МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ПЕРСОНАЛІЗОВАНИЙ МЕНЕДЖЕР ФІНАНСІВ З КРОСПЛАТФОРМНОЮ ПІДТРИМКОЮ ДЛЯ ВЕБ ТА МОБІЛЬНИХ ЗАСТОСУНКІВ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Артем КИЛІВНИК**

«__» _____ 20__ р.

Керівник роботи

канд. техн. наук, доцент _____ **Євген ДАВИДЕНКО**

«__» _____ 20__ р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Килівника Артема

1. Тема кваліфікаційної роботи **Персоналізований менеджер фінансів з кросплатформною підтримкою для веб та мобільних застосунків** затверджена наказом ректора ЧНУ ім. Петра Могили № 315 від « 13 » Листопада 2024 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

– очікуваний результат: кросплатформний програмний комплекс, що дозволяє користувачам вести облік доходів/витрат, планувати бюджет та отримувати персоналізовану аналітику;

– початкові дані: вимоги до інтерфейсу користувача (UI/UX), перелік категорій фінансових операцій, архітектурні патерни для кросплатформної

розробки, вимоги до безпеки даних.

4. Перелік питань, що підлягають розробці:

– аналіз наявних рішень для управління фінансами та обґрунтування розробки власного продукту;

– проєктування архітектури бази даних та вибір технологічного стеку;

– розробка серверної частини (API) для синхронізації даних між платформами;

– проєктування адаптивного та інтуїтивно зрозумілого інтерфейсу;

– реалізація модулів аналітики, звітності та системи сповіщень;

– тестування програмного забезпечення та оцінка ефективності кросплатформного рішення.

5. Перелік графічних матеріалів:

Презентація

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «____» _____ 20___ р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Персоналізований менеджер фінансів з кросплатформною підтримкою для веб та мобільних застосунків**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	13.11.2024	13.11.2024	Виконано
2.	Аналіз предметної області та існуючих аналогів	01.02.2026	10.02.2026	Виконано
3.	Формування вимог до системи	11.02.2026	15.02.2026	Виконано
4.	Розробка технічного завдання	16.02.2026	20.02.2026	Виконано
5.	Проєктування архітектури системи	21.02.2026	28.02.2026	Виконано
6.	Проєктування структури бази даних	01.03.2026	07.03.2026	Виконано
7.	Розробка серверної частини	08.03.2026	25.03.2026	Виконано
8.	Розробка веб-інтерфейсу	15.03.2026	30.03.2026	Виконано
9.	Розробка мобільного застосунку	20.03.2026	10.04.2026	Виконано
10.	Реалізація механізму синхронізації даних	01.04.2026	15.04.2026	Виконано
11.	Тестування та виправлення помилок	15.04.2026	25.04.2026	Виконано
12.	Відгук керівника КБР	25.04.2026	05.05.2026	Виконано
13.	Оформлення КБР та презентації	05.05.2026	26.05.2026	Виконано
14.	Попередній захист	27.05.2026	27.05.2026	Виконано
15.	Завершення оформлення КБР та презентації	28.05.2026	12.06.2026	Виконано
16.	Рецензування			
17.	Захист кваліфікаційної роботи			

Здобувач _____ **Артем КИЛІВНИК**

«__» _____ 20__ р.

Керівник роботи

канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Персоналізований менеджер фінансів з кросплатформною підтримкою для веб та мобільних застосунків»

Здобувач 409 гр.: Килівник Артем

Керівник: канд. техн. наук, доцент Давиденко Євген

Актуальність полягає у зростальній потребі користувачів у зручних інструментах для управління особистими фінансами, аналізу витрат та планування бюджету в умовах сучасного цифрового середовища.

Метою роботи є розробка персоналізованого менеджера фінансів з кросплатформною підтримкою, що забезпечує облік доходів і витрат, управління категоріями та фінансовими цілями.

Для досягнення мети під час виконання роботи було проаналізовано сучасні підходи до розробки фінансових застосунків, визначено функціональні та нефункціональні вимоги до системи, спроєктовано архітектуру програмного забезпечення, розроблено інтерфейс користувача та реалізовано основний функціонал застосунку.

Об’єктом дослідження є процес управління особистими фінансами користувачів із використанням сучасних інформаційних технологій.

Предметом дослідження є методи та програмні засоби реалізації кросплатформної інформаційної системи персоналізованого управління фінансами з функціями обліку, аналізу, планування та синхронізації даних.

Програмний продукт забезпечує можливість ведення обліку транзакцій, управління категоріями доходів і витрат, встановлення фінансових цілей та отримання аналітичної інформації. Реалізована кросплатформна архітектура дозволяє використовувати систему як у вебсередовищі, так і на мобільних пристроях.

Отримані результати можуть бути використані для подальшого розвитку програмного продукту та впровадження у практичну діяльність.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність теми, визначено об’єкт та предмет дослідження, сформульовано мету та основні завдання кваліфікаційної роботи, а також визначено практичне значення отриманих результатів.

У першому розділі проведено аналіз предметної області персоналізованого управління фінансами, розглянуто сучасні програмні рішення для обліку доходів і витрат та виконано їх порівняльний аналіз.

У другому розділі виконано моделювання предметної області, проаналізовано сучасний стан інструментарію, моделей і методів розробки, сформовано функціональні та нефункціональні вимоги до системи, а також розроблено специфікацію вимог до програмного забезпечення.

У третьому розділі виконано проєктування програмного забезпечення, розроблено UML-діаграми, діаграму класів, діаграми послідовностей, станів та варіантів використання системи.

У четвертому розділі описано програмну реалізацію системи, зокрема серверної частини, вебзастосунку та мобільного застосунку, а також проведено тестування програмного продукту.У висновках наведено результати виконаної роботи та визначено перспективи подальшого розвитку програмного продукту.

Кваліфікаційна робота викладена на 86 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 21 найменування та 0 додатків. Праця містить 11 таблиць та 37 рисунків.

Ключові слова: менеджер фінансів, транзакції, категорії, фінансові цілі, кросплатформний застосунок, вебзастосунок, мобільний застосунок, аналіз витрат.

ABSTRACT

to the qualifying bachelor's thesis

Personalized finance manager with cross-platform support for web and mobile applications

Student of 409 group: Kylivnyk Artem

Supervisor: Ph.D. in Engineering, Associate Professor Davydenko Yevhen

The relevance of the topic lies in the growing need for convenient tools for personal finance management, expense analysis, and budget planning in the modern digital environment.

To achieve this aim, modern approaches to the development of financial applications were analyzed, functional and non-functional requirements for the system were defined, software architecture was designed, the user interface was developed, and the core functionality of the application was implemented.

The software product provides the ability to track transactions, manage income and expense categories, set financial goals, and obtain analytical information. The implemented cross-platform architecture allows the system to be used both in a web environment and on mobile devices.

The object of the research is the process of personal finance management using modern information technologies.

The subject of the research is methods and software tools for implementing a cross-platform information system for personalized finance management, including accounting, analysis, planning, and data synchronization functions.

The obtained results can be used for further development of the software product and its implementation in practical activities.

The qualification work consists of an introduction, 4 sections, conclusions and a list of references.

The introduction substantiates the relevance of the research topic, defines the object and subject of the study, formulates the purpose and main objectives of the qualification work, and determines the practical significance of the obtained results.

The first chapter presents an analysis of the personal finance management domain, reviews modern software solutions for income and expense tracking, and provides their comparative analysis.

The second chapter focuses on domain modeling, analyzes current development tools, models and methods, defines the functional and non-functional requirements of the system, and develops the software requirements specification.

The third chapter describes the software design process and includes the development of UML diagrams, class diagrams, sequence diagrams, state diagrams, and use case diagrams of the system.

The fourth chapter presents the implementation of the software system, including the server-side application, web application, and mobile application, as well as the testing of the developed software product.The qualification work is presented on 86 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 21 titles and 0 appendices. The work contains 11 tables and 37 figures.

Keywords: finance manager, transactions, categories, financial goals, cross-platform application, web application, mobile application, expense analysis.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному цифровому середовищі управління особистими фінансами стає важливою складовою повсякденного життя користувачів. Стрімке зростання кількості фінансових операцій, що підтверджується аналітичними прогнозами Statista щодо глобального масштабування ринку цифрових платежів [4], а також необхідність контролю витрат, планування бюджету та аналізу фінансової активності формують потребу у використанні спеціалізованих програмних засобів для обліку та управління фінансами.

Останніми роками значно зріс попит на мобільні та вебзастосунки для ведення персонального бюджету, оскільки користувачі прагнуть отримувати швидкий доступ до фінансової інформації з різних пристроїв. Особливої актуальності набувають кросплатформні рішення, які дозволяють синхронізувати дані між веб та мобільними платформами, забезпечуючи зручність та доступність використання системи.

В умовах економічної нестабільності, зростання цін та необхідності більш ефективного планування особистих витрат користувачі все частіше звертаються до цифрових інструментів фінансового контролю. Це підвищує актуальність розробки сучасних інформаційних систем для управління доходами, витратами та фінансовими цілями.

Метою кваліфікаційної роботи є розробка персоналізованого менеджера фінансів з кросплатформною підтримкою, що забезпечує облік доходів і витрат, управління категоріями та фінансовими цілями.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

– провести аналіз предметної області та існуючих програмних рішень у сфері управління особистими фінансами;

– обґрунтувати вибір технологій та інструментальних засобів розробки програмного продукту;

– виконати моделювання та проєктування програмного забезпечення;

- розробити структуру бази даних системи;
- реалізувати серверну частину програмного продукту;
- розробити вебзастосунок для взаємодії користувача із системою;
- реалізувати мобільний застосунок для роботи з системою;
- реалізувати механізм синхронізації даних та обміну інформацією;
- виконати тестування програмного продукту.

Об’єктом дослідження є процес управління особистими фінансами користувачів із використанням сучасних інформаційних технологій.

Предметом дослідження є методи та програмні засоби реалізації кросплатформної інформаційної системи персоналізованого управління фінансами з функціями обліку, аналізу, планування та синхронізації даних.

Практичне значення розроблюваного програмного продукту полягає у створенні кросплатформної інформаційної системи для обліку та аналізу особистих фінансів. Розроблений застосунок дозволить користувачам вести облік транзакцій, керувати категоріями, фінансовими цілями, аналізувати фінансову активність за допомогою статистичних звітів та графіків, а також забезпечувати синхронізацію даних між веб та мобільною версіями системи в режимі реального часу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз предметної області персоналізованого управління фінансами

У сучасних умовах розвитку цифрових технологій управління особистими фінансами стає важливою складовою повсякденного життя користувачів. Зростання кількості безготівкових платежів, використання електронних банківських сервісів та онлайн-покупок призводить до збільшення кількості фінансових операцій, які потребують контролю та аналізу [4].

Потреба у веденні обліку доходів і витрат зростає через необхідність ефективного планування бюджету та контролю фінансової активності. Багато користувачів стикаються з проблемами нераціонального використання коштів, відсутності систематизованого обліку витрат та складністю аналізу фінансової інформації. Як зазначають аналітики індустрії, формування чіткого бюджету є фундаментальним елементом управління капіталом, без якого неможливо забезпечити фінансову стабільність та досягнення довгострокових цілей [20]. Використання програмних засобів дозволяє автоматизувати ці процеси та забезпечити зручний доступ до фінансових даних.

Сучасні системи персонального фінансового менеджменту дозволяють користувачам вести облік транзакцій, створювати категорії витрат, формувати статистику та аналізувати структуру використання коштів. Додатково такі системи можуть підтримувати встановлення фінансових цілей, нагадування та аналітичні інструменти для оцінки фінансової активності користувача.

Однією з важливих тенденцій розвитку сучасного програмного забезпечення є кросплатформність. Користувачі очікують можливість працювати із системою як через вебзастосунок, так і за допомогою мобільних пристроїв. Це забезпечує доступність інформації незалежно від місця перебування користувача та підвищує зручність використання програмного продукту.

Крім того, сучасні інформаційні системи повинні забезпечувати швидкий доступ до даних, зручний інтерфейс користувача та можливість синхронізації

інформації між різними платформами. Це дозволяє підвищити ефективність використання системи та покращити користувацький досвід.

Таким чином, розробка персоналізованого менеджера фінансів із кросплатформною підтримкою є актуальним напрямом створення сучасного програмного забезпечення, що дозволяє користувачам ефективно контролювати фінансову активність та планувати особистий бюджет.

1.2 Функціональні особливості систем персонального фінансового менеджменту

Сучасні системи персонального фінансового менеджменту призначені для комплексної автоматизації процесів обліку, аналізу та стратегічного планування особистих фінансових ресурсів користувача [20]. Головною метою впровадження програмних рішень цього класу є надання кінцевому користувачу гнучких інструментів для операційного контролю витрат і доходів, оптимізації споживчих звичок та формування стійкого бюджету. Стрімкий розвиток індустрії цифрових платежів та фінтех-технологій загалом зумовлює високий попит на інтелектуальні системи управління капіталом [4], які дозволяють не лише фіксувати рух грошових коштів, а й здійснювати глибокий ретроспективний аналіз фінансової поведінки.

Одним із фундаментальних функціональних блоків систем персоналізованого фінансового менеджменту є модуль ведення первинного обліку транзакцій. Інформаційна система забезпечує можливість інтерактивного створення фінансових операцій із фіксацією таких метаданих, як точна сума, тип транзакції (надходження або вибуття коштів), часова мітка, текстовий коментар та аналітична категорія. Накопичення цих даних дозволяє згенерувати несуперечливу та структуровану історію фінансової активності, що є базисом для подальших розрахунків.

Ефективність аналізу структури витрат безпосередньо залежить від ручної категоризації транзакцій. Для забезпечення високої ергономіки інтерфейсу всі операції класифікуються за фіксованими або користувацькими доменами (наприклад: транспорт, логістика, продукти харчування, комунальні платежі,

розваги, заробітна плата). Впровадження ієрархічної категоризації дозволяє локалізувати зони найбільшого фінансового навантаження на бюджет та оцінити загальний розподіл капіталу за звітний період.

Ключовою відмінністю професійних фінансових сервісів є наявність розвинених інструментів бізнес-аналітики та інтерактивної візуалізації даних. Використання динамічних кругових і лінійних діаграм, гістограм та зведених статистичних звітів дозволяє трансформувати масиви сирих даних у наочні аналітичні зрізи. Завдяки графічному представленню інформації користувач може оперативно оцінювати баланс рахунків, відстежувати динаміку витрат у часі та виявляти приховані тенденції дефіциту чи профіциту бюджету.

