатку та забезпечує швидкість оновлення інтерфейсу навіть при великих обсягах даних.

У масштабних додатках для централізованого управління станом застосовується Redux. Він дозволяє зберігати всі дані у єдиному сховищі (store) та керувати передачею інформації через дії (actions) та редюсери (reducers). Це сприяє чіткій організації даних та полегшує відстеження змін у додатку, забезпечуючи зручну роботу з великими масивами даних.

Архітектура React базується на поєднанні компонентного підходу, ефективного оновлення інтерфейсу через Virtual DOM та централізованого управління даними з Redux. Це рішення забезпечує масштабованість та гнучкість додатків, дозволяючи з однаковою ефективністю розробляти як невеликі проєкти, так і складні системи.

Важливим аспектом активна підтримка спільноти React та регулярні оновлення фреймворку. Це сприяє впровадженню нових функцій, виправленню виявлених недоліків та адаптації до сучасних вимог ринку, що робить React актуальним та перспективним інструментом для розробників.

1.4 Можливості бібліотеки Tailwind CSS для створення адаптивного дизайну додатків

У сучасній розробці веб-додатків значну увагу приділяють швидкості створення адаптивного дизайну, гнучкості та індивідуальності візуальних рішень. Tailwind CSS класифікується як утилітарний CSS-фреймворк, який дозволяє розробникам створювати унікальні та адаптивні дизайни без необхідності писати власні стилі з нуля. Популярність Tailwind CSS зумовлена його простотою, широким спектром можливостей та гнучкістю у налаштуваннях. Основна концепція Tailwind CSS полягає в утилітарному підході, який дозволяє створювати складні дизайни за допомогою набору простих класів, таких як класи для встановлення розміру, кольору, розташування та інше. Це дозволяє розробникам будувати складні макети без зайвих зусиль.

Утилітарний підхід забезпечує' зменшення дублювання коду, висока швидкість розробки та легке налаштування дизайну на різних етапах проєкту. Tailwind CSS підтримує створення адаптивного дизайну завдяки вбудованим медіа-запитам, що дозволяє налаштовувати стилі для різних розмірів екранів без написання додаткового CSS. Наприклад, класи sm, md, lg, xl дозволяють працювати з різними розмірами екранів, забезпечуючи простоту зміни стилів для мобільних пристроїв, планшетів та настільних комп'ютерів.

Tailwind CSS дозволяє налаштовувати дизайн відповідно до потреб конкретного проєкту. За допомогою конфігураційного файлу tailwind.config.js розробники можуть змінювати кольорову палітру, шрифти, розміри тощо, створюючи унікальні дизайни. Ключовими можливостями кастомізації є зміна стандартних налаштувань та додавання власних класів для розширення функціональності. Tailwind CSS також підтримує оптимізацію файлів стилів за допомогою інструментів, таких як PurgeCSS, що видаляють невикористовувані класи, зменшуючи розмір файлів і покращуючи продуктивність веб-додатків.

У порівнянні з іншими CSS-фреймворками, Tailwind CSS вирізняється утилітарним підходом, використанням готових класів для стилізації, вбудованими

медіа-запитами для різних пристроїв, можливістю кастомізації через конфігураційний файл та зменшенням розміру файлів за допомогою PurgeCSS. Це робить Tailwind CSS особливо привабливим для розробників, які цінують швидкість, гнучкість та ефективність.

Приклади використання Tailwind CSS охоплюють широкий спектр вебрішень, серед яких особливо вирізняються сайти електронної комерції. Завдяки високому рівню кастомізації можна створювати унікальні, візуально привабливі інтерфейси для онлайн-магазинів, які відповідають брендовому стилю. Крім того, Tailwind CSS активно використовується для корпоративних порталів, де важливо забезпечити адаптивність дизайну, відповідність сучасним UI-трендам і зручність масштабування проєкту. Ще одним поширеним напрямом є розробка односторінкових додатків (SPA), де Tailwind CSS забезпечує високу швидкість верстки, гнучкість налаштувань і полегшену підтримку, що робить його ідеальним вибором для швидких і адаптивних інтерфейсів.

Таблиця 1.5

Основні класи Tailwind CSS та їх використання

Клас	Призначення	Приклад використання
text-blue500	Встановлення синього кольору тексту	Текст із синім відтінком
bg-green-200	Світло-зелений фон	Елемент із світло-зеленим фоном
p-4	Встановлення внутрішнього відступу	4 одиниці відступу з усіх боків
sm:text-sm	Розмір тексту для маленьких екранів	Текст розміром small для sm

Завдяки наведеним прикладам класів стає очевидним, що Tailwind CSS значно полегшує процес стилізації елементів, надаючи розробникам гнучкість та можливість швидко налаштовувати візуальні аспекти додатка. Додатково варто розглянути ще кілька важливих аспектів, які роблять Tailwind CSS потужним інструментом для сучасної розробки.

Підтримка псевдокласів належить до важливих функціональних особливостей Tailwind CSS, що дозволяє стилізувати елементи у різних станах, таких як :hover, :focus або ::before і ::after. Це додає більше можливостей для створення інтерактивних та динамічних інтерфейсів.

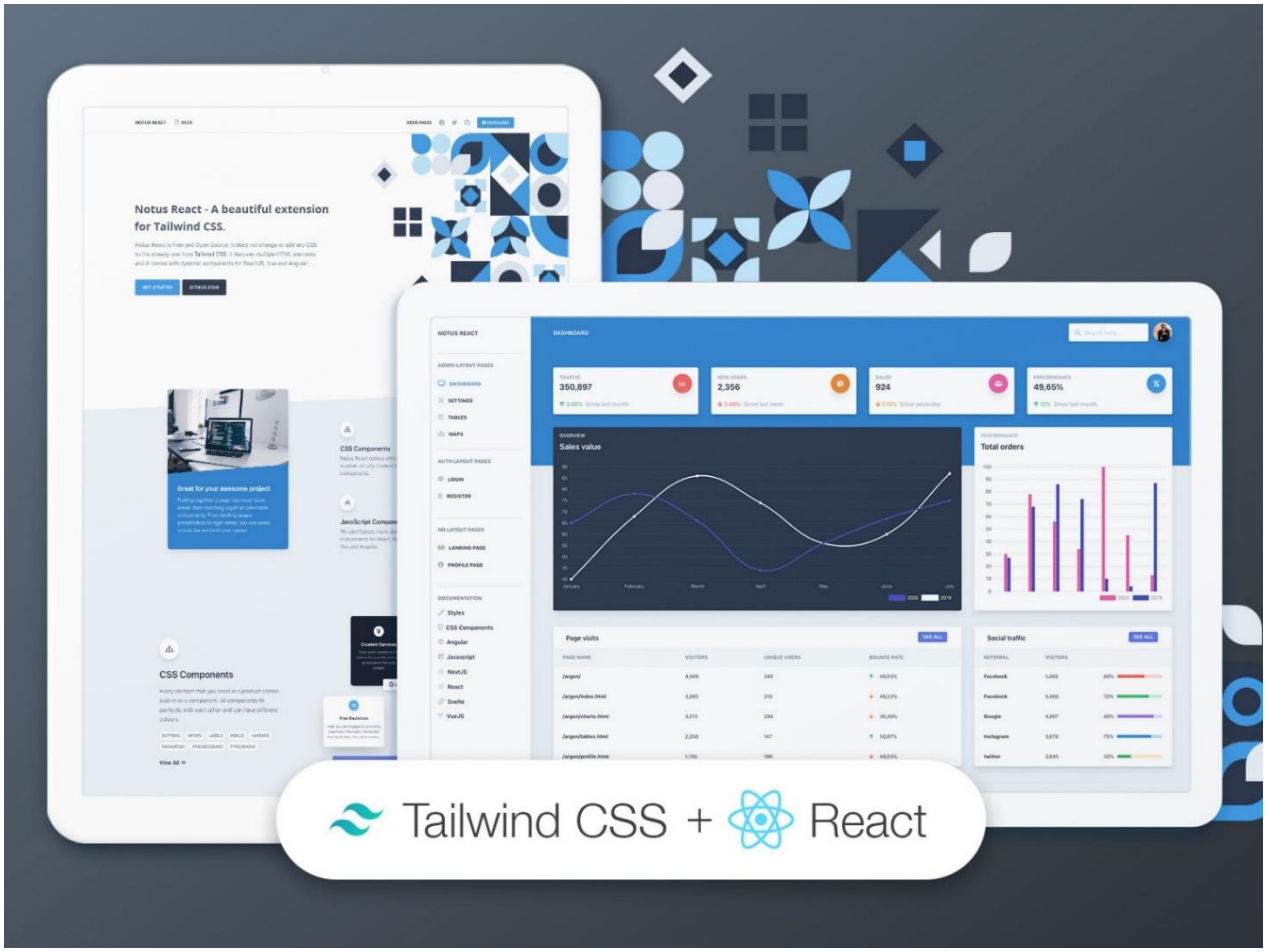


Рис. 1.10 Дизайн панелі управління на Tailwind CSS та React [10]

Також важливо зазначити, що Tailwind CSS інтегрується з сучасними фреймворками та бібліотеками JavaScript, такими як React, Vue.js, Angular, що робить його універсальним вибором для різних типів проєктів. Інтеграція з цими

фреймворками дозволяє створювати компоненти з унікальними стилями, зберігаючи при цьому загальну структуру та логіку програми.

Завершуючи розгляд можливостей Tailwind CSS, варто акцентувати увагу на його активному розвитку та підтримці спільнотою. Tailwind CSS має не лише потужний функціонал для адаптивної розробки, але й багатий набір готових шаблонів та компонентів, які можна використовувати для прискорення розробки інтерфейсів. Спільнота Tailwind CSS постійно створює та оновлює бібліотеки з попередньо стилізованими компонентами, що дозволяє розробникам швидко реалізовувати складні макети без написання додаткового CSS-коду. До важливих аспектів Tailwind CSS належить інтеграція із сучасними інструментами автоматизації розробки, зокрема PostCSS та Webpack, що сприяє ефективному налаштуванню та оптимізації робочих процесів. Це забезпечує зручну компіляцію стилів та можливість додавання нових функцій через плагіни. Наприклад, Tailwind CSS можна легко інтегрувати з PostCSS, що дозволяє використовувати додаткові плагіни для генерації кольорових схем, адаптивних відступів або анімацій.

Також Tailwind CSS надає можливості для побудови кастомних тем, що особливо актуально для проєктів, де важливо зберігати єдиний стиль оформлення на всіх сторінках додатку. За допомогою конфігураційного файлу tailwind.config.js можна налаштовувати глобальні стилі, такі як кольори, типографіка, розміри шрифтів та інші параметри. Це забезпечує гнучкість у створенні унікальних тем та можливість швидкого оновлення стилів при зміні вимог проєкту.

Інтеграція Tailwind CSS з фреймворками, такими як Next.js та Gatsby, відкриває додаткові можливості для створення серверних рендерингів та оптимізованих статичних сторінок. Це забезпечує швидке завантаження контенту та підвищує продуктивність додатків, що особливо важливо для великих проєктів зі складними візуальними елементами.

Tailwind CSS не лише полегшує створення адаптивних інтерфейсів, але й забезпечує інструменти для масштабування проєктів, інтеграції з сучасними фреймворками та оптимізації продуктивності.

2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ДЛЯ ДОДАВАННЯ ФІНАНСОВИХ ЗАПИСІВ

2.1 Побудова архітектури веб-додатку для управління фінансами

Ретельно спроєктована архітектура становить основу успішної реалізації веб-додатку для управління фінансами, забезпечуючи високу модульність, легке масштабування та зручність подальшої підтримки й розвитку проєкту. Для реалізації клієнтської частини застосовано бібліотеку React, що дозволяє розбити інтерфейс на незалежні компоненти з власною логікою, що значно спрощує процес розробки та тестування. Водночас для стилізації компонентів та забезпечення адаптивності інтерфейсу використано бібліотеку Tailwind CSS, яка дозволяє швидко налаштовувати стилі за допомогою спеціалізованих класів без необхідності створення окремих CSS-файлів.

Архітектуру веб-додатку сформовано на основі компонентного підходу, у межах якого ключову роль виконує компонент-контейнер MainLayout.jsx, відповідальний за формування загальної структури сторінок. Цей компонент відповідає за основну розмітку сторінок, включаючи бокове меню навігації та контентну область для виведення дочірніх компонентів. Основні компоненти додатку Dashboard.jsx, Transactions.jsx, Analytics.jsx, Settings.jsx вбудовуються в цей контейнер відповідно до поточного маршруту.

Керування маршрутизацією здійснюється за допомогою бібліотеки React Router, яка відповідає за динамічну зміну компонентів у залежності від обраного маршруту. Це дозволяє користувачам плавно та швидко перемикатися між сторінками без перезавантаження додатку. До важливих аспектів архітектурного проєктування належить централізація даних і логіки, що відповідає за отримання та обробку транзакцій. Для цього створено контекст TransactionContext.jsx, який надає доступ до глобального стану транзакцій.

Взаємодія з зовнішнім API Monobank для отримання транзакцій реалізована у файлі monobank.js, що розташований у теці utils. Тут зібрані функції, які відповідають за інтеграцію з API та подальше форматування отриманих даних для зручної роботи в контексті додатку.

На рисунку 2.1 зображено повну схему архітектури веб-додатку, де наочно представлені всі компоненти, їхні взаємозв'язки, а також потоки даних між ними. Ця схема надає чітке уявлення про логіку взаємодії компонентів між собою.

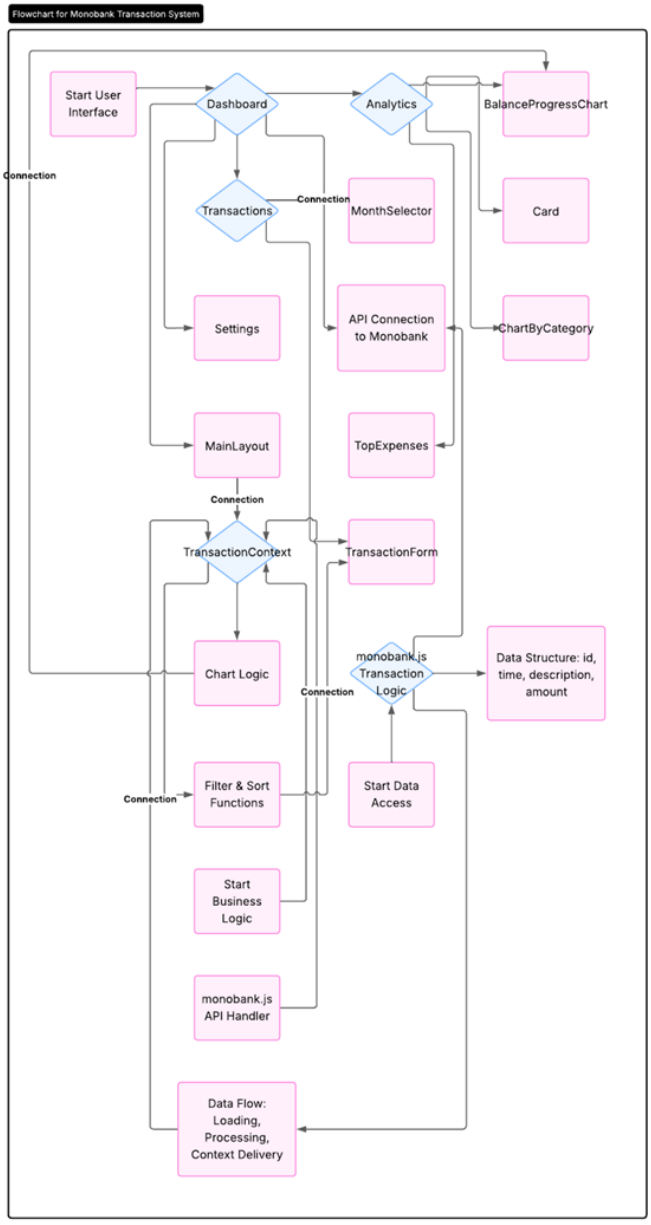


Рис. 2.1 Архітектура веб-додатку та взаємозв'язок основних компонентів

Для наочності, на рисунку 2.2 наведено структуру файлів і папок проєкту. Ця ієрархія дозволяє чітко зрозуміти, як організовано компоненти, контексти, сторінки та утиліти, що додатково спрощує орієнтацію в проєкті та підтримку коду в майбутньому.

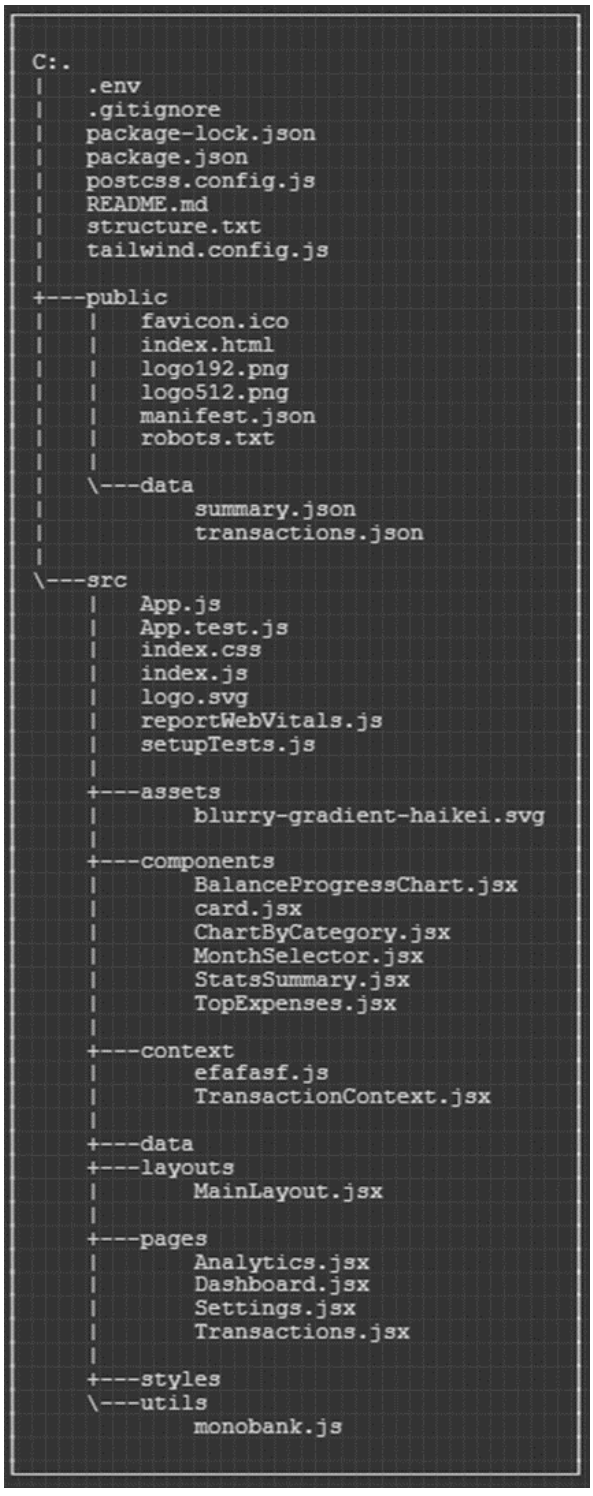


Рис. 2.2 Структура проєкту: компоненти, контексти, сторінки, утиліти

На схемі зображено загальну логіку роботи веб-додатку для фінансового менеджменту. Відправною точкою є зберігання токена користувача у локальному середовищі, який використовується для автентифікації запитів до Monobank API. Після успішного отримання транзакцій функцією fetchAllTransactions() дані передаються до TransactionContext, який виступає як централізоване сховище стану. Оброблені дані передаються до основних компонентів інтерфейсу — Dashboard.jsx, Analytics.jsx та Transactions.jsx, де здійснюється їх візуалізація у вигляді графіків, списків і таблиць. Така архітектура дозволяє забезпечити повторне використання даних, ефективну передачу стану та розділення логіки між рівнями застосунку.

