Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ОБЛІКУ ТА АНАЛІЗУ ОСОБИСТИХ ФІНАНСІВ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-3
Кліменко Я. К.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Шафроненко А. Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кліменку Ярославу Костянтиновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебзастосунку для обліку та аналізу особистих фінансів

затверджена наказом університету від 19 травня 2025 року № 381Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 29 травня 2025 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, фреймворк React, фреймворк Express.js, платформа Node.js, мова програмування JavaScript, бібліотека TensorFlow.js, система керування базами даних.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд існуючих вебзастосунків для обліку та аналізу особистих фінансів.
2. Проєктування вархітектури застосунку та моделювання бази даних.
3. Розробка користувацького інтерфейсу вебзастосунку.
4. Тестування розробленого вебзастосунку та оцінка його ефективності.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми ефективного управління особистими фінансами, постановка задачі, схема бази даних, архітектура вебзастосунку, інтерфейс головної сторінки застосунку, інтерфейс сторінки обліку транзакцій, інтерфейс сторінки встановлення та перегляду цілей, розділ аналітики, візуалізація процесу прогнозування бюджету, інтерфейс сторінки налаштувань користувача.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Дослідження технічних інструментів	15.04.25-20.04.25	
5	Проєктування архітектури застосунку	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
13	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Шафроненко А. Ю.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 79 с., 14 табл., 19 рис., 31 джерело.

ВЕБЗАТОСУНОК, БАЗА ДАНИХ, JAVASCRIPT, REACT, NODE.JS, EXPRESS.JS, REST API, MONGODB, TENSORFLOW.JS, АУТЕНТИФІКАЦІЯ GOOGLE, ФІНАНСОВИЙ АНАЛІЗ, ПРОГНОЗУВАННЯ БЮДЖЕТУ.

Об’єктом роботи є процес створення та забезпечення функціонування вебзастосунку для централізованого обліку особистих фінансів та їхнього аналізу.

Метою роботи є розробка інтуїтивно зрозумілого та ефективного вебсервісу, що надасть користувачам можливість контролювати свої фінансові потоки та отримувати прогнози щодо майбутнього бюджету.

Спроєктовано архітектуру вебзастосунку та модель бази даних. Реалізовано функціонал аутентифікації користувачів, обліку транзакцій, аналізу фінансових даних і прогнозування бюджету на основі алгоритмів машинного навчання. Створено зручний користувацький інтерфейс для ефективної взаємодії з усіма можливостями застосунку.

У результаті здійснена програмна реалізація вебзастосунку, що забезпечує комплексний підхід до обліку та аналізу особистих фінансів.

WEB APPLICATION, DATABASE, JAVASCRIPT, REACT, NODE.JS, EXPRESS.JS, REST API, MONGODB, TENSORFLOW.JS, GOOGLE AUTHENTICATION, FINANCIAL ANALYSIS, BUDGET FORECASTING.

The object of this work is the development and implementation of a web application for centralized personal finance tracking and analysis.

The aim is to create an intuitive and efficient web service that enables users to control their financial flows and receive forecasts for future budgets.

The architecture of the web application and the database model were designed. Functionality for user authentication, transaction tracking, financial data analysis, and budget forecasting based on machine learning algorithms was implemented. A user-friendly interface was developed to ensure effective interaction with all application features.

As a result, a fully functional web application was created that provides a comprehensive approach to personal finance management.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JS – JavaScript (мова програмування JavaScript)

JSON – JavaScript Object Notation

MERN – MongoDB, Express.js, React, Node.js

REST API – Representational State Transfer Application Programming Interface (інтерфейс програмування застосунків з представленням стану)

СУБД – система управління базами даних

БД – база даних

CRUD – Create, Read, Update, Delete/Destroy

UI – User Interface (інтерфейс користувача)

Mono – Monobank

БД – база даних

CSS – Cascading Style Sheets (каскадні таблиці стилів)

UI – User Interface (інтерфейс користувача)

Mono – Monobank

ML – Machine Learning (машинне навчання)

HMR – Hot Module Replacement (гаряча заміна модулів)

JSX – JavaScript XM

ВСТУП

У сучасному світі ефективне управління особистими фінансами має вирішальне значення для забезпечення фінансової стабільності та досягнення особистих цілей. Уміння контролювати доходи і витрати, складати бюджет і аналізувати фінансові потоки дозволяє приймати зважені рішення, мінімізувати стрес і впевнено рухатися до фінансової незалежності. У цьому контексті вебзастосунок для фінансового обліку стає незамінним інструментом, що пропонує зручні та функціональні рішення для управління грошима.

Інструмент для обліку та аналізу особистих фінансів є відповіддю на ці актуальні виклики сучасного суспільства. Він розробляється з метою надання користувачам дієвого та інтуїтивно зрозумілого інструменту для організації, моніторингу та аналізу їхніх фінансових операцій. Завдяки інтегрованим функціям обліку доходів, витрат, планування бюджету та візуалізації фінансових даних, цей застосунок покликаний допомогти користувачам отримати повний контроль над своїми коштами, підвищити фінансову грамотність та сприяти досягненню поставлених фінансових цілей.

Актуальність роботи полягає у зростаючій потребі суспільства в зручних та функціональних інструментах для управління особистими фінансами. Розробка вебзастосунку, що поєднує в собі можливості обліку, аналізу та прогнозування, є своєчасним та затребуваним рішенням, здатним значно полегшити фінансове життя користувачів, підвищити їхню продуктивність у сфері управління власними коштами та сприяти досягненню фінансової стабільності.

1 ТЕОРЕТИЧНИЙ ОГЛЯД ФУНКЦІОНАЛУ ТА МЕТОДІВ СТВОРЕННЯ ВЕБЗАСТОСУНКУ

1.1 Огляд предметної області

Вебзастосунок для обліку фінансів це інноваційний інструмент, призначений для комплексного управління фінансовими ресурсами користувачів. У контексті даної роботи, розглядається розробка вебзастосунку, що забезпечує не лише відстеження фінансових операцій, але й їхню систематизацію та аналіз з метою оптимізації фінансового планування. До ключових аспектів розробки належать:

– забезпечення функціональності для фіксації, класифікації та моніторингу транзакцій користувача;

– реалізація аналітичної функціональності, включаючи візуалізацію фінансових даних у вигляді графіків;

– імплементація інструментів для автоматизованої та ручної категоризації доходів і витрат;

– створення інтуїтивно зрозумілого інтерфейсу.

Метою розробки є надання користувачам зручного та ефективного застосунку для контролю особистих фінансів, сприяючи підвищенню фінансової грамотності та прийняттю обґрунтованих фінансових рішень завдяки спрогнозованим значенням балансу, витрат та доходів.

1.2 Аналіз функціональних можливостей популярних вебзастосунків для обліку та аналізу фінансів

На ринку представлено безліч вебзастосунків, призначених для обліку та аналізу особистих фінансів. Серед популярних рішень можна виділити Money Lover, Money Manager, Wallet, Buddy, Saldo, Monobudget, Spendee, Goodbudget.

Застосунки для управління фінансами пропонують широкий спектр функціональних можливостей, спрямованих на покращення контролю над особистими чи сімейними бюджетами. Однією з основних функцій є відстеження доходів і витрат: користувачі можуть фіксувати всі свої фінансові операції, класифікуючи їх за категоріями. Деякі застосунки дозволяють автоматично синхронізуватися з банківськими рахунками.

Доволі поширеною є можливість бюджетування та планування. Застосунки дають змогу створювати бюджети, встановлювати ліміти витрат за категоріями та контролювати їх виконання. У деяких випадках реалізовано метод «конвертів», який дозволяє більш наочно розподіляти фінанси. Його суть полягає в тому, що користувач розподіляє доступні кошти між окремими категоріями витрат, подібно до того, як у класичному варіанті гроші розкладаються по фізичних конвертах із різноманітними позначками.

Крім того, деякі рішення включають інструменти для базового інвестиційного аналізу, хоча для повноцінної роботи з інвестиціями зазвичай використовуються окремі інструменти. Багато з представлених застосунків також підтримують інтеграцію з банківськими рахунками та кредитними картками, що забезпечує автоматичне оновлення даних і зручність користування. Іноді навіть реалізовані базові функції прогнозування, які на основі попередніх даних дозволяють користувачеві оцінити майбутній фінансовий стан, хоча така функціональність ще не є популярною.

1.3 Порівняльний аналіз ключових компонентів розглянутих фінансових вебзастосунків

Для кращого розуміння функціональних можливостей різних вебзастосунків для обліку фінансів доцільно провести їх порівняльний аналіз. Порівняння ключових компонентів представлено у таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця функцій різних застосунків

Назва застосунку	**Облік фінансів**	**Аналітика**	**Банківська інтеграція**	**Платформи**	**Вартість**
1	**2**	**3**	**4**	**5**	**6**
Money Lover	Автоматичний дохід/витрати, бюджетування за категоріями	Детальні звіти, візуалізації	Так	Web / iOS / Android	Безкоштовно / Підписка
Money Manager	Базовий облік, бюджетування за категоріями	Детальні звіти, статистика	Так	iOS / Android	Безкоштовно
Wallet	Автоматичний облік, цільове бюджетування	Детальні аналітичні звіти	Так	iOS / Android	Безкоштовно
Buddy	Спільний облік, бюджетування за категоріями	Базові звіти	Ні	iOS	Безкоштовно / Платна версія
Saldo	Автоматичний облік	Базові звіти	Так (Monobank)	iOS / Android	Безкоштовно

Подовження таблиці 1.1

1	2	3	4	5	6
Spendee	Автоматичний облік, бюджетування за категоріями	Візуалізації	Так	iOS / Android	Безкоштовно / Підписка
Goodbudget	Ручний облік, бюджетування за конвертами	Аналіз витрат, планування	Часткова (імпорт виписок)	Web / iOS / Android	Безкоштовно / Підписка
Monefy	Ручний облік, бюджетування за категоріями	Базові звіти	Ні	iOS / Android	Безкоштовно

Спільними рисами більшості розглянутих застосунків є можливість відстеження доходів та витрат, інструменти для бюджетування та створення базових звітів. Відмінності полягають у наявності таких функцій, як автоматична інтеграція з банками, інструменти для інвестиційного аналізу та прогнозування. Перевагою автоматичної інтеграції є зручність та економія часу, але вона може викликати питання щодо безпеки даних. Ручне введення даних забезпечує більший контроль, але є менш зручним. Різні підходи до бюджетування (за категоріями, за цілями, за методом конвертів) задовольняють різні потреби користувачів.

Таким чином, розробники застосунків для обліку фінансів намагаються забезпечити користувачів максимально зручним і функціональним інструментом. Продукти поєднують у собі базові можливості обліку доходів і витрат із розширеними функціями аналітики, бюджетування, банківської інтеграції та візуалізації даних. Водночас кожен із розглянутих сервісів має свої сильні сторони, які орієнтовані на різні типи користувачів: від тих, хто віддає перевагу ручному контролю, до тих, кому більше до душі автоматизація та глибока аналітика. Загальна тенденція полягає у поступовому переході до більш персоналізованих і технологічно складних рішень, що враховують

індивідуальні фінансові потреби, поведінкові звички та високі вимоги до інтуїтивно зрозумілого інтерфейсу.

1.4 Огляд технологій для розробки вебзастосунку для обліку та аналізу фінансів

Створення вебзастосунку для керування фінансами потребує застосування сучасних технологій, які забезпечують ефективну роботу, безпеку даних, гнучкість у розвитку та комфорт для користувача.

Для реалізації серверної логіки доцільно використовувати платформу Node.js разом із фреймворком Express.js, що дозволяє створювати швидкі та масштабні вебсервери.

Клієнтська частина розробляється за допомогою мови JavaScript і бібліотеки React, що забезпечує динамічний користувацький інтерфейс.

Функціонал прогнозування реалізується за допомогою TensorFlow.js, потужної бібліотеки для машинного навчання, яка працює безпосередньо у браузері.

Як система зберігання даних найкраще підійде MongoDB, що надає можливості для зручного управління неструктурованими даними та легко масштабується.

Комплексне використання цих інструментів дозволяє створити сучасний, надійний і «розумний» застосунок із широкими можливостям.

1.4.1 Огляд платформи Node.js та фреймворка Express.js для розробки серверної частини

Node.js є середовищем виконання JavaScript, що дозволяє розробникам запускати код JavaScript на сервері. Express.js – це легкий та гнучкий

фреймворк для вебзастосунків на Node.js, який надає набір потужних функцій для створення вебзастосунків.

Переваги використання:

– використання єдиної мови програмування для розробки як клієнтської, так і серверної частин застосунку сприяє підвищенню ефективності розробки та спрощує обмін знаннями між членами команди;

– Node.js демонструє високу швидкість обробки запитів завдяки своїй неблокуючій архітектурі, що є критично важливим для застосунків з великою кількістю одночасних користувачів;

– наявність великої кількості готових бібліотек та інструментів (через менеджер пакетів NPM) значно прискорює процес розробки та розширює функціональні можливості застосунку;

– фреймворк Express.js надає необхідний мінімум інструментів для швидкого створення API та вебзастосунків, залишаючи розробникам свободу у виборі архітектурних рішень.

Потенційні недоліки: обробка інтенсивних до процесора операцій може призвести до блокування основного потоку, хоча існують механізми та методи для обходу цього обмеження.

Node.js у тісному поєднанні з Express.js є ефективним та сучасним рішенням для розробки серверної частини вебзастосунку обліку фінансів, забезпечуючи високу продуктивність, масштабованість та зручність розробки завдяки використанню JavaScript на всіх рівнях.

1.4.2 Огляд мови програмування JavaScript та бібліотеки React

JavaScript є універсальною та популярною мовою програмування. Насамперед, вона відома для фронтенд-розробки, що забезпечує інтерактивне та динамічне наповнення для вебсторінок. React, у свою чергу, це популярна бібліотека JavaScript для створення інтерфейсів користувача або UI

компонентів. Вона використовує компонентно-орієнтовану архітектуру та віртуальний DOM для ефективних оновлень та рендерингу. Компонентно-орієнтована архітектура React сприяє повторному використанню коду та полегшує управління складними інтерфейсами, що є достатньо корисним та практичним для багатофункціональних фінансових застосунків. Інструмент для аналізу особистих фінансів матиме різні розділи, а структура React допомагає організувати та підтримувати кодову базу.

Переваги використання:

– React дозволяє створювати чутливі та динамічні інтерфейси, що забезпечують високий рівень взаємодії із користувачем;

– розбиття інтерфейсу на незалежні компоненти сприяє повторному використанню кода, полегшує розробку, та покращує підтримку складних структур;

– використання віртуальної DOM-моделі підвищує продуктивність шляхом оптимізації оновлень інтерфейсу;

– React має активну спільноту розробників та велику кількість доступних бібліотек та інструментів, що збільшує комфорт самої розробки.

Потенційні недоліки: залежно від складності та використовуваних бібліотек, кінцевий бандл JavaScript може стати великим, що потенційно впливає на час початкового завантаження.

1.4.3 Огляд бібліотеки TensorFlow.js для реалізації функціоналу прогнозування

TensorFlow.js являє собою бібліотеку, розробленою Google, з відкритим вихідним кодом, яка дозволяє розробникам запускати моделі машинного навчання в браузері або в Node.js. Вона надає інструменти для навчання та розгортання ML моделей безпосередньо у вебзастосунку. Інтеграція можливостей машинного навчання у вебзастосунок для особистих фінансів

забезпечить розширений функціонал прогнозованого бюджетування та персоналізованого фінансового «менторства». Використовуючи історичні фінансові дані, моделі можуть виявляти закономірності та тенденції, які будуть використовуватися для надання користувачам більш інтелектуальних порад.

Але з іншого боку розробка та навчання ефективних моделей прогнозування вимагає значних обсягів якісних даних.

1.4.4 Огляд системи керування базами даних MongoDB

MongoDB – це популярна NoSQL база даних, яка зберігає дані у гнучких, схожих на JSON, документах. Вона відома своєю масштабованістю, високою продуктивністю та простотою використання. Гнучка схема цієї бази даних дозволяє легко адаптуватися до змінних вимог до даних, що може бути особливо корисним у застосунку для особистих фінансів, де користувачі можуть захотіти відстежувати різні типи доходів, витрат, цілей або фінансових категорій з часом [1]. На відміну від реляційних баз даних із жорстко визначеними схемами, ця БД дозволяє вміщувати такі варіації без необхідності складних міграцій або змін у структурі. Це робить її зручною для швидкої розробки та гнучкого масштабування функціоналу в залежності від потреб користувача.

1.5 Постановка задачі

Об’єктом роботи є процес створення та забезпечення функціонування вебзастосунку для централізованого обліку особистих фінансів та їхнього аналізу.