Окреме місце в архітектурі функціонала посідають модулі довгострокового бюджетування та контролю фінансових цілей. Системи дозволяють встановлювати фінансові цілі, визначати цільові суми накопичень, розраховувати дедлайни та відображати поточний прогрес їх досягнення. Це безпосередньо сприяє підвищенню персональної фінансової дисципліни.

У сучасній практиці проектування додатків персоналізованого фінансового менеджменту обов'язковою вимогою є забезпечення парадигми кросплатформності. Ефективне управління фінансами передбачає безперешкодний доступ до профілю як зі стаціонарних робочих станцій (веб-версії), так і за допомогою мобільних пристроїв. Це реалізується шляхом збереження даних на сервері та впровадження механізмів синхронізації між клієнтськими платформами.

Оскільки фінансова інформація є конфіденційною, важливе значення мають механізми захисту даних користувачів. Безпека системи забезпечується за рахунок автентифікації користувачів, контролю доступу до персональних даних та логічного розмежування інформації між обліковими записами. Це дозволяє запобігти несанкціонованому доступу до фінансової інформації та забезпечити коректну роботу системи.

1.3 Аналіз існуючих аналогів та програмних рішень

На сучасному ринку програмного забезпечення існує велика кількість застосунків для управління особистими фінансами, які дозволяють вести облік доходів і витрат, аналізувати фінансову активність користувачів та планувати бюджет. Перед початком розробки власного програмного продукту було проведено аналіз існуючих систем персонального фінансового менеджменту.

Під час аналізу були визначені наступні критерії оцінювання програмних продуктів:

- назва застосунку;
- дистриб’ютор;
- архітектура програмного забезпечення;
- мова реалізації;
- основні функції та характеристики;
- переваги та недоліки;
- джерело інформації.

Для аналізу було обрано наступні програмні рішення:

- 1Money [1];
- YNAB [21];
- PocketGuard [11].

1.3.1 Аналіз 1Money

Одним із популярних мобільних застосунків для управління особистими фінансами є 1Money. Даний застосунок призначений для ведення обліку доходів і витрат користувача, аналізу фінансової активності та планування бюджету. Система підтримує роботу з категоріями транзакцій, побудову статистики та засоби візуалізації фінансової інформації.

Основні характеристики застосунку 1Money наведено у таблиці 1.1.

Таблиця 1.1 – Характеристики застосунку 1Money

Назва	1Money - Expense Tracker, Money Manager, Budget
Дистриб’ютор	Esin Dmitrii
Архітектура	Клієнт-серверний мобільний застосунок (Client-server mobile application)
Мова реалізації	Swift (iOS), Java/Kotlin (Android), cloud backend technologies
Основні функції/ характеристики	– ведення доходів та витрат вручну; – категоризація транзакцій; – побудова діаграм для аналізу; – планування бюджету по категоріях; – управління рахунками та балансами; – захист доступу (PIN / біометрія).
Переваги	– інтуїтивно зрозумілий інтерфейс; – проста і швидка робота з витратами; – зручна візуалізація у вигляді діаграм; – можливість деталізації категорій; – відображення балансів по рахунках.
Недоліки	– відсутність повноцінної вебверсії; – обмежена кросплатформна взаємодія; – частина функціоналу доступна лише у платній версії; – обмежені можливості інтеграції з банківськими сервісами;
Посилання	Джерело інформації [1].

Підсумовуючи аналіз 1Money, варто зазначити, що цей програмний продукт позиціонується насамперед як мобільний трекер «на щодень». Застосунок орієнтований на аудиторію, якій важлива швидкість фіксації витрат безпосередньо в момент покупки. Інтуїтивна ергономіка та якісна інфографіка полегшують первинне освоєння програми. Водночас жорстка прив'язка до мобільних платформ

та відсутність десктопного чи веб-інтерфейсу суттєво обмежують можливості користувача при необхідності глибокого ретроспективного аналізу великих масивів даних.

1.3.2 Аналіз YNAB

Одним із найбільш функціонально розвинених застосунків для управління особистими фінансами є YNAB. Даний програмний продукт орієнтований на детальне планування бюджету, контроль витрат та формування фінансової дисципліни користувачів. Система підтримує механізми синхронізації між пристроями, управління фінансовими цілями та аналіз фінансової активності.

Основні характеристики застосунку YNAB наведено у таблиці 1.2.

Таблиця 1.2 – Характеристики застосунку YNAB

Назва	YNAB
Дистриб’ютор	YNAB Inc.
Архітектура	SaaS (web + mobile, client-server)
Мова реалізації	JavaScript (frontend), cloud backend technologies
Основні функції/характеристики	– планування витрат наперед; – встановлення фінансових цілей; – аналіз поведінки користувача; – синхронізація між пристроями; – управління боргами.
Переваги	– глибокий функціонал для детального бюджетування; – ефективне формування фінансової дисципліни; – розвинені інструменти управління боргами та фінансовими цілями; – високий рівень безпеки та синхронізації між пристроями.

Кінець таблиці 1.2

Недоліки	– платна підписка; – високий поріг входу; – потрібно багато ручного введення.
Посилання	Офіційний сайт застосунку [21].

Дослідження системи YNAB дозволяє стверджувати, що вона є не просто інструментом обліку, а комплексною методологічною платформою для формування фінансової дисципліни. Головна цінність рішення полягає у превентивному плануванні (розподілі ще не витрачених коштів), що кардинально відрізняє його від класичних трекерів постфактум-витрат. Проте висока ефективність системи вимагає від користувача значних часових інвестицій на рутинне введення даних та готовності до тривалого навчання, що разом із виключно платною моделлю дистрибуції створює відчутний поріг входження.

1.3.3 Аналіз PocketGuard

PocketGuard є сучасною системою персонального фінансового менеджменту, орієнтованою на спрощення процесу контролю щоденних витрат користувача. Застосунок дозволяє аналізувати доходи та витрати, планувати бюджет, контролювати фінансову активність та синхронізувати дані між пристроями.

Основні характеристики застосунку PocketGuard наведено у таблиці 1.3.

Таблиця 1.3 – Характеристики застосунку PocketGuard

Назва	PocketGuard
Дистриб’ютор	PocketGuard Inc.
Архітектура	Кросплатформна клієнт-серверна система (web + mobile)
Мова реалізації	JavaScript (web frontend), Kotlin/Swift (mobile applications), cloud backend technologies

Кінець таблиці 1.3

Основні функції/характеристики	– аналіз доходів і витрат; – “скільки можна витратити” (In My Pocket); – відстеження підписок; – планування бюджету; – управління боргами; – синхронізація банківських рахунків.
Переваги	– простота використання; – мінімальний час налаштування; – фокус на щоденних витратах.
Недоліки	– обмежені можливості фінансової аналітики; – недостатньо гнучкі інструменти планування бюджету; – обмежена кастомізація функціоналу; – менший набір функцій у порівнянні з комплексними фінансовими системами.
Посилання	Офіційний сайт застосунку [11].

На відміну від попередніх аналогів, PocketGuard робить ставку на концепцію мінімалізму та концепт "In My Pocket" (оперативний розрахунок залишку вільних коштів). Продукт націлений на автоматичний моніторинг регулярних платежів та підписок, що звільняє користувача від необхідності тримати в голові постійні витрати. З іншого боку, зворотним боком такої спрощеної моделі є надто лінійна аналітика: система не здатна задовольнити потреби користувачів, які шукають інструменти для детального стратегічного планування довгострокових фінансових цілей чи інвестицій.

Висновки до розділу 1

У першому розділі було проведено аналіз предметної області систем персонального фінансового менеджменту та визначено основні особливості

сучасних програмних засобів для обліку й контролю фінансової активності користувачів.

У процесі дослідження встановлено, що розвиток цифрових технологій, зростання кількості фінансових операцій та поширення мобільних сервісів формують потребу у використанні спеціалізованих програмних рішень для управління доходами, витратами та особистим бюджетом. Визначено, що сучасні системи персонального фінансового менеджменту повинні забезпечувати зручний інтерфейс користувача, підтримку аналітичних інструментів, категоризацію транзакцій та синхронізацію даних між платформами.

Також було розглянуто функціональні особливості систем персонального фінансового менеджменту, зокрема механізми обліку транзакцій, категоризації операцій, управління фінансовими цілями та забезпечення кросплатформної взаємодії. Встановлено, що важливими характеристиками сучасних систем є підтримка веб та мобільних платформ, централізоване збереження даних та забезпечення зручного доступу користувача до фінансової інформації.

У межах аналізу існуючих програмних рішень було досліджено системи 1Money, YNAB та PocketGuard. Визначено їх основні функціональні можливості, переваги та недоліки. Проведений аналіз дозволив встановити, що сучасні програмні продукти забезпечують широкий набір функцій для фінансового менеджменту, проте мають окремі обмеження щодо кросплатформності, гнучкості налаштування або доступності окремого функціоналу.

Отримані результати аналізу предметної області та існуючих програмних рішень створюють основу для подальшого моделювання та формування вимог до програмної системи персоналізованого управління фінансами.

2 МОДЕЛЮВАННЯ ОБ'ЄКТА ТА ПРЕДМЕТА РОБОТИ

2.1 Аналіз сучасного стану інструментарію, моделей та методів

Сучасний розвиток інформаційних технологій характеризується активним переходом до кросплатформних, масштабованих та клієнт-серверних програмних систем. Особливо актуальним це є для сфери персонального фінансового менеджменту, де програмне забезпечення повинно забезпечувати швидку обробку даних, доступність із різних пристроїв, синхронізацію інформації та підтримку роботи в режимі реального часу. У зв'язку з цим при розробці сучасних web- та mobile-систем значна увага приділяється вибору технологічного стеку, архітектурних моделей та інструментів розробки.

Одним із найбільш поширених підходів до побудови сучасних інформаційних систем є клієнт-серверна архітектура. Даний підхід передбачає розділення програмного забезпечення на frontend- та backend-компоненти, що взаємодіють між собою через мережеві протоколи та API-інтерфейси. Така модель дозволяє забезпечити незалежний розвиток клієнтської та серверної частин системи, спрощує масштабування програмного забезпечення та покращує підтримуваність проєкту.

У сучасній frontend-розробці домінуючим підходом є використання SPA-архітектури (Single Page Application), яка забезпечує асинхронне оновлення інтерфейсу без повного перезавантаження сторінки. Одним із найбільш популярних інструментів реалізації SPA-застосунків є бібліотека React [14]. Згідно з глобальним аналітичним дослідженням 2025 Stack Overflow Developer Survey, React стабільно посідає провідні позиції як одна з найпопулярніших та найбільш використовуваних frontend-технологій серед професійних розробників у всьому світі, що пояснюється її компонентною архітектурою, високою продуктивністю та широкою екосистемою готових рішень [2].

Важливою тенденцією сучасної веброзробки є активне використання TypeScript [17]. У порівнянні зі стандартним JavaScript, дана мова програмування забезпечує підтримку статичної типізації, що дозволяє суттєво зменшити кількість

помилок у великих програмних системах та покращити супровід програмного коду. Згідно з аналітичним звітом «Статистика розробників програмного забезпечення 2024 - Звіт про стан екосистеми розробників» від компанії JetBrains, TypeScript демонструє стабільне зростання популярності серед інженерів, інтегруючись як обов'язковий стандарт при створенні корпоративних та масштабованих web-застосунків [16].

Для реалізації мобільних застосунків останніми роками активно використовуються кросплатформні технології, що дозволяють створювати Android- та iOS-застосунки на основі єдиної кодової бази. Одним із найбільш популярних рішень у даній сфері є React Native, який базується на екосистемі React та JavaScript. Використання кросплатформного підходу дозволяє суттєво скоротити витрати на розробку мобільного програмного забезпечення та спростити підтримку системи на різних платформах [16].

Для реалізації серверної частини сучасних інформаційних систем широко використовуються багаторівневі архітектури та REST-підхід до організації взаємодії між клієнтом і сервером. REST API забезпечує стандартизований механізм обміну даними через HTTP-протокол із використанням JSON-формату. Подібний підхід дозволяє реалізувати незалежну взаємодію між web-, mobile- та backend-компонентами системи, а також спрощує інтеграцію із зовнішніми сервісами.

У межах даної роботи для реалізації серверної частини обрано платформу Jakarta EE [8], яка забезпечує набір стандартизованих специфікацій для створення корпоративних інформаційних систем. Використання Jakarta EE дозволяє реалізовувати REST-сервіси, організовувати транзакційну взаємодію з базами даних, підтримувати багаторівневу архітектуру та забезпечувати масштабованість програмного забезпечення.

Для збереження та обробки фінансової інформації у сучасних інформаційних системах широко використовуються реляційні системи керування базами даних. Однією з найбільш популярних СКБД є PostgreSQL [12]. За результатами дослідження 2025 Stack Overflow Developer Survey, PostgreSQL залишається

найбільш затребуваною та популярною СКБД серед професійної спільноти [2], оскільки забезпечує повну підтримку транзакційності та відповідність стандартам сучасних баз даних [7], високу стабільність роботи під навантаженням та ефективну обробку складних SQL-запитів, що є критичним для систем обліку фінансів. Використання PostgreSQL дозволяє забезпечити цілісність даних та підтримувати горизонтальне і вертикальне масштабування системи.

У сучасній практиці розробки програмного забезпечення також активно застосовуються ORM-технології. Одним із найбільш поширених рішень у Java-екосистемі є JPA [10], яка забезпечує об’єктно-реляційне відображення між Java-сутностями та таблицями бази даних. Це дозволяє спростити взаємодію із СКБД та зменшити обсяг низькорівневого SQL-коду.

Окрему роль у сучасних web-системах відіграють технології роботи в режимі реального часу. Для забезпечення миттєвої синхронізації даних між сервером і клієнтом широко використовується технологія WebSocket, офіційна специфікація якої визначена інтернет-стандартом RFC 6455 [19]. Вона дозволяє організувати постійне двостороннє з’єднання без необхідності регулярного надсилання HTTP-запитів. Подібний підхід особливо актуальний для інформаційних систем, де необхідне оперативне оновлення даних користувача.

Однією з ключових тенденцій сучасної розробки є контейнеризація програмного забезпечення. Для цього активно використовується Docker [3], який забезпечує ізоляцію середовища виконання та стандартизацію процесу розгортання програмного забезпечення. Згідно з сучасними аналітичними дослідженнями, контейнеризація дозволяє зменшити проблеми сумісності системних залежностей та спрощує перенесення програмного забезпечення між різними середовищами.