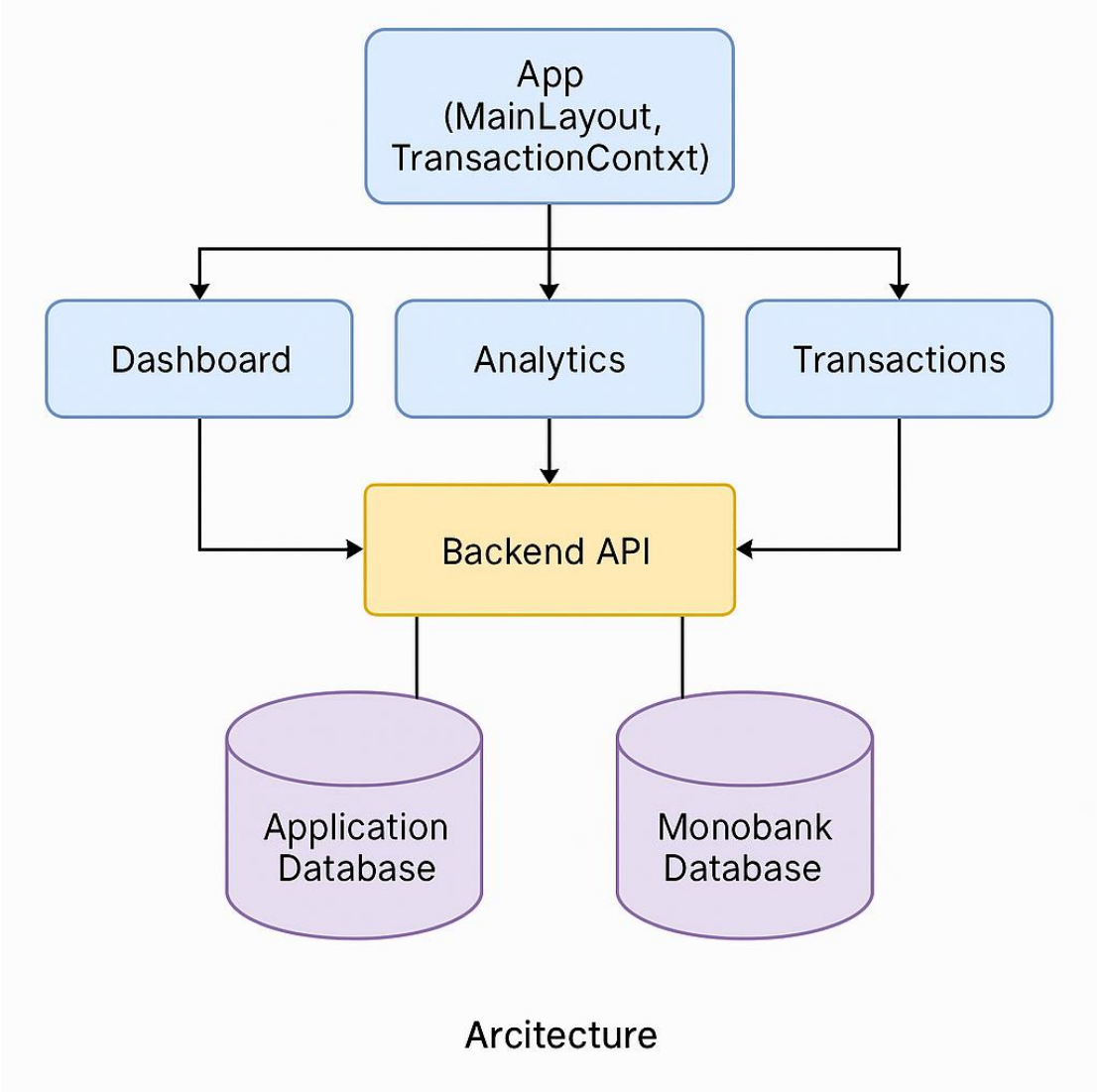


Рис 2.3 Архітектурна схема веб-додатк

Для створення користувацького інтерфейсу веб-додатку для управління фінансами була використана низка сучасних бібліотек і технологій, що дозволяють забезпечити високу ефективність, адаптивність та інтерактивність. Одна з основних технологій, яка була обрана для побудови додатку, — це бібліотека React. Вона дозволяє створювати динамічні інтерфейси, які автоматично оновлюються при зміні стану компонента. React забезпечує модульність проєкту, що дозволяє розбивати інтерфейс на невеликі, незалежні компоненти, кожен з яких виконує свою функцію. Цей підхід дозволяє спростити як процес розробки, так і подальшу підтримку коду. Завдяки React інтерфейс додатку стає більш чуйним і ефективним, оскільки оновлення відбуваються тільки в тих компонентах, які змінюються, замість перезавантаження всієї сторінки.

Окрім React, для стилізації і адаптивного дизайну додатку використано Tailwind CSS. Зазначена бібліотека виступає ефективним інструментом для створення гнучкого та адаптивного інтерфейсу, забезпечуючи можливість роботи зі стилями без потреби у написанні значного обсягу власного CSS-коду. Tailwind CSS використовує утилітні класи, які дають змогу налаштовувати стиль кожного елементу інтерфейсу прямо в JSX-коді, що значно пришвидшує розробку. Адаптивний дизайн реалізовано за допомогою класів Tailwind CSS, що забезпечують підтримку різних розмірів екранів.

Це дозволяє автоматично підлаштовувати компоненти інтерфейсу під мобільні телефони, планшети та десктопи без необхідності створювати окремі стилі для кожної версії екрану. Наприклад, елементи меню автоматично змінюють свій вигляд на мобільних пристроях, забезпечуючи зручний доступ до всіх функцій без шкоди для користувацького досвіду

Для реалізації маршрутизації між сторінками було використано React Router, що дозволяє організувати навігацію між різними частинами веб-додатку без перезавантаження сторінок. React Router дає можливість визначити маршрути для кожної сторінки додатку та відображати відповідні компоненти в залежності від URL. Це робить інтерфейс додатку більш динамічним і зручним для користувача,

оскільки переходи між сторінками виконуються миттєво, без затримок, не потребуючи перезавантаження всього додатку.

До числа важливих бібліотек, застосованих у проєкті, належить Recharts, яка використовується для побудови графічних візуалізацій даних. Ця бібліотека спеціалізується на створенні графіків і діаграм на основі даних, що дозволяє ефективно візуалізувати фінансові дані користувача. Recharts підтримує безліч типів графіків, таких як лінійні графіки, гістограми, кругові діаграми та інші, що дає можливість ефективно відображати дані про витрати, доходи та інші показники. Вибір Recharts зумовлено її простотою в інтеграції з React, а також гнучкістю в налаштуваннях графіків, що дозволяє швидко створювати наочні діаграми, які автоматично оновлюються при зміні даних.

Для більш детального управління станом компонентів та ефективного взаємодії між різними частинами додатку в проекті активно використовуються вбудовані хуки React. Наприклад, хук useState використовується для зберігання та оновлення стану компонентів, що становить основу реалізації інтерактивної поведінки веб-додатку. Хук useEffect забезпечує виконання певних дій після рендерингу компонентів, наприклад, для завантаження даних або виконання асинхронних запитів до API. useContext і useLocation використовуються для централізованого управління станом додатку та забезпечення доступу до даних у різних частинах інтерфейсу, що дозволяє зручно передавати дані між компонентами без необхідності використання пропсів.

Використаний набір бібліотек та інструментів забезпечує ефективність, адаптивність і зручність користування, що є важливим чинником для веб-додатків цього типу. Завдяки поєднанню сучасних технологій, таких як React, Tailwind CSS та Redux, забезпечується не лише стабільність роботи додатку, але й його здатність швидко реагувати на зміни даних та коректно відображати інформацію на різних пристроях. Всі технології та інструменти були обрані з огляду на потреби проекту і найкращі практики для створення сучасних веб-додатків.

2.2 Реалізація функціональності вебдодатку

Основна мета реалізації функціональності вебдодатку полягає у створенні інтерфейсу для управління фінансовими операціями, що забезпечує користувачу доступ до статистики витрат та доходів, аналізу динаміки фінансових потоків та взаємодії із транзакціями. Архітектура вебдодатку організована за компонентним принципом, що дозволяє ізолювати логіку кожного елемента та забезпечити їхню повторну використовуваність.

Основну сторінку додатку сформовано на базі компонента Dashboard.jsx, який відображає ключові фінансові показники, динаміку змін за останні сім днів, список останніх транзакцій і фінансові поради. Усі дані, які використовуються на цьому екрані, надходять із контексту TransactionContext.jsx, що дозволяє централізовано управляти транзакціями та забезпечувати актуальність інформації у режимі реального часу.

Вітальний блок, розташований у верхній частині інтерфейсу, виступає першим елементом, що відображається на головній сторінці. Він включає заголовок та коротке повідомлення для користувача, які формуються статично у компоненті. У той же час, блок із вибором місяця MonthSelector забезпечує динамічне оновлення даних за обраний період, використовуючи хук useState. Така структура дозволяє підтримувати єдиний стиль та логіку на всіх сторінках додатку, забезпечуючи чіткий і зрозумілий користувацький досвід, що сприяє швидкому орієнтуванню користувача у функціоналі додатку. Крім того, інтеграція MonthSelector з основним контекстом додатку забезпечує синхронізацію даних та їх актуалізацію у реальному часі.

```
79      <div className="bg-gradient-to-br from-white via-purple-50 to-purple-100 p-6 rounded-xl shadow-md">
80        <div className="flex justify-between items-center mb-4">
81          <div>
82            <h2 className="text-3xl font-bold">👋 Ласкаво просимо</h2>
83            <p className="text-sm text-purple-800">Тут буде ваша фінансова статистика 💰</p>
84          </div>
85          <MonthSelector selected={selectedDate} onChange={setSelectedDate} />
86        </div>
```

Рис. 2.4 Вітальний блок із вибором місяця у Dashboard.jsx

Вітальний блок залишається на екрані незалежно від обраних користувачем даних, зберігаючи єдину стилістику інтерфейсу. Це дозволяє не лише створити зрозумілу навігацію для користувача, а й зосередити увагу на основних даних про фінансові показники. Водночас, завдяки такому підходу забезпечується цілісність дизайну додатку, що робить інтерфейс більш впізнаваним та легким для сприйняття. Крім того, вітальний блок виконує роль орієнтира для користувача, підтримуючи логічну структуру відображення інформації незалежно від обраного контенту.

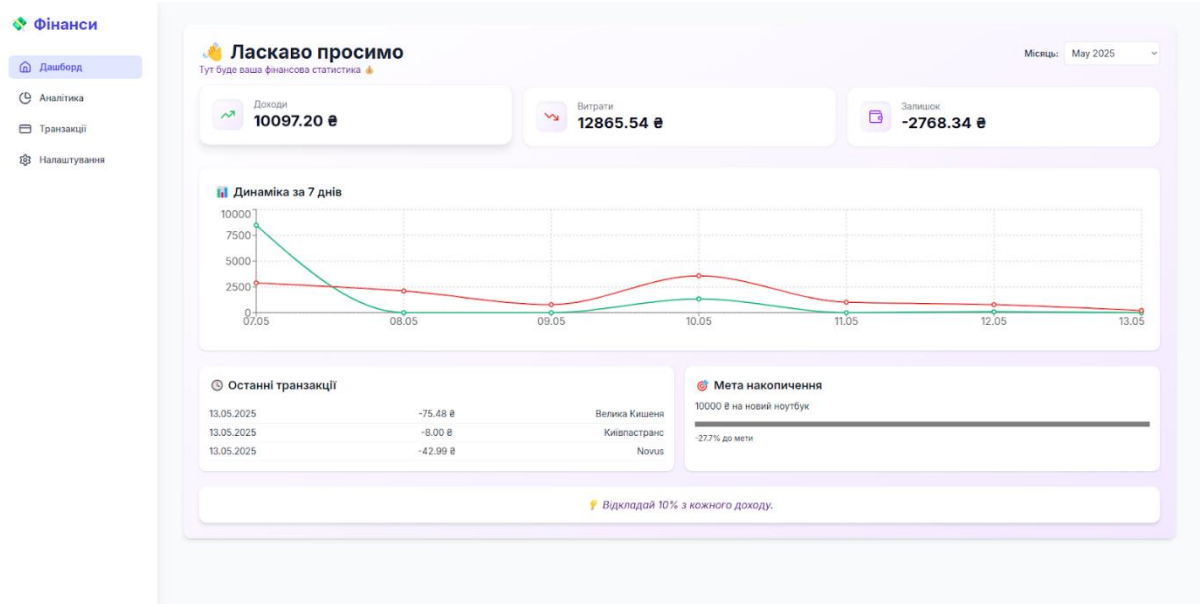


Рис. 2.5 Фінансові показники (Доходи, Витрати, Залишок) у Dashboard.jsx.

Блок фінансових показників, який розміщується після вітального елементу, містить три інформаційні картки: доходи, витрати та залишок коштів. Компонент Card.jsx використовується для створення кожної картки та приймає дані через пропси, що дозволяє застосовувати єдину структуру для різних типів фінансових показників. Дані для карток формуються через функції income, expenses та balance, які розраховуються на основі транзакцій за обраний період. Взаємодія між Dashboard.jsx та Card.jsx побудована за принципом передачі пропсів, що забезпечує гнучкість у відображенні даних.

```
88          <div className="grid grid-cols-1 sm:grid-cols-3 gap-4 mb-8">
89            <Card title="Доходи" value={(income / 100).toFixed(2) + " ₴"} icon={<TrendingUp className="text-green-500" />} />
90            <Card title="Витрати" value={(Math.abs(expenses) / 100).toFixed(2) + " ₴"} icon={<TrendingDown className="text-red-500" />} />
91            <Card title="Залишок" value={(balance / 100).toFixed(2) + " ₴"} icon={<Wallet className="text-purple-500" />} />
92          </div>
```

Рис 2.6 Код компонента фінансових карток у Dashboard.jsx

Компонент Card.jsx стилізовано так, щоб кожна картка мала унікальний колір, що відповідає типу показника: зелений для доходів, червоний для витрат та фіолетовий для залишку. Це дозволяє користувачу швидко орієнтуватися у фінансових даних та візуально розрізняти типи операцій.

Рис 2.7 Список останніх транзакцій у Dashboard.jsx

Подальша реалізація інтерфейсу включає компонент LineChart, який відповідає за візуалізацію динаміки фінансів за останні 7 днів. Цей компонент побудований за допомогою бібліотеки Recharts та приймає дані про доходи та витрати, які формуються у масиві weeklyData. Визначення даних для графіка реалізовано через .map(), який проходить по останніх 7 днях та обчислює загальну суму доходів та витрат за кожен день.

```
51  const weeklyData = Array(7).fill(0).map((_, i) => {
52    const date = new Date();
53    date.setDate(date.getDate() - (6 - i));
54    const day = date.toLocaleDateString();
55    const incomeDay = filtered.filter(t => t.amount > 0 && new Date(t.time * 1000).toLocaleDateString() === day).reduce((acc, t) => acc + t.amount, 0);
56    const expenseDay = filtered.filter(t => t.amount < 0 && new Date(t.time * 1000).toLocaleDateString() === day).reduce((acc, t) => acc + t.amount, 0);
57    return {
58      name: day.slice(0, 5),
59      income: incomeDay / 100,
60      expense: Math.abs(expenseDay / 100),
61    };
62  });
```

Рис 2.8 Код формування масиву даних для графіка за 7 днів у Dashboard.jsx

У графіку використано дві лінії — для доходів та витрат, які мають окремі кольори, що дозволяє чітко розділяти два типи фінансових потоків та візуально сприймати інформацію. Це рішення підвищує зрозумілість даних та дозволяє користувачу оцінити динаміку фінансових операцій.

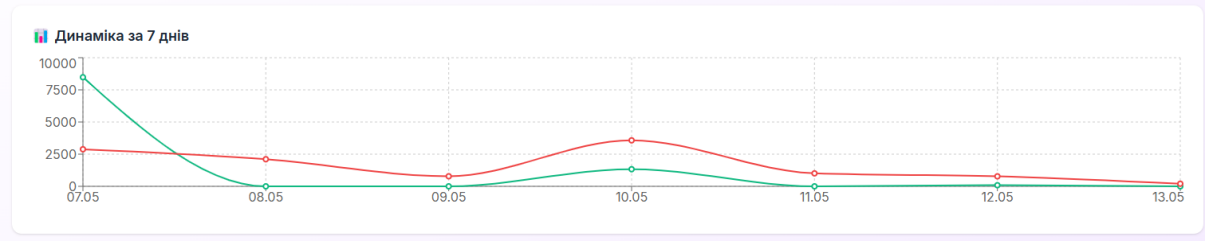


Рис. 2.9 Інтерфейс графіка динаміки фінансів за 7 днів у Dashboard.jsx

Секція останніх транзакцій реалізована як інтерактивний список останніх трьох операцій, дані для яких надходять із контексту TransactionContext.jsx. Рендеринг блоку здійснюється через .map(), що дозволяє оновлювати список у режимі реального часу при зміні даних у контексті. Візуально кожна транзакція відображає дату, суму та опис, що забезпечує користувачу швидкий доступ до останніх змін у фінансах.

```
109                <div className="bg-white rounded-xl shadow p-4">
110                  <h3 className="font-semibold mb-4 text-lg">🕓 Останні транзакції</h3>
111                  <ul className="text-sm">
112                    {recent.map(tx => (
113                      <li key={tx.id} className="grid grid-cols-3 gap-2 py-1 border-b border-gray-100">
114                        <span>{new Date(tx.time * 1000).toLocaleDateString()}</span>
115                        <span className="text-center">{(tx.amount / 100).toFixed(2)} ₴</span>
116                        <span className="text-right">{tx.description}</span>
117                      </li>
118                    ))}
119                  </ul>
120                </div>
```

Рис. 2.10 Код компонента відображення останніх транзакцій у Dashboard.jsx

Секція останніх транзакцій оформлена у вигляді таблиці з єдиною стилістикою, що відповідає загальному дизайну додатку та забезпечує візуальну узгодженість між різними елементами інтерфейсу.

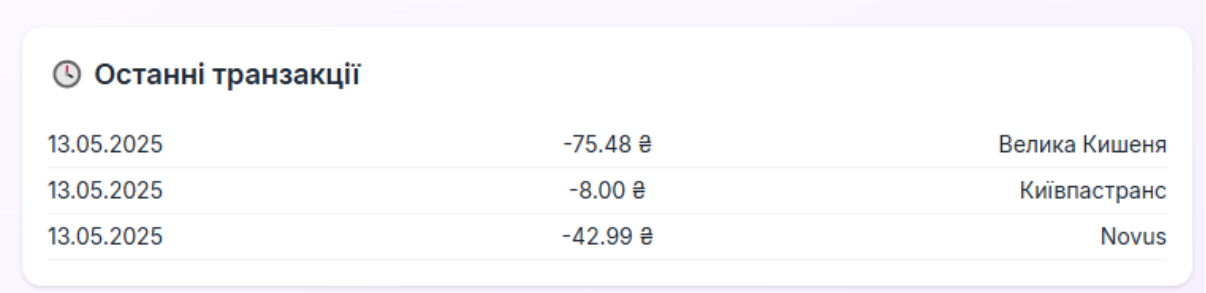


Рис. 2.11 Інтерфейс блоку останніх транзакцій у Dashboard.jsx

Фінансові поради реалізовані у вигляді випадкових повідомлень, які вибираються із масиву adviceList при завантаженні сторінки. Логіка вибору поради здійснюється через хук useEffect, що оновлює текст повідомлення при кожному рендері сторінки. Такий підхід дозволяє підвищити інтерактивність додатку та додати мотиваційний елемент для користувача.

```
130        <div className="bg-white text-purple-900 p-4 rounded-xl text-center shadow-md text-base italic">
131          {advice}
132        </div>
```

Рис 2.12 Код компонента блоку фінансових порад у Dashboard.jsx

Фінансові поради оформлені як окремий блок, який виділяється кольоровим фоном та стилізованим текстом, що створює додатковий акцент у загальній структурі інтерфейсу. Це рішення забезпечує баланс між основним контентом та супутніми повідомленнями, не перевантажуючи інтерфейс. Компонент Dashboard.jsx поєднує у собі кілька функціональних секцій, що дозволяють користувачу оперативно отримувати дані про основні фінансові показники, переглядати динаміку доходів та витрат за останній тиждень та бачити останні транзакції. Вся логіка управління даними централізована через контекст TransactionContext.jsx, що дозволяє підтримувати актуальність даних у режимі реального часу та забезпечує цілісність фінансових даних в інтерфейсі.

Функціональність вебдодатку у розділі аналітики реалізована за допомогою компонента Analytics.jsx, який дозволяє користувачу переглядати динаміку доходів, витрат та балансу за певний період часу. Цей компонент використовує дані про транзакції з контексту TransactionContext та виводить їх у вигляді інтерактивних графіків і списків, що надає можливість наочно оцінити фінансові показники. Компонент Analytics.jsx забезпечує гнучкість у налаштуванні періоду відображення даних, дозволяючи користувачеві обирати конкретний місяць, квартал або рік для детального аналізу. Динамічне оновлення інформації відбувається за допомогою хуків useEffect та useState, що гарантує актуальність відображених даних у режимі реального часу.

Крім того, для візуалізації фінансових показників використовуються інтерактивні графіки, побудовані на базі бібліотеки Recharts, які дозволяють відстежувати зміни у доходах та витратах за обраний період. Це забезпечує не лише зрозумілу подачу інформації, але й можливість глибшого аналізу даних через інтуїтивно зрозумілу графічну форму подання.