Метою роботи є розробка інтуїтивно зрозумілого та ефективного вебсервісу, що надасть користувачам можливість контролювати свої фінансові потоки та отримувати прогнози щодо майбутнього бюджету.

Для досягнення мети необхідно вирішити такі завдання:

– проаналізувати потреби користувачів у сфері особистого фінансового планування;

– розробити інтерфейс для швидкої та зручної реєстрації фінансових транзакцій;

– реалізувати функціонал обліку доходів і витрат із поділом на категорії;

– створити інструменти для формування бюджетів і встановлення лімітів за категоріями витрат;

– забезпечити адаптивний дизайн для роботи на різних пристроях;

– додати інтеграцію з банківськими API для автоматичного імпорту транзакцій;

– реалізувати фінансове прогнозування на основі зібраної статистики.

Для створення застосунку обрано стек MERN (MongoDB, Express.js, React, Node.js), оскільки він надає низку суттєвих переваг [2]. Повна розробка на JavaScript забезпечує єдність технологій на всіх рівнях застосунку, що значно спрощує взаємодію між фронтендом і бекендом. Використання Node.js і React дозволяє досягти високої продуктивності. Express.js і React мають потужні та розвинені екосистеми, що сприяють швидкій розробці завдяки великій кількості доступних бібліотек і готових рішень.

Архітектура застосунку базується на принципах RESTful API і це забезпечує наочне розділення між клієнтською та серверною частинами. Усе це дозволяє в майбутньому легко розширювати функціонал та інтегрувати додаткові сервіси без суттєвих змін в існуючому коді. MongoDB, як NoSQL база даних, забезпечує гнучкість та швидкість у роботі з різноманітними структурами фінансових даних та легко масштабується під зростаючі потреби користувачів. Документо-орієнтована модель даних MongoDB ідеально підходить для зберігання складних фінансових записів з різною кількістю полів

та категорій. Також увага приділяється безпеці даних користувачів. Планується реалізація багаторівневої системи захисту, включаючи шифрування чутливих даних, використання JSON Web Tokens для автентифікації, валідацію всіх вхідних даних на серверній стороні та захист від основних типів вебатак.

Прогнозування фінансових показників планується реалізувати з використанням алгоритмів машинного навчання, що аналізуватимуть історичні дані користувача та виявлятимуть закономірності у витратах і доходах. Це дозволить надавати персоналізовані рекомендації щодо оптимізації особистого бюджету та досягнення фінансових цілей.

Планується підключення до API банківських установ для автоматичного імпорту транзакцій, що дозволить користувачам синхронізувати свої рахунки та отримувати актуальну інформацію про фінансові операції в режимі реального часу. Також передбачається інтеграція з курсами валют через відкриті API для надання точної інформації про обмінні курси та автоматичного перерахунку сум у різних валютах.

Розгортання застосунку планується здійснити з використанням хмарних сервісів, що забезпечить високу доступність та масштабованість системи. Передбачається використання контейнеризації через Docker для забезпечення консистентності середовища розробки та продакшену. Система моніторингу та логування дозволить відстежувати продуктивність застосунку в реальному часі, виявляти потенційні проблеми та оптимізувати роботу системи на основі аналізу користувацької поведінки.

Продуктивность у застосунку буде поліпшена через впровадження сучасних технік оптимізації. Планується використання серверного кешування для зменшення навантаження на базу даних, клієнтського кешування для прискорення завантаження сторінок, та ледачого завантаження для компонентів інтерфейсу.

2 ВИБІР ТЕХНОЛОГІЙ ТА ПРИНЦИПІВ ПРОЄКТУВАННЯ

2.1 Перелік функціональних можливостей вебзастосунку для обліку особистих фінансів

Вебзастосунок для обліку та аналізу особистих фінансів включає клієнтську і серверну частини, а також базу даних, що дозволяє реалізувати основні можливості для ефективного фінансового планування. Основний функціонал системи передбачає:

– реєстрацію та авторизацію користувачів з урахуванням вимог безпеки та конфіденційності;

– ведення обліку доходів і витрат із можливістю їх редагування та розподілу за категоріями;

– формування персональних бюджетів, встановлення фінансових цілей та відстеження їх досягнення;

– перегляд узагальненої фінансової інформації за допомогою інтерактивної аналітичної панелі;

– отримання прогнозів щодо майбутнього фінансового стану на основі попередньої активності користувача;

– автоматичне оновлення інформації про транзакції для зменшення ручної роботи користувача;

– адаптивний інтерфейс, що забезпечує комфортну взаємодію з платформою на різних пристроях.

На відміну від розглянутих аналогів, запропонований вебзастосунок поєднує в собі функціональність, гнучкість і використання сучасних технологій. Його унікальність полягає у глибокій інтеграції функцій прогнозування та цілепокладання, які дозволяють не лише відображати фінансовий стан, але й активно впливати на фінансову поведінку кожного окремого користувача.

Додатково реалізована система збереження історії змін та перегляду фінансових періодів, що сприяє аналізу динаміки фінансів у довгостроковій перспективі. Це робить застосунок не просто засобом обліку, а повноцінною платформою для прийняття обґрунтованих фінансових рішень.

2.2 Архітектура вебзастосунку

Архітектура вебзастосунку для обліку та аналізу особистих фінансів спроєктована за класичною трирівневою моделлю, що забезпечує чітке розділення відповідальності між різними частинами системи та сприяє її масштабованості та підтримуваності. Кожен рівень виконує свою специфічну роль у забезпеченні функціональності спільного інструмента [3].

Клієнтська частина вебзастосунку є фундаментальною складовою системи, що забезпечує безпосередню взаємодію з користувачем. Вона розроблена з використанням бібліотеки React, яка дозволяє створювати швидкий та динамічний інтерфейс із можливістю миттєвого оновлення контенту без повного перезавантаження сторінки [4]. Це суттєво покращує користувацький досвід та сприяє підвищенню зручності при роботі з додатком.

Для управління станом застосунку використовується Redux, що забезпечує централізоване зберігання даних і дозволяє ефективно координувати взаємодію між компонентами. Цей підхід спрощує обробку складних сценаріїв користувацької активності, забезпечує передбачувану поведінку застосунку та покращує його підтримку й масштабування.

Функціональність клієнтської частини охоплює низку ключових розділів, серед яких сторінки реєстрації та авторизації користувача, дашборд з візуалізацією фінансових даних, інтерфейси для обліку доходів і витрат, налаштування профілю користувача, а також перегляд аналітики та прогнозів. Ці елементи забезпечують зручну взаємодію користувача з системою та дозволяють ефективно контролювати особисті фінанси. Завдяки інтуїтивному

інтерфейсу та наочному відображенню даних, користувач може швидко орієнтуватися у своїх фінансових операціях та приймати обґрунтовані рішення.

Клієнтська частина здійснює обмін даними з серверною частиною через REST API, що дозволяє надсилати запити для отримання, оновлення та збереження даних. Отримана інформація обробляється та відображається у зручному форматі.

Важливим аспектом є реалізація адаптивного дизайну, що гарантує коректне відображення інтерфейсу на різних пристроях.

Серверна частина виконує роль центрального логічного ядра вебзастосунку, обробляючи запити, що надходять від клієнтської частини, та забезпечуючи реалізацію основної бізнес-логіки. Вона побудована на платформі Node.js. Як основний фреймворк для побудови API використовується Express.js, який завдяки своїй легкості та гнучкості спрощує створення маршрутизації та обробку HTTP-запитів.

Програмна частина на стороні сервера забезпечує критично важливі функції, серед яких:

– аутентифікація та авторизація користувачів;
– обробка фінансових транзакцій;
– керування бюджетами;
– збір та агрегація даних;
– реалізація функцій прогнозування на основі даних користувача.

Крім цього, сервер підтримує можливість автоматичного імпорту фінансових транзакцій з зовнішніх джерел, що дозволяє зменшити навантаження на користувача та забезпечити більшу актуальність даних.

Особливу увагу приділено безпеці даних: реалізовано механізми захисту персональної та фінансової інформації, зокрема за допомогою токенів, хешування паролів та захищених маршрутів. Серверна частина працює як надійний посередник між інтерфейсом користувача і базою даних, забезпечуючи цілісність, безперервність і логічну узгодженість усіх операцій, що здійснюються у системі.

Для забезпечення надійного та гнучкого зберігання даних у вебзастосунку використовується документоорієнтована база даних MongoDB.

У MongoDB зберігається основна інформація:

– облікові записи користувачів;
– дані про доходи та витрати;
– налаштування профілю кожного користувача;
– історія транзакцій;
– дані про фінансові цілі.

Завдяки високій продуктивності MongoDB здатна ефективно обробляти великі обсяги даних у режимі реального часу, забезпечуючи швидкий та стабільний доступ [5].

2.3 Проєктування бази даних

Під час проєктування бази даних для вебзастосунку було враховано специфіку предметної області та необхідність забезпечення гнучкої, масштабованої та ефективної моделі зберігання даних. Архітектура бази даних побудована таким чином, щоб підтримувати динамічну структуру інформації, забезпечувати швидкий доступ до актуальних даних та можливість подальшого розширення функціональності без суттєвих змін у структурі сховища. Концепція організації даних орієнтована на простоту обробки, логічну цілісність і можливість гнучкого адаптування до різноманітних сценаріїв використання системи.

Окрім цього особливу увагу було приділено логічному структуруванню сутностей та взаємозв’язків між ними. Це забезпечує ефективність під час виконання типових операцій, таких як збереження, пошук, фільтрація та агрегація даних. Проєктування бази даних також враховує принципи безпеки та консистентності інформації, що зберігається, із можливістю подальшої оптимізації для роботи з великими обсягами інформації. Таким чином було

створено надійний фундамент для стабільної роботи вебзастосунку, забезпечуючи одночасно продуктивність, масштабованість і гнучкість у подальшому розвитку системи.

2.3.1 Опис сутностей та їхніх атрибутів

У структурі бази даних вебзастосунку зберігаються дані, що відображають ключові аспекти взаємодії користувача із системою. Ці дані охоплюють різні інформаційні площини: від загальних відомостей до аналітичних та інтеграційних механізмів. Щоб забезпечити цілісність, логічність і масштабованість інформаційної моделі, важливо на концептуальному рівні окреслити основні структурні одиниці даних, з якими працює система.

Кожна сутність являє собою певну цілісну одиницю інформації, яка має унікальні ознаки, відносну автономність і здатність до встановлення логічних зв’язків з іншими сутностями. Наприклад, незалежно від контексту застосунку, сутністю може виступати будь-який елемент, що має чітку ідентифікацію, описується певним набором атрибутів та виконує конкретну роль у структурі даних [6, 7].

Встановлення зв’язків між сутностями відбувається через систему посилань, що дозволяє ефективно структурувати складні фінансові операції та забезпечувати цілісність даних при їх модифікації. Кожна сутність також має власний життєвий цикл з чітко визначеними станами та правилами переходу між ними, що гарантує коректність обробки бізнес-логіки застосунку.

Формалізація сутностей є основою побудови цілісної моделі даних, що дозволяє системі залишатися узгодженою, передбачуваною та здатною до ефективного оброблення запитів користувачів. Вона також створює передумови для подальшого розширення функціоналу.

Сутності у базі даних описані у таблицях 2.1 – 2.6.

Таблиця 2.1 – Опис сутностей колекції users

Назва поля	**Тип даних**	**Опис сутності**
1	**2**	**3**
_id	ObjectId	Унікальний ідентифікатор користувача в базі даних.
name	String	Ім’я користувача (за замовчуванням порожній рядок).
email	String	Електронна адреса користувача, використовується для входу в систему, повинна бути унікальною та відповідати шаблону email.
password	String	Хешований пароль користувача для безпечного зберігання.
currency	String	Обрана користувачем валюта для відображення фінансових даних (за замовчуванням «UAH», можливі значення: «UAH», «USD», «EUR»).
lastCurrencyUpdate	Date	Дата останнього оновлення курсу валюти (за замовчуванням поточна дата).
balance	Number	Поточний баланс користувача

Продовження таблиці 2.1

1	2	3
lastBalanceUpdate	Date	Дата останнього оновлення балансу користувача (за замовчуванням поточна дата).
createdAt	Date	Дата та час створення запису про користувача.
updatedAt	Date	Дата останнього оновлення запису про користувача.

Таблиця 2.2 – Опис сутностей колекції transactions

Назва поля	Тип даних	Опис сутності
1	2	3
_id	ObjectId	Унікальний ідентифікатор транзакції в базі даних.
userId	ObjectId	Ідентифікатор користувача, якому належить транзакція (посилання на колекцію users).
type	String	Тип транзакції («income» – дохід, «expense» – витрата), обов’язкове поле.

Продовження таблиці 2.2

1	2	3
createdAt	Date	Дата та час створення запису про транзакцію.
updatedAt	Date	Дата останнього оновлення запису про транзакцію.
amount	Number	Сума транзакції, обов’язкове поле.
category	String	Категорія транзакції, обов’язкове поле (наприклад, «Продукти», «Транспорт», «Зарплата»).
description	String	Додатковий опис транзакції
date	Date	Дата здійснення транзакції (за замовчуванням поточна дата).
source	String	Джерело транзакції («manual» - введена вручну, «monobank» - імпортована з MonoBank), за замовчуванням «manual».

Таблиця 2.3 – Опис сутностей колекції forecast

Назва поля	**Тип даних**	**Опис сутності**
1	**2**	**3**
_id	ObjectId	Унікальний ідентифікатор прогнозу в базі даних.

Продовження таблиці 2.3

1	2	3
userId	ObjectId	Ідентифікатор користувача, для якого створено прогноз (посилання на колекцію users).
budgetForecasts	Array	Масив прогнозів бюджету на різні періоди (містить дату, прогнозовані доходи/витрати/баланс, прогнози за категоріями, рівень впевненості та оцінку ризику).
quickEstimates	Array	Масив швидких прогнозів на найближчі місяці для швидкого відображення (містить місяць, прогнозовані доходи/витрати/баланс, рівень впевненості та дату розрахунку).
thirtyDayBudget	Object	Прогноз бюджету на наступні 30 днів (містить прогнозовані доходи/витрати/баланс, рівень впевненості та дату розрахунку).
calculationStatus	String	Статус процесу розрахунку прогнозу («pending», «in_progress», «completed», «failed»).
calculationProgress	Number	Прогрес розрахунку прогнозу

Продовження таблиці 2.3

1	2	3
goalForecast	Object	Прогноз досягнення фінансової цілі (містить ідентифікатор цілі, очікувану/оптимістичну/песимістичну кількість місяців, щомісячні заощадження, ймовірність досягнення, фактори ризику та швидку оцінку).
lastUpdated	Date	Дата останнього оновлення прогнозу.
forecastMethod	String	Метод, що використовувався для створення прогнозу (за замовчуванням «Advanced-AI-Enhanced-v4»).
confidenceScore	Number	Загальний показник впевненості прогнозу (від 0 до 100).
calculationTime	Number	Час, витрачений на розрахунок прогнозу (у мілісекундах).
dataQuality	Object	Показники якості даних, що використовувалися для прогнозування (кількість транзакцій, кількість місяців з даними, повнота даних у відсотках).
createdAt	Date	Дата та час створення запису про прогноз.
updatedAt	Date	Дата останнього оновлення запису про прогноз.

Таблиця 2.4 – Опис сутностей колекції goals

Назва поля	**Тип даних**	**Опис сутності**
1	**2**	**3**
_id	ObjectId	Унікальний ідентифікатор фінансової цілі в базі даних.
userId	ObjectId	Ідентифікатор користувача, який поставив ціль (посилання на колекцію users).
title	String	Назва фінансової цілі, обов’язкове поле.
targetAmount	Number	Цільова сума, яку користувач хоче досягти, обов’язкове поле, мінімальне значення 0.
currentAmount	Number	Поточна накопичена сума для досягнення цілі (за замовчуванням 0, мінімальне значення 0).
deadline	Date	Кінцевий термін досягнення цілі, обов’язкове поле.

Продовження таблиці 2.4

1	2	3
isActive	Boolean	Статус активності цілі (true – активна, false – неактивна, за замовчуванням false).
highestAmount	Number	Найвища сума, яка була досягнута у процесі накопичення цілі (за замовчуванням 0).
createdAt	Date	Дата та час створення запису про фінансову ціль.
updatedAt	Date	Дата та час останнього оновлення запису про фінансову ціль.