Таким чином, сучасний стан розвитку інструментарію, моделей та методів web- і mobile-розробки демонструє активне використання клієнт-серверної архітектури, SPA-підходу, REST API, кросплатформних технологій та контейнеризації програмного забезпечення. Аналіз сучасних тенденцій та технологічних підходів підтверджує доцільність використання React, TypeScript,

React Native, Jakarta EE, PostgreSQL та Docker для розробки програмної системи персоналізованого управління фінансами.

2.2 Моделювання предметної області

Концептуальне проектування кросплатформної системи «FinanceManager» базується на формалізації процесів обліку, розподілу та моніторингу приватного капіталу користувача. Основне завдання моделювання предметної області полягає в оптимізації інформаційних потоків між клієнтськими застосунками та реляційною базою даних для забезпечення високої швидкодії та цілісності інформації. На основі аналізу функціональних вимог виділено ключові абстракції системи (користувач, рахунок, категорія, транзакція, фінансова ціль), що утворюють логічний каркас бази даних.

Для технічної реалізації розробленої моделі предметної області визначено багаторівневий архітектурний стек, який структурує взаємодію між компонентами системи (табл. 2.1).

Таблиця 2.1 – Технологічний стек системи «FinanceManager»

Шар системи	Технологія	Обґрунтування вибору
Клієнтська частина	React + TypeScript	React забезпечує компонентну архітектуру та високу швидкодію SPA-застосунків, а TypeScript дозволяє реалізувати статичну типізацію та покращити підтримуваність програмного коду
Серверна частина	Jakarta EE	Платформа забезпечує реалізацію REST API, підтримку багаторівневої архітектури, роботу з транзакціями та взаємодію з базою даних
Система керування базами даних	PostgreSQL	PostgreSQL підтримує ACID-транзакції, забезпечує стабільність роботи та ефективну обробку SQL-запитів

Кінець таблиці 2.1

ORM-рівень	JPA	Забезпечує об’єктно-реляційне відображення між Java-сутностями та таблицями бази даних
Робота в режимі реального часу	WebSocket	Дозволяє реалізувати двосторонню синхронізацію даних між сервером та клієнтом
Контейнеризація	Docker	Забезпечує ізольованість середовища виконання та стандартизує процес розгортання системи

Архітектурна взаємодія компонентів системи організована за класичною розподіленою клієнт-серверною моделлю. Користувач ініціює операції через клієнтський інтерфейс, який за допомогою асинхронних HTTP-запитів звертається до REST-ендпоінтів сервера. Серверна частина, розгорнута в ізольованому Docker-контейнері, інкапсулює бізнес-логику: здійснює валідацію вхідних даних, автентифікацію сесії та через рівень JPA-репозиторіїв взаємодіє із СКБД PostgreSQL. Обмін даними між шарами системи уніфіковано за допомогою текстового формату JSON.

Особливістю моделювання процесів у системі «FinanceManager» є потреба в оперативному реагуванні на тригери зміни балансу. Це забезпечується інтеграцією протоколу WebSocket. Будь-яка транзакційна подія (створення, редагування або видалення транзакції, категорії чи фінансової цілі) автоматично транслюється сервером на активні клієнтські сесії без додаткових HTTP-запитів, що гарантує консистентність відображення фінансового стану в реальному часі.

Побудована модель предметної області є слабозв'язаною та інваріантною до операційного середовища. Чітке розмежування відповідальності між рівнем представлення (React), бізнес-сервісами (Jakarta EE) та рівнем збереження даних (PostgreSQL) спрощує супровід програмного коду. Окрім того, використання Docker-контейнеризації нівелює ризики несумісності системних залежностей при переносі системи між різними хост-машинами.

Проводячи подальшу характеристику обраного технологічного стеку, необхідно окреслити, яким чином його базові елементи підтримують повний життєвий цикл системи «FinanceManager» – від поточного прототипу до потенційної комерційної експлуатації. Нижче наведено аналіз ключових площин розвитку архітектури.

Масштабованість. Застосована архітектурна модель є «горизонтально гнучкою». За умови зростання користувацької бази, frontend-частина, яка є статичним набором файлів, може бути легко делегована на мережі доставки контенту. Серверна частина на Jakarta EE, завдяки ізоляції в контейнерах Docker [3], масштабується шляхом запуску додаткових екземплярів додатку (інстансів) під керуванням балансувальника навантаження. Це не потребує модифікації коду системи, оскільки сесії є безстанційними. На рівні даних PostgreSQL підтримує механізми реплікації, що дозволяє масштабувати аналітичні запити без зниження швидкодії основної бази.

Портативність та кросплатформність. Обрані технологічні рішення є повністю інваріантними до операційного середовища. Веб-інтерфейс адаптований під будь-які сучасні браузери, мобільний застосунок функціонує на базі операційних систем Android та iOS з єдиної кодової бази, а серверний контейнер Docker разом із СКБД PostgreSQL може бути розгорнутий на будь-кому Unix-подібному або Windows-сервері. Конфігурація розгортання та керування версіями вихідного коду повністю інкапсульовані в межах Git [6], що спрощує міграцію проєкту між різними хмарними провайдерами чи локальними хостингами.

Обслуговуваність та підтримка. Статична типізація TypeScript мінімізує виникнення логічних помилок на етапі компіляції клієнтського коду, а архітектура шарів Jakarta EE (контролери, сервіси, репозиторії) забезпечує суворе розмежування відповідальності. Такий підхід полегшує проведення рев'ю коду, модульного тестування та інтеграцію нових розробників у проєкт. Документування API спрощується завдяки автоматичній генерації специфікацій безпосередньо з анотацій вихідного коду серверної частини.

Безпека даних. Оскільки система оперує конфіденційною фінансовою інформацією користувачів, важливим аспектом архітектури є забезпечення контрольованого доступу до персональних даних. У системі реалізовано механізми реєстрації та автентифікації користувачів, що дозволяють ідентифікувати власника фінансового профілю та обмежити доступ до його транзакцій, категорій і фінансових цілей. Логічне розмежування даних здійснюється за ідентифікатором користувача, завдяки чому кожен обліковий запис працює лише з власною фінансовою інформацією. Додатково взаємодія з базою даних виконується через ORM-рівень JPA, що зменшує ризики помилок під час формування SQL-запитів та підвищує надійність роботи з даними.

Економічна доцільність розробки. Усі обрані компоненти стеку (React, Jakarta EE, PostgreSQL, Docker) поширюються за вільними ліцензіями з відкритим вихідним кодом (Open Source), що виключає необхідність ліцензійних відрахувань. Наявність великих глобальних спільнот навколо цих технологій забезпечує доступ до розгалуженої документації, готових рішень та спрощує подальший супровід системи.

Функціональний розвиток системи залишається прогнозованим завдяки модульній архітектурі програмного забезпечення. Поточна версія системи реалізує основні функції персонального фінансового менеджменту, зокрема облік доходів і витрат, керування категоріями транзакцій, управління фінансовими цілями та синхронізацію даних між веб- і мобільною платформами. Використання багаторівневої архітектури на базі Jakarta EE дозволяє розширювати функціональні можливості системи без суттєвих змін існуючих компонентів. У подальшому до системи можуть бути додані нові модулі, зокрема засоби розширеної фінансової аналітики, формування звітів, інтеграція із зовнішніми сервісами та інші інструменти підтримки прийняття фінансових рішень користувачем.

Таким чином, сформований стек технологій (React / TypeScript / React Native / Jakarta EE / PostgreSQL / Docker) демонструє оптимальний баланс між швидкістю розробки, надійністю та архітектурною гнучкістю, що підтверджує його доцільність для створення системи персоналізованого фінансового менеджменту.

2.3 Формування функціональних та нефункціональних вимог

Після проведення аналітичного огляду предметної області та дослідження наявних програмних аналогів було сформовано комплекс вимог до проектованої інформаційної системи. Етап специфікації вимог є критично важливою фазою життєвого циклу розробки, оскільки він окреслює межі проєкту, визначає логіку взаємодії користувача з інтерфейсом та задає критерії оцінки якості майбутнього програмного забезпечення.

Відповідно до фундаментальних засад інженерії програмного забезпечення, процес виявлення та структурування вимог є першочерговою фазою аналізу та моделювання системи. Як зазначає Роджер Прессман, чітка класифікація вимог дозволяє повноцінно описати очікувану поведінку програмного забезпечення, визначити сценарії взаємодії користувачів із системою та закласти надійну основу для верифікації й тестування майбутнього продукту [13].

Створюваний персоналізований менеджер фінансів «FinanceManager» має забезпечувати гнучкий облік транзакцій, аналітичну обробку фінансових показників та безперешкодну асинхронну взаємодію між веб- та мобільними клієнтськими платформами. Для деталізації архітектурних завдань усі вимоги розподілено на дві категорії: функціональні та нефункціональні.

2.3.1 Функціональні вимоги

Функціональні вимоги визначають конкретні сервіси, алгоритми та операції, які повинна виконувати інформаційна система для задоволення потреб кінцевого користувача.

До переліку основних функціональних вимог системи «FinanceManager» належать:

– автентифікація, реєстрація та авторизація користувачів із розмежуванням прав доступу до персональних профілів;

– виконання операцій створення, перегляду та видалення фінансових транзакцій;

– роздільне ведення обліку операцій за типами (доходи, витрати);

– динамічне керування користувацькими категоріями фінансових операцій;

– логічна прив’язка транзакцій до відповідних категорій користувача;

– фільтрація, сортування та відображення деталізованої історії фінансової активності;

– агрегація даних, аналіз фінансового стану та графічна візуалізація статистики (інфографіка);

– встановлення довгострокових та короткострокових фінансових цілей, а також моніторинг прогресу їх досягнення;

– асинхронна синхронізація стану та балансів між веб-інтерфейсом і мобільним додатком;

– реал-тайм сповіщення клієнтських застосунків про зміну фінансових метрик на сервері;

– забезпечення стандартизованої API-взаємодії між клієнтськими платформами та backend-шаром.

2.3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики якості програмного продукту, продуктивність системи та особливості її використання.

Основними нефункціональними вимогами є:

– забезпечення кросплатформної підтримки веб та мобільних пристроїв;

– зручність та зрозумілість інтерфейсу користувача;

– швидка обробка запитів користувача;

– забезпечення безпечного доступу до даних користувача;

– можливість масштабування системи у майбутньому;

– стабільна робота програмного продукту при одночасній роботі декількох користувачів;

– підтримка синхронізації даних у режимі реального часу;

– забезпечення збереження даних користувача у базі даних;

- модульність архітектури програмного забезпечення;
- можливість подальшого розширення функціоналу системи.

Таким чином, сформовані функціональні та нефункціональні вимоги чітко окреслюють вектор проектування та реалізації системи «FinanceManager». Систематизований опис цих параметрів виступає в ролі технічного базису, який мінімізує ризики архітектурних помилок при подальшій декомпозиції системи та безпосередній програмній реалізації її компонентів.

2.4 Специфікація вимог до програмного забезпечення

2.4.1 Призначення та межі проєкту

Персоналізований менеджер фінансів призначений для забезпечення користувачів зручним інструментом для ведення обліку доходів і витрат, аналізу фінансової активності та планування особистого бюджету.

Програмний продукт забезпечує можливість роботи із фінансовими даними через вебзастосунок та мобільний застосунок, а також підтримує синхронізацію інформації між платформами.

2.4.2 Межі проєкту

Проєкт охоплює розробку:

- серверної частини системи;
- вебзастосунку;
- мобільного застосунку;
- системи авторизації користувачів;
- механізмів обліку транзакцій та категорій;
- системи синхронізації даних між платформами.

Проєкт не включає:

- інтеграцію з реальними банківськими API;
- автоматичний імпорт транзакцій із банківських систем;
- реалізацію платіжних систем;

– розробку desktop-застосунку.

2.4.3 Загальний опис

Сфера застосування

Сфера застосування програмного продукту включає системи персонального фінансового менеджменту та програмні засоби для обліку особистих фінансів користувачів.

Застосунок дозволяє користувачам вести облік фінансових операцій, аналізувати структуру витрат та планувати бюджет.

Характеристики користувачів

Користувачами системи є особи, які потребують інструментів для:

– ведення обліку доходів і витрат;

– аналізу фінансової активності;

– контролю фінансових цілей.

Користувачі повинні мати базові навички роботи із веб та мобільними застосунками.

Загальна структура і склад системи

Система складається з наступних компонентів:

– вебзастосунок;

– мобільний застосунок;

– серверна частина;

– база даних.

Веб та мобільний застосунки забезпечують взаємодію користувача із системою, тоді як серверна частина відповідає за обробку запитів, авторизацію користувачів та взаємодію з базою даних.

Загальні обмеження

Основним обмеженням системи є необхідність доступу до мережі Інтернет для синхронізації даних та взаємодії клієнтських застосунків із серверною частиною.

2.4.4 Функції системи

Авторизація та аутентифікація

Опис функції

Система забезпечує можливість реєстрації та авторизації користувачів для доступу до персональних фінансових даних.

Вхідна і вихідна інформація

Вхідна інформація:

- електронна пошта;
- пароль користувача.

Вихідна інформація:

- інформація про користувача;
- статус авторизації.

Функціональні вимоги

- реєстрація нового користувача;
- авторизація користувача у системі;
- перевірка правильності введених даних;
- обмеження доступу до персональної інформації інших користувачів.

Управління транзакціями

Опис функції

Система забезпечує можливість створення, перегляду та видалення фінансових транзакцій користувача.

Вхідна і вихідна інформація

Вхідна інформація:

- сума транзакції;
- тип транзакції;
- опис;
- категорія.

Вихідна інформація:

- список транзакцій користувача;

– оновлена фінансова статистика.

Функціональні вимоги

– створення транзакцій;

– видалення транзакцій;

– відображення історії фінансових операцій.

Управління категоріями

Опис функції

Система забезпечує можливість створення та редагування категорій доходів і витрат.

Вхідна і вихідна інформація

Вхідна інформація:

– назва категорії.

Вихідна інформація:

– список категорій користувача.

Функціональні вимоги

– створення категорій;

– редагування категорій;

– видалення категорій;

– прив’язка транзакцій до категорій.

Аналіз фінансової активності

Опис функції

Система забезпечує можливість аналізу фінансової активності користувача та візуалізації статистичних даних.

Вхідна і вихідна інформація

Вхідна інформація

– список транзакцій користувача.

Вихідна інформація:

– статистика доходів і витрат;

– діаграми та аналітична інформація.