На початку компонента виконується імпорт ключових бібліотек, включаючи Recharts для побудови графіків, lucide-react для візуалізації іконок та інших компонентів, які відповідають за відображення фінансових даних: ChartByCategory, TopExpenses, BalanceProgressChart, StatsSummary. Це забезпечує єдину структуру подання інформації, спрощуючи інтеграцію нових компонентів та підтримку загальної логіки відображення фінансових показників. Додатково, така організація компонентів сприяє підтримці єдиного стилю в інтерфейсі додатку.

```
1   // src/pages/Analytics.jsx
2   import { useContext, useState } from "react";
3   import MainLayout from "../layouts/MainLayout";
4   import { TransactionContext } from "../context/TransactionContext";
5   import {
6     BarChart,
7     Bar,
8     XAxis,
9     YAxis,
10    Tooltip,
11    ResponsiveContainer,
12  } from "recharts";
13
14  import { PieChart, Pie, Cell, Legend } from "recharts";
15  import { CircleDollarSign, LineChart } from "lucide-react";
16  import ChartByCategory from "../components/ChartByCategory";
17  import TopExpenses from "../components/TopExpenses";
18  import BalanceProgressChart from "../components/BalanceProgressChart";
19  import StatsSummary from "../components/StatsSummary";
20
```

Рис. 2.13 Код імпорту основних компонентів та бібліотек у Analytics.jsx

Компонент Analytics.jsx призначено для здійснення розширеного аналізу фінансових даних за вибраний період часу. Цей компонент дозволяє користувачеві вибирати часові інтервали для детального перегляду статистики витрат та доходів, що надає можливість здійснювати глибокий аналіз фінансових операцій. Для цього реалізовано набір кнопок, які відображають доступні часові відрізки — день, тиждень, два тижні, місяць та три місяці. Усі ці періоди зберігаються у спеціальній константі PERIODS, яка визначає тривалість кожного з них, що дозволяє легко розширювати функціональність компонента.

Вибір періоду здійснюється за допомогою хуку useState, який зберігає поточне значення змінної period і оновлює його при зміні вибору користувача. Зміна періоду автоматично ініціює повторний рендеринг компонента, що, у свою чергу, активує фільтрацію даних та оновлення вмісту відповідно до нового періоду. Завдяки цьому компонент динамічно підлаштовується під нові умови перегляду, дозволяючи швидко аналізувати транзакції за різні часові відрізки. Такий підхід забезпечує високу інтерактивність, зручність навігації й дозволяє користувачу оперативно оцінювати фінансову інформацію в обраному контексті.

```
21  const PERIODS = {
22    day: 1,
23    week: 7,
24    twoWeeks: 14,
25    month: 30,
26    threeMonths: 90,
27  };
28
29  export default function Analytics() {
30    const { transactions, loading } = useContext(TransactionContext);
31    const [period, setPeriod] = useState("month");
32
33    const now = Date.now();
34    const startTime = now - PERIODS[period] * 24 * 60 * 60 * 1000;
```

Рис. 2.14Ініціалізація періодів для аналізу даних у Analytics.jsx

Важливо зазначити, що структура компонента Analytics.jsx складається з кількох основних блоків, кожен з яких виконує окрему функцію в рамках аналізу фінансових даних. Перший блок — це набір кнопок вибору періоду, які надають користувачу можливість обрати потрібний часовий інтервал для аналізу транзакцій. Ці кнопки реалізовано у вигляді масиву даних, де кожен елемент відповідає певному періоду, такому як день, тиждень, два тижні, місяць або три місяці. Для генерації цих елементів використовується метод .map(), який проходить по масиву, створюючи кнопки з відповідними значеннями періодів. Кожна кнопка має власний обробник подій, який при натисканні змінює поточний період і запускає повторний рендеринг компонента. Це забезпечує інтерактивність та динамічність інтерфейсу, дозволяючи користувачу миттєво перемикатися між різними періодами та отримувати актуальні дані для обраного часового проміжку.

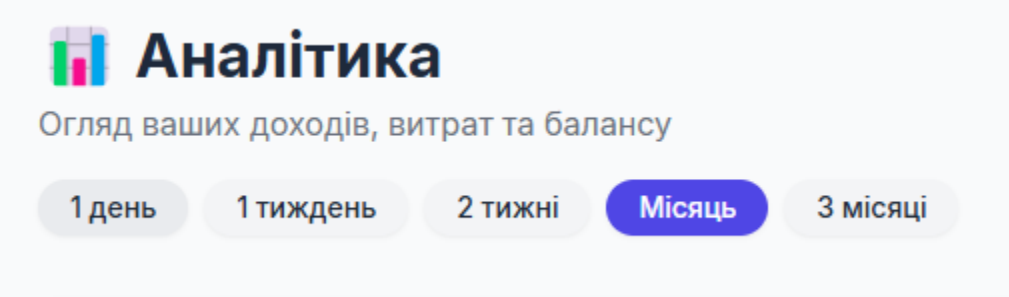


Рис. 2.15 Вибір періоду аналізу у вкладці "Аналітика"

Після вибору періоду дані автоматично фільтруються та передаються до наступних компонентів, які відповідають за їх візуалізацію, забезпечуючи наочне відображення фінансових показників у зрозумілому форматі. Цей процес відбувається за допомогою контексту TransactionContext, що дозволяє динамічно оновлювати дані у реальному часі. Користувач отримує актуальну інформацію про доходи, витрати та баланс за обраний період, що значно полегшує аналіз фінансової ситуації. Усі зміни автоматично синхронізуються з відповідними компонентами, що забезпечує консистентність даних у всьому інтерфейсі. Завдяки такому підходу, навіть при зміні періоду, користувач миттєво отримує оновлені фінансові дані без необхідності ручного перезавантаження сторінки.

Серед основних компонентів: ChartByCategory — цей компонент відіграє ключову роль у представленні витрат за категоріями, що дозволяє користувачеві швидко оцінити структуру своїх витрат за обраний період, визначити пріоритетні напрямки витрат та виявити потенційні зони надмірних витрат. Він приймає дані про транзакції, аналізує їх, формує масив категорій та обчислює суму витрат для кожної категорії, акцентуючи увагу на найзначніших витратах. Візуалізація цих даних здійснюється за допомогою Recharts PieChart, що дозволяє створити кругову діаграму з кольоровим розмежуванням категорій, де кожен сектор відображає відповідну категорію витрат, що забезпечує чітке та зручне сприйняття інформації про фінансові операції. Такий підхід сприяє кращому розумінню користувачем фінансової ситуації та прийняттю більш обґрунтованих рішень щодо управління витратами.

```
31    return (
32      <div className="bg-white rounded-xl p-6 shadow">
33        <h3 className="font-semibold mb-4">📊 Витрати за категоріями</h3>
34        <ResponsiveContainer width="100%" height={300}>
35          <PieChart>
36            <Pie
37              data={data}
38              dataKey="value"
39              nameKey="name"
40              cx="50%"
41              cy="50%"
42              outerRadius={100}
43              fill="#8884d8"
44              label
45            >
46              {data.map((_, index) => (
47                <Cell key={index} fill={COLORS[index % COLORS.length]} />
48              ))}
49            </Pie>
50            <Tooltip formatter={(value) => `${value.toFixed(2)} ₴`} />
51          </PieChart>
52        </ResponsiveContainer>
53      </div>
54    );
55  }
```

Рис. 2.16 Код ChartByCategory.jsx, який реалізує кругову діаграму

На круговій діаграмі кожна категорія має свій унікальний колір, що дозволяє користувачу не лише швидко ідентифікувати основні категорії витрат, але й оцінити їхню відносну вагу у загальній структурі витрат. Завдяки чіткому кольоровому розмежуванню, користувач може одразу звернути увагу на найбільш витратні категорії, виділивши ті, що займають найбільшу частку у загальному бюджеті. Наприклад, якщо певна категорія, така як «Розваги», має яскравий червоний колір та займає значну частину діаграми, це може свідчити про надмірні витрати в цьому сегменті. Відображення даних відбувається з використанням потужної бібліотеки Recharts, яка не лише забезпечує динамічну візуалізацію даних, але й додає інтерактивність графіка.

Завдяки цьому користувач може навести курсор на окремий сектор діаграми та отримати додаткову інформацію про суму витрат у певній категорії, що значно полегшує процес аналізу фінансових даних та прийняття рішень щодо їх оптимізації. Такий підхід робить компонент інтуїтивно зрозумілим та зручним для щоденного використання, забезпечуючи високий рівень візуалізації фінансової інформації.

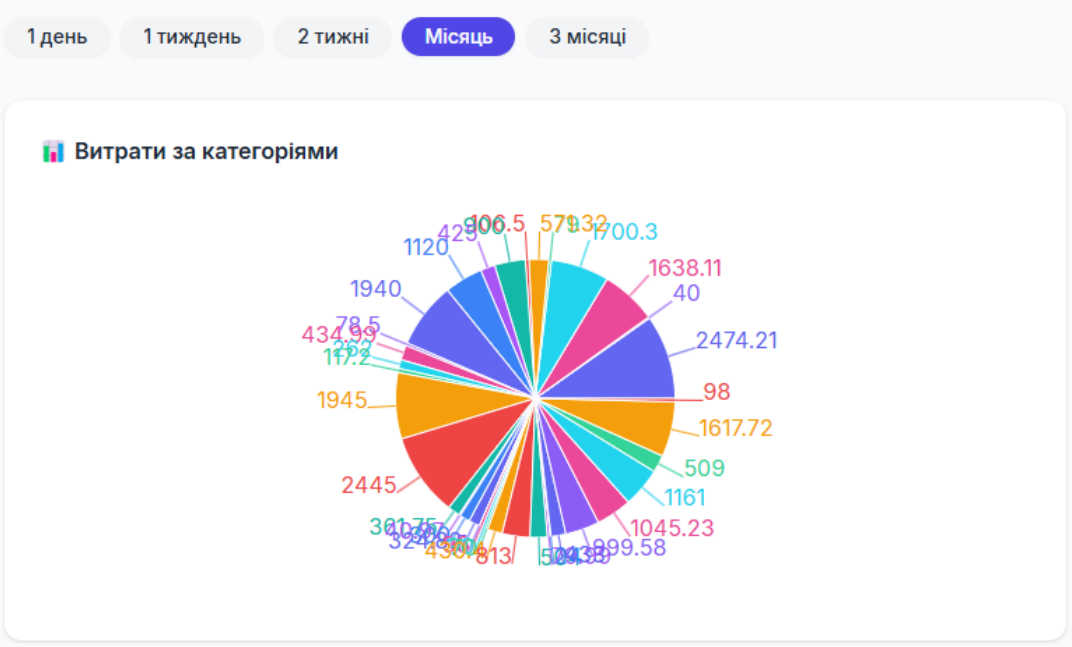


Рис. 2.17 Візуалізація витрат за категоріями у вигляді кругової діаграми

TopExpenses — наступний компонент, який відповідає за відображення найбільших витрат за обраний часовий проміжок. Логіка компонента побудована на сортуванні масиву транзакцій за сумою витрат, що дозволяє визначити найбільш вагомі фінансові операції за вказаний період. Компонент здійснює вибір кількох найбільших транзакцій, які мають найбільший вплив на загальний бюджет користувача, надаючи можливість акцентувати увагу на найбільш значущих витратах. Такий підхід допомагає користувачу вчасно ідентифікувати потенційні зони надмірних витрат та приймати обґрунтовані рішення щодо їх оптимізації. Це робить компонент не лише інформативним, але й важливим інструментом для контролю за фінансовими потоками.

```jsx
// Файл: src/pages/TopExpenses.jsx
export default function TopExpenses({ transactions }) {
  if (!Array.isArray(transactions)) {
    return (
      <div className="bg-white rounded-xl p-6 shadow text-gray-500 text-center">
        Немає даних для відображення.
      </div>
    );
  }

  const expenses = transactions
    .filter(tx => tx.amount < 0)
    .sort((a, b) => a.amount - b.amount)
    .slice(0, 5);

  if (expenses.length === 0) {
    return (
      <div className="bg-white rounded-xl p-6 shadow text-gray-500 text-center">
        Немає витрат.
      </div>
    );
  }

  return (
    <div className="bg-white rounded-xl p-6 shadow">
      <h3 className="font-semibold mb-2">🔥 Топ витрат</h3>
      <ul className="text-sm space-y-1">
        {expenses.map(tx => (
          <li key={tx.id} className="flex justify-between border-b border-gray-100 py-1">
            <span className="truncate max-w-[200px]">{tx.description || "Невідомо"}</span>
            <span className="text-red-500">{(tx.amount / 100).toFixed(2)} ₴</span>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

Рис. 2.18 Код TopExpenses.jsx, відповідає за відображення топових витрат

У інтерфейсі компонент TopExpenses відображається у вигляді вертикального списку, який містить чітко структуровану інформацію про транзакції, включаючи категорію витрат, суму та дату операції. Такий формат відображення створює візуальний акцент на найбільших витратах, що дозволяє користувачу швидко ідентифікувати ключові фінансові операції за обраний період та оцінити їхній вплив на загальний бюджет. Крім того, подібна структура списку сприяє зручному перегляду даних, адже користувач одразу може побачити найбільші транзакції зверху, що полегшує процес аналізу фінансових витрат та прийняття рішень щодо подальших дій.

Важливо зазначити, що такий підхід забезпечує не лише оперативний доступ до основних витрат, але й дозволяє порівнювати їх між собою, що особливо корисно для виявлення потенційних зон перевитрат та оптимізації бюджету. Компонент TopExpenses виступає в ролі ефективного аналітичного інструменту, який допомагає користувачеві краще розуміти власні фінансові потоки, своєчасно помічати надмірні витрати та приймати більш обґрунтовані фінансові рішення.

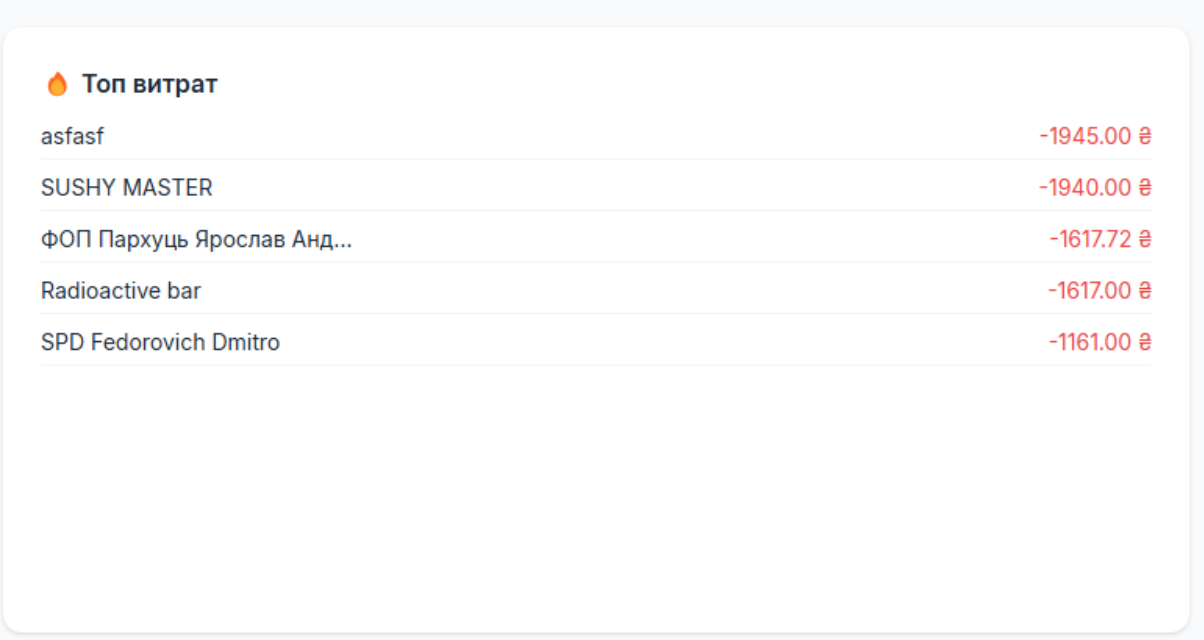


Рис. 2.19 Виведення списку топових витрат за обраний період

BalanceProgressChart — компонент, що відповідає за відображення прогресу досягнення фінансової цілі. Він приймає масив транзакцій, обчислює загальну суму доходів та витрат та формує графік у вигляді лінійної діаграми. Це дозволяє користувачу оцінити, наскільки близько він знаходиться до досягнення поставленої фінансової цілі.

```jsx
// src/components/BalanceProgressChart.jsx
import {
  ResponsiveContainer,
  LineChart,
  Line,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
} from "recharts";
import { BarChart3 } from "lucide-react";

export default function BalanceProgressChart({ transactions }) {
  const days = Array.from({ length: 14 }, (_, i) => {
    const date = new Date();
    date.setDate(date.getDate() - (13 - i));
    return date.toLocaleDateString();
  });

  const data = days.map((day) => {
    const dayTransactions = transactions.filter(
      (t) => new Date(t.time * 1000).toLocaleDateString() === day
    );
    const income = dayTransactions
      .filter((t) => t.amount > 0)
      .reduce((acc, t) => acc + t.amount, 0);
    const expense = dayTransactions
      .filter((t) => t.amount < 0)
      .reduce((acc, t) => acc + t.amount, 0);
    const balance = income + expense;
    return {
      name: day.slice(0, 5),
      balance: balance / 100,
    };
  });

  return (
    <div className="bg-white rounded-xl shadow p-5">
      <div className="flex items-center gap-2 mb-4">
        <BarChart3 className="text-blue-500" />
        <h3 className="text-lg font-semibold">Прогрес балансу за 14 днів</h3>
      </div>
      <ResponsiveContainer width="100%" height={200}>
        <LineChart data={data}>
          <CartesianGrid strokeDasharray="3 3" />
          <XAxis dataKey="name" />
          <YAxis />
          <Tooltip />
          <Line
            type="monotone"
            dataKey="balance"
            stroke="#6366f1"
            strokeWidth={2}
            dot={false}
          />
        </LineChart>
      </ResponsiveContainer>
    </div>
  );
}
```

Рис. 2.20 Код BalanceProgressChart.jsx, що відповідає за побудову графіка

Компонент StatsSummary виконує функцію підсумкового блоку, що відображає загальні фінансові показники за обраний період і завершує структуру сторінки аналітики. У компоненті реалізовано відображення загальної суми доходів, витрат та балансу, що дозволяє користувачу отримати загальну картину фінансового стану.

```jsx
// src/components/StatsSummary.jsx
import { PackageCheck } from "lucide-react";

export default function StatsSummary({ transactions }) {
  const total = transactions.length;
  const totalIncome = transactions
    .filter((t) => t.amount > 0)
    .reduce((acc, t) => acc + t.amount, 0);

  const totalExpense = transactions
    .filter((t) => t.amount < 0)
    .reduce((acc, t) => acc + t.amount, 0);

  const avgIncome = totalIncome / transactions.filter((t) => t.amount > 0).length || 0;
  const avgExpense = totalExpense / transactions.filter((t) => t.amount < 0).length || 0;

  const lastTransaction = transactions[0]
    ? new Date(transactions[0].time * 1000).toLocaleDateString()
    : "—";

  const stats = [
    { label: "Кількість транзакцій", value: total },
    { label: "Середній дохід", value: (avgIncome / 100).toFixed(2) + " ₴" },
    { label: "Середня витрата", value: (Math.abs(avgExpense) / 100).toFixed(2) + " ₴" },
    { label: "Остання транзакція", value: lastTransaction },
  ];

  return (
    <div className="bg-white rounded-xl shadow p-6">
      <div className="flex items-center gap-2 mb-4">
        <PackageCheck className="text-yellow-500" />
        <h3 className="text-lg font-semibold">Загальна статистика</h3>
      </div>
      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-4 gap-4 text-sm">
        {stats.map((stat, idx) => (
          <div
            key={idx}
            className="bg-gray-50 border rounded-xl px-4 py-3 shadow-sm text-center"
          >
            <p className="text-gray-500">{stat.label}</p>
            <p className="font-semibold text-lg">{stat.value}</p>
          </div>
        ))}
      </div>
    </div>
  );
}
```

Рис. 2.21 Код який обчислює та відображає загальну статистику

Візуальне оформлення компонента Analytics.jsx відповідає загальній стилістиці додатку, використовуючи спільні кольори, шрифти та елементи інтерфейсу. Це дозволяє зберегти єдність дизайну та створити узгоджену структуру для перегляду фінансових даних у динаміці.

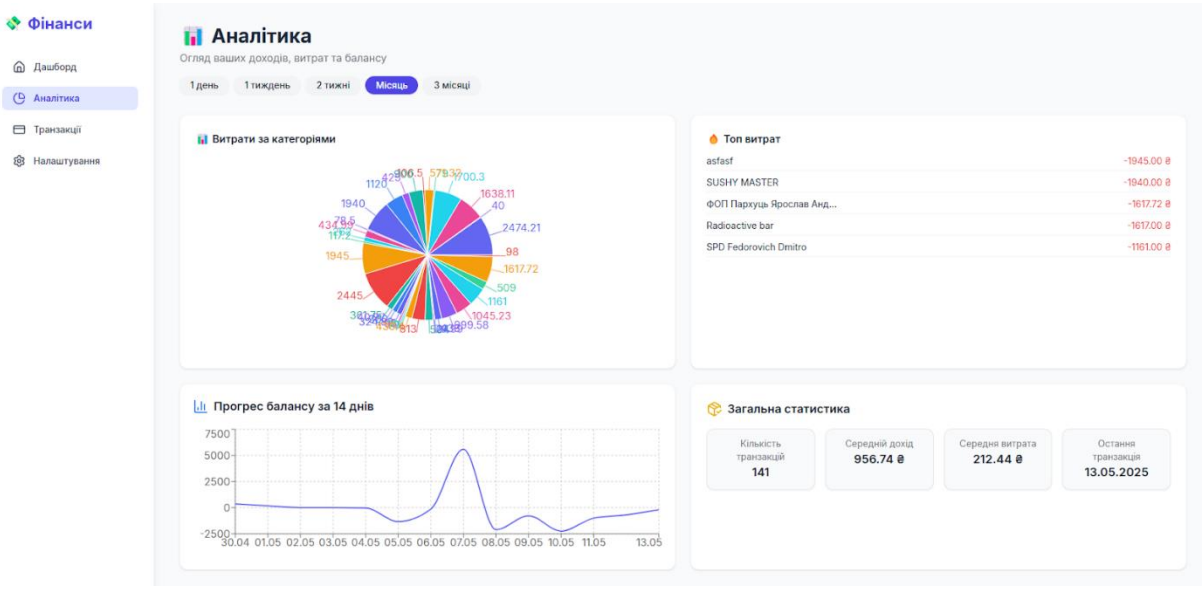


Рис. 2.22 Загальний вигляд вкладки "Аналітика" з усіма секціями

Компонент Analytics.jsx виступає в якості централізованого інструменту для огляду та аналізу фінансових даних за обраний часовий проміжок. Він забезпечує інтерактивну візуалізацію даних за допомогою графіків, списків та діаграм, що дозволяє користувачу детально розглянути динаміку витрат, доходів та зміни балансу. Це особливо корисно для тих, хто прагне контролювати свої фінансові потоки, відстежувати виконання запланованих цілей та вчасно виявляти можливі фінансові ризики. Окрім цього, компонент дозволяє здійснювати порівняльний аналіз даних за різні періоди, що робить його важливим елементом для всебічного контролю за фінансовими операціями. Analytics.jsx не лише робить вебдодаток інформативним, але й значно підвищує його зручність у використанні, сприяючи глибшому розумінню фінансових показників та допомагаючи користувачам приймати більш обґрунтовані рішення.

Компонент Transactions.jsx реалізовує функціональність перегляду всіх транзакцій користувача у вигляді таблиці. Цей компонент інтегрує дані з контексту TransactionContext, що дозволяє отримувати актуальний масив транзакцій та фільтрувати їх за вибраними параметрами. Компонент призначено для надання користувачу функціональності перегляду всіх транзакцій за обраний період, із можливістю їх сортування та пошуку. На початку компонента виконується імпорт бібліотек та необхідних компонентів, включаючи TransactionContext, MainLayout.