Таблиця 2.5 – Опис сутностей колекції monobanktokens

Назва поля	**Тип даних**	**Опис сутності**
1	**2**	**3**
_id	ObjectId	Унікальний ідентифікатор запису про токен MonoBank в базі даних.
userId	ObjectId	Ідентифікатор користувача, якому належить токен

Продовження таблиці 2.5

1	2	3
encryptedToken	String	Зашифрований токен доступу до MonoBank API, обов’язкове поле (шифрування забезпечує безпеку).
iv	String	Унікальний вектор ініціалізації, що використовується для шифрування токена, обов’язкове поле.
lastSync	Date	Дата та час останньої успішної синхронізації транзакцій з MonoBank (може бути null, якщо синхронізація ще не проводилась).
accounts	Array	Масив об’єктів, що містять інформацію про рахунки користувача в MonoBank
createdAt	Date	Дата та час створення запису про токен MonoBank.
updatedAt	Date	Дата та час останнього оновлення запису про токен MonoBank.

Таблиця 2.6 – Опис сутностей колекції sessions

Назва поля	**Тип даних**	**Опис сутності**
1	**2**	**3**
_id	ObjectId	Унікальний ідентифікатор сесії користувача в базі даних.
userId	ObjectId	Ідентифікатор користувача, якому належить сесія (посилання на колекцію users), обов’язкове поле.
accessToken	String	Токен доступу, що використовується для аутентифікації запитів користувача, обов’язкове поле.
refreshToken	String	Токен оновлення, що використовується для отримання нового токена доступу без повторної авторизації, обов’язкове поле.
accessTokenValidUntil	Date	Дата та час закінчення терміну дії токена доступу, обов’язкове поле.

Продовження таблиці 2.6

1	2	3
refreshTokenValidUntil	Date	Дата та час закінчення терміну дії токена оновлення, обов’язкове поле.
createdAt	Date	Дата та час створення запису про сесію.

2.3.2 Схема бази даних

Схема бази даних відображає логічну структуру зберігання інформації, а також взаємозв’язки між основними сутностями, що забезпечують функціонування системи. Основною метою проєктування було створення гнучкої, масштабованої та безпечної архітектури, здатної ефективно обробляти великі обсяги фінансових даних, надаючи користувачам зручні інструменти для ведення бюджету, планування витрат і досягнення фінансових цілей.

Центральною ланкою всієї структури є колекція users, яка містить дані про зареєстрованих користувачів застосунку. Ця колекція виконує роль основи для встановлення зв’язків з іншими сутностями, кожна з яких зберігає специфічну інформацію, пов’язану з конкретним користувачем. Для забезпечення безпеки, паролі зберігаються у вигляді хешів, що виключає можливість їх прямого отримання у разі компрометації даних.

Також важливу роль у функціоналі застосунку відіграє колекція transactions, яка містить фінансові операції користувачів. Кожен запис пов’язаний із користувачем через поле userId, що дозволяє здійснювати персоналізований облік доходів та витрат. Для аналітики та організації фінансів транзакції класифікуються за категоріями, які наразі зберігаються у

вигляді рядкового поля, однак структура передбачає можливість винесення категорій в окрему колекцію для ієрархічної класифікації й більш детального аналізу фінансових звичок у майбутньому.

З метою розширення можливостей фінансового планування у схемі передбачені додаткові колекції. Зокрема, колекція goals використовується для зберігання фінансових цілей користувачів, таких як накопичення певної суми до заданої дати чи зменшення витрат у певній категорії. У комбінації із цим функціонує колекція forecast, яка містить прогнозовані дані, такі як очікувані доходи та витрати, а також передбачення досягнення поставлених цілей. Знов таки, обидві ці колекції тісно пов’язані з користувачем через поле userId, що дозволяє реалізовувати персоналізовану аналітику та динамічні поради щодо фінансового управління.

Однією з ключових функціональних можливостей застосунку є інтеграція з MonoBank API для автоматичного імпорту транзакцій. Для цього у схемі реалізовано колекцію monobanktokens, яка зберігає зашифровані токени доступу до банківських рахунків користувача. Задля безпеки кожен токен зберігається у зашифрованому вигляді, що гарантує захист чутливих даних навіть у разі витоку інформації. Крім того, ця колекція містить допоміжну інформацію про рахунки, яка дозволяє пов’язувати фінансові події з конкретним джерелом. Для підтримки механізмів автентифікації та управління сесіями у базі даних присутня колекція sessions. Вона зберігає дані про активні сесії користувачів, включаючи токени доступу, токени оновлення та терміни їх дії. Ці дані дозволяють забезпечити безпечну та зручну роботу з застосунком, автоматично продовжуючи сеанси, не змушуючи користувача повторно входити в систему [8, 9].

Важливим питанням під час проєктування схеми стало забезпечення безпеки та конфіденційності. Зокрема, воно було вирішено тим, що всі критично важливі дані такі як паролі, токени доступу до банківських рахунків, всі вони зберігаються з використанням надійних алгоритмів хешування або шифрування. Крім того, взаємозв’язки між колекціями реалізовані через

ідентифікатор користувача, що дозволяє ефективно виконувати запити та об’єднувати інформацію, зберігаючи при цьому чітке логічне розмежування між даними різних користувачів. Додатково реалізовані обмеження доступу на рівні API, які перевіряють авторизацію кожного запиту за допомогою токена, що унеможливлює несанкціоноване отримання чи зміну даних іншого користувача.

Архітектура бази даних була спроєктована з урахуванням можливого розширення функціональності в майбутньому. Наприклад, у разі потреби реалізації детального поділу категорій транзакцій або додавання нових типів фінансових аналітик, можуть бути створені нові колекції або додані додаткові атрибути до наявних. Це робить систему гнучкою і адаптивною до майбутніх потреб користувачів. Також завдяки використанню MongoDB можливо масштабувати сховище горизонтально, що є перевагою у випадку збільшення кількості активних користувачів і обсягу збережених даних.

Структура бази даних підтримує версіонування схем, що дозволяє безболісно оновлювати формати даних без втрати інформації існуючих користувачів. Реалізовано систему індексації для оптимізації швидкості виконання складних запитів з фільтрацією за датами, категоріями та сумами транзакцій, що особливо важливо при обробці великих обсягів фінансових даних.

Також забезпечено цілісність даних і їх захист. Реалізовано механізми валідації на рівні моделі, що запобігає збереженню некоректної або неповної інформації. Для підвищення безпеки чутливі дані, зокрема облікові записи користувачів, зберігаються у зашифрованому вигляді. Окрім цього передбачено регулярне резервне копіювання бази даних, що дозволяє мінімізувати ризики втрати інформації у разі збоїв у системі. А Система логування змін забезпечує можливість відстеження історії дій користувачів та адміністратора, що сприяє прозорості й полегшує аудит.

Наочно продемонстровані взаємозв’язки між основними колекціями та їхні атрибути у візуальному представленні схеми бази даних (рис. 2.1).

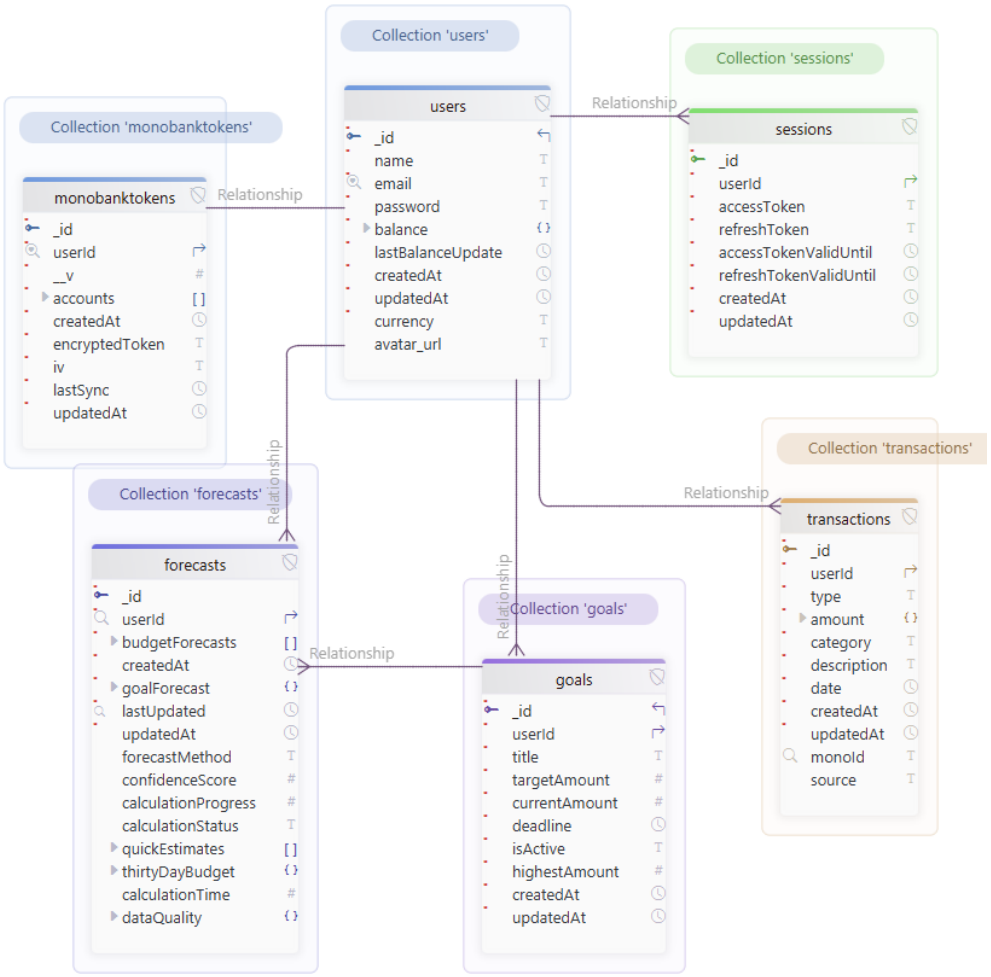

Рисунок 2.1 – Опис структури таблиць

2.3.3 Вибір СУБД

Вибір системи керування базами даних є одним із ключових архітектурних рішень при створенні вебзастосунку для обліку та аналізу особистих фінансів. У цьому проєкті було обрано MongoDB – документоорієнтовану NoSQL базу даних, що ідеально відповідає як поточним технічним вимогам, так і майбутнім завданням, які передбачає розвиток застосунку. Рішення використовувати саме цю СУБД є стратегічно виваженим кроком, що забезпечує гнучкість, продуктивність і масштабованість, необхідні для успішного функціонування сервісу.

MongoDB відрізняється від традиційних реляційних СУБД насамперед своєю гнучкою структурою зберігання даних. Замість таблиць із фіксованими схемами, дані у MongoDB зберігаються у вигляді документів формату BSON, бінарного представлення JSON. Це дозволяє зберігати в одній колекції об’єкти з різною структурою, що є надзвичайно важливим для вебзастосунку, що постійно обробляє фінансові дані. Оскільки типи транзакцій можуть варіюватися, а нові функціональні можливості додаватися без попереднього визначення суворої схеми, ця база даних дозволяє уникати складних міграцій бази даних, які часто виникають у реляційних СУБД при зміні структури.

Вагомою перевагою є її здатність до горизонтального масштабування. Завдяки архітектурі з підтримкою шардування, дані можуть бути розподілені між численними серверами, що дозволяє зберігати й обробляти великі обсяги інформації без зниження продуктивності. Це надзвичайно важливо для вебзастосунку такого типу, який потенційно може обслуговувати тисячі або навіть мільйони користувачів, кожен із яких генерує численні фінансові записи.

У рамках технологічного стека MERN що використовується для розробки, MongoDB органічно поєднується з іншими компонентами. Завдяки уніфікованому використанню JavaScript на фронтенді, у серверній частині та в базі даних, забезпечується цілісність і швидкість розробки, мінімізуються помилки трансформації даних між компонентами застосунку. Зберігання транзакцій з різними атрибутами, створення персоналізованих категорій витрат і доходів, підтримка прогнозних моделей зі змінною структурою, усе це можливо завдяки гнучкості цієї СУБД.

Підсумовуючи, вибір MongoDB для вебзастосунку для обліку та аналізу особистих фінансів є обґрунтованим з погляду гнучкості, масштабованості, зручності розробки та відповідності сучасним вимогам до динамічних вебрішень. Ця технологія не лише оптимізує процеси зберігання та обробки даних, а й забезпечує надійну основу для подальшого розширення функціональності інструменту.

2.4 Проєктування серверної частини

Серверна частина вебзастосунку виконує основну роль у забезпеченні його функціональності, стабільності та безпеки. Вона виступає посередником між користувачем і базою даних, обробляє запити, реалізує бізнес-логіку і відповідає за управління ресурсами та даними. Під час проєктування серверної частини більша частина часу приділяється структурованій організації коду, чіткому розділенню відповідальностей між модулями, а також реалізації ефективних і безпечних механізмів взаємодії з клієнтською частиною.

До основних функцій серверної частини належить обробка запитів користувачів, управління сесіями, валідація вхідних та вихідних даних, виконання обчислень згідно з правилами застосунку, а також зберігання та обробка інформації в базі даних. Особливу увагу приділено механізмам автентифікації та авторизації, що дозволяють ідентифікувати користувачів та обмежити доступ до захищених ресурсів. Сервер також може інтегруватися з зовнішніми API або мікросервісами. Ретельне проєктування серверної частини дозволяє досягти високої продуктивності, надійності й адаптивності вебзастосунку [10, 11].

У процесі реалізації серверної логіки зазвичай застосовуються сучасні фреймворки, які спрощують створення REST API, полегшують обробку запитів та підключення до бази даних. Такий підхід сприяє підтримці модульності, повторного використання коду й спрощує тестування функціональності. Крім того, дотримання принципів безпеки, є невід’ємною частиною будування архітектури.

2.4.1 Вибір технологій для розробки серверної частини

Для реалізації серверної частини вебзастосунку було обрано середовище Node.js разом з вебфреймворком Express.js, що є надійною платформою для

створення REST API. Express.js надає зручні засоби для організації маршрутизації, обробки запитів та проміжного програмного забезпечення, що забезпечує стабільність і гнучкість під час розробки та використання. Підключення змінних середовища реалізується через бібліотеку dotenv, що дозволяє зберігати конфіденційну інформацію окремо від основного коду.

Зберігання даних здійснюється за допомогою бібліотеки mongoose, яка надає зручні інтерфейси для моделювання структури документів, валідації даних і виконання CRUD-операцій. Авторизація та автентифікація і хешування паролів реалізуються за допомогою bcrypt та cookie-parser, який обробляє куки користувача. Для захисту міждоменної взаємодії застосовується cors, що забезпечує обробку запитів з клієнтської частини, розміщеної на іншому домені.

Для побудови валідації вхідних даних у запитах використовується бібліотека joi, що дозволяє створювати надійні та прості схеми перевірки. Зовнішні API, включно з банківськими сервісами, інтегруються за допомогою axios, що забезпечує швидкий та зрозумілий механізм відправлення HTTP-запитів. У разі використання Google-авторизації, до проєкту також включено бібліотеку google-auth-library, яка дозволяє реалізувати OAuth 2.0.

У якості логування застосовується зв’язка pino та pino-pretty, яка забезпечує ефективне ведення журналу подій з можливістю зручного форматування для аналізу. Для роботи з датами й часом використовується бібліотека date-fns. Обробка помилок стандартизована через http-errors, що дозволяє створювати HTTP-помилки з відповідними статус-кодами.

Особливістю серверної частини даного вебзастосунку є реалізація функціоналу прогнозування фінансових показників користувача за допомогою бібліотеки tensorflow/tfjs. Завдяки використанню моделей машинного навчання стало можливим здійснювати інтелектуальне передбачення майбутнього балансу, планованих доходів, витрат і досягнення фінансових цілей. Побудовані моделі аналізують історичні дані користувача та формують прогнози, які допомагають приймати обґрунтовані фінансові рішення.

2.4.2 Опис компонентів серверної частини

Серверна частина вебзастосунку виконує роль ядра всієї системи, вона обробляє запити від клієнтської частини, виконує бізнес-логіку, взаємодіє з базою даних та зовнішніми сервісами, забезпечує авторизацію й автентифікацію користувачів, валідує введені дані та формує відповіді у відповідному форматі. Код серверу структуровано відповідно до принципів чистої архітектури з чітким розділенням відповідальностей між компонентами: маршрутизаторами, контролерами, сервісами, моделями, middleware, утилітами тощо. Це дозволяє легко масштабувати проєкт, імплементувати нові функції та підтримувати існуючі без ускладнень [12, 13].