Функціональні вимоги

- формування статистики витрат;
- візуалізація фінансових даних;
- оновлення інформації у режимі реального часу.

2.4.5 Вимоги до інформаційного забезпечення

Джерела і зміст вхідної інформації

Основним джерелом вхідної інформації є користувач системи. Користувач вводить дані про фінансові операції, категорії витрат та фінансові цілі.

Вимоги до організації та збереження інформації

Для реалізації системи збереження даних повинна використовуватися реляційна система керування базами даних, яка забезпечує повну відповідність вимогам ACID (транзакційність, ізольованість, надійність), підтримує роботу зі складними реляційними зв'язками, зовнішніми ключами та має високу швидкість обробки аналітичних SQL-запитів при масштабуванні.

База даних забезпечує збереження інформації про

- користувачів;
- транзакції;
- категорії;
- фінансові цілі.

2.4.6 Вимоги до технічного забезпечення

Для роботи системи необхідний персональний комп’ютер або мобільний пристрій із доступом до мережі Інтернет та підтримкою сучасних веббраузерів або мобільних операційних систем.

2.4.7 Вимоги до програмного забезпечення

Архітектура програмної системи

Архітектура системи включає:

- вебзастосунок;

- мобільний застосунок;
- серверну частину;
- базу даних.

Системне програмне забезпечення

Серверна частина системи реалізується на платформі Jakarta EE з підтримкою REST-сервісів, об'єктно-реляційного відображення та модульної архітектури застосунків. Для реалізації клієнтської частини вебзастосунку має використовуватися компонентно-орієнтована бібліотека розробки користувацьких інтерфейсів із підтримкою віртуального DOM. Для мобільного застосунку необхідно застосувати кросплатформну технологію моделювання інтерфейсів на основі єдиної бази коду для операційних систем Android та iOS.

Програмне забезпечення ведення інформаційної бази

Для реалізації системи збереження даних використовується PostgreSQL.

Засоби розробки

Розробка програмного продукту здійснюється із використанням IntelliJ IDEA, WebStorm, Docker та системи контролю версій Git.

2.4.8 Вимоги до зовнішніх інтерфейсів

Інтерфейс користувача

Інтерфейс користувача повинен забезпечувати простоту використання, зручність навігації та доступність основного функціоналу системи.

Програмний інтерфейс

Для взаємодії між клієнтськими застосунками та серверною частиною використовується REST API та WebSocket-з’єднання.

2.4.9 Властивості програмного забезпечення

Доступність

Система повинна бути доступною для користувача з різних типів пристроїв.

Переносимість

Програмний продукт повинен підтримувати роботу у сучасних веббраузерах та мобільних операційних системах.

Продуктивність

Система повинна забезпечувати швидку обробку запитів користувача та стабільну роботу при одночасній взаємодії декількох користувачів.

Надійність та безпечність

Доступ до персональних даних користувача повинен здійснюватися лише після проходження авторизації.

Система повинна забезпечувати захист інформації та обмеження доступу до даних інших користувачів.

Висновки до розділу 2

В даному розділі було проведено аналіз сучасного стану інструментарію, моделей та методів web- і mobile-розробки, що використовуються при створенні сучасних інформаційних систем персонального фінансового менеджменту. У процесі дослідження було визначено актуальні технологічні підходи, архітектурні моделі та програмні засоби, які забезпечують масштабованість, кросплатформність та підтримуваність програмного забезпечення.

На основі аналізу сучасних аналітичних досліджень та технологічних тенденцій було обґрунтовано доцільність використання React, TypeScript, React Native, Jakarta EE, PostgreSQL, Docker та WebSocket для реалізації програмної системи «FinanceManager». Встановлено, що використання SPA-архітектури, REST API та контейнеризації програмного забезпечення відповідає сучасним тенденціям розробки web- і mobile-систем.

У межах моделювання предметної області було визначено основні сутності системи, структуру взаємодії між компонентами та архітектурну модель програмного продукту. Також було сформовано технологічний стек системи та проаналізовано особливості його використання з точки зору масштабованості, кросплатформності, безпеки та підтримуваності програмного забезпечення.

Крім того, у розділі було сформовано функціональні та нефункціональні вимоги до системи «FinanceManager», а також виконано специфікацію вимог до програмного забезпечення. Визначено призначення системи, межі проєкту, функціональні можливості, вимоги до програмного та технічного забезпечення, а також характеристики користувацького інтерфейсу та властивостей програмного продукту.

Отримані результати формують теоретичну та технологічну основу для подальшого етапу конструювання та програмної реалізації системи персоналізованого управління фінансами.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка UML-діаграм

3.1.1 Діаграма варіантів використання

На етапі проєктування персоналізованого менеджера фінансів було визначено основні сценарії взаємодії користувачів із системою. Для візуалізації функціональних можливостей програмного продукту та ролей користувачів було використано діаграму варіантів використання (Use Case Diagram), яка є одним із базових засобів UML-моделювання та дозволяє відобразити межі системи і взаємодію зовнішніх акторів із її функціоналом [5].

Діаграма варіантів використання демонструє основні функції системи з точки зору кінцевих користувачів, визначає перелік доступних сценаріїв роботи та дозволяє встановити розподіл відповідальності між різними ролями. Використання даного типу діаграм спрощує аналіз функціональних вимог та забезпечує формування цілісного уявлення про поведінку програмного забезпечення на етапі проєктування.

У процесі розробки системи було визначено два основні актори:

– користувач;

– адміністратор.

Користувач є основним актором системи та має можливість проходити процедури реєстрації й авторизації, здійснювати управління фінансовими транзакціями та категоріями, переглядати статистику фінансової активності, а також працювати з фінансовими цілями.

Адміністратор є спеціалізованою роллю користувача та додатково має доступ до функцій перегляду користувачів системи, отримання статистичної інформації про користувачів та видалення облікових записів у разі необхідності.

Діаграма варіантів використання персоналізованого менеджера фінансів наведена на рисунку 3.1.

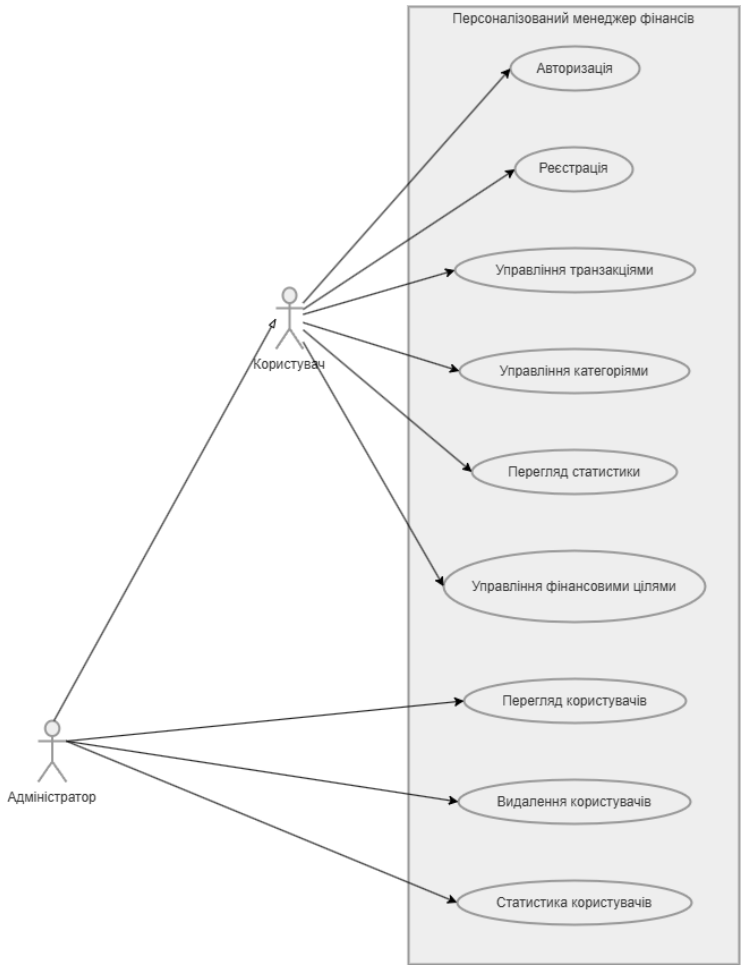

Рисунок 3.1 – Діаграма варіантів використання системи

Розроблена діаграма варіантів використання відображає основні сценарії взаємодії користувачів із системою «FinanceManager», демонструє розподіл функціональних можливостей між ролями користувача та адміністратора, а також визначає межі програмного продукту. Побудована модель стала основою для подальшого проєктування архітектури системи, UML-діаграм структурного та поведінкового рівнів, а також програмної реалізації компонентів застосунку.

3.1.2 Розробка діаграми класів

Одним із важливих етапів проєктування програмного забезпечення є розробка діаграми класів, яка дозволяє відобразити логічну структуру системи, основні сутності предметної області, їх атрибути, методи та взаємозв’язки між ними [5].

Діаграма класів використовується для моделювання архітектури програмного забезпечення та визначення основних компонентів системи. У процесі проєктування персоналізованого менеджера фінансів FinanceManager було визначено основні класи, що забезпечують реалізацію функціональних можливостей системи.

До основних сутностей системи належать:

– User – клас користувача системи;

– Transaction – клас фінансових транзакцій;

– Category – клас категорій доходів і витрат;

– Goal – клас фінансових цілей користувача.

Для реалізації бізнес-логіки були визначені сервісні класи:

– AuthService – сервіс авторизації та аутентифікації користувачів;

– TransactionService – сервіс управління транзакціями;

– CategoryService – сервіс управління категоріями;

– GoalService – сервіс управління фінансовими цілями.

Між класами системи реалізовано асоціативні зв’язки типу «один до багатьох». Один користувач може мати декілька транзакцій, категорій та фінансових цілей. Кожна транзакція належить певному користувачу та категорії, а фінансова ціль може бути пов’язана з набором транзакцій, що використовуються для відстеження прогресу її виконання.

Розроблена діаграма класів персоналізованого менеджера фінансів наведена на рисунку 3.2.

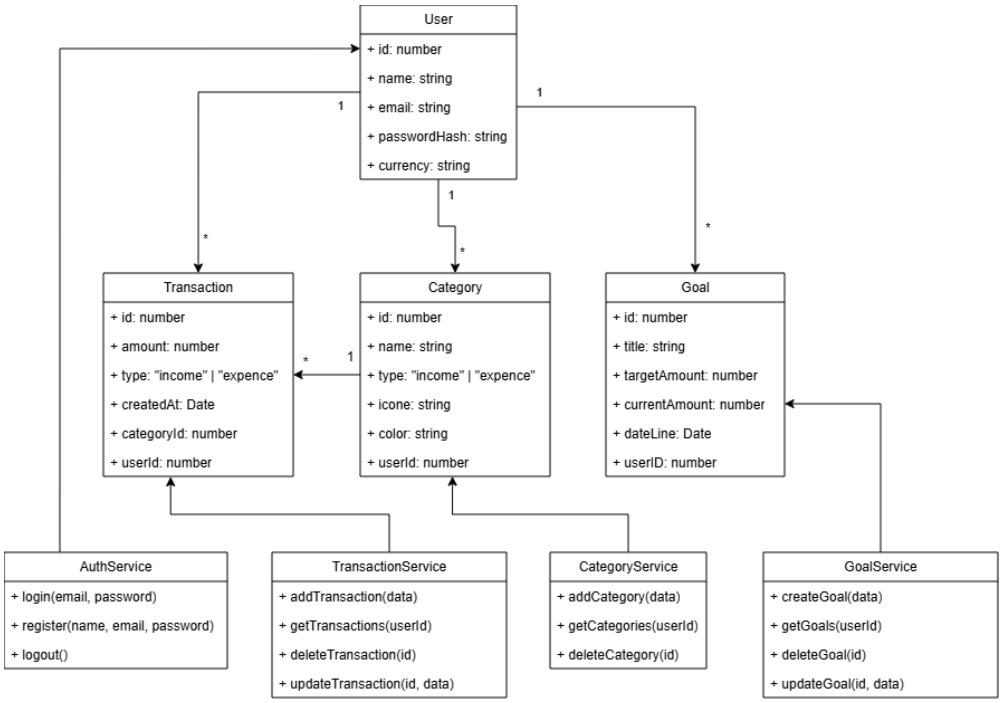

Рисунок 3.2 – Діаграма класів

Таким чином, розроблена діаграма класів відображає основні компоненти системи персоналізованого менеджера фінансів, їх атрибути, методи та взаємозв’язки між ними.

Розроблена структура системи забезпечує можливість реалізації функціоналу авторизації користувачів, управління транзакціями, категоріями та фінансовими цілями. Використання сервісних класів дозволяє розділити бізнес-логіку системи та забезпечити модульність програмного забезпечення.

Структура діаграми класів створює основу для подальшої реалізації серверної частини програмного продукту та взаємодії між компонентами системи.

3.1.3 Розробка діаграми послідовностей дій системи

Для моделювання динамічної поведінки та специфікації процесів взаємодії між компонентами кросплатформного менеджера фінансів було застосовано апарат діаграм послідовностей (Sequence Diagram). Цей тип поведінкових моделей UML дозволяє деталізувати логіку функціонування розподіленої системи шляхом хронологічного впорядкування повідомлень, якими обмінюються об'єкти та лінії життя (Lifelines) у межах виконання конкретних прецедентів [5].

Впровадження діаграм послідовностей на етапі проєктування дозволяє формалізувати протоколи обміну даними між клієнтськими та серверними шарами архітектури, визначити межі транзакцій та спрогнозувати поведінку системи під час паралельних запитів. У межах розробки системи «FinanceManager» було деталізовано та побудовано діаграми послідовностей для таких базових сценаріїв:

– автентифікація та реєстрація користувача в системі;

– створення нової фінансової транзакції з реал-тайм сповіщенням;

– динамічна категоризація фінансових операцій;

– генерація аналітичних звітів та статистичних зрізів.

Діаграма послідовності реєстрації користувача

Графічне представлення процесу створення нового облікового запису наведено на рисунку 3.3. Модель візуалізує інтеграційну взаємодію між актором, клієнтським SPA-компонентом, REST-контролером бекенду та рівнем персистентності.