```
1  import { useContext } from "react";
2  import MainLayout from "../layouts/MainLayout";
3  import { TransactionContext } from "../context/TransactionContext";
```

Рис. 2.23 Імпорти необхідних модулів та контексту

Після імпорту відбувається ініціалізація основних станів компонента за допомогою хуку useContext, який отримує дані про транзакції та їхній статус завантаження з TransactionContext. Цей підхід дозволяє централізовано керувати всіма даними про фінансові операції, забезпечуючи узгодженість інформації між різними компонентами додатку. Використання хуку useContext спрощує доступ до даних у будь-якому компоненті, що мінімізує кількість необхідних пропсів та зменшує обсяг коду. Крім того, у випадку зміни даних у контексті, усі компоненти, які використовують цей контекст, автоматично оновлюються, що забезпечує актуальність інформації у реальному часі без додаткових запитів до сервера.

```
5  export default function Transactions() {
6    const { transactions, loading } = useContext(TransactionContext);
```

Рис. 2.24 Використання контексту для отримання транзакцій

Основну частину компонента становить таблиця з транзакціями. Вона складається з кількох колонок: Дата, Сума, Опис. Кожен рядок таблиці рендериться за допомогою методу .map(), який приймає дані транзакції та форматує їх для відображення.

```
<table className="w-full text-left border">
  <thead>
    <tr className="bg-gray-100">
      <th className="p-2">Дата</th>
      <th className="p-2">Сума</th>
      <th className="p-2">Опис</th>
    </tr>
  </thead>
  <tbody>
    {transactions.map((tx) => (
      <tr key={tx.id} className="border-t">
        <td className="p-2">{new Date(tx.time * 1000).toLocaleDateString()}</td>
        <td className="p-2 ${tx.amount < 0 ? "text-red-500" : "text-green-600"}`}>
          {(tx.amount / 100).toFixed(2)} ₴
        </td>
        <td className="p-2">{tx.description}</td>
      </tr>
    ))}
  </tbody>
</table>
```

Рис. 2.25 Логіка рендерингу таблиці транзакцій.

Для більшої зручності користувача реалізовано можливість сортування транзакцій за сумою або датою. Для цього використовується метод .sort(), який сортує масив транзакцій у зростаючому або спадному порядку, залежно від обраного параметра сортування. Це дозволяє користувачу швидко отримувати доступ до найбільш важливих транзакцій, фокусуючись на найбільших витратах або останніх операціях. Крім того, у компоненті реалізовано можливість перемикання між різними режимами сортування — за зростанням або спаданням, що забезпечує більш гнучкий підхід до аналізу фінансових даних. Таке рішення сприяє покращенню користувацького досвіду, оскільки надає можливість миттєво змінювати порядок відображення транзакцій без необхідності перезавантаження сторінки. Додатково реалізовано візуальне відображення активного режиму сортування за допомогою іконок,

```
<tr key={tx.id} className="border-t">
  <td className="p-2">{new Date(tx.time * 1000).toLocaleDateString()}</td>
  <td className="p-2 ${tx.amount < 0 ? "text-red-500" : "text-green-600"}`}>
    {(tx.amount / 100).toFixed(2)} ₴
  </td>
  <td className="p-2">{tx.description}</td>
</tr>
```

Рис. 2.26 Рендеринг суми транзакцій із стилям

Інтерфейс компонента Transactions.jsx відповідає загальній стилістиці додатку, включаючи колірні рішення та структуру таблиці. Це забезпечує узгодженість дизайну та покращує загальний користувацький досвід.

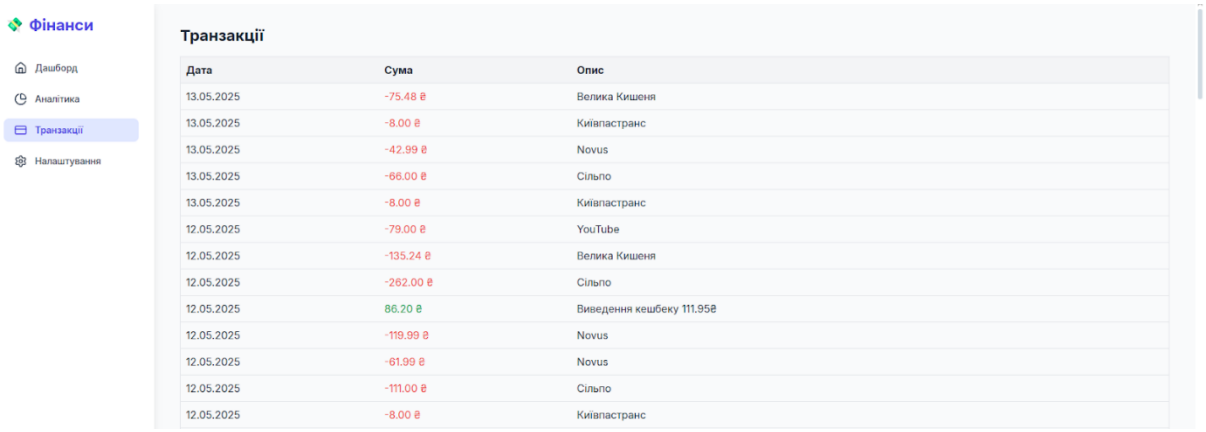


Рис. 2.27 Інтерфейс таблиці транзакцій

Компонент Transactions.jsx інтегрує декілька важливих функцій: відображення усіх транзакцій за обраний період та їх форматування у вигляді таблиці з можливістю сортування даних. Це дозволяє користувачу легко аналізувати фінансові операції та зберігати контроль над своїми витратами. Фінансова інформація користувача у вебдодатку отримується за допомогою інтеграції з API Monobank. Автоматичне оновлення даних про доходи, витрати та баланс слугує основою для роботи ключових компонентів, таких як Dashboard.jsx, Analytics.jsx та Transactions.jsx.

Функція fetchAllTransactions у файлі monobank.js відповідає за інтеграцію з API Monobank та отримання даних про транзакції користувача за останні 31 день. Вона використовується для збору фінансових даних, які потім відображаються у таких компонентах, як Dashboard.jsx, Analytics.jsx та Transactions.jsx. У процесі виконання запиту функція також обробляє можливі помилки, забезпечуючи стабільність додатку навіть у випадку відсутності з'єднання з API. Це дозволяє гарантувати коректне завантаження даних та їх подальшу обробку у компонентах інтерфейсу.

На початку функції відбувається запит до API Monobank для отримання інформації про акаунт користувача. Цей запит надсилається на /personal/client-info з використанням заголовка X-Token, що містить токен доступу користувача. Якщо запит неуспішний або акаунт не знайдено, функція повертає відповідне повідомлення про помилку, яке зберігається у стані компонента для подальшого відображення користувачу. Додатково передбачено обробку помилок з боку сервера, що дозволяє відобразити користувачу більш інформативне сповіщення про можливі технічні проблеми або некоректний токен доступу.

```
3   export async function fetchAllTransactions(token) {
4     const API_URL = "https://api.monobank.ua/personal/statement";
5
6     // Получение ID аккаунта
7     const accountsRes = await fetch("https://api.monobank.ua/personal/client-info", {
8       headers: { "X-Token": token },
9     });
```

Рис. 2.28 Запит до Monobank API – оголошення URL

Після успішного отримання ID акаунта, формується запит до API Monobank для отримання транзакцій. В якості інтервалу використовується поточний час та значення 31 день тому, що дозволяє охопити останній місяць фінансових операцій користувача. Цей запит дозволяє завантажити усі транзакції за останні 31 день, включаючи витрати, надходження та інші фінансові операції. У разі отримання від API некоректної відповіді або даних, що не відповідають структурі масиву транзакцій, функція повертає порожній масив, запобігаючи помилкам у компонентах, які працюють із очікуваним форматом даних.

```
11    if (!accountsRes.ok) throw new Error("Не вдалося отримати акаунт");
12
13    const accountsData = await accountsRes.json();
14    const accountId = accountsData.accounts[0]?.id;
15
16    if (!accountId) throw new Error("Акаунт не знайдено");
17
```

Рис. 2.29 Обробка помилок Monobank API

Дані, отримані від API, обробляються шляхом перетворення масиву транзакцій у формат, що відповідає структурі програми. Кожна транзакція містить такі поля, як id, time, description та amount. Ці дані потім використовуються для візуалізації у компонентах Dashboard.jsx, Analytics.jsx та Transactions.jsx.

```
19  const now = Math.floor(Date.now() / 1000);
20  const from = now - 60 * 60 * 24 * 31; // 31 день
21
22  const url = `${API_URL}/${accountId}/${from}/${now}`;
23  const res = await fetch(url, {
24    headers: { "X-Token": token },
25  });
26
```

Рис. 2.30 Формування запиту транзакцій

Функцію fetchAllTransactions визначено як ключовий елемент інтеграції з API Monobank, що забезпечує централізований доступ до даних про фінансові операції користувача. Вона дозволяє отримувати інформацію про всі транзакції за останні 31 день, обробляти ці дані та використовувати їх у різних компонентах вебдодатку для подальшого аналізу, візуалізації та формування фінансових звітів. Завдяки цьому, додаток може не лише відображати актуальні дані, але й динамічно оновлювати інформацію у реальному часі, що підвищує інформативність інтерфейсу та забезпечує зручність для користувача.

Адаптивність інтерфейсу становить один із ключових елементів сучасних вебдодатків, оскільки користувачі можуть взаємодіяти з додатком на різних пристроях — від смартфонів до десктопів. Вебдодаток для управління фінансами не становить винятку, реалізуючи адаптивний дизайн, що забезпечує коректне відображення контенту незалежно від розміру екрана. Tailwind CSS та вбудовані медіа-запити дозволяють налаштовувати макети для різних типів пристроїв, зберігаючи єдину стилістику та логіку відображення даних.

2.3 Адаптивність та оптимізація інтерфейсу

Для реалізації адаптивності використовуються медіа-запити (media queries), які дозволяють змінювати структуру та стилі компонентів відповідно до ширини екрану. У проекті застосовуються такі ключові точки переходу: 640px (мобільні пристрої), 768px (планшети) та 1024px (десктопи). Це дозволяє адаптувати інтерфейс для користувачів з різними пристроями, забезпечуючи коректне відображення контенту незалежно від розміру екрану. Крім того, такий підхід сприяє підтримці єдиної стилістики та зручної навігації на всіх етапах взаємодії з

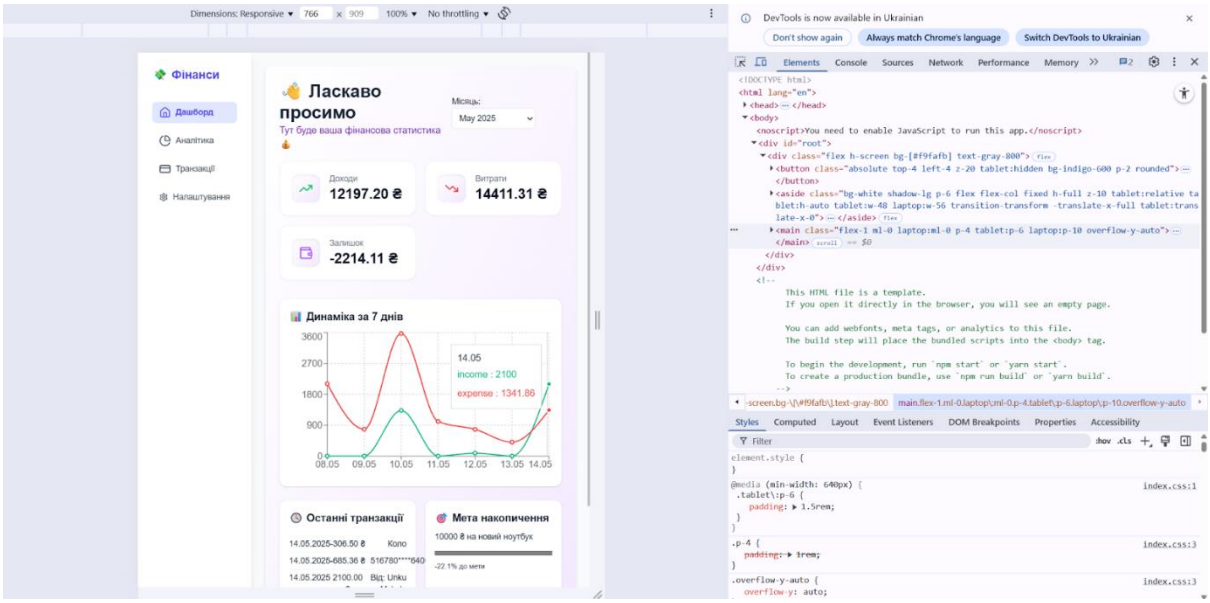


Рис. 2.31 Адаптив для ширини 766px

Під час розробки адаптивного інтерфейсу використовується метод flexbox для управління розташуванням компонентів. Це забезпечує динамічну зміну розмірів блоків залежно від ширини екрану. Наприклад, на головному екрані Dashboard.jsx блок з фінансовими картками (Доходи, Витрати, Залишок) реалізований як гнучкий контейнер із класами grid grid-cols-1 sm:grid-cols-3. Це означає, що на мобільних пристроях картки розташовуються вертикально, а на планшетах та десктопах — горизонтально. Такий підхід дозволяє зберігати структуру макету та забезпечувати користувачу єдиний візуальний досвід незалежно від розміру екрану.

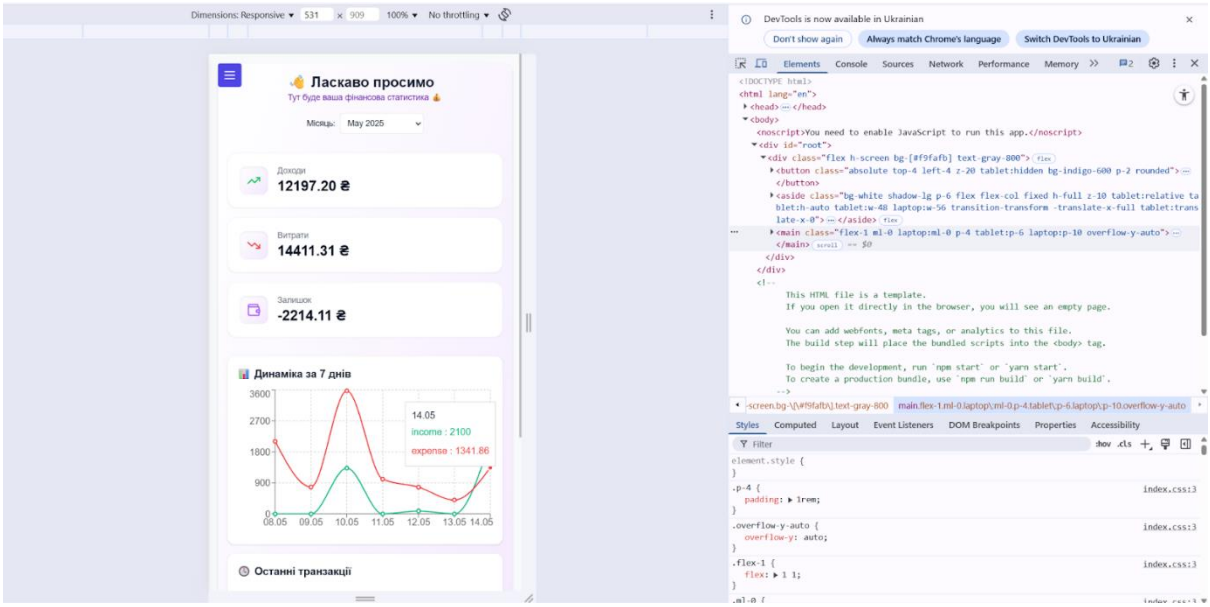


Рис. 2.32 Адаптив для ширини 531px

Адаптивна верстка передбачає також коректну зміну розміру графіків. В інтерфейсі використовуються компоненти з бібліотеки Recharts, такі як LineChart та PieChart, які автоматично підлаштовуються під ширину контейнера завдяки використанню ResponsiveContainer. Це дозволяє зберегти пропорційність графіків та забезпечити їх читабельність навіть на невеликих екранах. Додатково, налаштування мінімальної та максимальної ширини контейнера дозволяє уникнути спотворення графіків при зміні розміру вікна браузера.

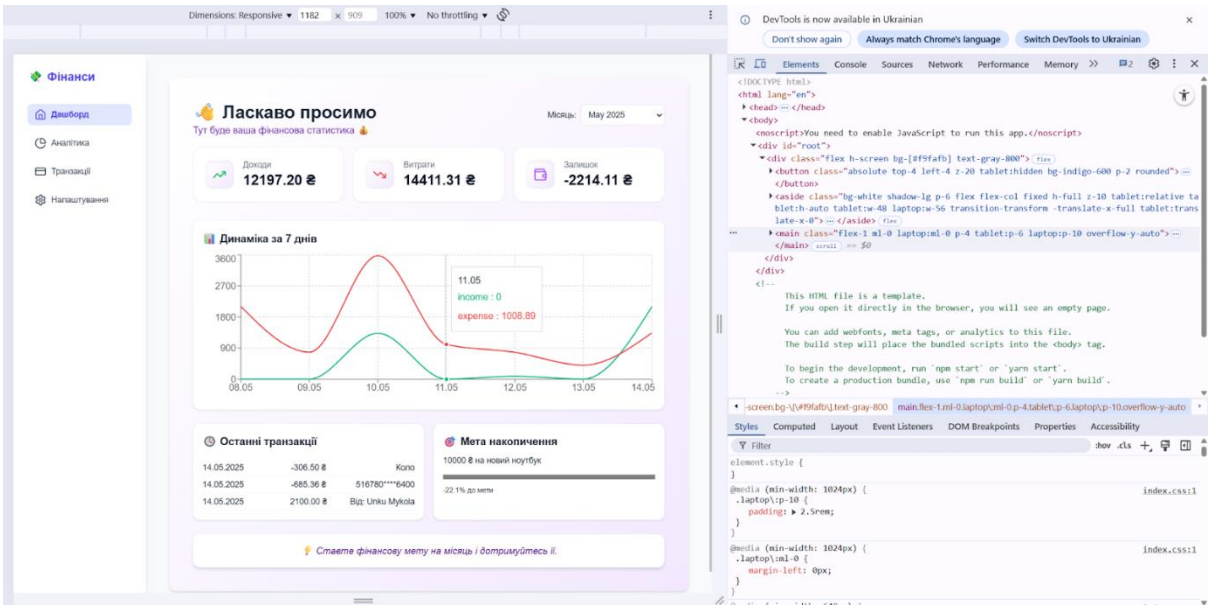


Рис. 2.33 Адаптив для ширини 1182px

Крім того, при адаптації інтерфейсу важливо враховувати відступи, розміри шрифтів та розташування елементів. Наприклад, кнопки вибору періоду в Analytics.jsx мають різні стилі залежно від розміру екрану. На мобільних пристроях кнопки розташовуються вертикально, а на планшетах та десктопах — горизонтально, що реалізовано за допомогою класів flex та flex-wrap.

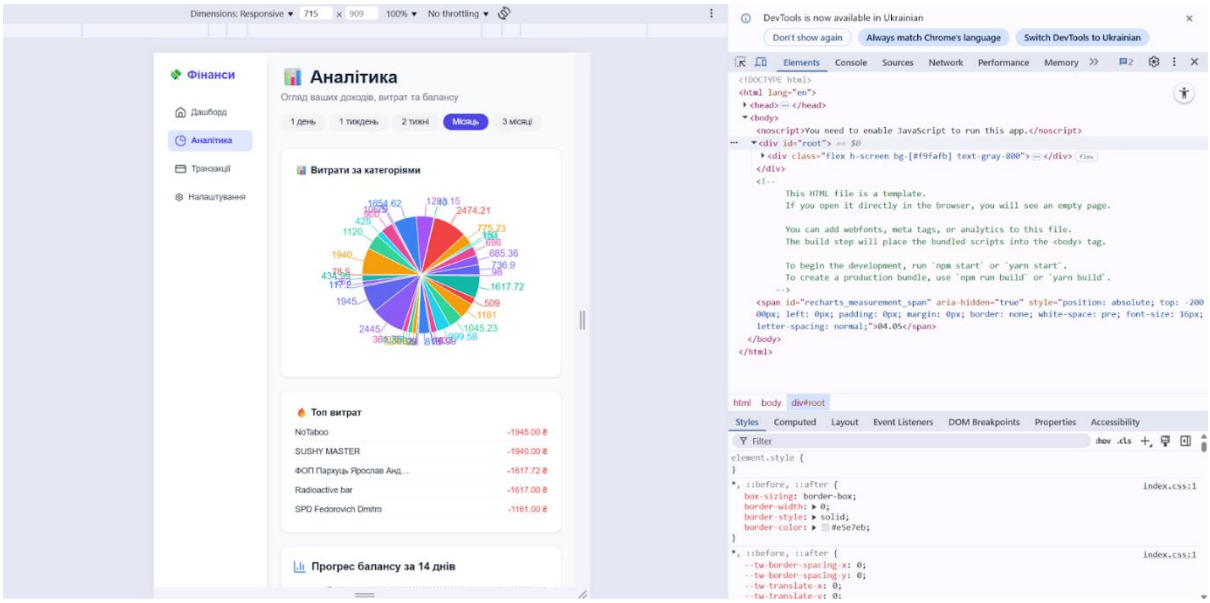


Рис. 2.34 Адаптив сторінки Analytics.jsx 715px

Використання DevTools у браузері дозволяє не лише перевіряти адаптивність інтерфейсу, але й відстежувати, які стилі застосовуються до компонентів у різних режимах перегляду. Це особливо важливо для налагодження адаптивних стилів та усунення можливих колізій між різними екранами. У процесі тестування використовуються вікна з різними розмірами: 375px (iPhone X), 768px (iPad), 1024px (десктоп).

Реалізація адаптивності у вебдодатку забезпечує зручність використання інтерфейсу на різних пристроях та сприяє поліпшенню користувацького досвіду, дозволяючи динамічно змінювати структуру та стилі компонентів залежно від розміру екрану.

3. ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ РОБОТИ ВЕБ-ДОДАТКУ ДЛЯ ФІНАНСОВОГО МЕНЕДЖМЕНТУ

3.1 Тестування адаптивності веб-додатку

Функціональність веб-додатку для фінансового менеджменту становить ключовий аспект, адже саме через основні компоненти користувач отримує доступ до фінансових даних, аналітики та статистики. У цьому пункті здійснено комплексне тестування компонентів Dashboard.jsx, Analytics.jsx та Transactions.jsx для перевірки їхньої коректної роботи, інтеграції з Monobank API, а також відображення та обробки даних у інтерфейсі додатку. Основна мета тестування полягала у виявленні можливих помилок у логіці роботи компонентів, перевірці коректності рендерингу інтерфейсу та аналізі відповідності відображених даних фактичним значенням, отриманим із API.

На початковому етапі тестування основну увагу зосереджено на компоненті Dashboard.jsx, який слугує стартовою сторінкою додатку та містить ключові фінансові показники користувача: доходи, витрати та залишок. У процесі тестування використовувалися як стандартні набори транзакцій з позитивними сумами доходів та негативними витратами, так і випадки, де відсутні будь-які транзакції. Для перевірки коректності роботи з API використовувалася функція fetchAllTransactions() з monobank.js, яка здійснює запит до API Monobank для отримання даних за останні 31 день. Основна увага була приділена правильному форматуванню дат, обробці сум транзакцій та валідації описів.

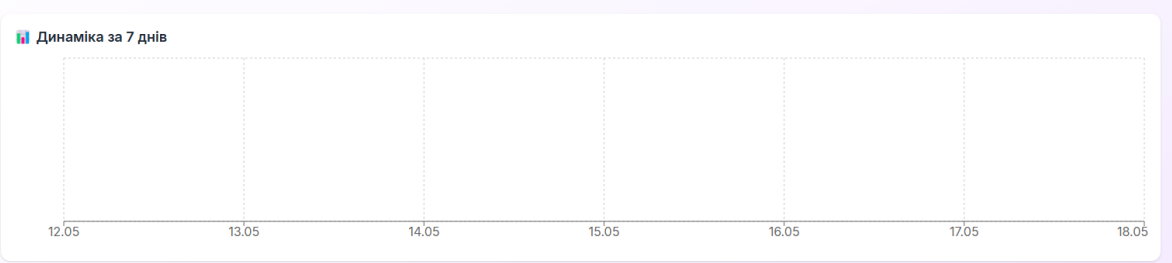


Рис. 3.1 Порожній графік динаміки за 7 днів

Для формування графіку використовується функція calculateWeeklyData(), яка обчислює суми доходів та витрат за останні 7 днів та формує масив даних для відображення у LineChart. У процесі тестування здійснювалася симуляція транзакцій з однаковими датами для перевірки коректності групування сум за днями. Важливо було переконатися, що дані на графіку оновлюються динамічно та відповідають фактичним значенням транзакцій. Наприклад, при завантаженні нових транзакцій дані у графіку повинні змінюватися автоматично.

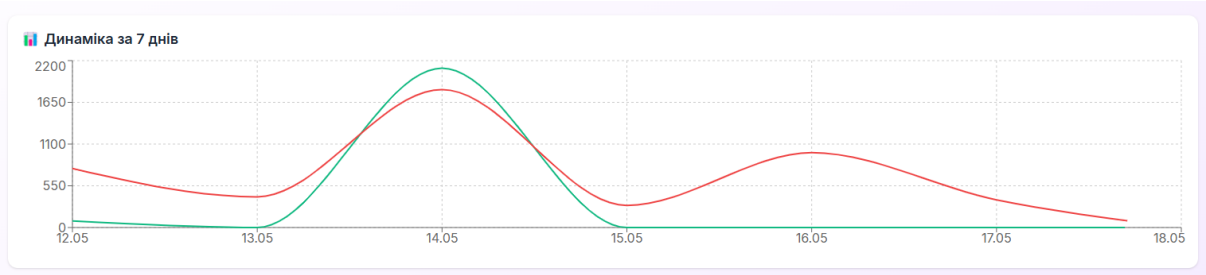


Рис 3.2 Заповнений графік динаміки за 7 днів

Наступним етапом тестування було відображення фінансових карток у Dashboard.jsx, які демонструють доходи, витрати та залишок. Кожна картка формує своє значення на основі даних, отриманих з API Monobank. Тестування включало сценарії, де сума доходів дорівнює нулю, витрати мають від'ємні значення, а залишок формується як різниця між доходами та витратами. У таких випадках особлива увага приділялася форматуванню суми у форматі toFixed(2), що забезпечує коректне відображення значень до двох десяткових знаків.