Нижче детально описані основні програмні компоненти серверної частини:

– маршрутизація (routers). Файли у папці routers відповідають за обробку HTTP-запитів до відповідних частин системи. Кожен модуль (наприклад, balance.js, goal.js, monobank.js) містить маршрути для відповідних функцій таких як, обробка даних про баланс, цілі, транзакції, тощо. Вони передають запити до відповідних контролерів;

– контролери (controllers). Ці компоненти реалізують логіку обробки запитів. Вони виступають посередниками між маршрутизаторами та сервісами. Наприклад, контролер forecast.js обробляє запити, пов’язані з прогнозуванням витрат і балансу, викликаючи відповідний сервіс з TensorFlow;

– сервіси (services). Сервіси містять бізнес-логіку, що реалізує основну функціональність застосунку. Наприклад, AI_ForecastService.js відповідає за прогнозування майбутнього фінансового стану користувача на основі історичних даних. Інші сервіси, як goal.js або transactions.js, реалізують CRUD-операції та обчислення;

– моделі (models). Вони описують структуру документів для таких сутностей, як User, Transaction, Goal, Session, Forecast тощо. Саме моделі забезпечують взаємодію з БД та її цілісність;

– проміжне програмне забезпечення (middleware). Папка middlewares містить функції для обробки запитів перед тим, як вони дійдуть до маршрутизаторів. Наприклад, authenticate.js перевіряє, чи має користувач дійсний токен доступу, перш ніж дозволити доступ до захищених маршрутів;

– валідація (validation). Файли у папці validation відповідають за перевірку вхідних даних від користувача. Вони використовують бібліотеку Joi для перевірки, чи відповідають надані дані очікуваним схемам. Це важливо для безпеки та надійності застосунку;

– утиліти (utils). Допоміжні функції, такі як googleOAuth2.js для автентифікації через Google, categoryMapper.js для роботи з категоріями транзакцій, або ctrlWrapper.js для обгортання асинхронних контролерів із обробкою помилок, розміщені у папці utils. Вони використовуються повторно в різних частинах системи;

– інтеграція штучного інтелекту. Модуль AI_ForecastService.js реалізує прогнозування майбутнього балансу користувача. Він використовує TensorFlow.js для створення моделей машинного навчання, що аналізують попередні доходи, витрати та фінансові цілі. На основі цих даних система може формувати прогнозовані значення майбутнього стану бюджету, надаючи користувачу корисні фінансові підказки;

– інтеграція із зовнішніми сервісами. Реалізована можливість підключення користувачів до особистого кабінету Monobank шляхом передачі токена. Після авторизації сервер отримує доступ до історії транзакцій користувача та може синхронізувати ці дані з власною базою. Для цього призначено окремі сервіси (monobank.js), контролери (monobank.js) і моделі (MonobankToken.js). Також застосунок підтримує авторизацію через обліковий запис Google. Інтеграція реалізована через бібліотеку google-auth-library і допоміжний файл googleOAuth2.js;

– конфігурації та інші компоненти. Dotenv використовується для збереження чутливих даних у .env файлі (ключі, URI), pino та pino-pretty забезпечують ефективне логування, cookie-parser оброблює cookies у запитах, http-errors генерує HTTP-помилки.

2.4.3 Функціональність API серверної частини

Серверна частина програмного комплексу реалізована за допомогою архітектури REST API, що забезпечує гнучкий та стандартизований інтерфейс взаємодії між клієнтськими додатками та сервером.

Реалізація надає наступні ендпоінти, згруповані за функціональним призначенням зазначені у таблицях 2.7 – 2.13.

Таблиця 2.7 – Аутентифікація та авторизація (/auth)

Метод	**Ендпоінт**	**Функціональність**	**Валідація**
POST	/register	Реєстрація нового користувача	Валідація тіла запиту за схемою registerUserSchema
POST	/login	Автентифікація користувача та видача токенів	Валідація тіла запиту за схемою loginUserSchema
POST	/logout	Завершення сеансу користувача	-
GET	/get-oauth-url	Отримання URL для OAuth авторизації через Google	-

Таблиця 2.8 – Управління користувачами (/users)

Метод	**Ендпоінт**	**Функціональність**	**Автентифікація**
GET	/current	Отримання даних поточного користувача	Потрібна
PATCH	/settings	Оновлення налаштувань користувача	Потрібна

Таблиця 2.9 – Управління балансом (/balance)

Метод	**Ендпоінт**	**Функціональність**	**Автентифікація**
GET	/	Отримання поточного балансу користувача	Потрібна
PUT	/	Оновлення поточного балансу	Потрібна

Таблиця 2.10 – Управління транзакціями (/transactions)

Метод	**Ендпоінт**	**Функціональність**	**Автентифікація**	**Валідація**
POST	/	Додавання нової транзакції	Потрібна	Валідація тіла запиту за схемою transactionValidationSchema
GET	/	Отримання списку транзакцій користувача	Потрібна	-

Таблиця 2.11 – Управління фінансовими цілями (/goal)

Метод	**Ендпоінт**	**Функціональність**	**Автентифікація**
POST	/	Створення нової фінансової цілі	Потрібна
GET	/	Отримання списку фінансових цілей	Потрібна
PATCH	/:goalId/activate	Активація фінансової цілі	Потрібна
PATCH	/:goalId/deactivate	Деактивація фінансової цілі	Потрібна
DELETE	/:goalId	Видалення фінансової цілі	Потрібна

Таблиця 2.12 – Прогнозування (/forecasts)

Метод	**Ендпоінт**	**Функціональність**	**Автентифікація**
GET	/	Отримання загальних прогнозів	Потрібна
GET	/quick	Отримання швидких прогнозів	Потрібна
GET	/categories	Отримання прогнозів за категоріями	Потрібна
GET	/goals	Отримання прогнозів	Потрібна

Таблиця 2.13 – Інтеграція з Монобанком (/api/monobank)

Метод	**Ендпоінт**	**Функціональність**	**Автентифікація**	**Валідація**
POST	/connect	Підключення API Монобанку	Потрібна	Валідація тіла запиту за схемою monobankTokenSchema
DELETE	/disconnect	Відключення API Монобанку	Потрібна	-
POST	/sync	Примусова синхронізація транзакцій	Потрібна	-
GET	/status	Перевірка статусу підключення до Монобанку	Потрібна	-

2.5 Проєктування клієнтської частини

Клієнтська частина вебзастосунку є відповідальною за безпосередню взаємодію з користувачем, відображення даних, отриманих із серверної частини, та ініціацію запитів до деяких API. Вона формує інтерфейс, через який користувачі взаємодіють із функціоналом застосунку, керують своїми даними, переглядають аналітику та отримують зворотний зв’язок. Основною метою при проєктуванні клієнтської частини є створення інтуїтивно зрозумілого, зручного та візуально привабливого інтерфейсу [14].

Під час проєктування клієнтської частини враховуються принципи адаптивності, модульності та повторного використання компонентів.

Вебінтерфейс реалізується з урахуванням структури даних, які передаються з серверної частини, що забезпечує ефективний зв’язок між усіма елементами застосунку. Застосовуються сучасні підходи до управління станом, маршрутизації та валідації форм.

Крім візуальної частини, клієнтська частина відповідає і за логіку взаємодії з користувачем, обробку подій, заповнення форм, динамічну зміну станів інтерфейсу тощо. Також вона забезпечує відображення повідомлень про помилки, завантаження даних у фоновому режимі, а також інтеграцію зі сторонніми сервісами через API [15 – 17]. Захисту користувацьких даних, забезпечується шляхом безпечної передачі токенів автентифікації.

2.5.1 Вибір технологій для розробки клієнтської частини

Основою проєкту є бібліотека React, яка дозволяє створювати динамічні, компонентні інтерфейси з реактивним оновленням даних. Завдяки декларативному підходу React сприяє зрозумілості коду та спрощує керування інтерфейсом користувача в залежності від стану застосунку.

Для керування станом на глобальному рівні використовується Redux Toolkit, це офіційний інструментарій для Redux, що забезпечує налаштування конфігурації та зменшує обсяг шаблонного коду. Разом із react-redux це дозволяє ефективно зв’язувати логіку стану з візуальними компонентами. Для збереження стану між сесіями застосовується redux-persist, який автоматично записує та відновлює стан з локального сховища браузера. Бібліотека axios, яка також згадувалися при проєктуванні бекенда, забезпечує зручний спосіб виконання HTTP-запитів до серверної частини, що використовується для отримання та надсилання фінансових даних користувача. React Router DOM дозволяє реалізувати маршрутизацію всередині застосунку на клієнті, забезпечуючи швидку навігацію між сторінками без перезавантаження. Для реалізації інтерактивних форм і валідації було обрано бібліотеку Formik у

поєднанні з Yup, що дозволяє будувати складні форми з гнучкими правилами перевірки введення. Це особливо корисно для форм реєстрації, входу, налаштувань профілю, створення фінансових цілей та введення транзакцій.

Для побудови графіків та аналітичних візуалізацій у застосунку використовується Recharts, що дозволяє інтегрувати інтерактивні діаграми та графіки з динамічним оновленням даних. Додаткові бібліотеки, такі як react-hot-toast, забезпечують зручне відображення повідомлень про події (наприклад, успішне збереження, помилки тощо), а swiper використовується для створення інтерактивних каруселей або слайдерів, що покращують UX.

З точки зору стилізації інтерфейсу, застосунок базується на normalize.css, що уніфікує стилі в різних браузерах. Також використовується lucide-react для додавання адаптивних іконок із сучасним та мінімалістичним зовнішнім виглядом [17].

Для збирання та запуску проєкту застосовується інструмент Vite, який забезпечує надзвичайно швидку розробку завдяки швидкому HMR та оптимізованій побудові. Плагін vitejs/plugin-react-swc використовується для ще більшої продуктивності за рахунок швидкої трансляції JSX [18].

Щодо забезпечення якості коду, використовуються ESLint разом з відповідними плагінами для React, що дозволяє виявляти помилки на ранніх етапах розробки.

2.5.2 Структура вебсторінок та компонентів інтерфейсу

Основу архітектури клієнтської частини становить розподіл коду на незалежні компоненти, які формують візуальні та функціональні блоки інтерфейсу. Усі ці компоненти згруповані в окрему директорію components, де кожен модуль виконує чітко визначене завдання, будучи потенційно незалежним від конкретного контексту сторінки.

В окремих підкаталогах зберігаються такі ключові елементи, як картки фінансового балансу, секції витрат, прогнозів, цілей і графіки транзакцій. Наприклад, BalanceCard, ExpensesCard, ForecastSection, GoalCard, TransactionChart та ExchangeRateCard відповідають за візуалізацію відповідних фінансових даних, забезпечуючи користувача стислою, але дуже інформативною інформацією. Для обробки більш складних дій, таких як відображення детального прогнозу або підключення банківського API, використовуються окремі модальні компоненти, як-от DetailedForecastModal та MonobankConnect. Компоненти ErrorBoundary та LoadingSpinner відповідають за обробку помилок та індикацію завантаження, що дозволяє забезпечити стабільність і зручність користування. Завдяки використанню бібліотеки Formik, усі форми у застосунку інтегруються з компонентами для валідації та керування введенням даних [19].

Організація глобального стану застосунку реалізована з використанням Redux Toolkit. Стан поділений на логічні домени, винесені в окремі категорії, такі як auth, balance, forecasts, goals, monobank і transactions. У кожній з них реалізовано slice’и, які описують структуру даних, редуктори для оновлення стану, а також асинхронні функції для обробки API-запитів. Центральна конфігурація Redux-стору зберігається у файлі store.js, де об’єднано всі редуктори, додано middleware, а також реалізовано інтеграцію з redux-persist, що забезпечує збереження стану між сесіями. Таким чином забезпечується централізоване керування даними, яке дозволяє ефективно взаємодіяти з бекендом і полегшує тестування функціональності застосунку.

Глобальний стан активно використовується для синхронізації даних між різними сторінками та компонентами інтерфейсу. Наприклад, після авторизації користувача інформація про нього зберігається в auth-слайсі та стає доступною для інших модулів таких як goals або transactions без необхідності повторного запиту. Це дозволяє підвищити продуктивність, зменшити навантаження на сервер і забезпечити узгодженість відображення інформації в UI.

Сторінка LandingPage виконує роль вітальної і містить лише заголовки, картинки, короткий опис та кнопки для переходу до реєстрації або входу. Вона не перевантажена контентом, адже її завдання представити застосунок і запросити користувача до подальшої взаємодії. Макет цієї сторінки представлений на рисунку 2.2.

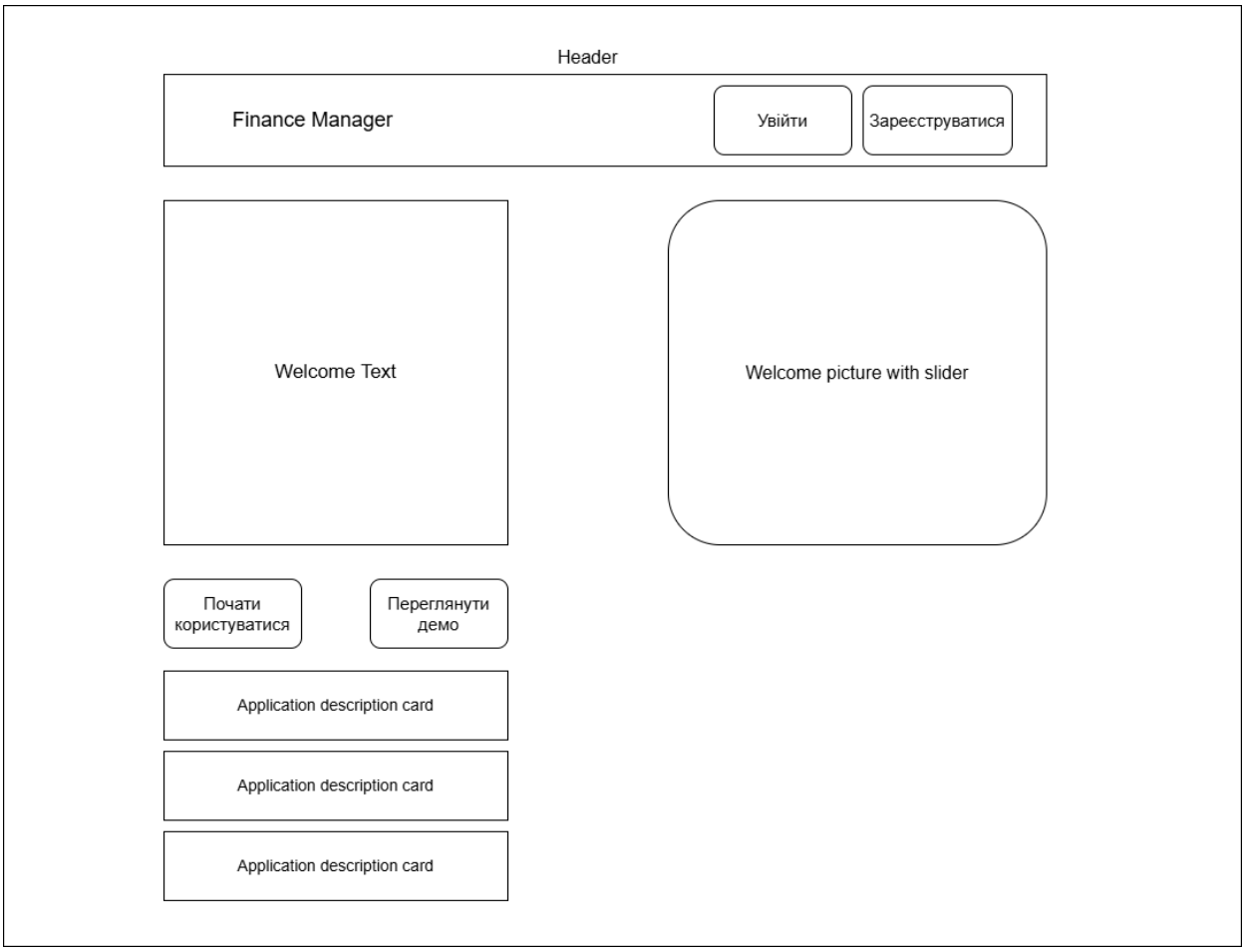

Рисунок 2.2 – Макет Landing Page застосунку

Вхід до застосунку передбачає використання сторінки авторизації (SignInPage) та реєстрації (SignUpPage). Ці сторінки не містять окремих компонентів і реалізовані як прості форми з використанням Formik, що забезпечує зручність у валідації та обробці введених даних. Для інтерактивності також застосовується yup-схема валідації. У разі успішного входу стан користувача оновлюється в глобальному сховищі Redux, а також відбувається навігація до головної сторінки застосунку. Якщо введені дані не відповідають

вимогам (наприклад, неправильний логін чи пароль), користувач отримує відповідне сповіщення про помилку. Також передбачена обробка можливих серверних помилок, таких як проблеми з мережею чи сервером, для поліпшення користувацького досвіду. Макет сторінок представлений на рисунку 2.3.