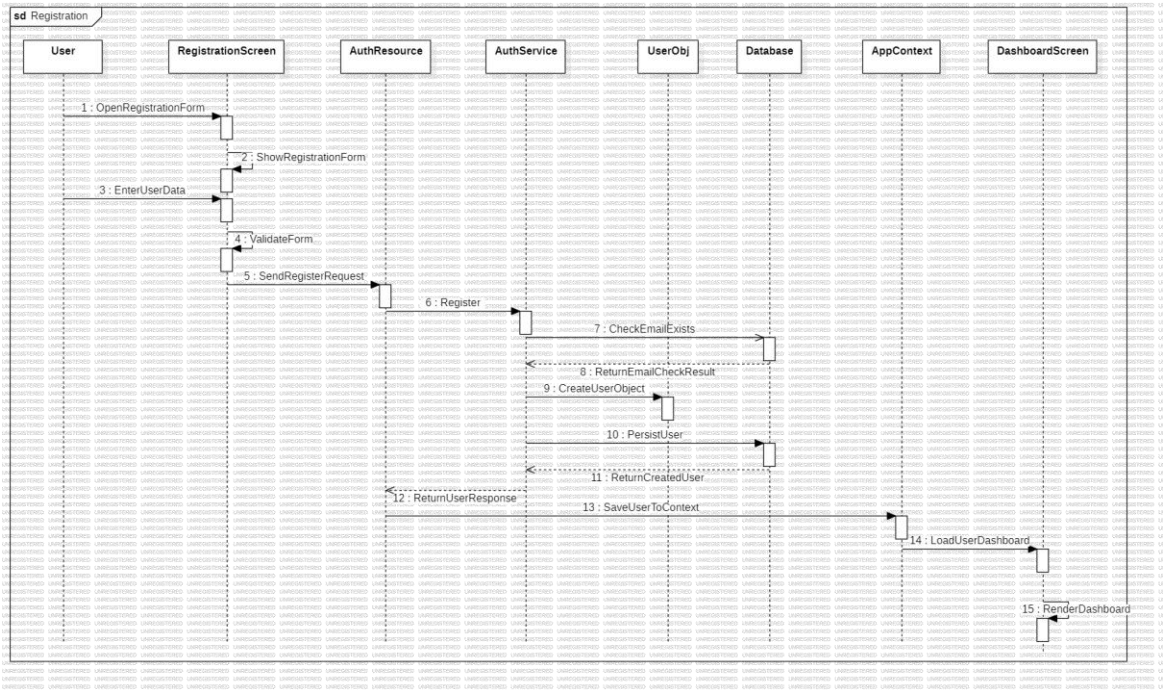

Рисунок 3.3 – Діаграма послідовності реєстрації користувача

Ініціація сценарію відбувається під час активації користувачем форми реєстрації та введення первинних метаданих (електронна пошта, пароль, ім'я, валюта). Фронтенд-компонент виконує клієнтську валідацію типів даних та

надсилає HTTP POST-запит на серверний ендпоінт. Серверний сервіс бізнес-логіки перевіряє унікальність ідентифікатора (адреси пошти) через механізми СКБД. За умови відсутності дублікатів, пароль хешується, новий об'єкт персиститься у реляційній базі даних, а сервер повертає статус 201 Created. Після отримання успішної відповіді клієнтський застосунок оновлює локальний глобальний контекст стану (State) та автоматично перенаправляє користувача на головний екран системи (Dashboard).

Діаграма послідовності авторизації користувача

Для деталізації процедури входу користувача до персонального профілю та ініціалізації сесії було побудовано діаграму послідовності автентифікації (рис. 3.4). Модель ілюструє покроковий обмін повідомленнями між презентаційним рівнем, сервісним шаром безпеки та базою даних.

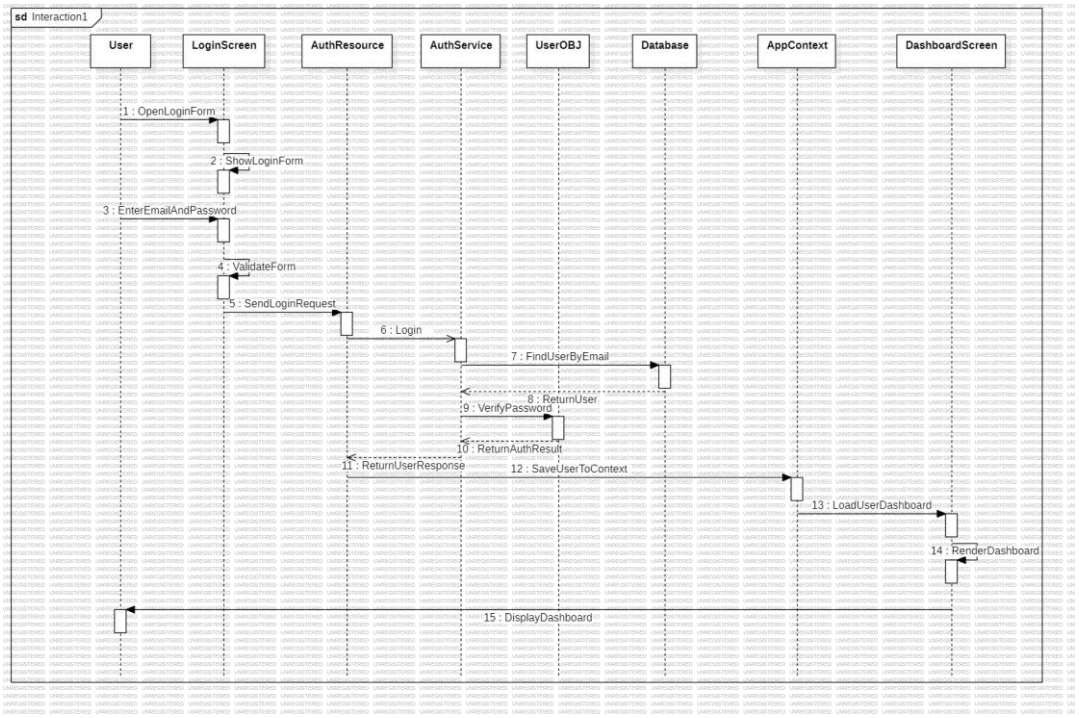

Рисунок 3.4 – Діаграма послідовності авторизації користувача

Процес стартує з виклику форми входу, яка після рендерингу приймає ідентифікаційні дані користувач. Після перевірки валідності заповнення полів інтерфейсний шар асинхронно транслює запит до REST-контролера AuthResource.

Серверний компонент ініціює виклик методу бізнес-логіки, що виконує пошуковий SQL-запит до сховища. Отриманий з бази об'єкт сутності проходить

верифікацію на відповідність хэшу пароля. Результат перевірки повертається до ресурсного контролера, який надсилає клієнту відповідь із авторизаційним токеном. Клієнтський застосунок записує токен та профайл у глобальний стейт, запускає процедуру підвантаження фінансових метрик, виконує рендеринг інтерфейсу та фінально відображає готову робочу панель.

Додавання транзакції

Сценарій створення нової операції надходження чи вибуття коштів наведено на рисунку 3.5. Особливістю цього процесу є інтеграція шару WebSocket для забезпечення концепції асинхронної реал-тайм синхронізації між розподіленими клієнтами.

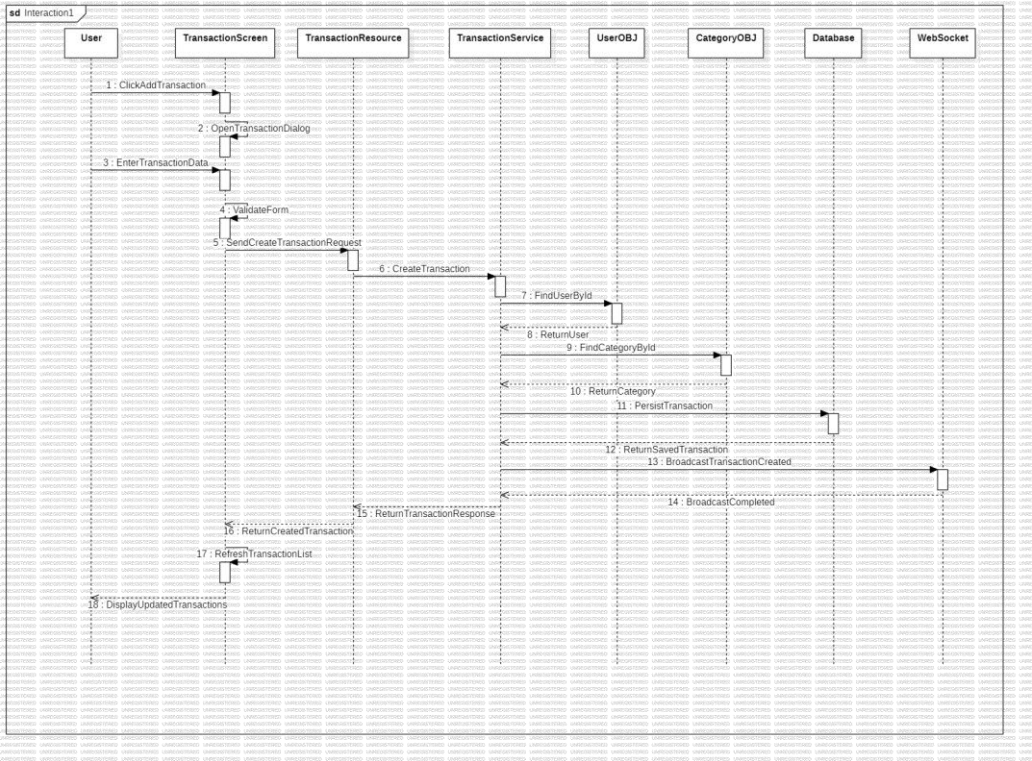

Рисунок 3.5 – Створення транзакції

Під час взаємодії з інтерфейсом користувач заповнює форму транзакції, вказуючи суму, тип операції, аналітичну категорію та опис. Клієнтський додаток валідує вхідні параметри та передає їх на сервер через REST API. Серверний шар бізнес-логіки перевіряє ліміти рахунку, прив'язує транзакцію до обраного ID категорії та користувача, після чого персистить запис у базі даних PostgreSQL.

У разі успішного завершення операції на рівні СКБД, серверний компонент WebSocket Endpoint генерує та розсилає подію оновлення (Event) усім активним клієнтським сесіям цього користувача. Фронтенд-частина перехоплює WebSocket-повідомлення, актуалізує локальний список операцій та динамічно перераховує поточний баланс без необхідності повного оновлення сторінки застосунку.

Аналітика

Сценарій отримання, агрегації та візуалізації фінансових звітів відображено на рисунку 3.6. Процес орієнтований на трансформацію сирих транзакційних даних у структуровані графічні масиви.

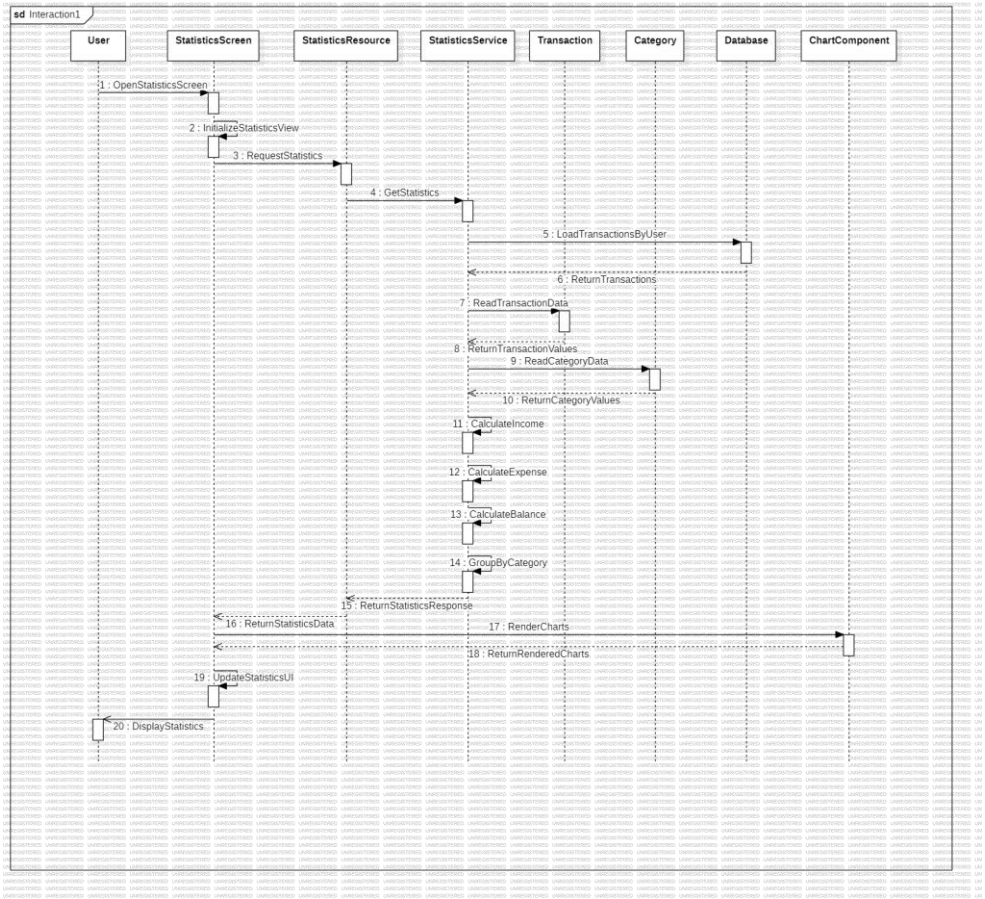

Рисунок 3.6 – Аналітика витрат користувача

Перехід користувача на вкладку аналітики тригерує автоматичне надсилання запиту до серверного модуля StatisticService. Сервісний шар виконує агрегаційні вибірки з бази даних, групуючи транзакції користувача за часовими інтервалами та категоріями видатків. На рівні сервера обчислюються інтегральні фінансові метрики: загальний обсяг доходів, сумарні витрати, чистий профіцит/дефіцит

бюджету та відсотковий розподіл капіталу. Агрегований JSON-масив повертається клієнту, де бібліотеки візуалізації фронтенду трансформують його у динамічні лінійні та кругові діаграми.

3.1.4 Розробка діаграми станів та переходів

Для моделювання динамічної поведінки системи під час додавання фінансової транзакції було використано діаграму станів та переходів (State Machine Diagram). Даний тип UML-діаграм дозволяє описати життєвий цикл об’єкта системи, зміну його внутрішніх станів залежно від дій користувача, внутрішніх подій та результатів обробки даних [5].