```
value={(income / 100).toFixed(2) + " ₴"} icon={<TrendingUp className="text-green-500" />} />
' value={(Math.abs(expenses) / 100).toFixed(2) + " ₴"} icon={<TrendingDown className="text-red-500" />} />
' value={(balance / 100).toFixed(2) + " ₴"} icon={<Wallet className="text-purple-500" />} />
```

Рис. 3.3 Код фінансових показників

Перехід до компонента Analytics.jsx дозволив зосередитися на аналізі транзакцій за різні періоди часу. Використовуючи набір періодів, реалізованих у PERIODS, користувач може обирати періоди від одного дня до трьох місяців.

```
<div className="mt-4 flex gap-2 flex-wrap">
  {Object.keys(PERIODS).map((key) => (
    <button
      key={key}
      onClick={() => setPeriod(key)}
      className={`px-4 py-1 rounded-full text-sm font-medium shadow-sm transition ${
        period === key
          ? "bg-indigo-600 text-white"
          : "bg-gray-100 hover:bg-gray-200"
      }`}
    >
      {
        {
          day: "1 день",
          week: "1 тиждень",
          twoWeeks: "2 тижні",
          month: "Місяць",
          threeMonths: "3 місяці",
        }[key]
      }
    </button>
  ))}
</div>
</div>
```

Рис 3.4 Код вибору періоду аналізу

У компоненті Analytics.jsx також здійснювалася перевірка рендерингу графіків витрат за категоріями. Для цього використовувався компонент ChartByCategory.jsx, який формує кругову діаграму витрат за категоріями, відображаючи їх у кольорових секторах. Тестування включало перевірку коректності розподілу секторів за категоріями, а також відповідність кольорових індикаторів обраним категоріям. В окремих тестових випадках перевірялося коректне відображення секторів для категорій з дуже малими значеннями, щоб визначити, чи не зникають такі категорії під час рендерингу. Додатково тестувалося оновлення даних у ChartByCategory.jsx після зміни вибраного періоду, що дозволяло оцінити динамічне оновлення кольорових секторів та їх пропорцій. Особлива увага приділялася випадкам, коли одна категорія займає понад 50% загальних витрат, що могло призвести до перекриття секторів або неправильного відображення даних. Було проведено тестування поведінки компонента при наявності декількох категорій з однаковими значеннями .Також перевірялася коректність масштабування діаграми при додаванні нових категорій

```
export default function ChartByCategory({ transactions }) {
  const expenses = transactions?.filter(tx => tx.amount < 0) || [];

  const categorySums = expenses.reduce((acc, tx) => {
    const category = tx.description || "Інше";
    acc[category] = (acc[category] || 0) + tx.amount;
    return acc;
  }, {});

  const data = Object.entries(categorySums).map(([key, value]) => ({
    name: key,
    value: Math.abs(value / 100),
  }));

  if (!data.length) {
    return (
      <div className="bg-white rounded-xl p-6 shadow text-center text-gray-500">
        Немає витрат для відображення.
      </div>
    );
  }
```

Рис. 3.5 Код компонента ChartByCategory.jsx

У компоненті Analytics.jsx основна увага була зосереджена на перевірці рендерингу графіків за різні періоди часу. Використовуючи набір періодів, реалізованих у PERIODS, користувач має змогу обирати періоди від одного дня до трьох місяців. У процесі тестування перевірялися сценарії зі збільшеною кількістю транзакцій, що дозволило проаналізувати поведінку графіка при великій кількості даних.

Логіка формування графіка у Analytics.jsx базується на функції calculatePeriodData(), яка обчислює загальну суму доходів та витрат за обраний період. Основний акцент під час тестування було зроблено на перевірці відповідності значень на графіку фактичним сумам транзакцій. Було виявлено, що при завантаженні великої кількості даних за тримісячний період графік міг частково втрачати коректне відображення осі X. Це було виправлено шляхом оптимізації структури масиву даних та додавання проміжних значень для днів, за які транзакції були відсутні.

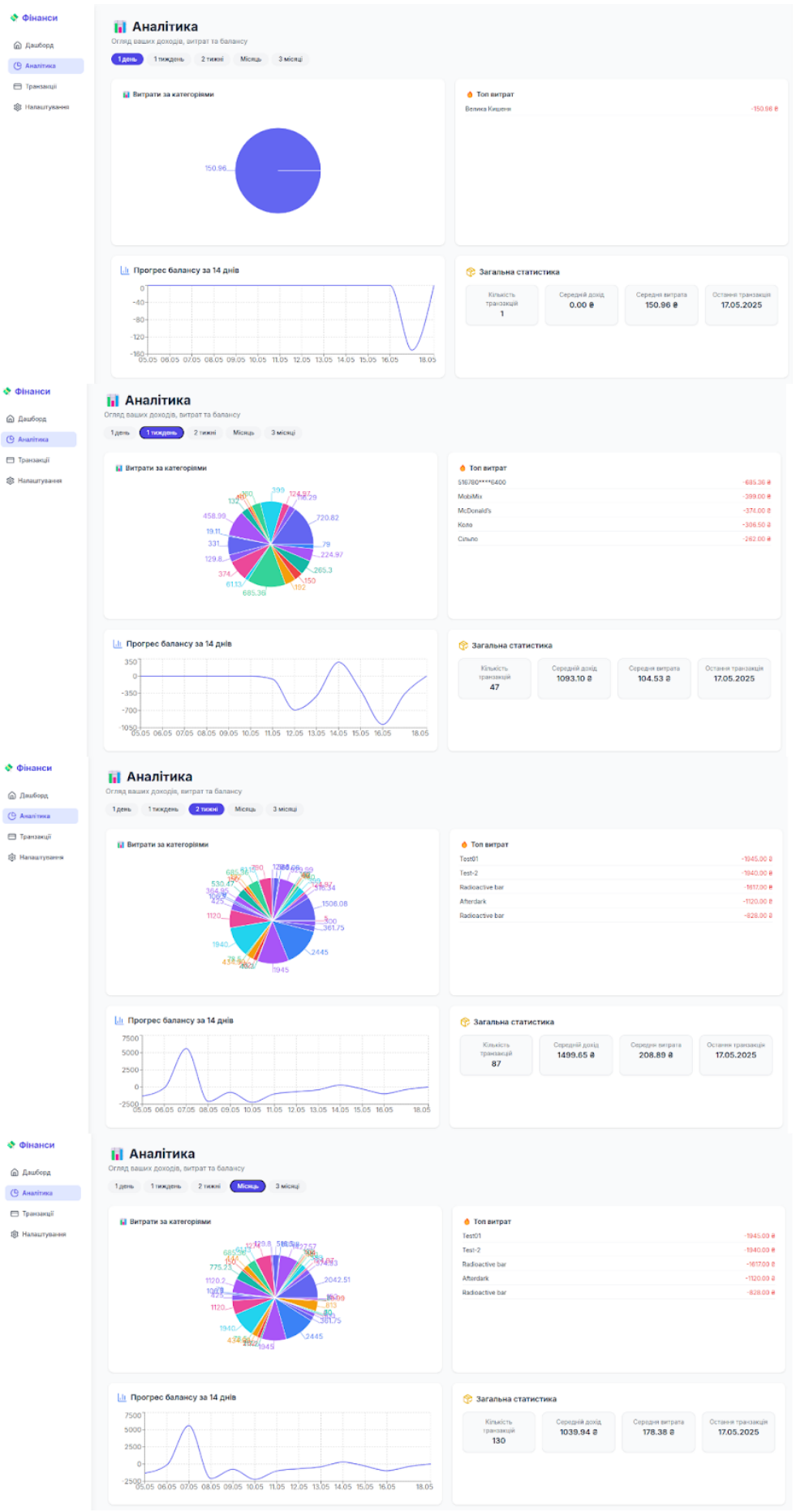


Рис 3.6 Інтерфейс вкладки "Аналітика"

Компонент Transactions.jsx відповідає за відображення списку транзакцій, що формуються на основі даних, отриманих із API. Основна увага була зосереджена на коректності форматування дати транзакцій, сум та описів. У тестовому наборі даних використовувалися транзакції з однаковими сумами, різними датами та порожніми описами, щоб перевірити, як компонент обробляє такі випадки.

Було перевірено, що кожен запис у таблиці відображає дату у форматі DD/MM/YYYY, суми доходів виділені зеленим кольором, а витрати — червоним. У разі відсутності транзакцій має з'являтися відповідне повідомлення. Додатково перевірялася коректність відображення даних у таблиці при зменшенні ширини екрану, коли кількість колонок перевищує розмір вікна браузера.

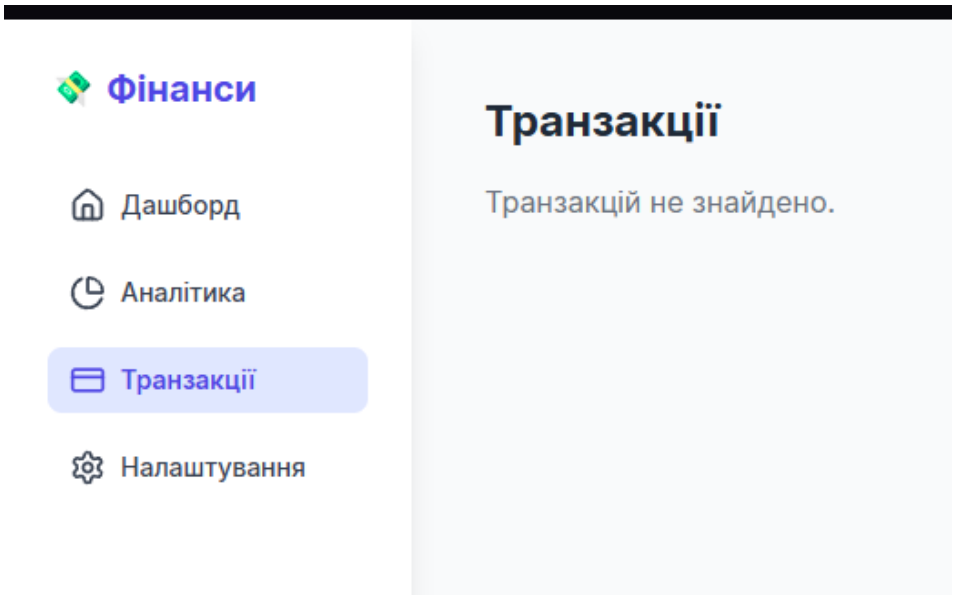


Рис 3.7 Порожній список транзакцій

Тестування компонентів веб-додатку підтвердило коректну обробку даних, отриманих з Monobank API, оновлення інтерфейсу у реальному часі при зміні даних та стабільне відображення графіків та таблиць незалежно від кількості транзакцій. Виявлені помилки були виправлені, зокрема у компоненті Analytics.jsx було скориговано відображення графіка при великій кількості даних, а у Transactions.jsx — реалізовано горизонтальний скролінг для збереження цілісності таблиці на мобільних пристроях.

3.2 Тестування інтеграції з Monobank API

Інтеграцію Monobank API визначено як одну з ключових складових функціональності веб-додатку для фінансового менеджменту, що забезпечує отримання актуальних даних про транзакції користувача у режимі реального часу. Реалізація такого механізму дозволяє автоматизувати процес оновлення фінансової інформації та підтримувати постійну актуальність даних у всіх основних компонентах додатку. Завдяки прямому підключенню до API банківської системи користувач має змогу переглядати свої останні операції без необхідності ручного введення або оновлення інформації, що значно підвищує зручність та ефективність взаємодії з додатком.

У процесі тестування особливу увагу зосереджено на перевірці коректності виконання HTTP-запитів до Monobank API, зокрема оцінювалася правильність авторизації за допомогою особистого токена користувача. Тестування охоплювало сценарії, пов’язані з обробкою стандартних та виняткових ситуацій, таких як перевищення ліміту запитів, недоступність сервісу чи помилкові відповіді з боку API. Для кожного з випадків здійснювалося логічне відстеження обробки помилки, з метою уникнення збоїв у роботі додатку та підтримки стабільності його функціонування у реальному середовищі.

Окремо протестовано ситуації, у яких API повертає пустий масив транзакцій — зокрема, у разі відсутності нових фінансових операцій у вибраний період. Перевірка таких випадків дозволила переконатися у стійкості логіки додатку до нестандартних або крайових умов, коли стандартний потік даних відсутній, але система має працювати без збоїв або критичних повідомлень.

Центральною функцією, відповідальною за роботу з Monobank API, виступає fetchAllTransactions(), яка реалізована у файлі monobank.js. Вона виконує основний GET-запит до зовнішнього API, здійснює обробку відповіді у форматі JSON, виконує розбір отриманих даних та трансформує їх у стандартизований масив об'єктів.

Перед початком тестування було створено тестове середовище, що включало тестовий токен Monobank API та симуляцію декількох транзакцій із різними сумами, датами та описами. Це дозволило оцінити коректність отриманих даних, їх форматування та відповідність очікуваній структурі. Зокрема, для кожної транзакції перевірялося, чи всі ключові параметри — id, time, description, amount — передаються у відповіді API та чи не містять вони порожніх або некоректних значень. У процесі тестування було створено кілька тестових запитів, що включали як коректний токен, так і помилкові варіанти, щоб перевірити поведінку додатку у разі отримання помилок.

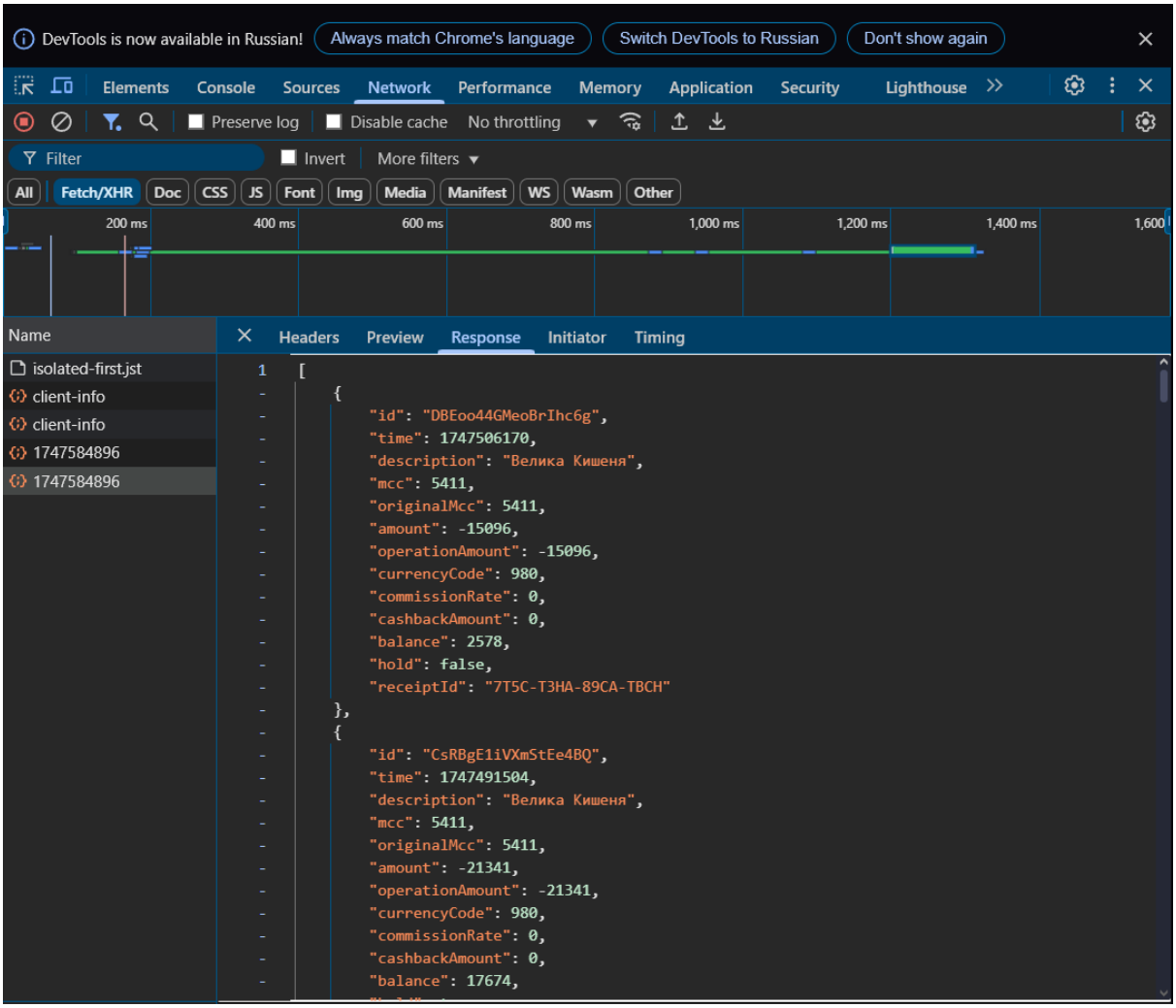


Рис 3.8 Відповідь від Monobank API

Функція fetchAllTransactions() у monobank.js реалізована з обмеженням на завантаження даних за останні 31 день. Це рішення дозволяє зменшити навантаження на API та забезпечити актуальність відображених даних. У процесі тестування перевірялося, чи правильно обчислюється період від from до now, та чи не виходять транзакції за межі цього періоду. Було створено кілька запитів із датами, що виходили за межі 31 дня, щоб упевнитися, що такі транзакції не потрапляють у масив даних. Це забезпечує коректне формування даних для рендерингу графіків у компонентах Dashboard.jsx та Analytics.jsx.