Рисунок 2.3 – Макет SignInPage/ SignUpPage застосунку

Головною сторінкою застосунку є MainPage, яка є найбільш насиченою за кількістю візуальних елементів. Саме тут зосереджено основну фінансову інформацію користувача – поточний баланс, структуру витрат, цілі, курси валют та прогнози, що відображаються за допомогою таких компонентів як BalanceCard, ExpensesCard, GoalCard, ForecastSection, TransactionChart і ExchangeRateCard. Важливою частиною є інтеграція модального вікна DetailedForecastModal, яке надає розгорнуту інформацію про майбутні

фінансові тенденції. Дизайн головної сторінки побудований за принципом дашборду, що дозволяє користувачу одразу отримати повну картину свого фінансового стану. Усі компоненти головної сторінки адаптивні та коректно відображаються на різних розмірах екранів [20]. На мобільних пристроях елементи автоматично перегруповуються у вертикальний макет з оптимальними розмірами. Макет сторінки представлений на рисунку 2.4.

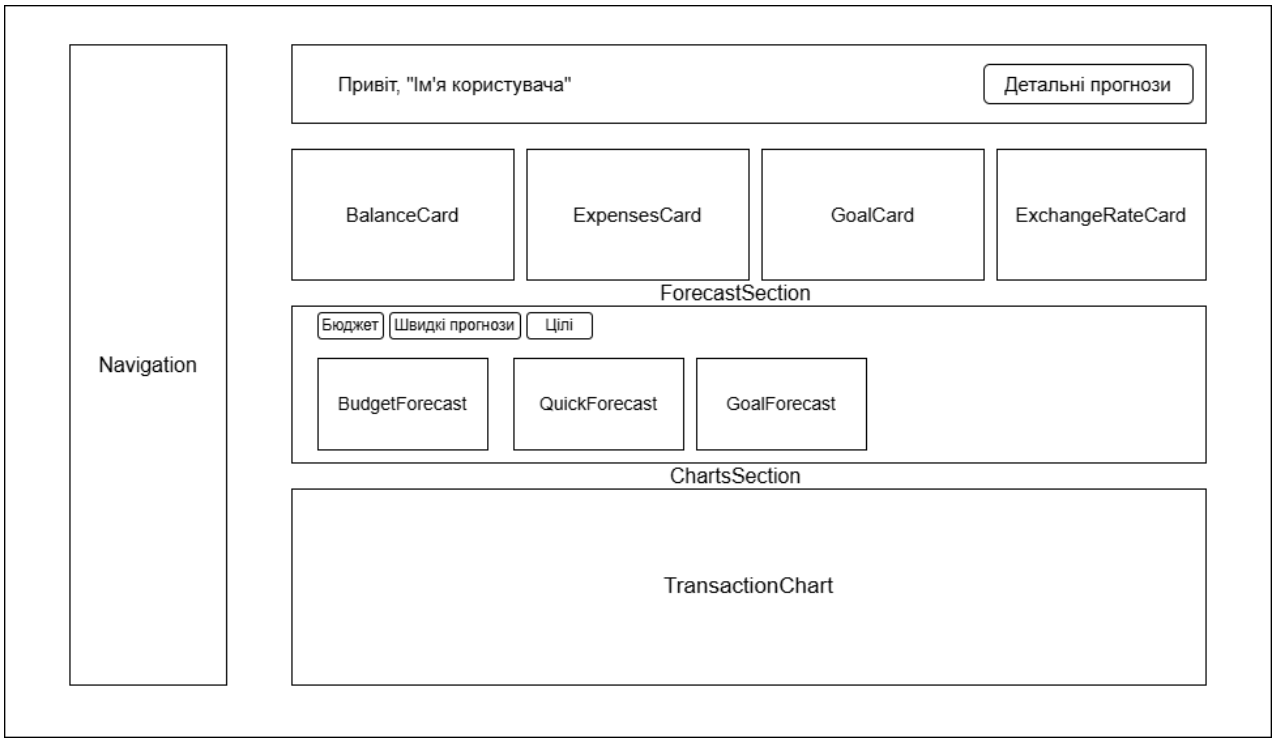

Рисунок 2.4 – Макет Main Page застосунку

Для роботи з фінансовими цілями використовується GoalsPage, яка включає компонент CurrencyDisplay і дозволяє створювати або активувати заплановані цілі. Інтерфейс надає користувачеві інструменти для довгострокового планування витрат. Користувач може задати назву цілі, очікувану суму накопичень та термін досягнення. Після створення цілі система автоматично розраховує необхідний щомісячний внесок. Усі активні цілі відображаються у вигляді сітки з динамічним оновленням статусу прогресу. Макет сторінки представлений на рисунку 2.5.

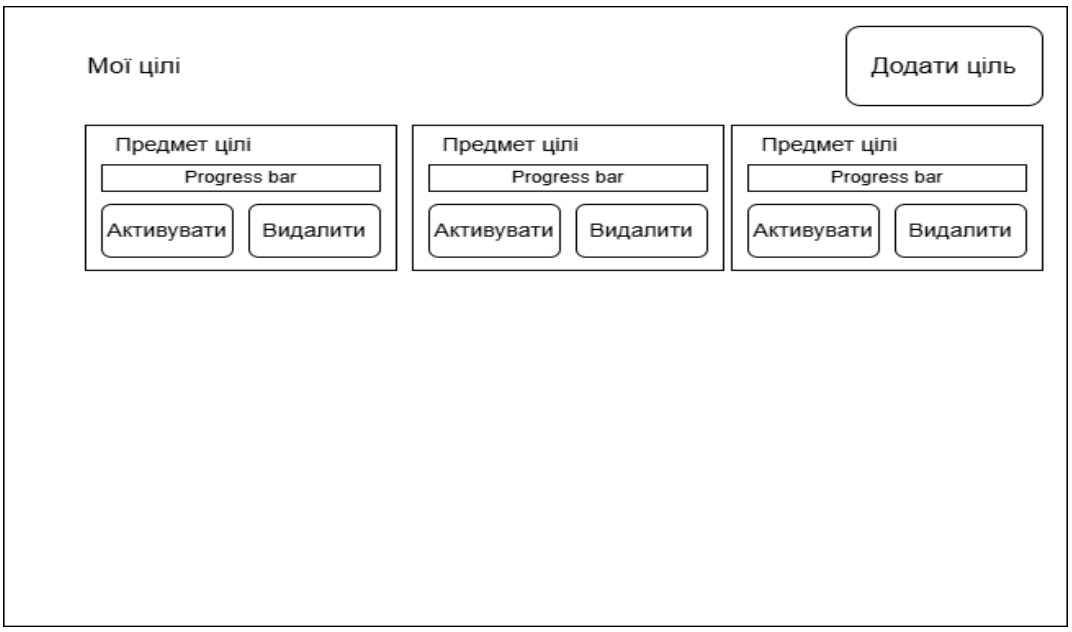

Рисунок 2.5 – Макет Goals Page застосунку

Для аналізу витрат та доходів призначена сторінка AnalyticsPage, де за допомогою бібліотеки Recharts реалізовано побудову графіків за категоріями. Така візуалізація дає змогу глибше аналізувати фінансову поведінку та виявляти тренди. Макет сторінки представлений на рисунку 2.6.

Рисунок 2.6 – Макет Analytics Page застосунку

Сторінка налаштувань профілю SettingsPage реалізована у вигляді окремої форми, що дозволяє змінити ім’я користувача, пароль та вибрати валюту. Макет сторінки представлений на рисунку 2.7.

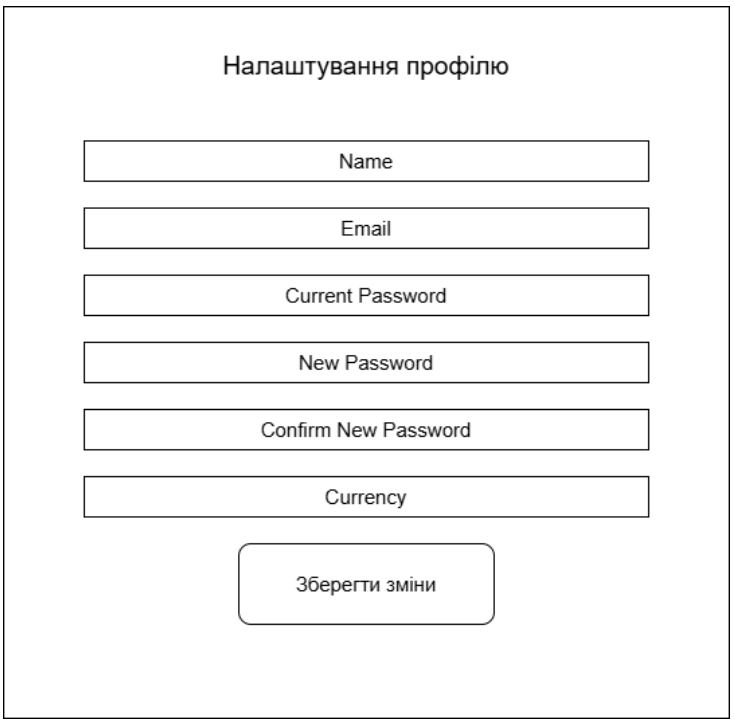

Рисунок 2.7 – Макет Settings Page застосунку

Окрему функціональну частину становить TransactionsPage. Ця сторінка об’єднує всі транзакції, проведені користувачем, надає можливість вручну додавати нові операції та містить функціональність підключення до банку Monobank. Таким чином, забезпечується повноцінний контроль за фінансовими потоками в одному місці, а інтеграція з банком дозволяє автоматизувати частину процесів.

Крім основного функціоналу, TransactionsPage підтримує пагінацію та автоматичне оновлення даних після додавання або редагування транзакцій. Завдяки адаптивному дизайну сторінка зручно відображається як на десктопах, так і на мобільних пристроях. У перспективі планується розширення можливостей аналітики на основі транзакцій безпосередньо на цій сторінці, що дозволить користувачеві отримувати миттєвий зворотний зв’язок щодо стану бюджету [21].

Щодо оптимізації продуктивності при роботі з великим обсягом транзакцій використовуються ефективні запити до бази даних та кешування часто використовуваних даних. Макет сторінки представлений на рисунку 2.8.

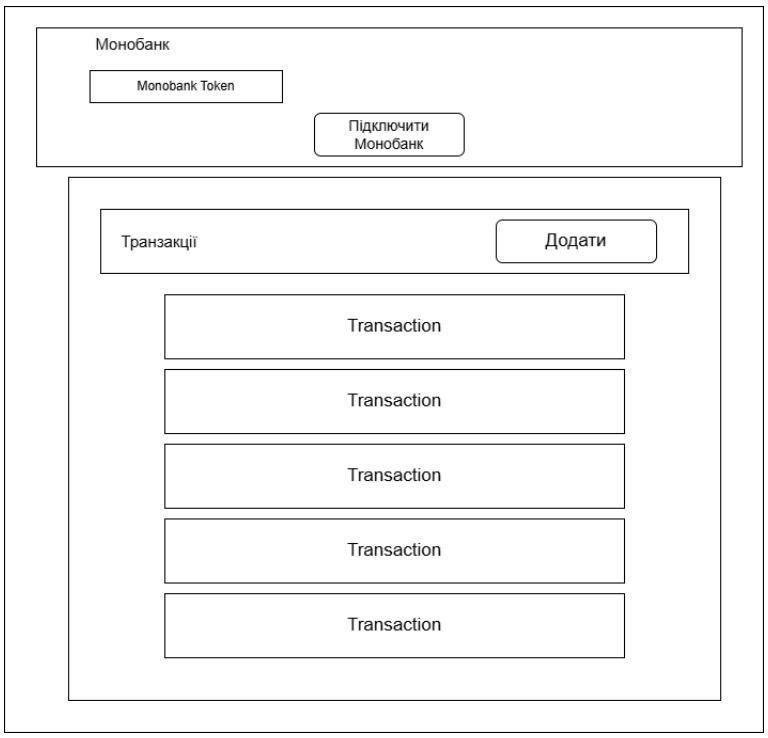

Рисунок 2.8 – Макет Transactions Page застосунку

2.5.3 Розробка інтерфейсу користувача

При розробці користувацького інтерфейсу увагу було зосереджено на забезпеченні інтуїтивно зрозумілої взаємодії між користувачем та системою. Головним пріоритетом стало надання швидкого доступу до основних функцій програми через мінімальну кількість дій та логічну навігацію, що дозволяє навіть початківцям легко орієнтуватися [22].

Інтерфейс побудовано за принципом «функціональність важливіша за форму», де кожен елемент має конкретне призначення. Створено уніфіковану систему базових макетів сторінок та стандартизовані шаблони для різних компонентів (форм, таблиць, карток, графіків). Це забезпечило візуальну послідовність інтерфейсу навіть при розширенні функціоналу.

Ергономіка стала ключовим фактором дизайну, ретельно підібрані розміри елементів керування та проміжків між ними забезпечують зручність використання на різноманітних пристроях. Єдина дизайн-система гарантує

цілісне сприйняття всього інтерфейсу. Інтерактивні компоненти суттєво покращують візуалізацію даних та спрощують взаємодію, а спеціально розроблені графіки та віджети миттєво відображають зміни у фінансових прогнозах відповідно до введених даних.

Система містить багаторівневий захист від помилок, включаючи клієнтську валідацію форм у реальному часі, серверну валідацію для критичних операцій, відображення помилок валідації безпосередньо біля відповідних полів форми. Автоматична валідація заповнених форм перед відправкою включає перевірку форматів email, телефонних номерів, сум грошових коштів та дат. Зрозумілі повідомлення при виявленні проблем формулюються зрозумілою мовою без технічних термінів [23, 24].

Адаптивний дизайн реалізовано за принципом «mobile-first» з використанням CSS Grid та Flexbox для створення гнучких макетів. Визначено чотири основні брейкпоінти: мобільні пристрої (до 768px), планшети (768px-1024px), десктопи (1024px-1440px) та великі екрани (понад 1440px).

Продуктивність інтерфейсу оптимізована через lazy loading компонентів, мемоізацію обчислень, віртуалізацію довгих списків транзакцій, оптимізацію зображень та використання Service Workers для кешування статичних ресурсів. Час завантаження початкової сторінки не перевищує 2 секунд навіть на повільних з’єднаннях [25].

В результаті створено адаптивний, гнучкий інтерфейс, який відповідає потребам користувачів, забезпечує ефективну роботу з даними, зберігаючи при цьому простоту та візуальну привабливість.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Підготовка середовища розробки

Для створення клієнтської частини вебзастосунку з обліку особистих фінансів було обрано сучасні та зручні інструменти розробки. Основним середовищем для написання коду став редактор VS Code, який завдяки своїй гнучкості, широкому вибору розширень та високій продуктивності значно полегшує процес розробки інтерфейсу користувача. Окрім VS Code, для налаштування середовища були використані інструменти Vite для швидкої збірки проєкту та керування залежностями, а також ESLint для забезпечення якості та стандартів коду [26]. Такий підхід дозволив оптимізувати розробку клієнтської частини, зробити її більш структурованою та «чистішою».

Спершу було сформовано колекцію директорій, яка логічно розділяє компоненти, сторінки, утиліти, сервіси, типи даних і модулі управління станом застосунку. Таким чином було значно полегшено навігацію у проєкті та створено основу для його подальшого розроблення.

Для керування версіями використовувалася система Git, яка дозволила зберігати історію змін, ефективно працювати з гілками та розробляти нову функціональність без виникнення конфліктів. Було також налаштовано файл .gitignore, що унеможливлює потрапляння у репозиторій зайвих файлів, таких як node_modules, результатів збірки чи локальних налаштування. Наступним етапом була розробка скриптів для швидкого запуску застосунку в режимі локального тестування, збирання продакшен-версії та перевірки коду на відповідність вимогам якості [27, 28].

Особливу роль відіграло налаштування середовища розробки для підтримки сучасних можливостей JavaScript і JSX. Система адаптована для автоматичного розпізнавання модулів, використання відносних шляхів імпорту та інтеграції зі статичним аналізом типів. Для підтримки чистоти коду було налаштовано використання лінтерів і форматерів, що автоматично

форматують файли під час збереження та допомагають дотримуватися єдиного стилю без додаткових зусиль.

У результаті реалізовані заходи забезпечили побудову надійної інфраструктури, яка заохочує до впорядкованої та ефективної розробки клієнтської частини вебзастосунку.

3.2 Розробка клієнтської частини вебзастосунку

Ключову роль у клієнтській частині застосунку відіграє головний компонент App, який відповідає за налаштування маршрутизації, авторизацію користувачів і підключення службових модулів, таких як система обробки помилок і механізм сповіщень. Під час першого завантаження програми виконується перевірка токена авторизації через запит до серверної частини, що забезпечує відновлення активної сесії користувача, якщо вона була.