Використання діаграми станів та переходів дозволяє наочно відобразити логіку роботи програмного забезпечення, а також визначити основні етапи обробки інформації під час виконання певного функціонального сценарію. У персоналізованому менеджері фінансів діаграма станів була побудована для процесу додавання фінансової транзакції користувачем. Під час взаємодії із системою транзакція проходить декілька послідовних станів – від відкриття форми введення даних до збереження інформації у базі даних та оновлення даних у клієнтському застосунку.

Під час виконання сценарію користувач відкриває форму створення транзакції та вводить необхідні дані фінансової операції. Після цього система виконує перевірку коректності введеної інформації. У разі виявлення помилки користувач отримує повідомлення про некоректність введених даних та можливість повторного введення інформації.

Якщо введені дані є коректними, система виконує збереження транзакції, оновлення списку фінансових операцій та надсилання повідомлення через WebSocket-з’єднання для синхронізації даних у режимі реального часу.

Діаграма станів та переходів процесу додавання транзакції наведена на рисунку 3.7.

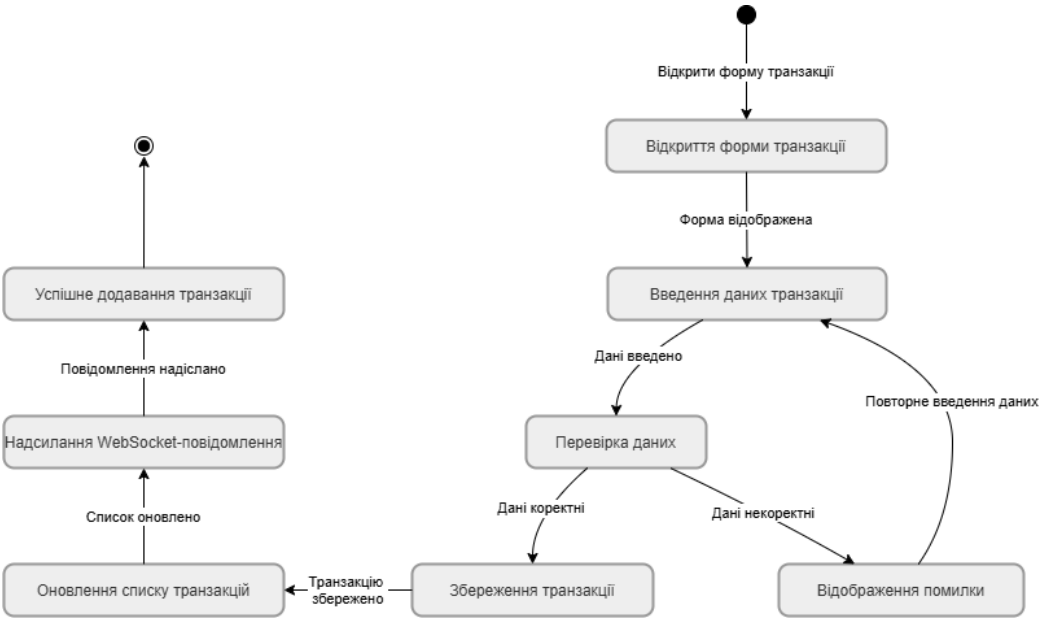

Рисунок 3.7 – Діаграма станів та переходів транзакції

Побудована діаграма станів та переходів демонструє логіку обробки фінансової транзакції та зміну станів системи під час виконання операції додавання даних. Розроблена модель також відображає механізм перевірки введеної інформації та обробку помилок під час взаємодії користувача із застосунком, що дозволяє формалізувати логіку майбутніх сервісів обробки даних на етапі кодування [5].

3.1.5 Діаграма розгортання

Для відображення фізичної архітектури програмного забезпечення було розроблено діаграму розгортання (Deployment Diagram). Діаграми даного типу використовуються для моделювання структури розміщення програмних компонентів, серверів, баз даних та каналів взаємодії між ними [5].

Діаграма розгортання дозволяє наочно продемонструвати архітектуру персоналізованого менеджера фінансів та показати спосіб взаємодії між клієнтською і серверною частинами системи. У межах проєкту було реалізовано клієнт-серверну архітектуру, що складається з web-застосунку, мобільного застосунку, серверної частини на базі Jakarta EE та бази даних PostgreSQL.

Користувач може взаємодіяти із системою через web-клієнт, реалізований за допомогою React і TypeScript, або через мобільний застосунок, створений із

використанням React Native та Expo. Обидва клієнти взаємодіють із серверною частиною через REST API для виконання основних операцій системи.

Серверна частина розгортається на сервері застосунків WildFly та містить набір сервісів, що відповідають за авторизацію користувачів, управління транзакціями, категоріями та фінансовими цілями. Для оновлення даних у режимі реального часу використовується WebSocket-компонент, який забезпечує передачу повідомлень між сервером і клієнтами

Зберігання даних здійснюється в реляційній базі даних PostgreSQL. Основними таблицями бази даних є users, categories, transactions та goals. Для спрощення розгортання та налаштування середовища база даних запускається в окремому Docker-контейнері.

Діаграма розгортання персоналізованого менеджера фінансів наведена на рисунку 3.8.

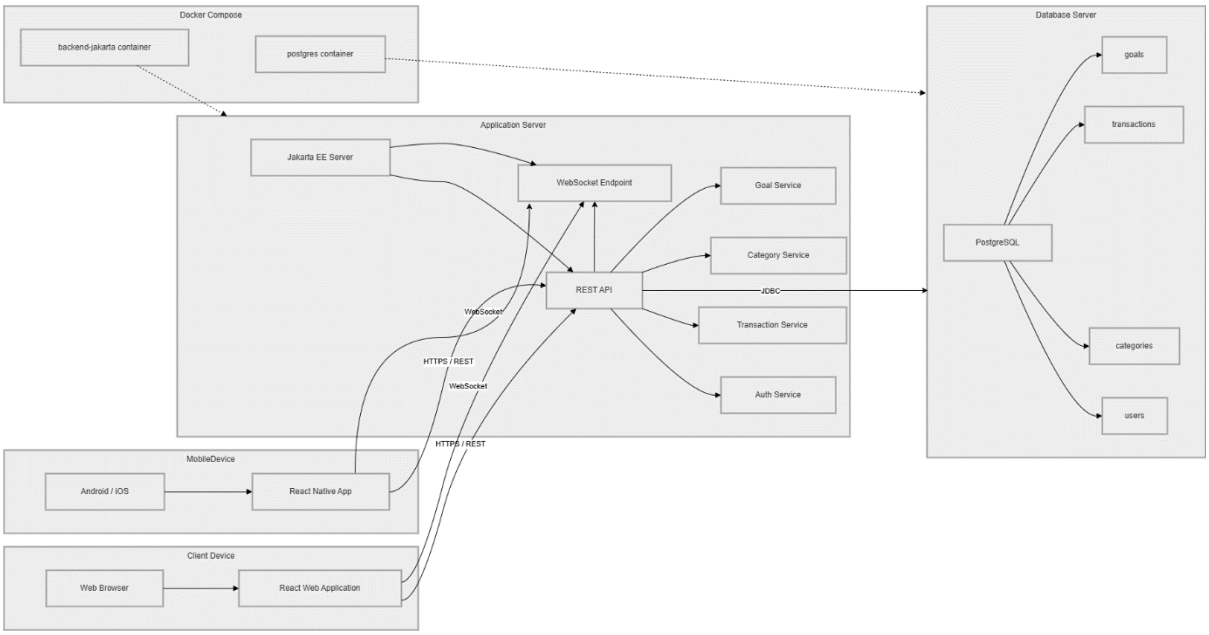

Рисунок 3.8 – Діаграма розгортання системи FinanceManager

Така архітектурна конфігурація забезпечує високу масштабованість системи, мінімізує час відгуку за рахунок використання WebSocket та гарантує повну платформонезалежність клієнтської частини, створюючи надійну основу для стабільного функціонування фінтех-додатка.

3.2 Варіанти використання системи (usecases)

Для деталізованого опису та формалізації функціональних можливостей кросплатформного додатка «FinanceManager» було розроблено текстові специфікації основних варіантів використання. Застосування цього методу дозволяє чітко регламентувати логіку взаємодії між актором (користувачем) та інформаційною системою, зафіксувати вхідні обмеження й визначити точну послідовність реакцій системи на дії оператора [5].

Під час аналізу вимог до розроблюваного менеджера фінансів було виділено та декомпоновано найбільш критичні прецеденти, що формують ядро бізнес-логіки системи. До них належать: реєстрація нового облікового запису, автентифікація, логування операцій руху коштів, генерація аналітичних звітів та встановлення лімітів лінійного бюджетування. Для кожного з наведених прецедентів було визначено склад учасників, тригери активації, передумови, основний успішний сценарій, альтернативні потоки виконання (обробка виключних ситуацій), а також кінцеві постумови.

Специфікація прецеденту «Зареєструвати акаунт» представлена у таблиці 3.1. Цей сценарій формалізує процедуру створення нового унікального профілю в системі, алгоритми первинної валідації вхідних полів на фронтенді та бекенді.

Таблиця 3.1 – Реєстрація користувача

Usecase section	**Comment**
Use Case Name	Зареєструвати акаунт
Scope	System
Level	User-goal
Primary Actor	Користувач
Stakeholders and interests	Користувач – отримати доступ до функціоналу системи Система – зберегти коректні дані користувача
Preconditions	Користувач не авторизований Є доступ до інтернету
Success guarantee	Акаунт створено Дані збережені в базі Користувач авторизований
Main Success Scenario	1. користувач відкриває застосунок; 2. обирає «Зареєструватися»; 3. вводить email, пароль, валюту; 4. підтверджує реєстрацію; 5. система створює акаунт; 6. система перенаправляє користувача на головний екран.

Кінець таблиці 3.1

Extensions	Некоректний email: – користувач вводить неправильний email; – система показує помилку. Email вже існує: – користувач вводить існуючий email; – система повідомляє про дубль. Слабкий пароль: – користувач вводить простий пароль; – система відхиляє введення.
Special Requirements	Валідація email Шифрування пароля Час відповіді ≤ 2 сек
Technology and Data Variations List	OAuth (Google, Apple) Email-підтвердження
Frequency of Occurrence	10%
Miscellaneous	Чи потрібна двофакторна автентифікація

Специфікація прецеденту «Увійти до системи» наведена у таблиці 3.2. У межах цієї специфікації задокументовано кроки проходження автентифікації користувача, верифікацію криптографічного хешу пароля, а також первинне підвантаження сесійного стану.

Таблиця 3.2 – Авторизація користувача

Usecase section	**Comment**
Use Case Name	Увійти до системи
Scope	System
Level	User-goal
Primary Actor	Користувач
Stakeholders and interests	Користувач – отримати доступ до власного акаунту Система – забезпечити безпечний доступ до даних
Preconditions	Користувач зареєстрований Акаунт активований Є доступ до інтернету
Success guarantee	Користувач успішно авторизований Сесія створена Дані користувача завантажені
Main Success Scenario	1. користувач відкриває застосунок; 2. обирає «Увійти»; 3. вводить email і пароль; 4. натискає кнопку входу; 5. система перевіряє дані; 6. система створює сесію; 7. система перенаправляє користувача на головний екран.
Extensions	Незаповнені поля: – користувач не вводить email або пароль; – система показує повідомлення про помилку. Невірний пароль: – користувач вводить неправильний пароль; – система відхиляє вхід та повідомлення про помилку. Користувача не існує: – користувач вводить неіснуючий email; – система повідомляє про відсутність акаунта. Заблокований акаунт: – користувач намагається увійти; – система блокує доступ.

Кінець таблиці 3.2

Special Requirements	Шифрування паролів Захист від brute-force атак Час відповіді ≤ 2 сек
Technology and Data Variations List	OAuth (Google, Apple) Двофакторна автентифікація (2FA)
Frequency of Occurrence	35%
Miscellaneous	Чи потрібна функція “Запам’ятати мене” Чи реалізовувати refresh token

Специфікація прецеденту «Додати витрату» висвітлена у таблиці 3.3. Сценарій регламентує процес створення операцій вибуття або надходження грошових ресурсів, перевірку зв'язків із рахунками та категоріями, механізми забезпечення транзакційності на рівні СКБД PostgreSQL, а також ініціацію реал-тайм подій через WebSocket для оновлення відкритих клієнтських інтерфейсів.

Таблиця 3.3 – Додавання витрати

Usecase section	**Comment**
Use Case Name	Додати витрату
Scope	System
Level	User-goal
Primary Actor	Користувач
Stakeholders and interests	Користувач – контроль витрат Система – коректний облік даних
Preconditions	Користувач авторизований
Success guarantee	Транзакція збережена Баланс оновлений
Main Success Scenario	1. користувач відкриває застосунок; 2. натискає «Додати транзакцію»; 3. вводить суму, категорію, опис; 4. натискає «Зберегти»; 5. система додає запис; 6. система оновлює історію.
Extensions	Порожні поля: – користувач залишає поля порожніми; – система показує помилку Некоректна сума: – користувач вводить від’ємну або нульову суму; – система блокує введення
Special Requirements	Швидкість ≤ 1 сек Офлайн режим
Technology and Data Variations List	Автопідбір категорій Голосове введення
Frequency of Occurrence	40%
Miscellaneous	Чи дозволяти редагування після створення

Специфікація прецеденту «Переглянути фінансову статистику» наведена у таблиці 3.4. Даний сценарій описує процес отримання та обробки фінансових даних користувача для формування статистичних показників. У межах виконання прецеденту система аналізує інформацію про транзакції користувача, обчислює

загальну суму доходів, витрат та поточний баланс, після чого відображає отримані результати у вигляді графіків та статистичних даних.

Таблиця 3.4 – Побудова звіту

Usecase section	Comment
Use Case Name	Переглянути фінансову статистику
Scope	System
Level	User-goal
Primary Actor	Користувач
Stakeholders and interests	Користувач – аналіз фінансів Система – генерація коректної аналітики
Preconditions	Є транзакції
Success guarantee	Побудовано графік Дані відображено коректно
Main Success Scenario	1. користувач відкриває «Статистика»; 2. обирає період; 3. система обробляє дані; 4. будує діаграму; 5. користувач переглядає результат.
Extensions	Немає даних: – система показує повідомлення Помилка завантаження: – повтор запиту
Special Requirements	Відображення ≤ 2 сек Адаптивний дизайн
Technology and Data Variations List	Chart.js / D3.js Кешування
Frequency of Occurrence	25%
Miscellaneous	Чи потрібен експорт у PDF

Таким чином, розроблені аналітичні специфікації варіантів використання дозволили виконати вичерпну декомпозицію функціональних вимог до системи «FinanceManager». Вони формалізували логіку поведінки системи на рівні атомарних кроків, що мінімізує архітектурні ризики та створює несуперечливу алгоритмічну базу для етапу безпосередньої розробки програмного коду сервісів.