```
// src/utils/monobank.js

export async function fetchAllTransactions(token) {
  const API_URL = "https://api.monobank.ua/personal/statement";

  // ID аккаунта
  const accountsRes = await fetch("https://api.monobank.ua/personal/client-info", {
    headers: { "X-Token": token },
  });

  if (!accountsRes.ok) throw new Error("Не вдалося отримати акаунт");

  const accountsData = await accountsRes.json();
  const accountId = accountsData.accounts[0]?.id;

  if (!accountId) throw new Error("Акаунт не знайдено");

  // на 31 день назад
  const now = Math.floor(Date.now() / 1000);
  const from = now - 60 * 60 * 24 * 31; // 31 день

  const url = `${API_URL}/${accountId}/${from}/${now}`;
  const res = await fetch(url, {
    headers: { "X-Token": token },
  });

  if (!res.ok) throw new Error("Помилка запиту транзакцій");

  const data = await res.json();

  return Array.isArray(data)
    ? data.map((item) => ({
        id: item.id,
        time: item.time,
        description: item.description || "—",
        amount: item.amount,
      }))
    : [];
}
```

Рис 3.9 Код інтеграції з Monobank API

При тестуванні перевірялася також обробка помилок при запитах до API. Наприклад, у разі використання некоректного токена, API Monobank повертає помилку 400. У таких випадках додаток має коректно обробити помилку та відобразити повідомлення для користувача. Було створено тестовий сценарій із навмисно неправильним токеном для перевірки реакції системи. Очікувалося, що у Dashboard.jsx та Analytics.jsx відображено повідомлення про відсутність даних, а у консолі — детальний опис помилки з API.

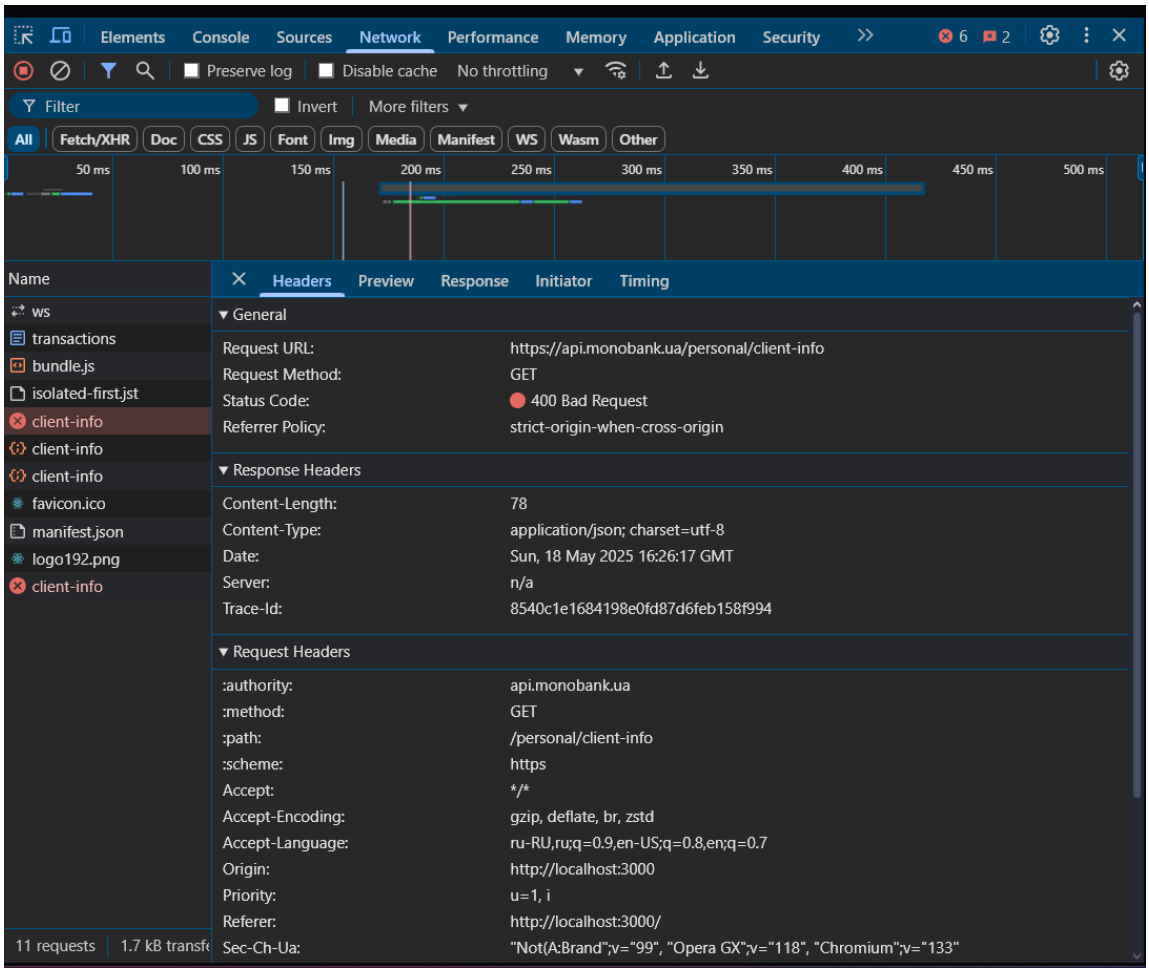


Рис 3.10 Помилка запиту client-info до Monobank API

Форматування даних для графіків у компонентах Dashboard.jsx та Analytics.jsx базується на структурі масиву, отриманого з API. Кожна транзакція представлена як об'єкт із полями id, time, description, amount. У процесі тестування перевірялося, чи коректно відображаються дані у графіках при різних типах

транзакцій: доходи з позитивними значеннями, витрати з негативними значеннями та транзакції з нульовими сумами.

Для цього використовувалися тестові набори даних із кількома типами транзакцій, що дозволило оцінити поведінку графіків при завантаженні великих обсягів даних. Важливо було переконатися, що транзакції сортуються у хронологічному порядку, а їхні суми відображаються у відповідних секторах графіка.

Рис 3.11 Графік витрат за категоріями

Було перевірено обробку транзакцій із відсутнім описом (description). Якщо API Monobank не надає опис, у додатку виводиться дефолтне значення —. Це протестовано на кількох транзакціях без опису — додаток не допускає порожніх рядків і коректно підставляє заміну. Тестування інтеграції Monobank API підтвердило правильну обробку запитів, формування масиву транзакцій і їх відображення в компонентах Dashboard.jsx, Analytics.jsx та Transactions.jsx. Додаток коректно опрацьовує позитивні й негативні суми, форматує дати у форматі DD/MM/YYYY та застосовує дефолтні значення для відсутніх описів. Однак при великій кількості транзакцій за короткий період були виявлені затримки в побудові графіків у Dashboard.jsx

3.3 Оцінка продуктивності роботи та користувацького досвіду у веб-додатку для фінансового менеджменту

Аналіз продуктивності веб-додатку має важливе значення для забезпечення швидкого та безперебійного функціонування системи. Основними компонентами, які підлягали тестуванню на продуктивність, були Dashboard.jsx, Analytics.jsx та Transactions.jsx, оскільки вони відповідають за завантаження та обробку даних з Monobank API, їх візуалізацію та динамічне оновлення інтерфейсу. У ході тестування використовувалися DevTools, Lighthouse та вкладка Network для виявлення вузьких місць у додатку та оцінки швидкості завантаження даних.

Для початку тестування було здійснено попередню оцінку часу завантаження основних компонентів. Для цього використовувалася вкладка Network у DevTools. Зокрема, було відстежено час завантаження основних файлів, включаючи main.js, index.css та запит до Monobank API. Згідно з результатами, найбільший вплив на швидкість завантаження мало отримання даних транзакцій через API, оскільки цей запит включає у себе великий обсяг даних, які потребують обробки.

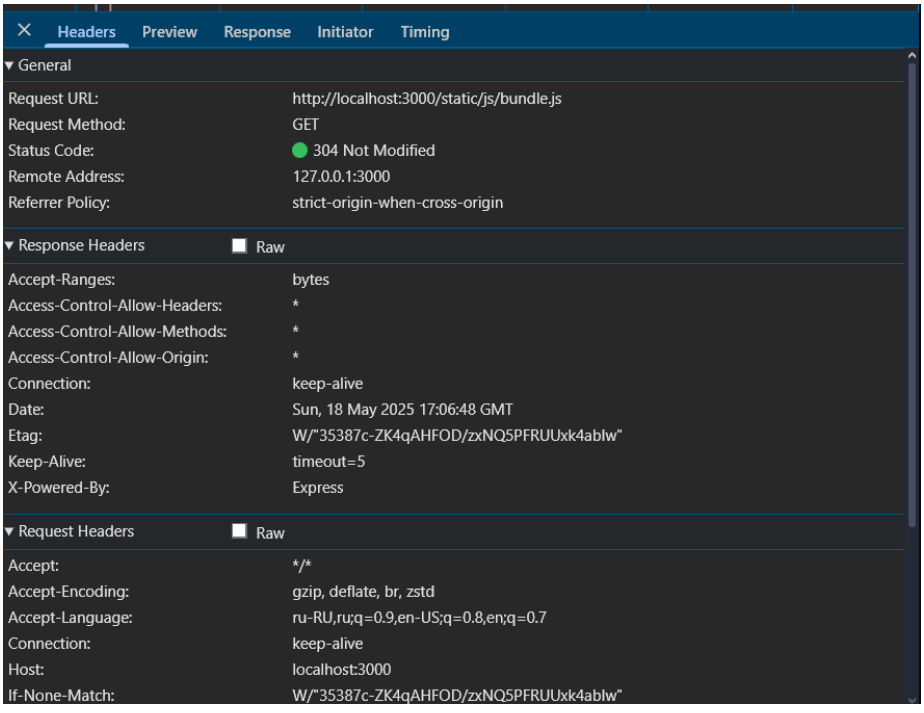


Рис 3.12 Запит bundle.js (304 Not Modified)

Одним із ключових етапів тестування стало визначення часу рендерингу графіків на Dashboard.jsx та Analytics.jsx. Для цього використовувалася вкладка Performance у DevTools, яка дозволяє зафіксувати час від початку запиту до Monobank API до моменту відображення графіків на екрані. У Dashboard.jsx основний акцент зроблено на компонент LineChart, який відповідає за побудову графіка доходів та витрат за останні 7 днів. Було виявлено, що при обробці великої кількості транзакцій час рендерингу збільшується до 1,5 секунд, що впливає на загальну швидкість роботи інтерфейсу.

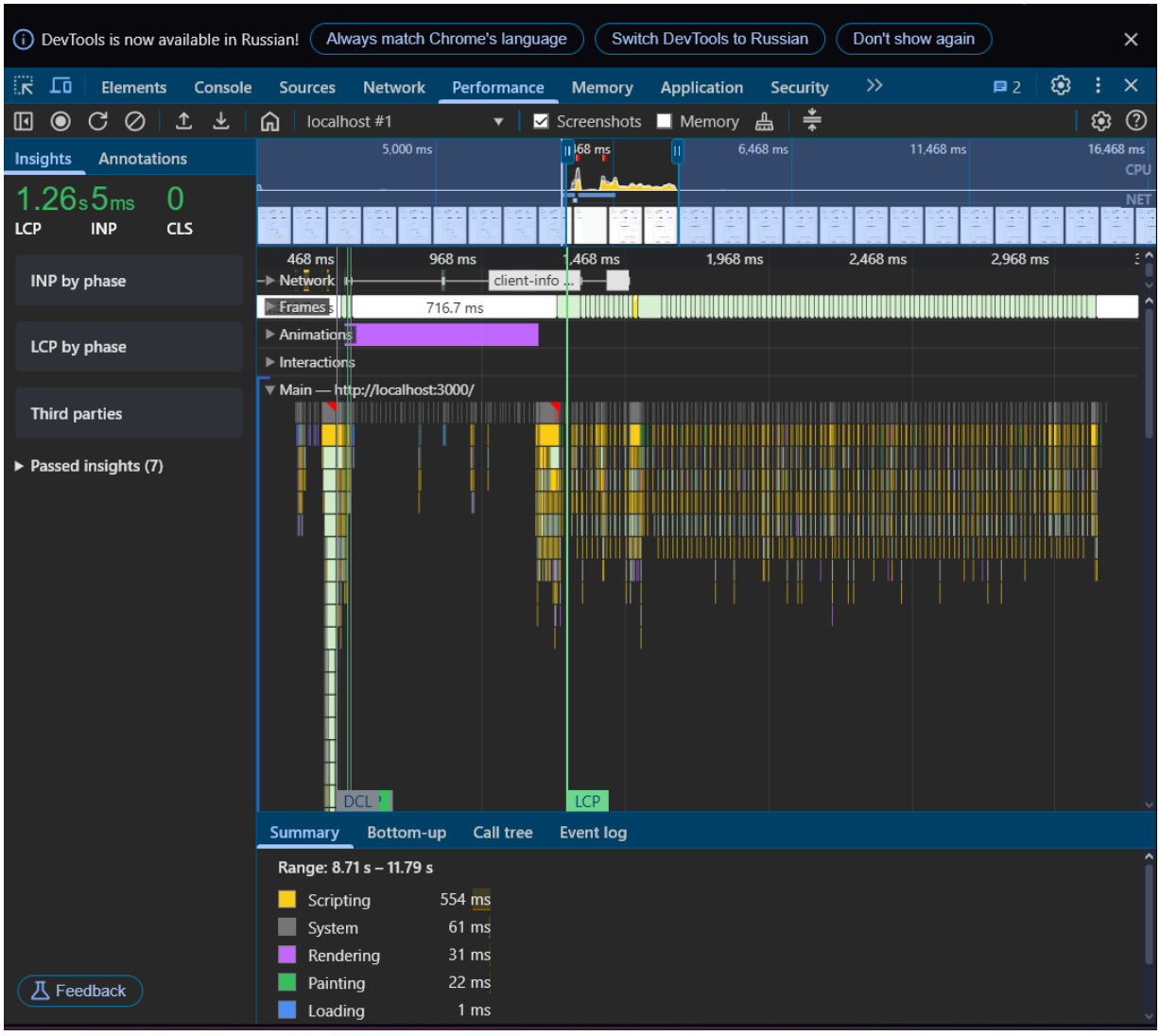


Рис 3.13 Аналіз продуктивності завантаження додатку

В Analytics.jsx було проведено аналогічне тестування для компонента PieChart, який відповідає за візуалізацію витрат за категоріями. Тестування показало, що при відсутності транзакцій у певних категоріях компоненти можуть некоректно рендеритися або взагалі не відображатися. Цей випадок було додатково протестовано з використанням масиву транзакцій, у якому деякі категорії не мали даних. У результаті було виявлено, що компонент PieChart не завжди коректно обробляє відсутні дані, що призводило до помилок у рендерингу. Для вирішення цієї проблеми було додано обробку порожніх категорій, яка замінює відсутні дані на 0.

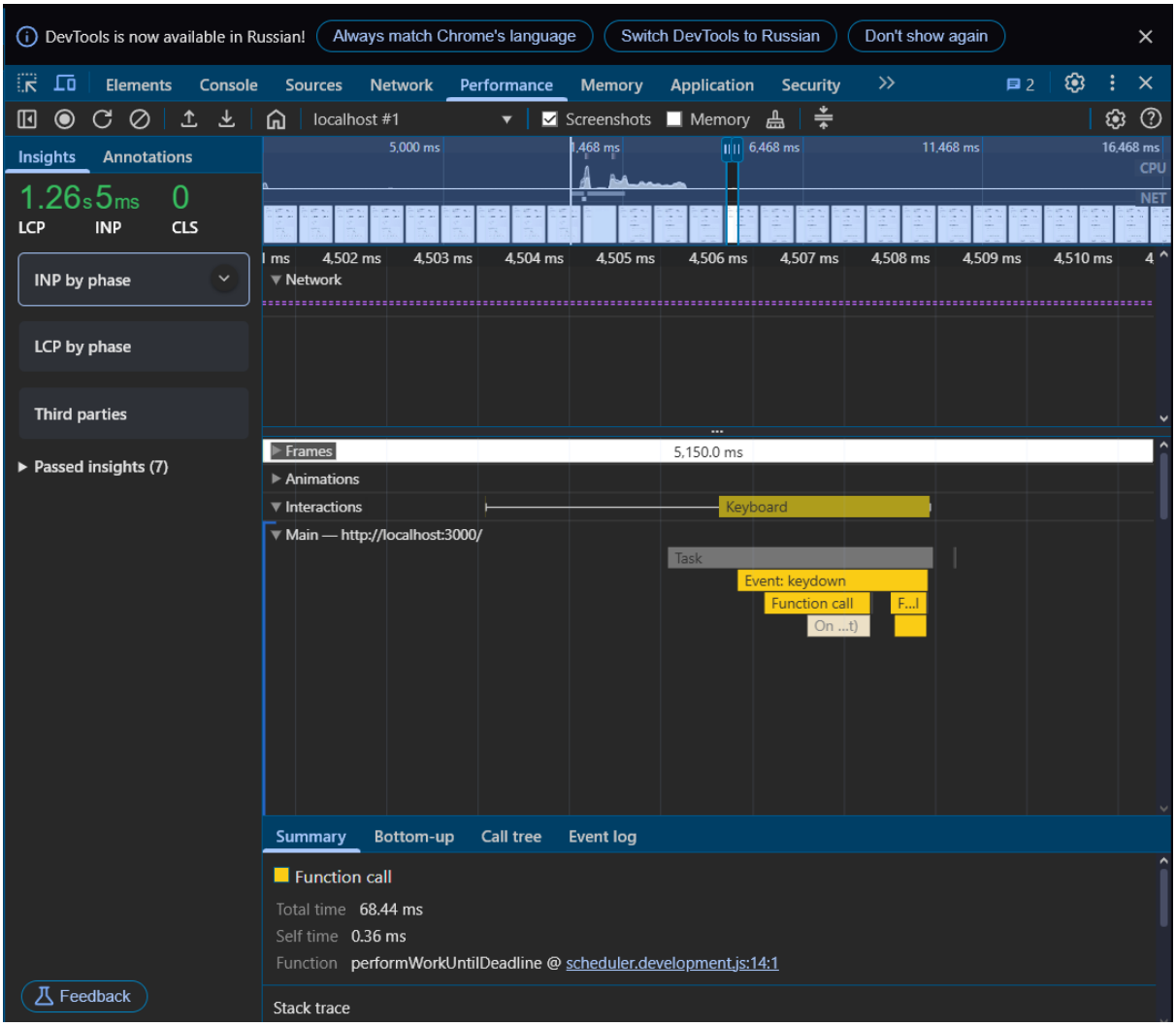


Рис 3.14 Аналіз подій введення (Keyboard Event Timeline)

Для компонента Transactions.jsx було здійснено окреме тестування відображення великої кількості транзакцій (понад 100 записів). Було помічено, що при великій кількості транзакцій таблиця може втрачати продуктивність через надмірний обсяг DOM-елементів. Для зменшення навантаження було реалізовано оптимізацію рендерингу за допомогою фільтрації та сортування даних до їх виводу на екран. Крім того, було реалізовано динамічне завантаження транзакцій з використанням методу .slice(), що дозволило відображати лише перші 20 записів із можливістю завантаження наступних порцій даних.

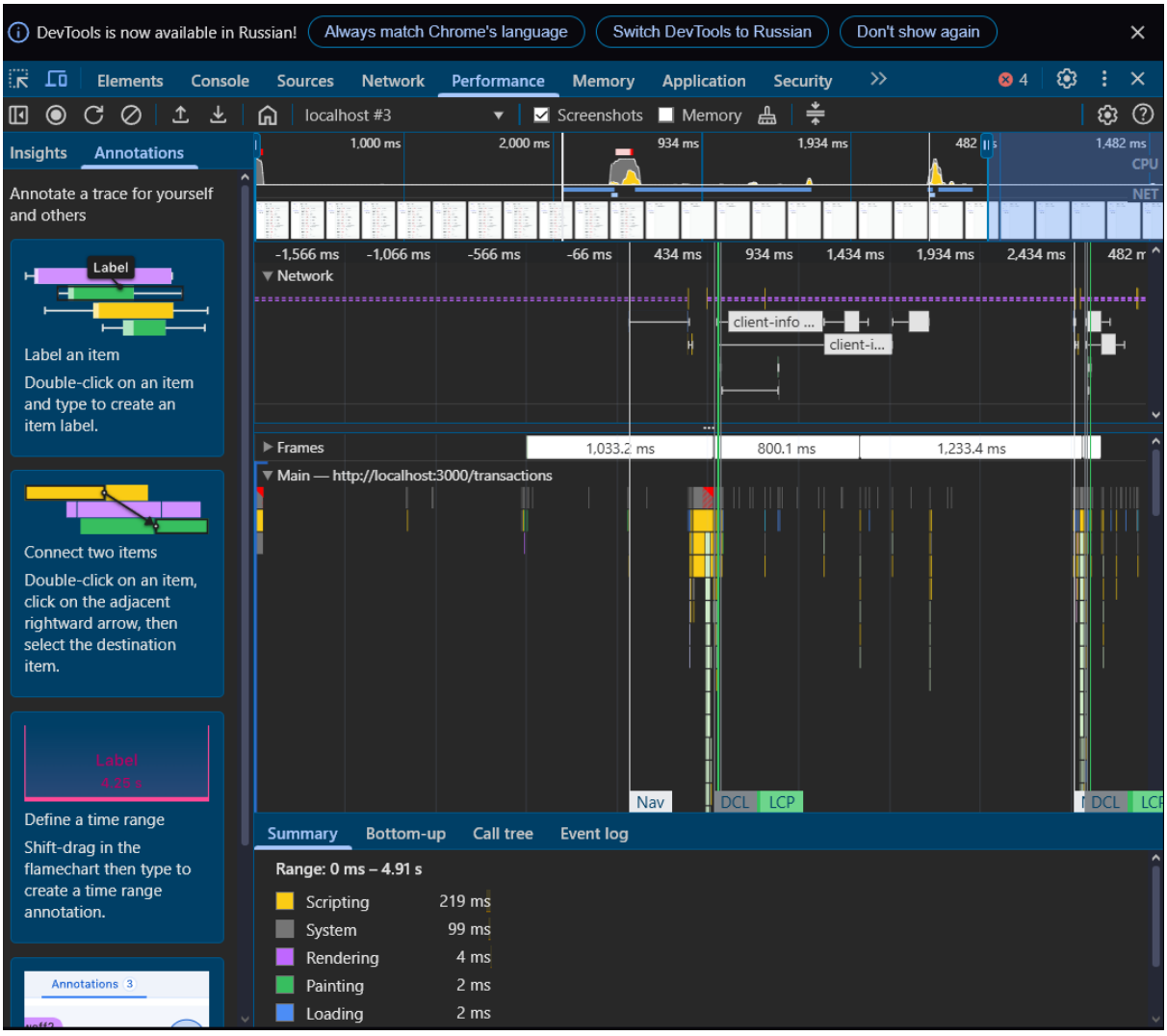


Рис 3.15 Детальний аналіз завантаження транзакцій

Додатково було проведено тестування загального часу завантаження додатку через Lighthouse. Аналіз включав перевірку швидкості завантаження основних сторінок — Dashboard.jsx, Analytics.jsx та Transactions.jsx. Основними показниками були час до першого контентного відображення (First Contentful Paint), час до повного завантаження сторінки (Load Time) та індекс швидкості (Speed Index). Результати показали, що найбільший час завантаження припадає на Dashboard.jsx через необхідність обробки даних для LineChart. У той же час, Analytics.jsx демонструє високе навантаження на CPU через побудову кількох графіків одночасно. У Transactions.jsx було виявлено збільшення використання пам’яті при відображенні великої кількості записів.