Перехід між сторінками організований через визначення маршрутів. Доступ до них залежить від автентифікаційного статусу користувача, для обмеження або надання доступу застосовуються обгортки PublicRoute і PrivateRoute. Публічні маршрути відкривають сторінки реєстрації й входу, тоді як приватні маршрути захищають основний функціонал від неавторизованих користувачів [29].

Для оптимізації швидкодії застосовується динамічне завантаження сторінкових компонентів через функцію lazy. Під час переходів між сторінками компонент Suspense відповідає за відображення індикатора завантаження, що позитивно впливає на користувацьке сприйняття інтерфейсу.

Головна структура застосунку також містить компоненти верхнього рівня, такі як Layout для приватних розділів і PublicLayout для публічних, що дозволяє чітко розмежувати інтерфейс. Це спрощує навігацію та забезпечує послідовність дизайну в межах кожного типу сторінок.

Лістинг 3.1 Основна конфігурація маршрутизації застосунку:

```
<ErrorBoundary>
<Suspense fallback={<LoadingSpinner />}>
<Routes>
<Route index element={<Navigate to="/landing" />} />
{/* Public routes with PublicLayout */}
<Route element={<PublicLayout />}>
<Route path="/landing" element={<PublicRoute component={<Suspense
fallback={<LoadingSpinner />}><LandingPage /></Suspense>}
redirectTo="/home" />} />
<Route path="/signup" element={<PublicRegisterRoute
component={<Suspense fallback={<LoadingSpinner />}><SignupPage
/></Suspense>} redirectTo="/signin" />} />
<Route path="/signin" element={<PublicRoute component={<Suspense
fallback={<LoadingSpinner />}><SinginPage /></Suspense>}
redirectTo="/home" />} />
</Route>
{/* Private routes */}
<Route path="/" element={<PrivateRoute component={<Layout />}
redirectTo="/signin" />}>
<Route path="home" element={<Suspense fallback={<LoadingSpinner
/>}><MainPage /></Suspense>} />
```

3.2.1 Реалізація інтерфейсу авторизації користувачів

Компонент інтерфейсу авторизації був реалізований як універсальний модуль, що автоматично визначає режим своєї роботи залежно від маршруту.

Управління базовими даними форми здійснюється через Formik, що забезпечує повний контроль над полями вводу, обробку помилок і відправлення даних. На етапі ініціалізації передбачено, що у режимі реєстрації рендериться додаткове поле для підтвердження пароля, тоді як під час входу залишаються лише поля для електронної пошти та пароля. Валідація даних реалізована за допомогою бібліотеки Yup, де встановлено правила для мінімальної довжини пароля, правильності формату email та збігу полів паролів.

У побудові інтерфейсу важливу роль відіграють локальні стани, зокрема passwordVisible та confirmPasswordVisible, які керують відображенням пароля у відкритому вигляді. Для зручності користувача реалізовано окремий компонент поля пароля з можливістю перемикання видимості. Процес обробки авторизації організований через Redux. Після відправки форми функцією handleSubmit викликається відповідна асинхронна дія logIn або register, залежно від режиму. Для обробки результатів використовується метод .unwrap(), що дозволяє безпосередньо отримати успішну відповідь або обробити помилку.

Крім стандартної форми авторизації, передбачено можливість входу через Google-акаунт. Для цього розроблений обробник handleGoogleSignIn, який виконує запит до маршруту /auth/get-oauth-url та перенаправляє користувача на сторінку авторизації Google.

Помилки, що виникають під час реєстрації або входу, відображаються безпосередньо під відповідними полями вводу, це забезпечує оперативне та лаконічне інформування користувача про причини невдачі.

3.2.2 Розробка головної сторінки

У процесі розробки головної сторінки застосунку, представленої компонентом Dashboard, стояла задача побудувати логічну, гнучку й

максимально динамічну архітектуру. Метою було забезпечити миттєве завантаження ключових даних користувача, ефективне керування станами та зручну взаємодію з інтерфейсом.

Щоб організувати правильний потік даних, спочатку було визначено, які саме інформаційні запити мають бути виконані одразу після завантаження сторінки, це баланс, список транзакцій, активні цілі та прогнози. Для цього ще на етапі проєктування було створено окремі сервіси на базі Redux, які можна централізовано викликати. При монтуванні компонента через useEffect здійснюється послідовний запуск необхідних запитів.

Лістинг 3.2 Запити при монтуванні для отримання даних:

```
useEffect(() => {
  dispatch(fetchGoals());
  dispatch(fetchTransactions());
  dispatch(fetchBalance());
  dispatch(calculateBudgetForecast());
  dispatch(calculateGoalForecast());
  dispatch(fetchGoalForecasts());
  dispatch(fetchCategoryForecasts());
}, [dispatch]);
```

Окремо було розроблено гнучке керування локальними станами. Наприклад, для відкриття детального прогнозу за цілями чи категоріями необхідно забезпечити перемикання вкладок у модальному вікні.

Для керування модальними вікнами було створено дві основні функції. Функція відкриття модального вікна прогнозу приймає параметр вкладки з значенням за замовчуванням «goals», встановлює активну вкладку через setForecastModalTab та відкриває модальне вікно шляхом встановлення стану setIsForecastModalOpen у значення true. Функція закриття модального вікна просто встановлює стан видимості модального вікна у значення false через

setIsForecastModalOpen, що забезпечує його приховування від користувача.

При створенні структури інтерфейсу було використано концепцію розділення відповідальностей, кожен тип даних представлений окремим компонентом. Так, фінансовий баланс відображає BalanceCard, витрати ExpensesCard, активні цілі GoalCard, а курс валют ExchangeRateCard. Для швидкого огляду прогнозів використовується ForecastSection, а аналіз витрат за категоріями візуалізовано через TransactionChart.

Додатковий акцент було зроблено на розширеній аналітиці через компонент DetailedForecastModal, який приймає необхідні пропси для контролю стану та відображення потрібної вкладки.

Лістинг 3.3 Імплементація компоненту DetailedForecastModal:

```
<DetailedForecastModal
  isOpen={isForecastModalOpen}
  onClose={handleCloseForecastModal}
  initialTab={forecastModalTab}
/>
```

3.2.3 Створення та інтеграція функціоналу управління цілями

На етапі розробки функціоналу управління фінансовими цілями було розроблено інтуїтивний і зручний інтерфейс для створення, активації, деактивації та видалення цілей, а також контролю їхнім станом і терміном виконання. Основою стало проєктування структури даних та визначення базових операцій, які має підтримувати система. Було вирішено використовувати бібліотеку Formik, яка є основною для створення форм у проєкті. Це дозволило одразу закласти фундамент для коректної перевірки введеної користувачем інформації, наприклад, обов’язковість заповнення полів, обмеження на мінімальну суму цілі.

Також було важливо забезпечити контроль над термінами виконання цілей. Для цього під час завантаження списку цілей відбувається перевірка, і якщо якась ціль прострочена і при цьому залишається активною, вона автоматично деактивується, а користувач отримує відповідне сповіщення через бібліотеку react-hot-toast. Це рішення підвищило надійність системи та мінімізувало ризик плутанини з активними, але неактуальними цілями.

Для створення нової цілі передбачене модальне вікно. Відкриття та закриття модального вікна керується локальним станом, що дозволяє плавно змінювати інтерфейс без перезавантаження сторінки. Валідація форми відбувається ще до надсилання даних на сервер, що дозволяє уникати непотрібних запитів і дає користувачеві зрозумілі підказки щодо виправлення помилок. Процес створення нової цілі включає заповнення трьох обов'язкових полів: назви, суми та кінцевої дати. Після успішного створення форма очищується, а модальне вікно закривається [30].

Кожна ціль на сторінці відображається як окремий елемент, де можна побачити її назву, прогрес досягнення, відображений через прогрес-бар, суму накопиченого та запланованого бюджету, а також крайній термін виконання. Додатково для прострочених цілей виводиться спеціальна мітка.

Щоб поліпшити UX, було реалізовано закриття модального вікна при натисканні на фон навколо форми. Це рішення дозволяє швидко і природно виходити із процесу створення цілі, не змушуючи користувача натискати окрему кнопку.

Крім базових операцій керування цілями, невід'ємним етапом стала інтеграція цього функціоналу з іншими модулями застосунку, зокрема з аналітикою транзакцій та системою прогнозування балансу.

Лістинг 3.4 Механізм перевірки наявності активної цілі і врахування її у прогнозі витрат:

```
if (activeGoal) {
  forecastedExpenses += activeGoal.targetAmount / monthsUntilDeadline;}
```

Тут monthsUntilDeadline – це кількість місяців до встановленої дати цілі. Таким чином до кінцевої суми витрат додається планований щомісячний внесок, необхідний для досягнення цілі вчасно.

Також у модулі аналітики доходів і витрат реалізовано відображення, наскільки цільова сума співвідноситься із загальним фінансовим потоком користувача. Наприклад, у графіках аналітики поряд із категоріями витрат може бути окремий сектор «Резерв на ціль».

3.2.4 Побудова модулю аналітики доходів та витрат

Розробка модуля аналітики стала важливою складовою для створення повноцінного фінансового інструменту, який не просто зберігає транзакції, а дозволяє користувачам аналізувати свою фінансову поведінку. Інтуїтивний та гнучкий інтерфейс, у стилі попередніх сторінок, став основою сторінки, він надає змогу оцінити розподіл витрат і доходів за категоріями як на основі реальних даних, так і на базі прогнозів на майбутнє. Функціональність модуля передбачає подвійний режим перегляду даних: «факт» і «прогноз». Для цього розроблено механізм перемикання між джерелами даних, який враховує як наявні транзакції користувача, так і прогнозовані значення, які генеруються системою прогнозування [31].

На початковому етапі було створено структуру стану, яка дозволяє отримувати транзакції та прогнози через селектори Redux. Для цього використовувалися хуки useSelector, що забезпечували доступ до відповідних частин глобального стану застосунку. Одночасно був організований асинхронний запит до сервера для завантаження прогнозних даних за допомогою dispatch(fetchCategoryForecasts()), який виконується один раз під час монтування компонента.

Одним із складних викликів стало правильне оброблення даних для побудови графіків. Оскільки фактичні транзакції та прогнозовані дані мають різну структуру, через це для кожного типу були розроблені окремі функції обробки. Для реальних даних відбувається групування транзакцій за категоріями з підсумовуванням сум. Для прогнозів же виділення передбачених сум на найближчий місяць. Таким чином обидва типи інформації конвертуються у формат, зручний для подальшого відображення.

Лістинг 3.5 Механізм групування реальних даних за категоріями:

```
const processTransactionsByCategory = () => {
  if (!transactions?.length) return [];
  const categoriesMap = transactions.reduce((acc, transaction) => {
    if (transaction.type === viewType) {
      acc[transaction.category] = (acc[transaction.category] || 0) +
transaction.amount;
    }
    return acc;
  }, {});
  return Object.entries(categoriesMap).map(([category, amount]) =>
({ category, amount }));
};
```

Для відображення аналітики використовувався компонент BarChart з бібліотеки recharts, який дозволяє будувати адаптивні гістограми з можливістю налаштування зовнішнього вигляду. В залежності від обраного режиму перегляду, на графіку можуть бути відображені або тільки реальні суми за категоріями, або комбінація реальних та прогнозованих значень. Для покращення взаємодії з користувачем додатково було реалізовано власний компонент підказки, який при наведенні на стовпець графіка показує детальну інформацію про суму фактичних витрат або прогнозованих значень.

Щоб забезпечити безперервність користувацького досвіду, були враховані можливі стани завантаження та відсутності даних. Якщо прогнозні дані ще завантажуються, відображається повідомлення про це, а якщо даних недостатньо показується відповідний інформативний текст замість порожнього графіка.

3.2.5 Реалізація модуля управління транзакціями та підключення зовнішніх сервісів

Робота з транзакціями відбувається у компоненті Transactions. При завантаженні сторінки автоматично виконується ініціалізація даних: завантажуються всі транзакції та поточний баланс користувача за допомогою функцій fetchTransactions та fetchBalance, що підключені через redux-thunk. Для цього використовується хук useEffect, який згадувався раніше, що викликається при монтуванні компонента.

Користувач може додати нову транзакцію через модальне вікно, яке відкривається після натискання кнопки «Додати». При цьому перевіряється готовність балансу, якщо він ще не завантажений, перед відкриттям модального вікна додатково викликається fetchBalance.

Для забезпечення інтеграції із зовнішніми фінансовими сервісами реалізовано окремий компонент MonobankConnect, який відповідає за підключення рахунку користувача до зовнішнього сервісу. Після підключення токена Монобанку користувач отримує доступ до перегляду своїх рахунків, синхронізації транзакцій, а також може у будь-який момент відключити інтеграцію. Інтерфейс підключення побудований таким чином, що інформація про рахунки і можливості синхронізації відображається у вигляді окремої секції, яку користувач може розгортати або згортати за потребою. Це реалізовано завдяки внутрішньому стану компонента і обробці події кліку.

Крім того передбачено механізм перевірки коректності введеного токена, що дозволяє уникнути помилок при підключенні та гарантує безпечний обмін даними. У разі втрати актуальності токена система автоматично повідомляє користувача про необхідність повторного підключення.

3.3 Демонстрація роботи вебзастосунку

3.3.1 Реєстрація користувача та налаштування профілю

Після першого завантаження сайту користувач потрапляє на лендінгову сторінку, де представлено коротку інформацію про можливості сервісу. Тут розміщено кнопки для переходу до сторінок авторизації та реєстрації нового акаунту (рис. 3.1). Інтерфейс лендінгової сторінки адаптований для мобільних пристроїв, це забезпечує зручність перегляду з будь-якого екрану. Додатково, сторінка містить візуальні елементи, які підкреслюють основні переваги інструменту. Завдяки простій та інтуїтивно зрозумілій навігації, користувачі можуть швидко орієнтуватися та знаходити необхідну інформацію.

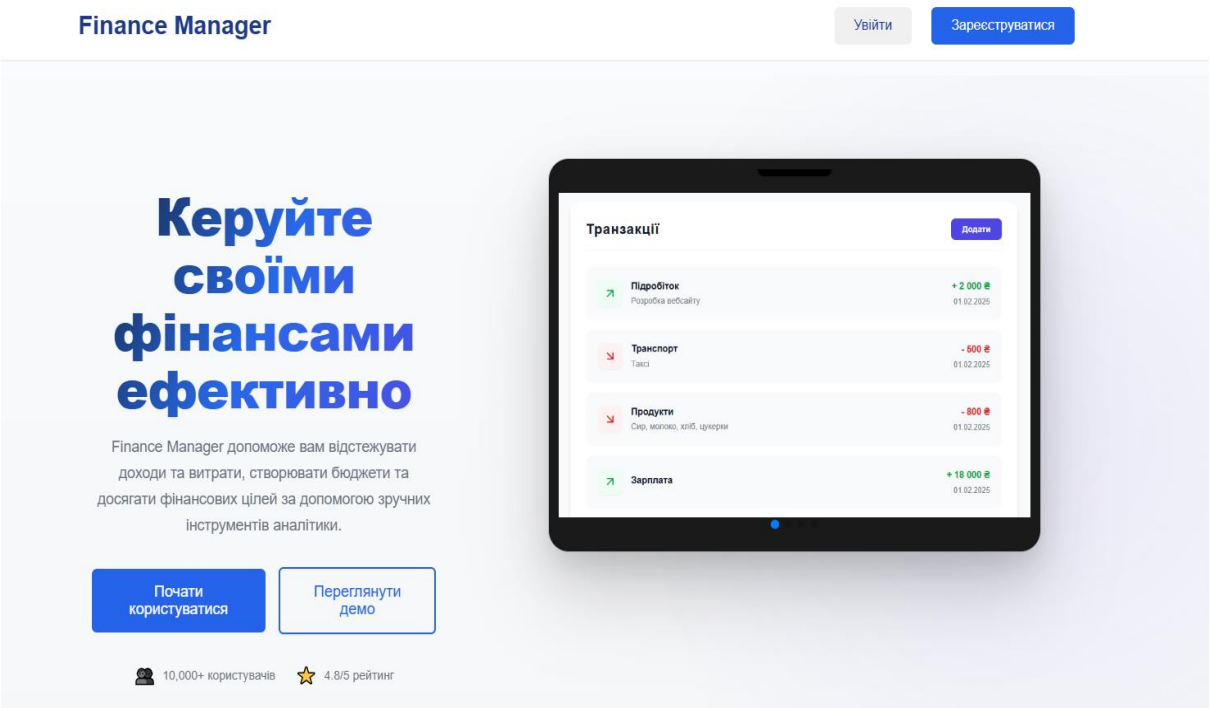

Рисунок 3.1 – Головна сторінка

Після натискання на відповідну кнопку користувач переходить на сторінку авторизації (рис. 3.2), де може увійти до свого акаунту або на сторінку реєстрації (рис. 3.3), де може створити новий профіль. Під час реєстрації необхідно вказати адресу електронної пошти, пароль та підтвердження паролю. Форма передбачає автоматичну перевірку введених даних: електронна пошта має відповідати встановленому формату, а пароль містити не менше шести символів. Крім того, система перевіряє, чи збігаються введені паролі для підтвердження.