Висновки до розділу 3

У третьому розділі було виконано проєктування програмного забезпечення персоналізованого менеджера фінансів «FinanceManager». На основі проведеного аналізу предметної області та сформованих вимог було розроблено комплекс UML-моделей, які відображають структуру системи, логіку взаємодії її компонентів та особливості архітектури програмного продукту.

У результаті проєктування було побудовано діаграму варіантів використання та розроблено текстові специфікації основних прецедентів, що дозволило

деталізувати функціональні можливості системи та визначити сценарії взаємодії користувача із застосунком. Для опису структури даних і взаємозв’язків між сутностями предметної області було створено діаграму класів.

Динамічні аспекти роботи системи були представлені за допомогою діаграм послідовностей, які демонструють порядок взаємодії між клієнтською частиною, сервером та базою даних під час виконання основних операцій. Крім того, було побудовано діаграму станів та переходів, що відображає життєвий цикл фінансової транзакції в системі.

Для опису фізичної архітектури програмного продукту розроблено діаграму розгортання, яка відображає взаємодію web-клієнта, мобільного застосунку, серверної частини на базі Jakarta EE, WebSocket-компонента та бази даних PostgreSQL.

Отримані результати проєктування дозволили сформувати цілісну архітектурну модель системи «FinanceManager» та створили основу для її подальшої програмної реалізації, тестування та розгортання, що розглядаються у наступному розділі дипломної роботи.

4 ПОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Після завершення етапу проєктування було виконано програмну реалізацію персоналізованого менеджера фінансів «FinanceManager». Під час розробки основна увага приділялася створенню зручного та зрозумілого інтерфейсу користувача, реалізації серверної частини для обробки даних, а також забезпеченню взаємодії між усіма компонентами системи.

Застосунок реалізовано за клієнт-серверною архітектурою. Серверна частина відповідає за авторизацію користувачів, обробку фінансових операцій, роботу з категоріями, цілями та статистикою, а також взаємодію з базою даних PostgreSQL. Для роботи з вебінтерфейсом було використано React та TypeScript, а мобільний клієнт реалізовано за допомогою React Native та Expo.

У цьому розділі наведено опис основних компонентів програмного продукту, особливостей їх реалізації та результати тестування функціональних можливостей системи. Також розглянуто основні екрани застосунку та приклади роботи реалізованого програмного забезпечення.

4.1 Реалізація серверної частини

Програмний продукт «FinanceManager» реалізовано як кросплатформну систему для ведення особистого фінансового обліку. Система складається із серверної частини, web-застосунку та мобільного клієнта, які взаємодіють між собою через REST API. Такий підхід дозволяє використовувати єдину серверну логіку для різних платформ та забезпечує однаковий функціонал незалежно від типу клієнтського застосунку.

Серверна частина розроблена на платформі Jakarta EE та відповідає за обробку запитів користувачів, виконання бізнес-логіки, взаємодію з базою даних та формування статистичних даних. Web-клієнт реалізовано за допомогою React та TypeScript, а мобільний застосунок створено з використанням React Native та Expo. Для зберігання інформації використовується реляційна база даних PostgreSQL.

Основу програмного коду становлять сутності предметної області, які відображають основні об'єкти системи та забезпечують збереження даних користувачів, фінансових операцій, категорій і цілей.

Таблиця 4.1 – Основні сутності системи

Сутність	**Призначення**	**Основні атрибути**
User	Зберігання інформації про користувача системи	id, username, email, passwordHash, currency, role, active
Transaction	Зберігання інформації про фінансові операції	id, amount, type, description, createdAt, userId, categoryId, goalId
Category	Зберігання категорій доходів та витрат	id, name, type, icon, color, userId
Goal	Зберігання інформації про цілі	id, title, targetAmount, currentAmount, deadline, userId

Між сутностями реалізовано зв'язки типу «один до багатьох» та «багато до одного». Один користувач може мати декілька категорій, транзакцій та фінансових цілей. Кожна транзакція пов'язана з конкретним користувачем і категорією, а також за потреби може бути прив'язана до фінансової цілі. Реалізація таких зв'язків здійснюється за допомогою анотацій @OneToMany та @ManyToOne.

Усі сутності реалізовано у вигляді окремих Java-класів із використанням анотацій JPA. Це дозволяє автоматично відображати об'єкти програмного коду на таблиці бази даних, реалізовувати зв'язки між сутностями та спрощує роботу із збереженням і отриманням даних.

Сутність User використовується для зберігання інформації про користувачів системи. Вона містить дані про ім'я користувача, електронну пошту, пароль, обрану валюту та роль у системі. Кожен користувач може мати власний набір транзакцій, категорій та фінансових цілей.

Сутність Transaction є основним елементом системи та призначена для зберігання інформації про фінансові операції. Для кожної транзакції фіксуються сума, тип операції, опис та дата створення. Крім цього, транзакція пов'язується із користувачем та категорією через відповідні зовнішні ключі. За необхідності

транзакція також може бути прив'язана до фінансової цілі для автоматичного відстеження прогресу її виконання.

Сутність Category використовується для групування доходів та витрат. Категорії допомагають структурувати фінансові операції користувача, спрощують аналіз витрат та використовуються під час побудови статистичних звітів. Кожна категорія належить певному користувачеві системи.

Для реалізації механізму накопичення коштів використовується сутність Goal. Вона дозволяє користувачеві створювати фінансові цілі, задавати необхідну суму накопичення та кінцевий термін виконання. Для кожної цілі зберігається поточний прогрес накопичення, який автоматично оновлюється під час додавання або видалення пов'язаних транзакцій.

Фрагмент реалізації сутності Transaction наведено на рисунку 4.1.

```java
@Entity
@Table(name = "transactions")
public class Transaction {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private Double amount;

    @Column(nullable = false)
    private String type; // income / expense

    private String description;

    @Column(name = "created_at")
    private LocalDateTime createdAt = LocalDateTime.now();

    @ManyToOne
    @JoinColumn(name = "category_id", nullable = false)
    private Category category;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne
    @JoinColumn(name = "goal_id")
    private Goal goal;

    public Transaction (){

    }

    public Transaction(Double amount, String type, String description, Category category, User user) {
        this.amount = amount;
        this.type = type;
        this.description = description;
        this.category = category;
        this.user = user;
        this.createdAt = LocalDateTime.now();
    }

```

Рисунок 4.1 – Реалізація сутності Transaction

Наведений клас демонструє використання механізмів JPA для опису структури фінансової транзакції та її зв'язків з іншими сутностями системи.

Завдяки використанню ORM-підходу взаємодія з базою даних здійснюється через об'єкти Java без необхідності ручного написання SQL-запитів для базових операцій.

Для реалізації бізнес-логіки системи було створено окремий набір сервісів. Кожен сервіс відповідає за виконання певного набору операцій та обробку даних конкретної предметної області.

```java
public class TransactionService {

    public Transaction createTransaction(
            Long userId,
            Long categoryId,
            Double amount,
            String type,
            String description,
            Long goalId
    ) {
        EntityManager em = JpaUtil.getEntityManager();

        try {
            em.getTransaction().begin();

            User user = findUserOrThrow(em, userId);
            Category category = findCategoryOrThrow(em, categoryId);

            Goal goal = null;

            if (goalId != null) {
                goal = em.find(Goal.class, goalId);

                if (goal == null) {
                    throw new RuntimeException("Goal not found");
                }
            }

            Transaction transaction = new Transaction(amount, type, description, category, user);
            transaction.setGoal(goal);

            em.persist(transaction);
            em.getTransaction().commit();

            return transaction;
        } catch (Exception e) {
            rollbackIfActive(em);
            throw new RuntimeException("Failed to create transaction", e);
        } finally {
            em.close();
        }
    }

    public List<Transaction> getTransactionsByUser(Long userId) {
        EntityManager em = JpaUtil.getEntityManager();

        try {
            TypedQuery<Transaction> query = em.createQuery(
                    "SELECT t FROM Transaction t WHERE t.user.id = :userId ORDER BY t.createdAt DESC",
                    Transaction.class
            );

            query.setParameter("userId", userId);
            return query.getResultList();
        } finally {
            em.close();
        }
    }
```

Рисунок 4.2 – Клас TransactionService

На рисунку 4.2 наведено фрагмент сервісу TransactionService, який використовується для роботи з фінансовими транзакціями. Саме цей клас

відповідає за створення нових записів, отримання списку операцій користувача, оновлення інформації та видалення транзакцій.

Таблиця 4.2 – Основні методи сервісу TransactionService

Метод	**Призначення**
createTransaction()	Створення нової фінансової транзакції
getTransactionsByUser()	Отримання списку транзакцій користувача
getTransactionById()	Отримання транзакції за ідентифікатором
updateTransaction()	Оновлення інформації про транзакцію
deleteTransaction()	Видалення транзакції та оновлення пов'язаної фінансової цілі
findUserOrThrow()	Пошук користувача з перевіркою існування
findCategoryOrThrow()	Пошук категорії з перевіркою існування
findTransactionOrThrow()	Пошук транзакції з перевіркою існування

Використання сервісного шару дозволяє відокремити бізнес-логіку від REST-ресурсів та зробити структуру програмного коду більш зрозумілою. Крім того, це спрощує подальше тестування та підтримку програмного продукту.

Взаємодія клієнтської частини із сервером здійснюється через набір REST-ресурсів, які обробляють HTTP-запити та повертають результати у форматі JSON. Для авторизації користувачів використовується AuthResource, для роботи з транзакціями – TransactionResource, для категорій – CategoryResource, а для фінансових цілей – GoalResource.

Структура серверного проєкту наведена на рисунку 4.3.

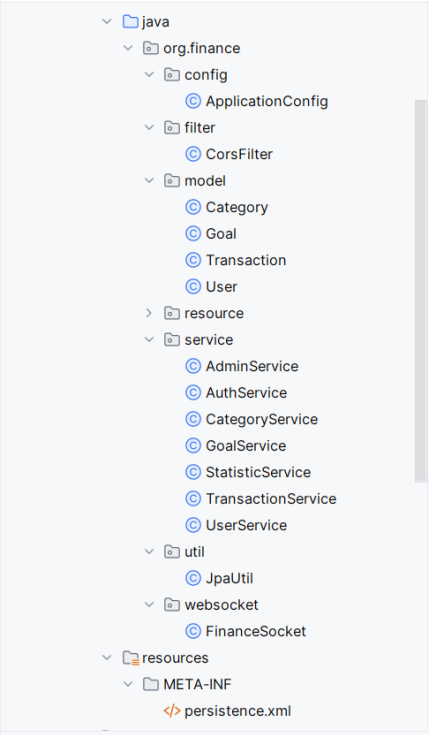

Рисунок 4.3 – Структура серверної частини проєкту

Структура проєкту побудована таким чином, щоб розділити моделі даних, сервіси бізнес-логіки та ресурси API. Це спрощує навігацію по програмному коду та дозволяє швидко знаходити необхідні компоненти під час розробки або супроводу системи.

Таким чином, структура програмного коду «FinanceManager» побудована за модульним принципом та забезпечує чіткий розподіл відповідальності між окремими компонентами системи. Використання сутностей, сервісів та REST-ресурсів дозволяє реалізувати необхідний функціонал і забезпечує можливість подальшого розвитку програмного продукту.

4.2 Реалізація вебзастосунку

4.2.1 Архітектура та програмна реалізація web-клієнта

Web-клієнт системи «FinanceManager» реалізовано як односторінковий застосунок, що забезпечує користувачу доступ до основних функцій персонального фінансового менеджера через браузер. Основним завданням клієнтської частини є відображення даних, отриманих із серверної частини, обробка дій користувача,

формування HTTP-запитів до REST API та оновлення інтерфейсу після виконання операцій.

Для реалізації web-клієнта було використано бібліотеку React у поєднанні з мовою TypeScript. React забезпечує компонентний підхід до побудови інтерфейсу, а TypeScript дозволяє явно описувати типи даних, які використовуються під час обміну між клієнтською та серверною частинами. Це особливо важливо для системи фінансового обліку, оскільки клієнт працює з транзакціями, категоріями, фінансовими цілями та даними користувача, структура яких повинна залишатися передбачуваною.

Під час розробки інтерфейсу було використано Tailwind CSS та shadcn/ui. Tailwind CSS застосовується для швидкого створення адаптивної верстки та стилізації елементів інтерфейсу, а shadcn/ui використовується як набір готових UI-компонентів, зокрема кнопок, форм, діалогових вікон, карток та елементів введення. Завдяки цьому інтерфейс має єдиний візуальний стиль і залишається зручним для подальшого розширення.

Структура web-клієнта побудована за модульним принципом. Основні сторінки системи винесено в окремі компоненти, а логіку взаємодії із сервером – в API-модулі. Такий підхід дозволяє не змішувати код інтерфейсу з кодом виконання HTTP-запитів. Наприклад, сторінка транзакцій відповідає за відображення списку операцій та взаємодію з користувачем, а окремий API-модуль виконує запити до серверного ресурсу `/transactions`.

Основними частинами web-клієнта є:

– модуль авторизації та реєстрації;
– глобальний контекст користувача;
– API-шар для взаємодії із сервером;
– сторінки транзакцій, категорій, цілей, аналітики та профілю;
– адміністративна панель;
– допоміжні утиліти для форматування валюти та обробки даних.

Для взаємодії із серверною частиною було реалізовано універсальний клієнтський модуль API. Він відповідає за формування HTTP-запитів, передавання

даних у форматі JSON, отримання відповіді сервера та обробку можливих помилок. Завдяки такому підходу всі сторінки застосунку використовують єдиний механізм обміну даними із backend-частиною.

Фрагмент реалізації модуля взаємодії з REST API наведено на рисунку 4.4.

```
const API_BASE_URL = import.meta.env.VITE_API_BASE_URL;

export async function apiRequest<T>(
  endpoint: string,
  options: RequestInit = {}
): Promise<T> {
  const response = await fetch(`${API_BASE_URL}${endpoint}`, {
    headers: {
      'Content-Type': 'application/json',
      ...(options.headers || {}),
    },
    ...options,
  });

  const data = await response.json().catch(() => null);

  if (!response.ok) {
    throw new Error(data?.message || 'API request failed');
  }

  return data as T;
}
```

Рисунок 4.4 – Реалізація модуля взаємодії з REST API

Наведений фрагмент коду демонструє централізований підхід до виконання HTTP-запитів. Замість того, щоб у кожному компоненті окремо описувати `fetch`, заголовки запиту, перевірку статусу відповіді та обробку помилок, ці дії винесено в окрему функцію. Це зменшує дублювання коду та робить клієнтську частину більш зручною для підтримки.