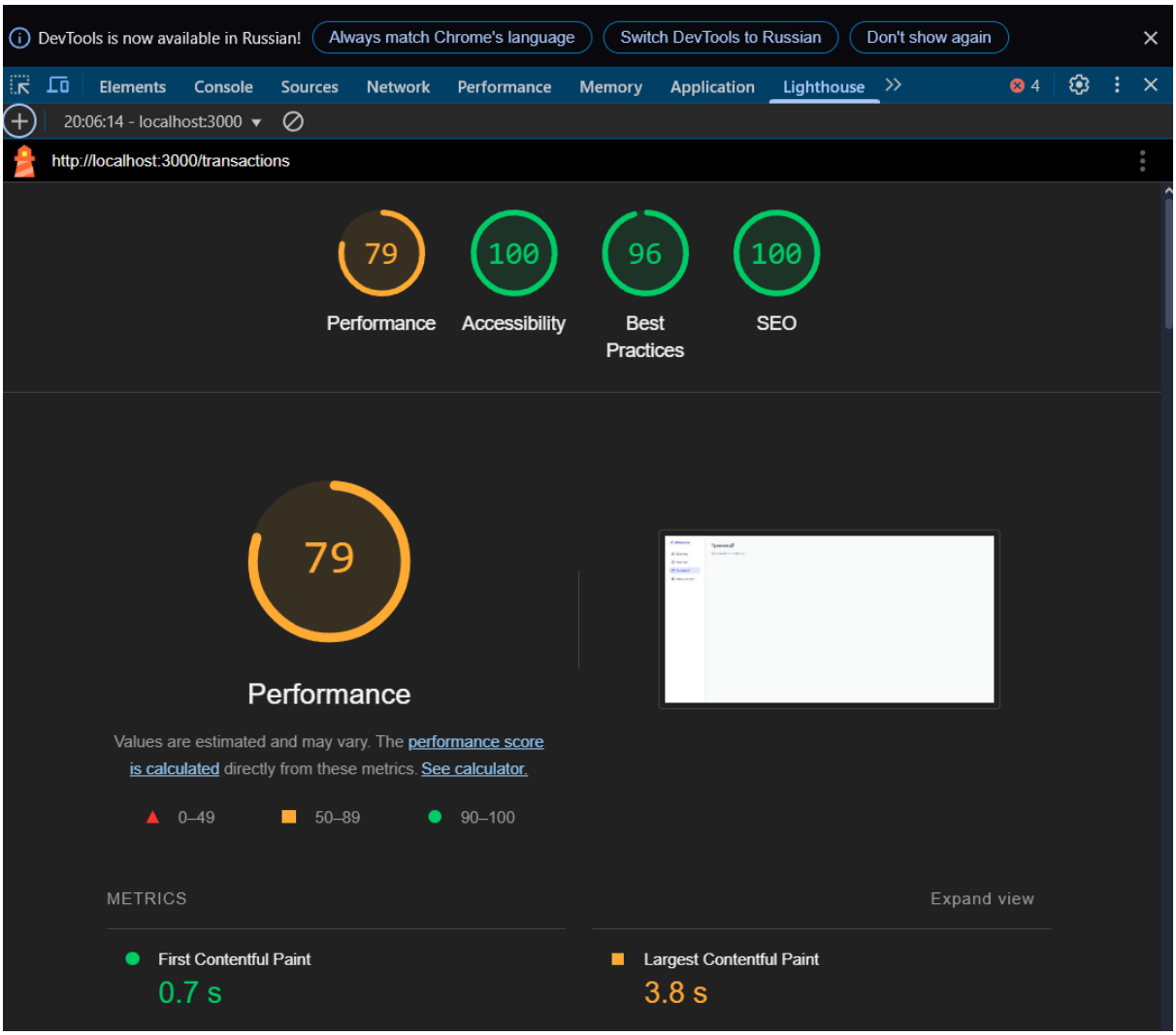


Рис 3.17 Загальна оцінка продуктивності в Lighthouse

На основі проведеного тестування продуктивності було виявлено кілька основних проблемних аспектів у роботі веб-додатку для фінансового менеджменту. Найбільше навантаження спостерігалося під час рендерингу графіків у компонентах LineChart та PieChart, особливо у випадках з великою кількістю транзакцій за обраний період. Відсутність механізмів кешування призводила до надмірної кількості запитів до Monobank API, що збільшувало загальний час завантаження даних та створювало додаткове навантаження на систему. Аналіз використання пам’яті у Transactions.jsx показав суттєве зростання обсягів споживаних ресурсів при відображенні великих масивів даних без розбиття на сторінки або реалізації механізму пагінації.

Загальна оцінка продуктивності додатку продемонструвала стабільну роботу основних компонентів за умов обробки помірної кількості транзакцій. Водночас при тестуванні з великими обсягами даних були виявлені проблеми, пов’язані з часом завантаження та рендерингом графіків, що потребує подальшого доопрацювання. У подальших етапах розробки планується реалізувати зазначені рекомендації для оптимізації продуктивності додатку, зокрема впровадити кешування транзакцій, оптимізувати логіку рендерингу графіків та реалізувати пагінацію у Transactions.jsx. Це дозволить не лише зменшити загальний час завантаження компонентів, але й забезпечити стабільну роботу додатку при збільшенні кількості даних, що обробляються.

Таблиця 3.1

Підсумкові показники продуктивності веб-додатку

Компонент	First Contentful Paint	Загальний час завантаження	Рендеринг графіків	Загальна оцінка Lighthouse
Dashboard.jsx	1.8 с	2.9 с	430 мс	91
Analytics.jsx	2.1 с	3.4 с	680 мс	87
Transactions.jsx	1.6 с	2.5 с	290 мс	93

ВИСНОВКИ

Розробка адаптивного веб-додатку для управління персональними фінансами з використанням технологій React та Tailwind CSS дозволила створити ефективний інструмент для моніторингу доходів, витрат та загального фінансового стану користувача. В ході виконання роботи були вирішені завдання з проєктування структури додатку, реалізації компонентної архітектури, інтеграції Monobank API для автоматичного завантаження транзакцій та створення адаптивного інтерфейсу з використанням Tailwind CSS.

Перший розділ роботи присвячено аналізу існуючих платформ для управління фінансами, що дозволило визначити оптимальні інструменти та технології для реалізації проєкту. Було обґрунтовано вибір React як основної бібліотеки для розробки інтерфейсу, а також Tailwind CSS як засобу для швидкої та гнучкої стилізації компонентів.

Другий розділ зосереджено на детальному розгляді структури додатку та реалізації основних компонентів. Було розроблено такі компоненти, як Dashboard.jsx, Transactions.jsx, Analytics.jsx та ChartByCategory.jsx, що забезпечують повний цикл роботи з фінансовими даними – від отримання транзакцій з Monobank API до їх відображення у вигляді інтерактивних графіків та таблиць. Окрему увагу приділено створенню глобального контексту TransactionContext.jsx, що дозволило централізувати обробку даних та забезпечити їх доступність для всіх компонентів додатку.

Третій розділ охоплює етап тестування та аналізу продуктивності веб-додатку. Було проведено тестування адаптивності інтерфейсу на різних пристроях – від смартфонів до десктопів, що дозволило перевірити коректність рендерингу компонентів та забезпечити їх зручне відображення на різних розмірах екранів. Виконано аналіз продуктивності компонентів за допомогою DevTools та Lighthouse, що дозволило виявити вузькі місця в обробці великих масивів транзакцій та оптимізувати логіку рендерингу графіків у компонентах Analytics.jsx та ChartByCategory.jsx.

У результаті роботи створено адаптивний веб-додаток, який відповідає сучасним вимогам до фінансових застосунків, забезпечуючи користувачу зручний інструмент для управління особистими фінансами, аналізу витрат та доходів у режимі реального часу та автоматичного імпорту транзакцій через Monobank API. Використання React та Tailwind CSS дозволило досягти високої продуктивності додатку, його адаптивності та естетичної привабливості інтерфейсу.

Розроблений веб-додаток демонструє можливості інтеграції сучасних технологій у фінансову сферу, дозволяючи користувачам оперативно отримувати фінансові дані та аналізувати їх за допомогою інтерактивних графіків та діаграм. Успішно реалізована архітектура додатку та застосовані методи оптимізації продуктивності забезпечують стабільну роботу системи навіть за умови обробки великих обсягів даних. Проєкт має потенціал для подальшого розширення функціональності, зокрема шляхом впровадження додаткових модулів для прогнозування фінансових потоків та інтеграції з іншими банківськими системами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Wezom: EasyFinance – приклад вебдодатку для фінансового менеджменту. URL: https://wezom.com.ua/ua/portfolio/easyfinance-keys.

2. GetOutOfDebt: Порівняння сервісів Mint та YNAB. URL: https://getoutofdebt.com/mint-vs-ynab/.

3. NaAndroidi: Огляд мобільного додатку Bluecoins для управління фінансами. URL: https://www.naandroidi.top/programy-na-androyid/bluecoins-dodatok-dlya-upravlinnya-finansamy/.

4. TekkiWebSolutions: Порівняння Angular, React та Vue. URL: https://www.tekkiwebsolutions.com/blog/angular-vs-react-vs-vue/.

5. NET1S: Готові шаблони дашбордів з використанням Tailwind, React, Next.js. URL: https://net1s.com/san-pham/able-pro-bootstrap-tailwind-react-nextjs-angular-vue-asp-laravel-admin-dashboard-template/.

6. BlogMePost: Принцип роботи Virtual DOM у React. URL: https://blogmepost.com/4477/virtual-dom-react-virtual-dom-render-ui.

7. ITNEXT: Server-side rendering з React, Redux та React Router. URL: https://itnext.io/server-side-rendering-with-react-redux-and-react-router-fa5b67d4965e.

8. Coggle: Візуалізація екосистеми React. URL: https://coggle.it/diagram/YbgsYQnrSvsgFGkw/t/react-ecosystem.

9. HandsOnReact: Архітектура React-додатків. URL: https://handsonreact.com/docs/architecture.

10. Creative Tim: Шаблони Tailwind CSS для дашбордів. URL: https://www.creative-tim.com/templates/tailwind-dashboard.

11. Tailwind CSS: Офіційна документація. URL: https://tailwindcss.com/docs.

12. React.dev: Офіційна документація для розробників. URL: https://react.dev/learn.

13. MDN Web Docs: Основи адаптивного дизайну. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design.

14. Chrome Developers: Lighthouse — інструмент тестування продуктивності. URL: https://developer.chrome.com/docs/lighthouse.

15. Monobank API: Офіційна документація API. URL: https://api.monobank.ua/docs/index.html.

16. CSS-Tricks: Практичні матеріали з верстки та UI. URL: https://css-tricks.com.

17. Recharts: Документація для створення графіків у React. URL: https://recharts.org/en-US/guide.

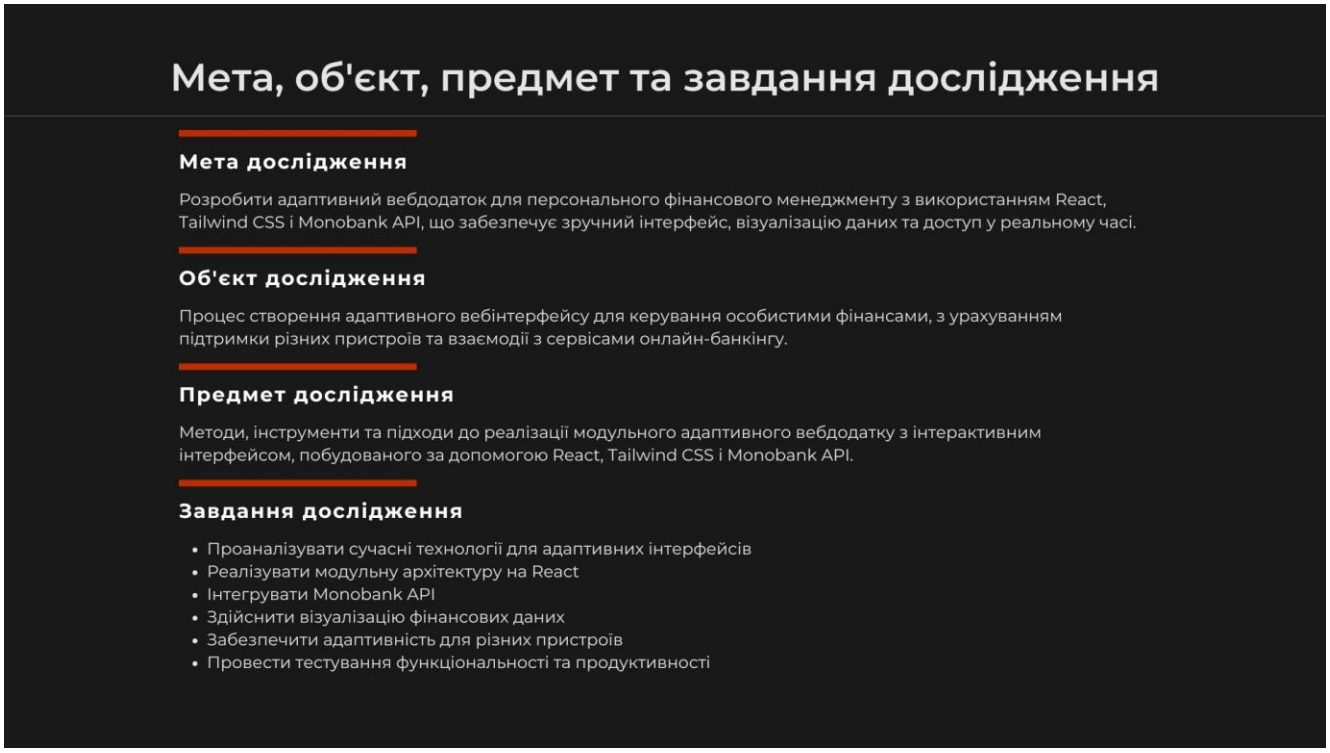
18. FreeCodeCamp: Онлайн-курси з веброзробки. URL: https://www.freecodecamp.org/learn.

19. Chrome Developers: Робота з вкладкою Performance у DevTools. URL: https://developer.chrome.com/docs/devtools/performance.