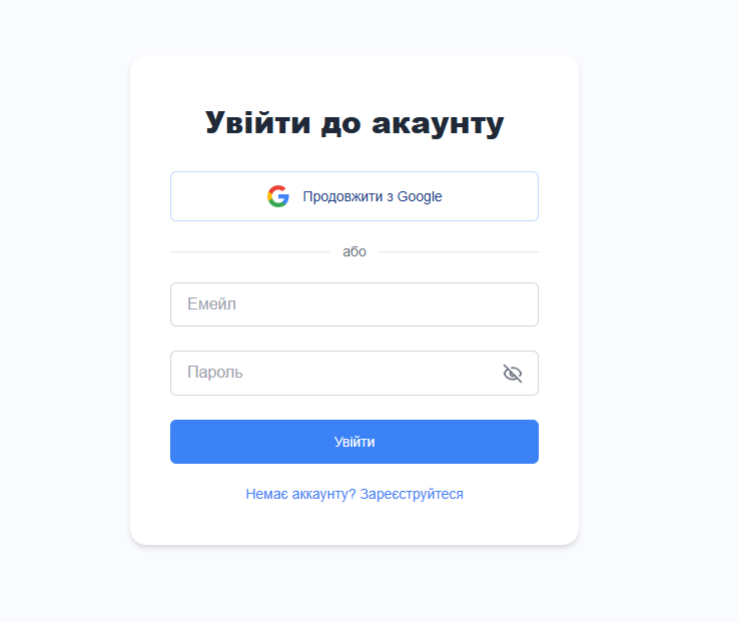

Рисунок 3.2 – Сторінка авторизації

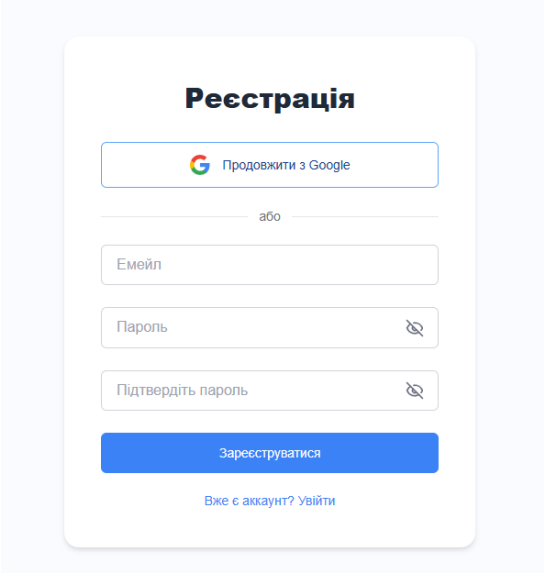

Рисунок 3.3 – Сторінка реєстрації

У разі помилки поруч із полем введення з’являється підказка, що допомагає швидко виправити неточності. Якщо користувач намагається зареєструватися за вже існуючою електронною поштою, система повідомить про це та запропонує увійти в акаунт. Після успішної реєстрації користувач автоматично перенаправляється на сторінку авторизації для входу в акаунт. Також передбачена можливість перегляду пароля для запобігання помилок при введенні.

Валідація електронної пошти здійснюється за допомогою регулярних виразів, що перевіряють наявність символу «@», домену та правильної структури адреси відповідно до міжнародних стандартів. Перевірка надійності паролю включає аналіз довжини, наявності спеціальних символів та цифр, а також порівняння з базою найпоширеніших паролів для запобігання використанню слабких комбінацій. Система також відстежує кількість невдалих спроб входу та тимчасово блокує доступ після трьох невірних спроб протягом п’яти хвилин для захисту від брутфорс-атак.

Для покращення користувацького досвіду всі дії з формою супроводжуються відповідними анімаціями, що додають інтерфейсу динамічності. Адаптивний дизайн забезпечує зручну роботу як на десктопних, так і на мобільних пристроях. Важливою складовою є швидкий зворотній зв’язок, так користувач отримує миттєві повідомлення про успішне виконання операцій або про помилки, що виникли під час заповнення форми. Такий підхід сприяє зменшенню кількості помилок і підвищує загальну задоволеність.

Окрім стандартної реєстрації, користувачеві надається можливість швидкої авторизації через обліковий запис Google. Натиснувши кнопку «Продовжити з Google», він переходить на сторінку підтвердження та після успішної авторизації автоматично отримує доступ до застосунку. У разі, якщо користувач вперше заходить через Google, система автоматично створює новий профіль, зберігаючи електронну пошту, отриману з облікового запису, та встановлює базові налаштування. Після реєстрації або входу користувач потрапляє на головну сторінку застосунку.

Якщо в акаунті ще не встановлено ім’я, відображається повідомлення із запрошенням перейти до налаштувань профілю (рис. 3.4). Там можна додати ім’я, змінити пароль або обрати валюту для обліку фінансів. Всі зміни автоматично зберігаються в базі даних, а оновлений профіль одразу синхронізується з інтерфейсом застосунку, що гарантує актуальність відображуваних даних. Можливість виходу з акаунту в будь-який момент через відповідну кнопку в налаштуваннях також предбачено. Це забезпечує додатковий рівень безпеки та зручність у користуванні застосунком на спільних пристроях.

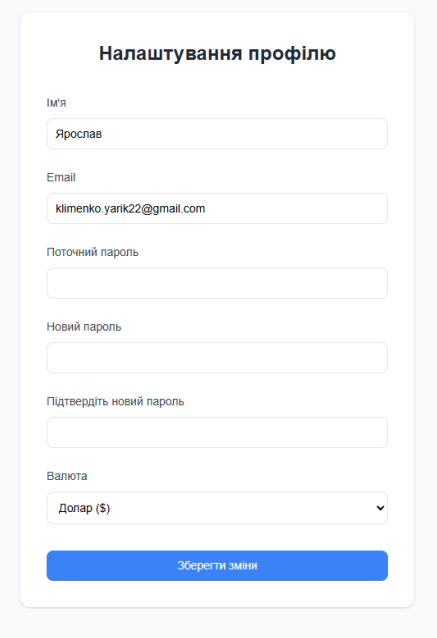

Рисунок 3.4 – Сторінка налаштувань профілю

3.3.2 Основний функціонал управління фінансами та цілями

Після налаштування профілю користувач потрапляє на головну сторінку вебзастосунку (рис. 3.5), де одразу відображається загальна фінансова картина. У верхній частині показано поточний баланс, який можна встановити лише один раз після реєстрації, загальні витрати та залишок до досягнення фінансової цілі. Поряд розміщено блок із актуальними валютними курсами.

Всі ці дані автоматично завантажуються з бекенду під час ініціалізації сторінки, використовуючи відповідні Redux-слайси та асинхронні запити. Інтерфейс побудований таким чином, щоб користувач з першого погляду міг оцінити своє фінансове становище без необхідності переходу до інших розділів.

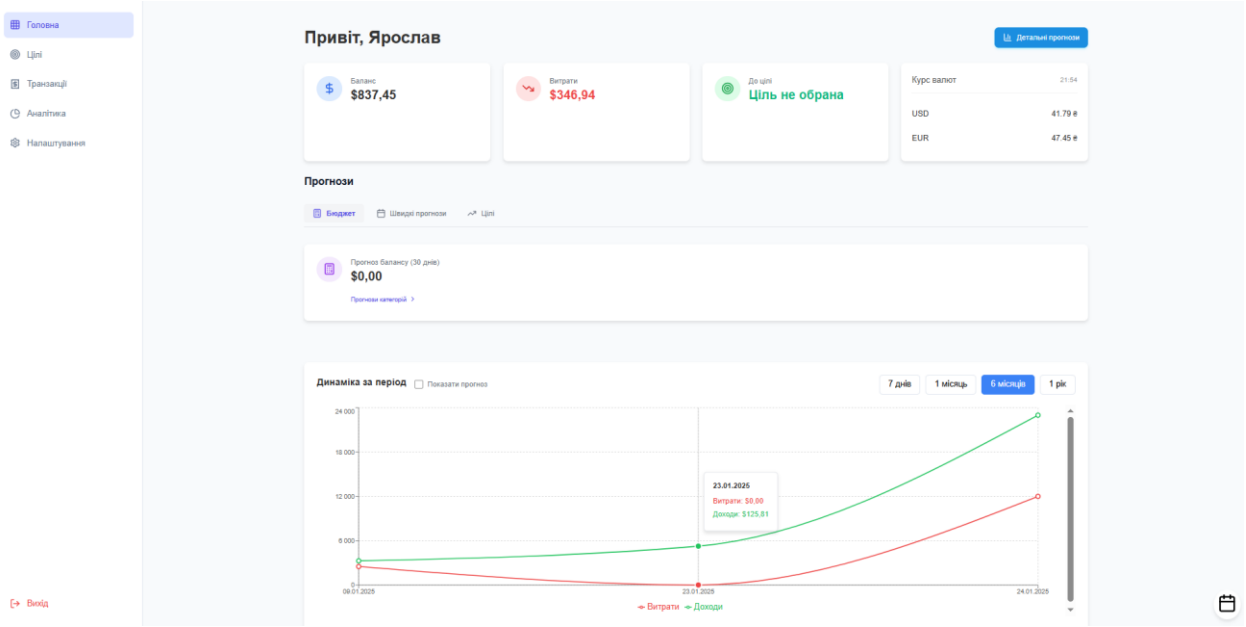

Рисунок 3.5 – Домашня сторінка

Нижче на головній сторінці представлений розділ прогнозів, де користувач може переглянути очікувану зміну балансу за обраний період часу. Інтерактивний графік демонструє динаміку доходів та витрат, що дозволяє візуально оцінити фінансові тенденції. Для глибшого аналізу передбачено окрему аналітичну сторінку. Тут можна переглядати розподіл доходів та витрат за категоріями у вигляді діаграм. Користувач має можливість перемикатися між фактичними та прогнозованими даними (рис. 3.6), що полегшує процес планування бюджету.

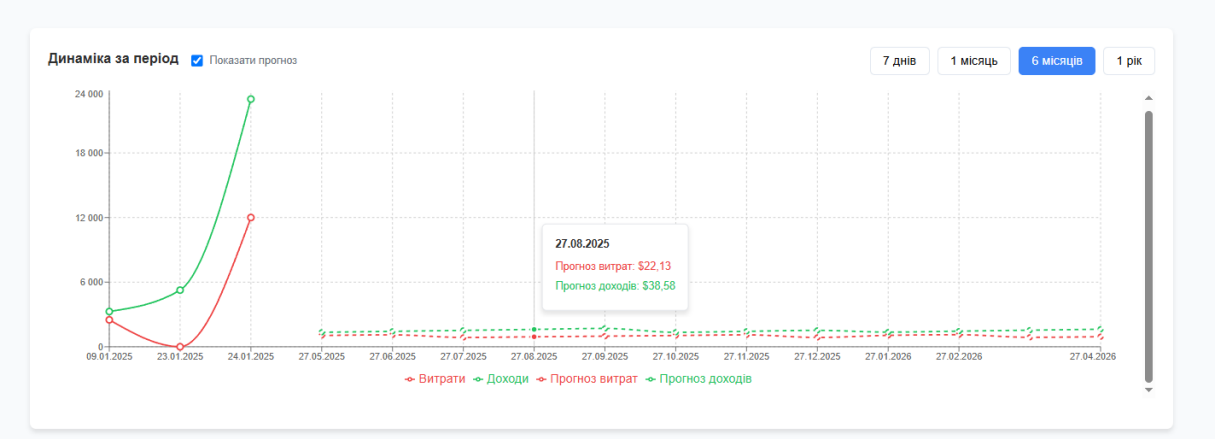

Рисунок 3.6 – Прогнози доходів та витрат

Управління фінансовими цілями здійснюється на спеціальній сторінці (рис. 3.7), яка призначена для планування, відстеження та аналізу прогресу накопичень користувача. Інтерфейс цієї сторінки передбачає зручне представлення усіх активних цілей, а також можливість швидкого додавання нових. Створення нової фінансової цілі відбувається через модальне вікно (рис. 3.8), де користувач заповнює обов’язкові поля: бажану суму накопичення та дату, до якої планується її досягнення. Після збереження нової цілі вона автоматично додається до списку активних, і для неї починається відстеження фінансового прогресу.

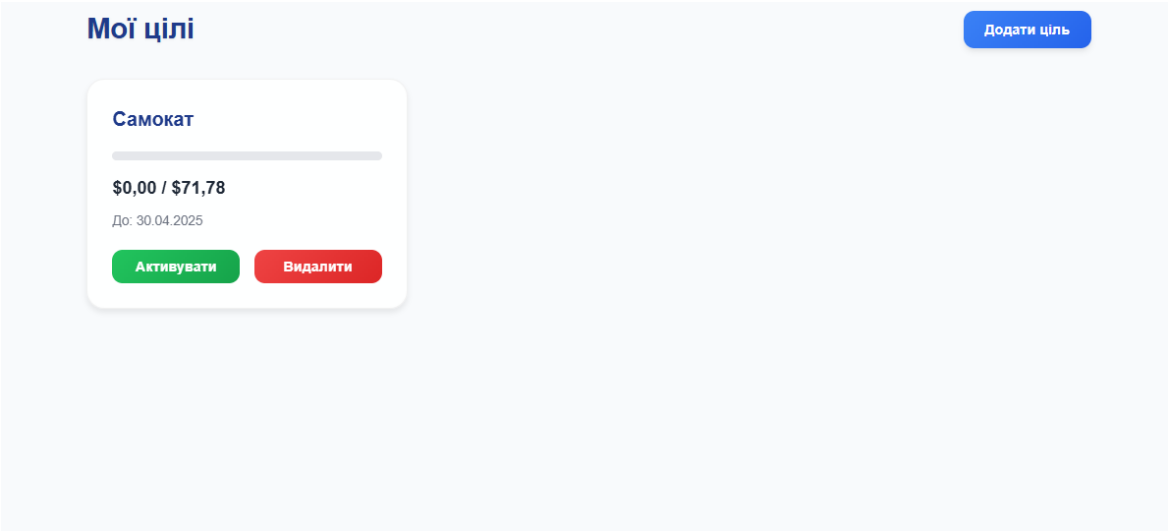

Рисунок 3.7 – Сторінка управління цілями

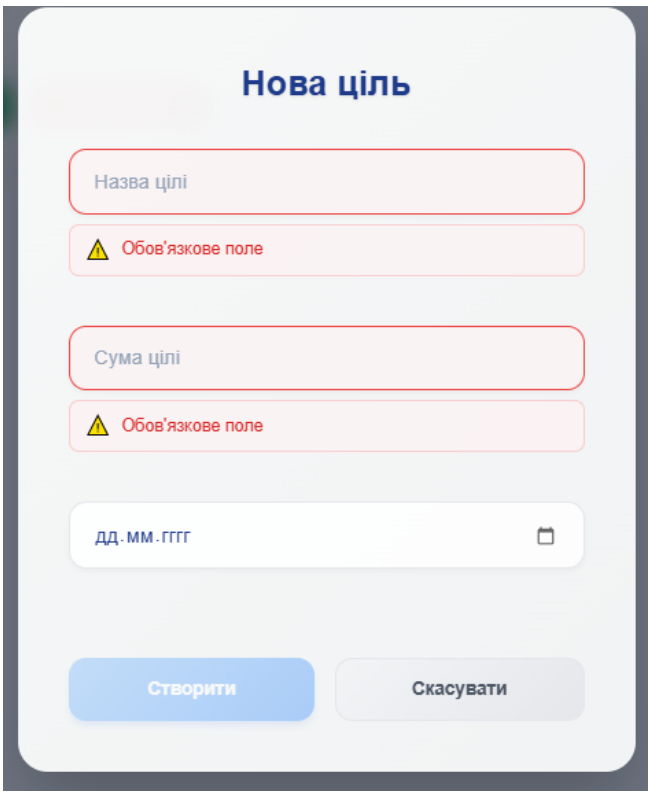

Рисунок 3.8 – Модальне вікно для створення цілі

Для кожної створеної цілі надається візуальне відображення поточного прогресу у вигляді шкали, що демонструє співвідношення між уже накопиченою сумою та загальною метою. У разі досягнення повної суми або настання вказаного терміну, система автоматично деактивує ціль і дозволяє користувачу видалити її зі списку.