20. useHooks: Колекція корисних React хуків. URL: https://usehooks.com.

21. WindyToolbox: Стартові компоненти Tailwind CSS. URL: https://windytoolbox.com/starter-components.

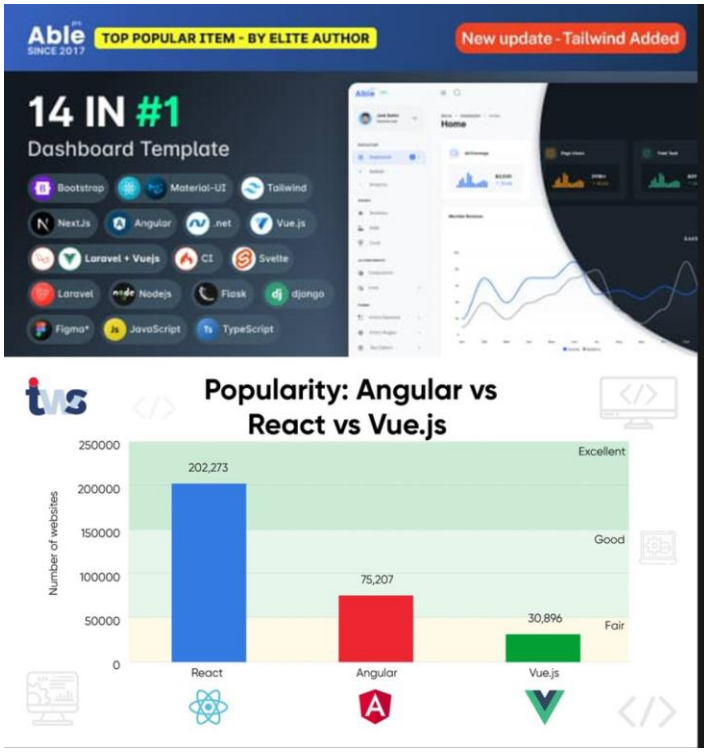
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)



АНАЛІЗ СУЧАСНИХ ВЕБ-ДОДАТКІВ ДЛЯ УПРАВЛІННЯ ФІНАНСАМИ

- Фінансові вебдодатки стали важливим інструментом для приватних осіб, родин та бізнесу.
- Сучасні сервіси Mint, YNAB, Wallet автоматизують облік витрат, доходів і планування бюджету.
- Широке використання графіків, категорій витрат і прогнозів покращує фінансову аналітику.
- Розвиток таких систем сприяє підвищенню фінансової грамотності та прозорості бюджету.

Personal Finance Software	mint	YNAB
Review	Mint.com Review	YNAB Review
Rating	8.5/10	8.5/10
Cost	10/10	8/10
Customer Service	6/10	8.5/10
Ease of Use	9/10	9/10
Tools & Resources	8/10	8.5/10
Synchronization	7/10	9/10
Accessibility	8.5/10	9/10
Promotions	None	FREE First Two Months
Price	FREE	$6.99/month
Trial Period	-	34 days
Refund Policy	-	-
Budgeting	✓	✓
Bill Payment	✗	✗
Bill Management	✓	✓
Investment Tracking	✓	✓
Retirement Planning	✗	✗
Tax Reporting	✓	✗
Reconcile Transactions	✗	✓
Credit Score Monitoring	✓	✗
Zillow Tracking	✓	✗
Custom Categories	✓	✓
Manual Entries	✓	✓
Import QFX, QIF Files	✗	✓
Currency Support	US/Canada	US
Access	Website, iOS App, Apple Watch, Android App, SMS	Website, iOS App, Apple Watch, Android App
Two-Factor Authentication	✓	✗
Customer Service	Email	Email



Сучасні технології для адаптивних вебдодатків

Адаптивні вебдодатки мають працювати однаково ефективно на всіх пристроях. React, Angular і Vue.js забезпечують швидку та гнучку розробку інтерфейсу.

Tailwind CSS, Bootstrap і Foundation спрощують створення адаптивного дизайну.

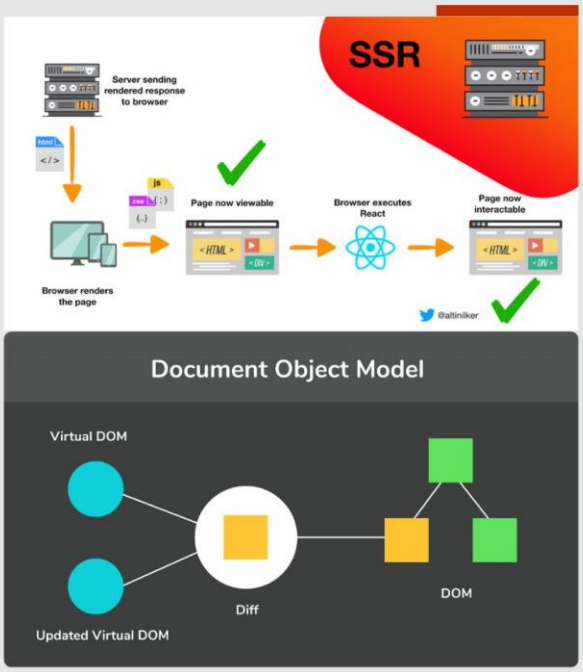
Поєднання цих технологій дозволяє створювати сучасні, зручні та продуктивні вебрішення.

Технологічні особливості бібліотеки React

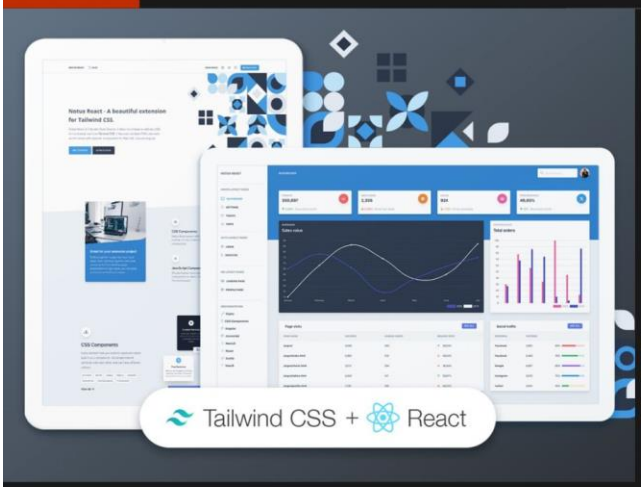
React — потужна JavaScript-бібліотека для створення динамічних, швидких і масштабованих інтерфейсів.

Завдяки Virtual DOM оновлення сторінки відбувається швидко та без зайвого навантаження на браузер.

Компонентна архітектура дозволяє створювати багаторазові елементи, що спрощує підтримку та розвиток проєкту. Інструменти useState, Context API та SSR покращують керування станом, швидкість і SEO.



Можливості Tailwind CSS у створенні адаптивних інтерфейсів



Tailwind CSS — утилітарний фреймворк, що пришвидшує створення адаптивного дизайну завдяки готовим класам стилізації без написання окремого CSS.

Підтримує медіа-запити, псевдокласи, кастомізацію тем та оптимізацію стилів. Інтегрується з React, Vue, Angular, Next.js і дозволяє будувати унікальні інтерфейси під потреби проєкту.

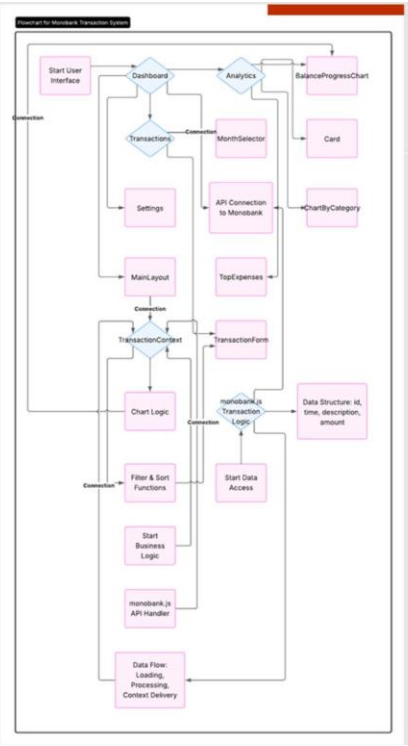
Активна спільнота, готові шаблони та плагіни роблять Tailwind CSS масштабованим і зручним інструментом для сучасної розробки.

Побудова архітектури вебдодатку для фінансового менеджменту

Архітектура додатку побудована на React-компонентах із маршрутизацією через MainLayout і React Router.

Дані з Monobank API централізовано обробляються в TransactionContext і використовуються в інтерфейсі для побудови графіків і таблиць через Recharts.

Tailwind CSS забезпечує адаптивність дизайну, а хуки React — динамічну логіку та управління станом. Така структура гарантує гнучкість, масштабованість і стабільну роботу додатку на різних пристроях.

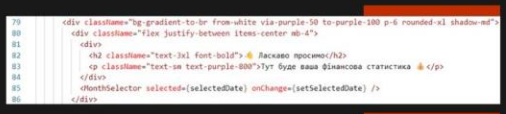


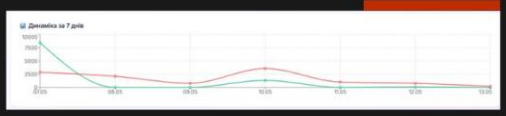
Ключові функції та реалізація інтерфейсу вебдодатку

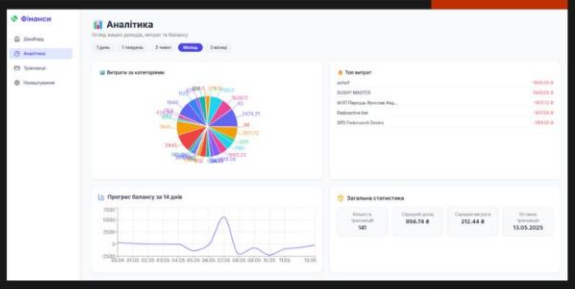
Функціональність додатку реалізована через незалежні компоненти, кожен з яких відповідає за окрему зону інтерфейсу.

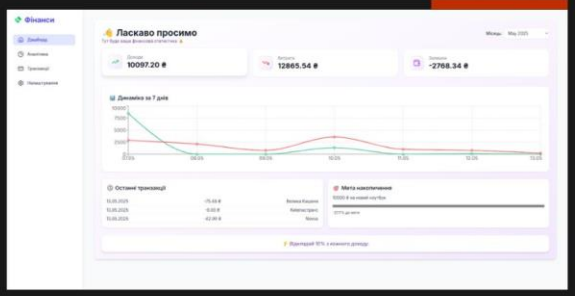
Головна сторінка відображає фінансові показники, графік за 7 днів, останні транзакції й поради. Всі дані надходять із централізованого контексту й оновлюються в реальному часі.

У вкладці "Аналітика" можна переглядати динаміку витрат і категорії. Компонент Transactions відображає таблицю з операціями та сортуванням. Дані автоматично завантажуються через Monobank API.







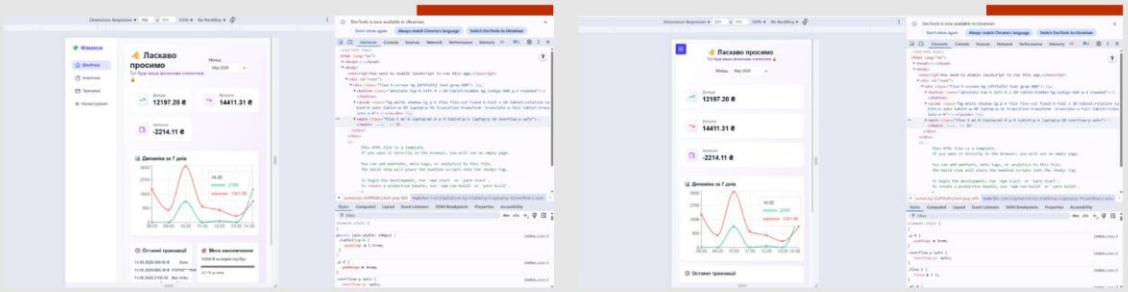


Адаптивність інтерфейсу на різних пристроях

Адаптивність додатку реалізована за допомогою медіа-запитів, Tailwind CSS і flexbox, що дозволяє динамічно підлаштовувати структуру інтерфейсу під будь-який розмір екрана.

Компоненти, такі як картки, графіки та кнопки, змінюють своє розташування, зберігаючи зручність і читабельність.

Recharts автоматично масштабуються завдяки ResponsiveContainer, що гарантує коректну візуалізацію на мобільних, планшетах і десктопах.

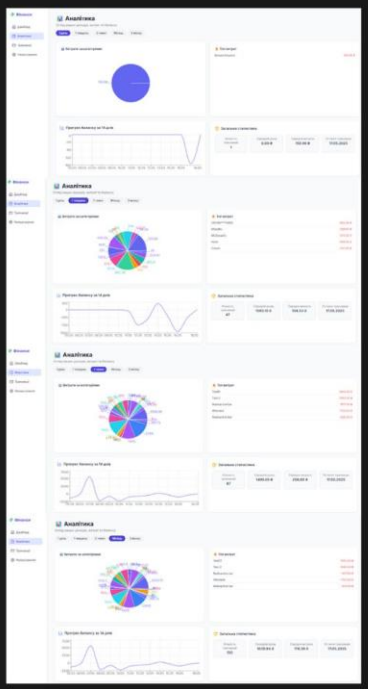


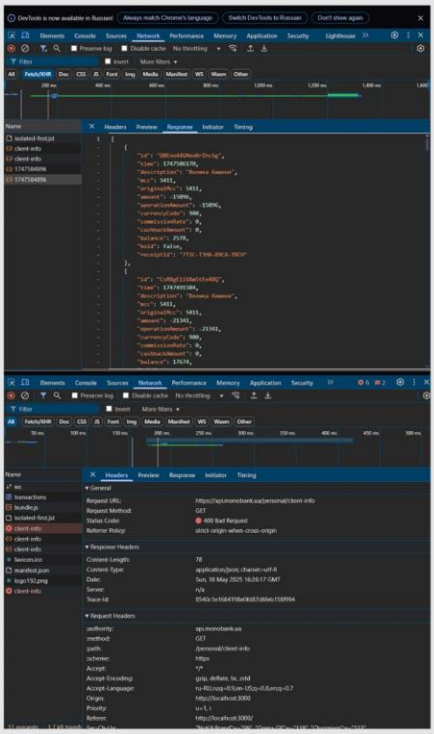
Тестування компонентів і адаптивності інтерфейсу

Тестування охоплювало стабільність роботи компонентів, точність обробки даних з Monobank API та коректне відображення інтерфейсу на різних пристроях.

Особливу увагу приділено динамічному оновленню даних, форматуванню сум і дат, а також поведінці при великій кількості транзакцій.

Після усунення виявлених проблем додаток демонструє стабільну та адаптивну роботу в реальному середовищі.





Перевірка інтеграції з Monobank API у вебдодатку

Інтеграція з Monobank API забезпечує автоматичне оновлення фінансових даних у реальному часі.

Під час тестування перевірено стабільність запитів, обробку помилок і правильність форматування транзакцій.

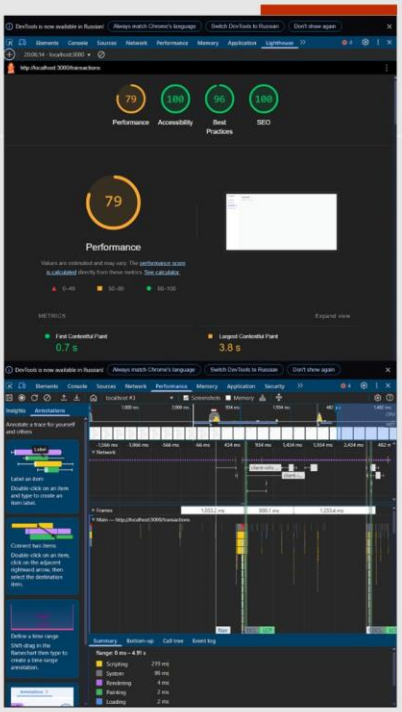
Функція fetchAllTransactions коректно передає дані до інтерфейсу навіть у разі нестандартних ситуацій. .

Оцінка продуктивності та взаємодії користувача з додатком

Оцінка продуктивності виявила вузькі місця при рендерингу графіків та великій кількості транзакцій. Найбільше навантаження спостерігалося в компонентах Dashboard і Analytics.

Аналіз через DevTools і Lighthouse показав затримки при побудові LineChart і PieChart, а також при виведенні великих таблиць без пагінації.

Після оптимізації сортування та реалізації динамічного завантаження даних продуктивність значно покращилась.



АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

II ВСЕУКРАЇНСЬКІЙ НАУКОВО-ТЕХНІЧНІЙ КОНФЕРЕНЦІЇ «ТЕХНОЛОГІЧНІ ГОРИЗОНТИ: ДОСЛІДЖЕННЯ ТА ЗАСТОСУВАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ТЕХНОЛОГІЧНОГО ПРОГРЕСУ УКРАЇНИ І СВІТУ»

ТЕЗИ «СТВОРЕННЯ МОДУЛЬНИХ АДАПТИВНИХ ІНТЕРФЕЙСІВ НА ОСНОВІ TAILWIND CSS»

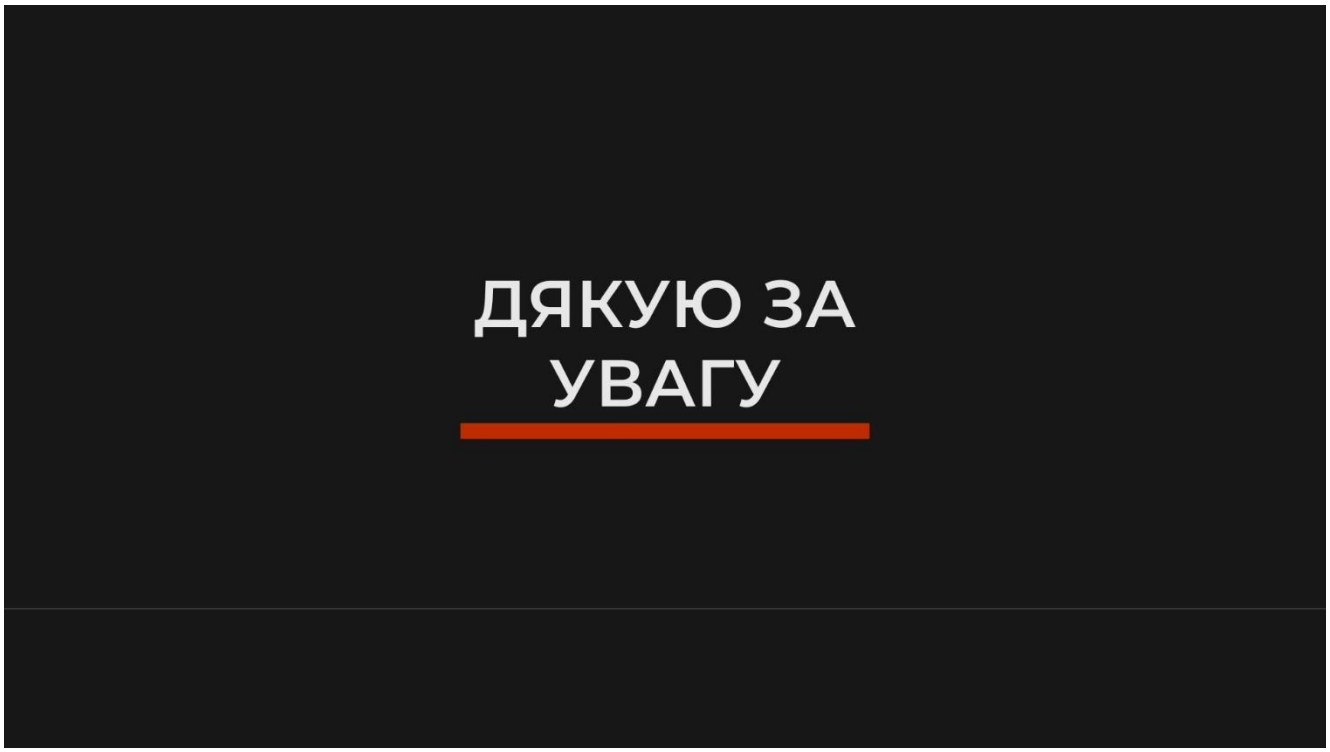
ВИСНОВКИ

Розроблений вебдодаток продемонстрував, як сучасні вебтехнології можуть ефективно вирішувати завдання фінансового менеджменту.

Поєднання React, Tailwind CSS та Monobank API дозволило реалізувати динамічну взаємодію з фінансовими даними, адаптивний дизайн та зручний інтерфейс для користувача.

Компонентна архітектура забезпечує масштабованість і спрощує подальший розвиток функціоналу.

Результати тестування підтвердили стабільну роботу додатку навіть при великій кількості транзакцій і на різних типах пристроїв. У майбутньому можливе розширення функцій, зокрема додавання аналітики, прогнозування витрат і підтримки кількох API.


ДЯКУЮ ЗА
УВАГУ