Для зручності користувацького досвіду передбачена можливість редагування та коригування параметрів фінансових цілей у будь-який момент. Також, реалізовано функцію сортування та фільтрації цілей за різними критеріями, що дозволяє швидко знаходити необхідні записи. Інтерфейс підтримує інтерактивні підказки, які допомагають користувачу ефективно планувати свої накопичення та досягати поставлених фінансових завдань.

Наступним пунктом у меню навігації є сторінка з історією транзакцій (рис. 3.9). Тут користувач може переглядати всі додані доходи та витрати, а також вручну додавати нові записи у модальному вікні (рис. 3.10) або синхронізувати дані з банківським рахунком Монбанк (рис. 3.11).

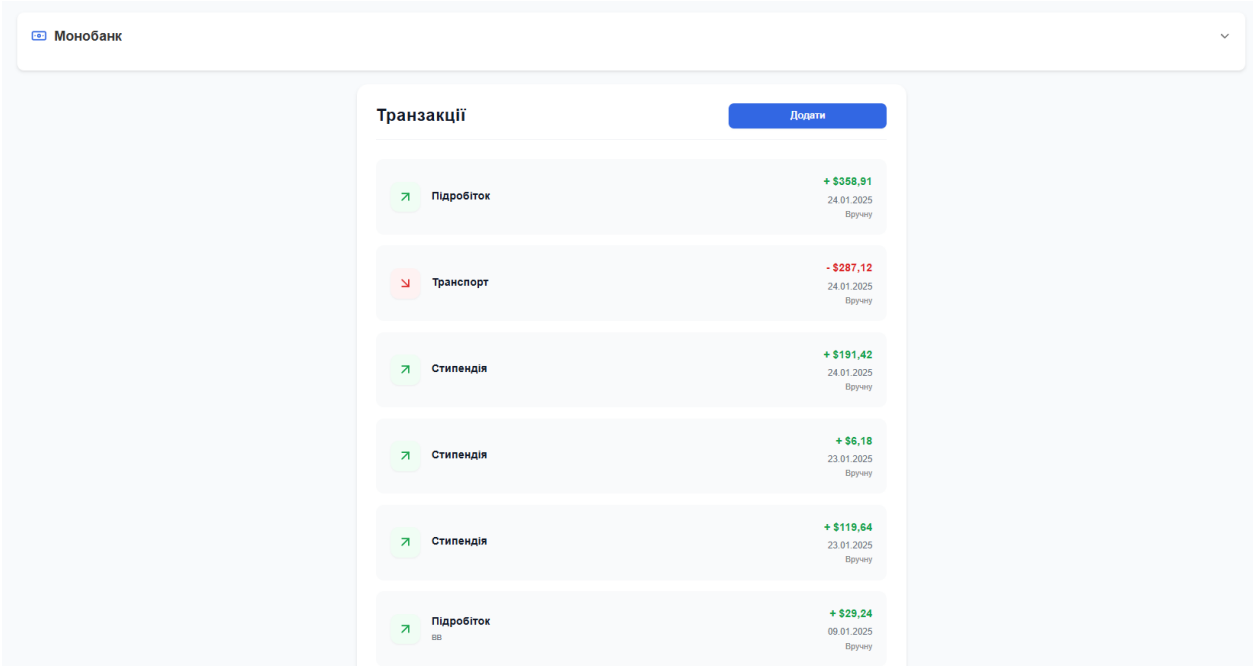

Рисунок 3.9 – Сторінка транзакцій

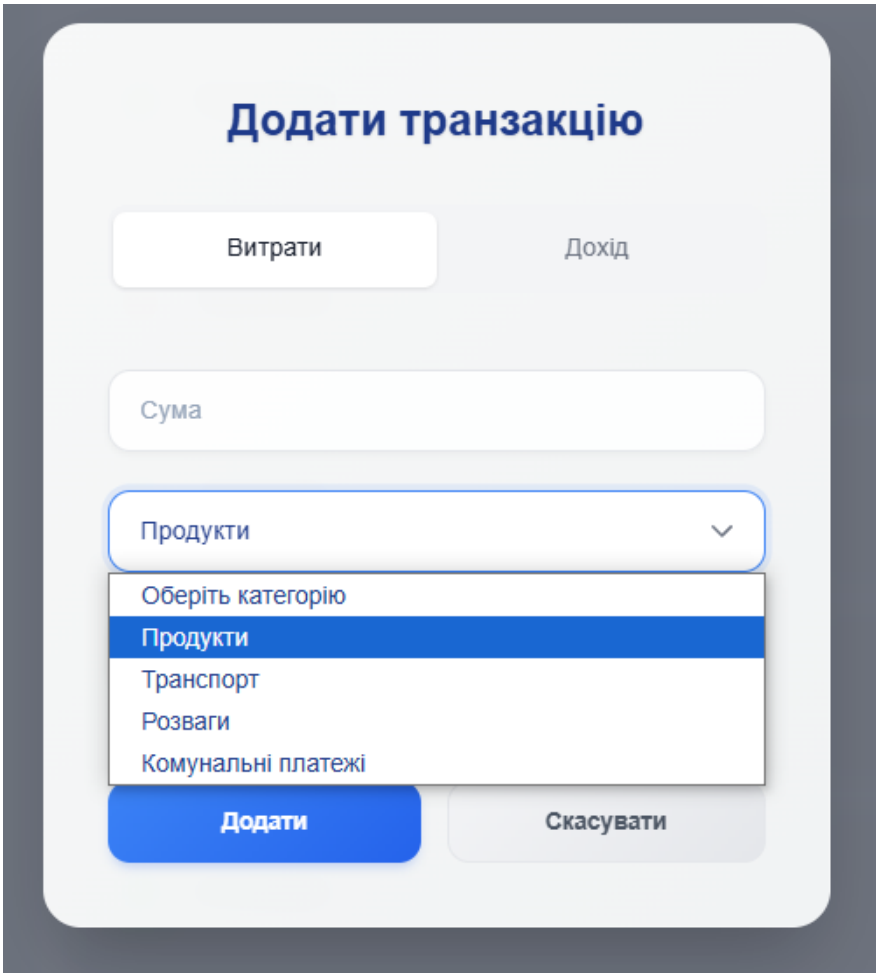

Рисунок 3.10 – Модальне вікно для додавання транзакції

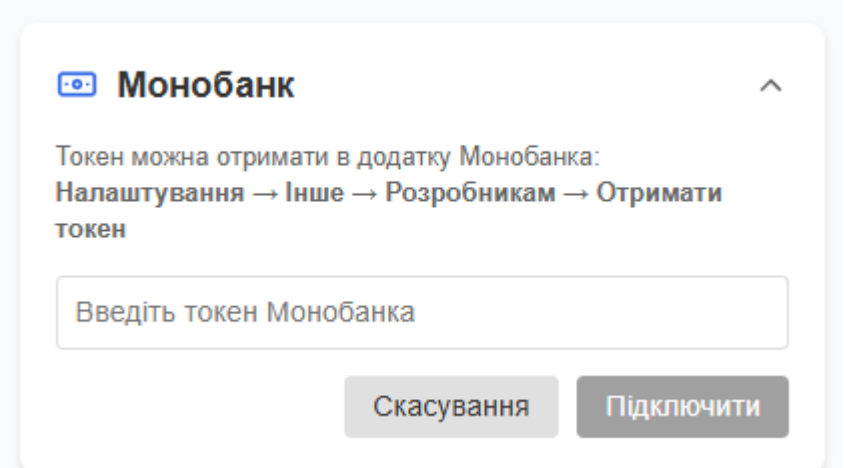

Рисунок 3.11 – Панель підключення Монобанк

Основний функціонал вебзастосунку охоплює як щоденне відстеження фінансових операцій, так і довгострокове планування, що робить його універсальним інструментом для управління особистими фінансами. Користувач має можливість швидко додавати доходи та витрати, вказуючи категорію, суму й опис, що сприяє створенню повної та точної картини фінансової активності.

Крім базового обліку, інструмент надає користувачу потужні аналітичні засоби: діаграми, графіки та зведені таблиці, які візуалізують фінансові потоки за обраний період. Це дозволяє швидко ідентифікувати найбільш витратні категорії, оцінити ефективність бюджету та виявити потенційні можливості для економії.

3.4 Перспективи розвитку вебзастосунку

На поточному етапі вебзастосунок уже пропонує зручний та функціональний інтерфейс для щоденного обліку фінансів, прогнозування витрат, управління цілями та синхронізації з банківськими сервісами. Завдяки сучасному дизайну, інтеграції з Google-акаунтом та базовій аналітиці, сервіс вигідно вирізняється серед подібних інструментів.

Однак застосунок має значний потенціал для подальшого розвитку і вдосконалення. Планується поступове розширення функціональності, що дозволить не лише утримувати конкурентну перевагу, а й випереджати інші подібні сервіси.

Одним із ключових напрямків розвитку є впровадження спільного управління фінансами. Це надасть змогу декільком користувачам створювати спільні рахунки, планувати загальні витрати, накопичення та цілі для родини, партнерських проєктів або малих команд. Кожному учаснику можна буде призначити роль і права доступу до фінансових даних.

Другим важливим кроком стане реалізація розширеної системи аналітики. Користувачі зможуть самостійно створювати фінансові звіти за вибраними категоріями, періодами та параметрами. Побудова графіків, динамічних таблиць і кастомізованих діаграм дасть змогу детально аналізувати фінансову поведінку та приймати стратегічні рішення щодо бюджету.

Крім того, планується впровадити систему нагадувань і персональних повідомлень. Користувачі зможуть встановлювати нагадування про регулярні платежі (наприклад, оренда, комунальні послуги), про необхідність поповнення цілей або проведення щомісячного аналізу бюджету. Така функціональність сприятиме формуванню фінансової відповідальності й дозволить уникати забутих витрат.

Ще одним перспективним напрямком є розробка офлайн-режиму роботи застосунку, що дозволить переглядати останні збережені дані навіть без доступу до інтернету. Це підвищить стабільність і зручність використання сервісу у будь-яких умовах.

Таким чином, подальший розвиток вебзастосунку орієнтований на підвищення гнучкості, зручності та інтелектуальності сервісу, що в майбутньому дозволить йому стати універсальним інструментом для особистого та спільного фінансового планування.

ВИСНОВКИ

У межах даної кваліфікаційної роботи було розроблено вебзастосунок, призначений для ефективного обліку та аналізу особистих фінансів користувачів. Процес створення проєкту розпочався з глибокого аналізу існуючих вебсервісів у сфері управління особистими фінансами. Було досліджено їхні функціональні можливості, особливості дизайну та зручність використання. Визначено загальні тенденції, переваги та недоліки для подальшого створення більш інноваційного та орієнтованого на потреби користувачів застосунку.

У ході розробки було опановано та застосовано необхідний стек технологій. Серверну частину застосунку було реалізовано з використанням платформи Node.js та фреймворку Express.js. Клієнтська частина розроблена за допомогою мови програмування JavaScript та бібліотеки React, що дозволило створити сучасний та інтерактивний користувацький інтерфейс. Для реалізації функціоналу прогнозування бюджету було інтегровано бібліотеку TensorFlow.js. Зберігання та управління даними забезпечено за допомогою системи керування базами даних MongoDB. Також було розроблено REST API для ефективної взаємодії між клієнтською та серверною частинами застосунку та реалізовано інтеграцію з MonoBank API для автоматичного імпорту банківських транзакцій.

Розроблений вебзастосунок надає користувачам можливість зручно фіксувати свої доходи та витрати, здійснювати їхню категоризацію та аналіз. Після проведеного тестування було визначено потенційні напрямки для подальшого вдосконалення та розширення функціональних можливостей застосунку з метою надання користувачам ще більш повного та зручного інструменту для управління їхніми особистими фінансами.

Результати роботи апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [31].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Покрокове підключення MongoDB до React.js. URL: https://medium.com/@kaklotarrahul79/step-by-step-guide-connecting-mongodb-with-react-js-for-seamless-full-stack-development-db51c34da282 (дата звернення 16.04.2025).

2. Як користуватися MERN Stack. URL: https://www.mongodb.com/en-us/resources/languages/mern-stack-tutorial (дата звернення 17.04.2025).

3. Структура проєкту React для масштабування. URL: https://www.developerway.com/posts/react-project-structure (дата звернення 17.04.2025).

4. React документація англійською. URL: https://react.dev/reference/react (дата звернення 17.04.2025).

5. MongoDB документація англійською. URL: https://www.mongodb.com/docs/ (дата звернення 17.04.2025).

6. Гороховатський, В. О., & Творошенко, І. С. (2022). Аналіз багатовимірних даних за описом у формі множини компонент.

7. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

8. Morgan N. JavaScript Crash Course: A Hands-On, Project-Based Introduction to Programming. San Francisco: No Starch Press, Mar 2024. 376 p.

9. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2-ге вид. Sebastopol, CA: O'Reilly Media, 2020. 310 p.

10. Herman D. Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript. 1-ше вид. Boston: Addison-Wesley Professional, 2012. 240 p.

11. Crockford D. JavaScript: The Good Parts. Sebastopol: O'Reilly Media, 2008. 172 p.

12. Документація Monobank API Postman. URL: https://documenter.getpostman.com/view/25272928/2sA3JGe3W4/ (дата звернення 21.04.2025).

13. Monobank.API.Core 1.3.0. URL: https://www.nuget.org/packages/Monobank.API.Core/ (дата звернення 21.04.2025).

14. Khan, A., & Gonsalvez, J. J. (2023). The MERN Stack Revolution: A Review of its Impact on Modern Web Development. International Journal of Global Science and Technology, 12(SI4), 1300–1320. [pp. 1300–1305].

15. Johnson, M. K. (2021). Clean architecture: Implementing robust software design. Journal of Software Architecture and Design, 19(2), 45-68.

16. Shah, H., & Soomro, T. R. (2017). Node.js Challenges in Implementation. Global Journal of Computer Science and Technology, 17(2), 1–10. [pp. 3–7].

17. Walke, J. (2024). ReactJS: A JavaScript Library. International Research Journal of Modernization in Engineering Technology and Science, 4(4), 1–5. [pp. 2–4].

18. Cherednichenko, O., Vovk, M., Yanholenko, O., & Yakovleva, O. (2020). Towards the technology of employers' requirements collection development. In Integrated Computer Technologies in Mechanical Engineering: Synergetic Engineering (pp. 228-239). Cham: Springer International Publishing.

19. Kuzomin, O., & Lyashenko, V. (2022). Key Elements of a Specialized Complex for Solving Modeling Problems.

20. Gorokhovatskyi V.A., Zamula A.A. (2016) Employment of Intelligent Technologies in Multiparametric Control Systems. Telecommunications and Radio Engineering. Vol. 75, No 19, p. 1775–1785.

21. Змінні та режими середовища – Vite. URL: https://vite.dev/guide/env-and- mode (дата звернення 13.05.2025).

22. Kobylin, O., & Lyashenko, V. (2020). Time series clustering based on the k-means algorithm.

23. npm scripts – npm Docs URL: https://docs.npmjs.com/cli/v8/using-npm/scripts/ (дата звернення 13.05.2025).

24. Додавання ESLint та Prettier до проєкту ViteJS React URL: https://dev.to/marcosdiasdev/adding-eslint-and-prettier-to-a-vitejs-react-project-2kkj (дата звернення 18.05.2025).

25. Wikipedia. React. URL: https://uk.wikipedia.org/wiki/React (дата звернення 19.05.2025).

26. Domain-Driven Design and Its Application in Modern Software Architecture. Journal of Applied Software Engineering, 15(1), 34–58.

27. Chen, L., & Wang, M. (2023). Analysis of the Application Efficiency of TensorFlow and PyTorch in Neural Network Development. Journal of Computational Intelligence, 20(4), 400–420. [pp. 405–410].

28. Wilson, J., & Thompson, R. (2024). Trends in Node.js Framework Evolution: A Comprehensive Study. Journal of Web Development Technologies, 22(1), 60–80. [pp. 65–70].

29. Brown, A., & Davis, L. (2023). React Example Viability for Efficient API Learning (REVEAL): A Tool to Aid Novice Programmers. *Journal of Software Engineering Education*, 25(3), 180–200. [pp. 185–190].

30. Підключення бекенду до фронтенду в React URL: https://forum.freecodecamp.org/t/how-do-you-connect-the-backend-to-the-frontend-in-react/497067 (дата звернення 18.05.2025)

31. Кліменко Я.К. (2025) Ймовірнісні методи інтелектуального аналізу та моделювання фінансових часових рядів. *Радіоелектроніка і молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16–19 квітня 2025 р.).* Харків: ХНУРЕ, 2025. Т. 7. С. 60-62.