Окремо було реалізовано API-модулі для кожної предметної області. Наприклад, для роботи з транзакціями створено функції отримання списку транзакцій користувача, створення нової транзакції та видалення запису. Дані функції звертаються до відповідних REST-ендпоінтів серверної частини та повертають типізовані результати.

Фрагмент реалізації API-модуля для роботи з транзакціями наведено на рисунку 4.5.

```ts
import { apiRequest } from './clientApi';

export interface TransactionDto {
    id: number;
    amount: number;
    type: 'income' | 'expense';
    description: string;
    createdAt: string;
    userId: number;
    categoryId: number;
    categoryName: string;
}

export interface TransactionRequest {
    userId: number;
    categoryId: number;
    amount: number;
    type: 'income' | 'expense';
    description: string;
    goalId?: number;
}

export function getTransactions(userId: number) {
    return apiRequest<TransactionDto[]>(`/transactions?userId=${userId}`);
}

export function createTransaction(data: TransactionRequest) {
    return apiRequest<TransactionDto>('/transactions', {
        method: 'POST',
        body: JSON.stringify(data),
    });
}

export function deleteTransaction(id: number) {
    return apiRequest<{ message: string }>(`/transactions/${id}`, {
        method: 'DELETE',
    });
}
```

Рисунок 4.5 – Реалізація API-модуля транзакцій

У даному фрагменті описано типи `TransactionDto` та `TransactionRequest`, які використовуються для обміну даними між клієнтом і сервером. Це дозволяє ще на етапі розробки контролювати правильність структури об’єктів, які надсилаються до backend-частини або отримуються від неї. Наприклад, під час створення транзакції обов’язково передаються ідентифікатор користувача, ідентифікатор категорії, сума, тип операції та опис.

Для зберігання інформації про авторизованого користувача у web-клієнті реалізовано глобальний контекст застосунку. Він містить дані поточного користувача, зокрема ідентифікатор, ім’я, електронну пошту, валюту та роль. Завдяки цьому будь-яка сторінка системи може отримати доступ до інформації про користувача без повторного запиту до сервера.

Фрагмент реалізації глобального контексту користувача наведено на рисунку 4.6.

```tsx
const AppContext = createContext<AppContextType | undefined>(undefined);

export function AppProvider({ children }: { children: ReactNode }) {
  const [user, setUserState] = useState<User | null>(null);
  const [categories, setCategories] = useState<Category[]>([]);
  const [transactions, setTransactions] = useState<Transaction[]>([]);
  const [goals, setGoals] = useState<Goal[]>([]);

  const [balance, setBalance] = useState(0);
  const [totalIncome, setTotalIncome] = useState(0);
  const [totalExpense, setTotalExpense] = useState(0);

  const loadUserData = async (userId: number) => {
    const [categoriesData, transactionsData, goalsData, statisticsData] =
      await Promise.all([
        getCategories(userId),
        getTransactions(userId),
        getGoals(userId),
        getStatistics(userId),
      ]);

    setCategories(categoriesData.map(mapCategory));
    setTransactions(transactionsData.map(mapTransaction));
    setGoals(goalsData.map(mapGoal));

    setTotalIncome(statisticsData.income);
    setTotalExpense(statisticsData.expense);
    setBalance(statisticsData.balance);
  };

  const login = async (email: string, password: string) => {
    const apiUser = await loginUser({ email, password });
    const mappedUser = mapUser(apiUser);

    setUser(mappedUser);
    await loadUserData(mappedUser.id);
  };

  const logout = () => {
    setUser(null);
    disconnectSocket();
  };

  return (
    <AppContext.Provider value={{
      user, login, logout,
      categories, transactions, goals,
      balance, totalIncome, totalExpense,
      loadUserData
    }}>
      {children}
    </AppContext.Provider>
  );
}

export function useApp() {
  const context = useContext(AppContext);

  if (!context) {
    throw new Error('useApp must be used within an AppProvider');
  }

  return context;
}
```

Рисунок 4.6 – Реалізація AppContext

AppContext використовується для зберігання даних авторизованого користувача, списку категорій, транзакцій, фінансових цілей та статистичних показників. У межах контексту також реалізовано метод loadUserData, який одночасно завантажує основні дані користувача з серверної частини. Це дозволяє централізовано оновлювати стан застосунку після авторизації, створення транзакції або отримання WebSocket-повідомлення.

Маршрутизація між сторінками реалізована за допомогою React Router. Після успішної авторизації користувач перенаправляється на головну сторінку системи.

Якщо користувач не авторизований, доступ до основних сторінок обмежується. Такий підхід дозволяє відокремити публічні сторінки, зокрема Welcome та Auth, від захищених сторінок, доступних лише після входу в систему.

Одним із ключових модулів web-клієнта є сторінка транзакцій. Вона поєднує в собі роботу з формою створення фінансової операції, отримання списку транзакцій користувача, відображення категорій та оновлення даних після виконання CRUD-операцій. Саме цей модуль можна розглядати як приклад повного циклу взаємодії клієнтської частини із сервером.

Фрагмент реалізації сторінки транзакцій наведено на рисунку 4.7.

```
const {
  user,
  transactions,
  categories,
  addTransaction,
  deleteTransaction,
} = useApp();

const [formData, setFormData] = useState({
  type: 'expense',
  amount: '',
  categoryId: '',
  description: '',
  goalId: '',
});

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();

  await addTransaction({
    type: formData.type as 'income' | 'expense',
    amount: Number(formData.amount),
    categoryId: Number(formData.categoryId),
    description: formData.description,
    goalId: formData.goalId ? Number(formData.goalId) : undefined,
  });

  setFormData({
    type: 'expense',
    amount: '',
    categoryId: '',
    description: '',
    goalId: '',
  });
};
```

Рисунок 4.7 – Фрагмент реалізації сторінки транзакцій

Компонент отримує дані з глобального контексту за допомогою хука useApp, після чого використовує готові методи для додавання та видалення фінансових операцій. Під час створення транзакції введені користувачем дані перетворюються у формат запиту та передаються до методу addTransaction, який уже всередині AppContext виконує звернення до REST API та оновлює дані застосунку.

Важливою частиною реалізації є робота з категоріями. Під час створення транзакції користувач обирає категорію зі списку, який також завантажується із серверної частини. Для зручності в інтерфейсі категорія відображається не лише назвою, а й кольором та іконкою. Це покращує сприйняття фінансових операцій і робить інтерфейс більш зрозумілим.

У web-клієнті також реалізовано підтримку фінансових цілей. Користувач може створювати цілі, переглядати поточний прогрес і пов’язувати транзакції з конкретною ціллю. Такий підхід дозволяє не просто вести облік витрат і доходів, а й контролювати накопичення коштів для певної мети.

Окремим функціональним модулем є сторінка профілю користувача. Вона дозволяє змінити ім’я користувача, валюту та пароль. Після оновлення даних відповідний запит надсилається на сервер, а клієнтський стан оновлюється без необхідності повторної авторизації.

Для адміністратора реалізовано окрему адміністративну панель. Вона дозволяє переглядати список користувачів, змінювати їхній статус, блокувати або розблоковувати акаунти та переглядати статистику системи. Доступ до адміністративної панелі залежить від ролі користувача, яка зберігається у глобальному контексті після авторизації.

Таким чином, програмна реалізація web-клієнта побудована навколо кількох ключових принципів: компонентної структури, типізації даних, централізованої роботи з REST API та використання глобального стану для збереження інформації про користувача. Це дозволило створити клієнтську частину, яка може працювати з усіма основними модулями системи та залишається зручною для подальшого розширення.

4.2.2 Інтерфейс користувача web-застосунку

Після реалізації програмної логіки web-клієнта було створено користувацький інтерфейс, який забезпечує доступ до основних функцій системи. Інтерфейс побудований таким чином, щоб користувач міг швидко перейти до

потрібного розділу, переглянути фінансову інформацію та виконати необхідні дії без зайвих переходів.

Після запуску застосунку користувач потрапляє на початкову сторінку, з якої може перейти до авторизації або реєстрації. Після успішного входу система зберігає дані користувача у глобальному контексті та відкриває головну сторінку.

Головна сторінка системи наведена на рисунку 4.8.

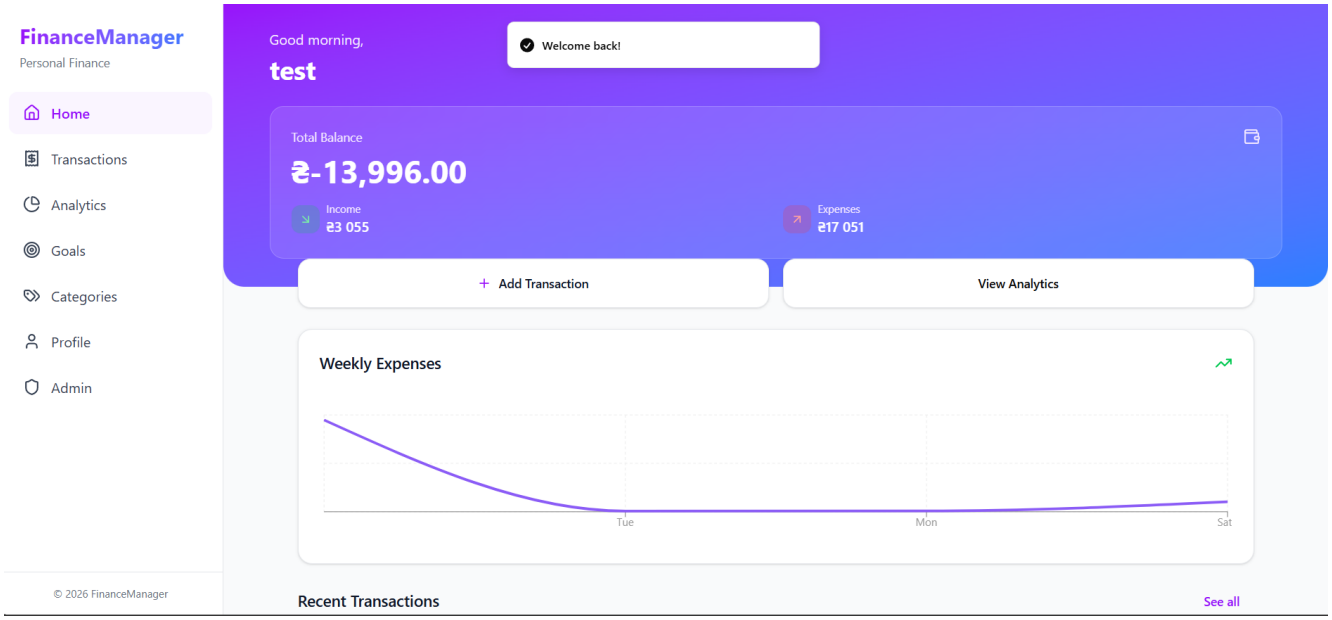

Рисунок 4.8 – Головна сторінка системи FinanceManager

Головна сторінка виконує роль інформаційної панелі. На ній відображаються основні фінансові показники користувача: поточний баланс, загальна сума доходів, сума витрат та коротка інформація про останні операції. Завдяки цьому користувач одразу після входу отримує загальне уявлення про стан власних фінансів.

Сторінка транзакцій є одним із головних розділів системи. Вона дозволяє користувачу створювати нові фінансові операції, переглядати історію транзакцій, видаляти помилкові записи та контролювати рух коштів.

Інтерфейс сторінки транзакцій наведено на рисунку 4.9.

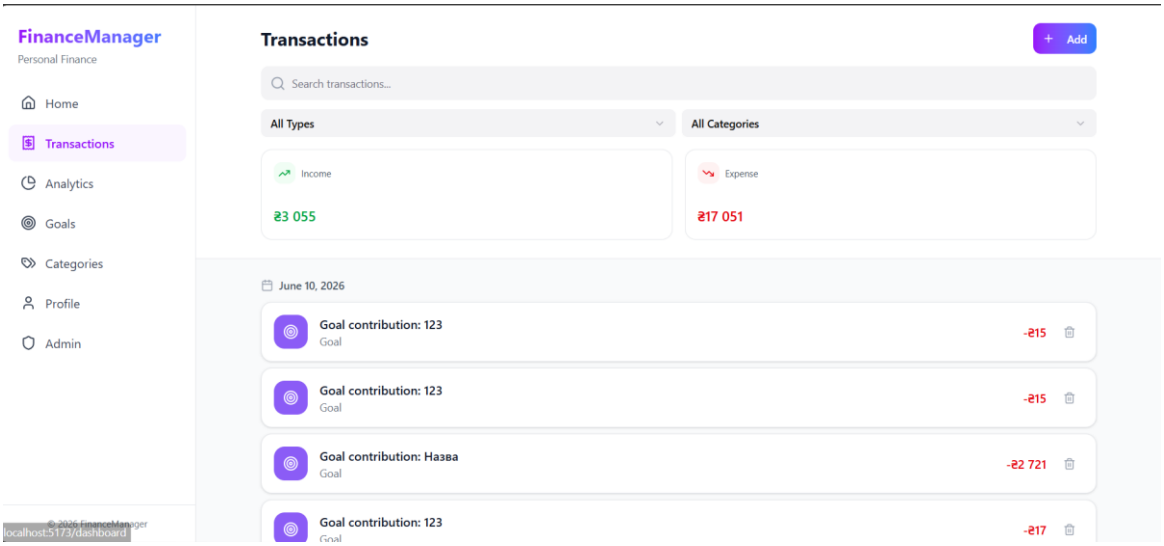

Рисунок 4.9 – Сторінка управління транзакціями

На сторінці транзакцій користувач заповнює форму, в якій вказує суму, тип операції, категорію та опис. Після збереження запис додається до історії фінансових операцій. Якщо транзакція пов’язана з фінансовою ціллю, система також враховує її під час розрахунку прогресу цілі.

Для зручного групування фінансових операцій реалізовано сторінку категорій. Вона дозволяє створювати категорії доходів і витрат, обирати для них колір та іконку. Завдяки цьому користувач може самостійно налаштовувати структуру обліку під власні потреби.

Інтерфейс сторінки категорій наведено на рисунку 4.10.

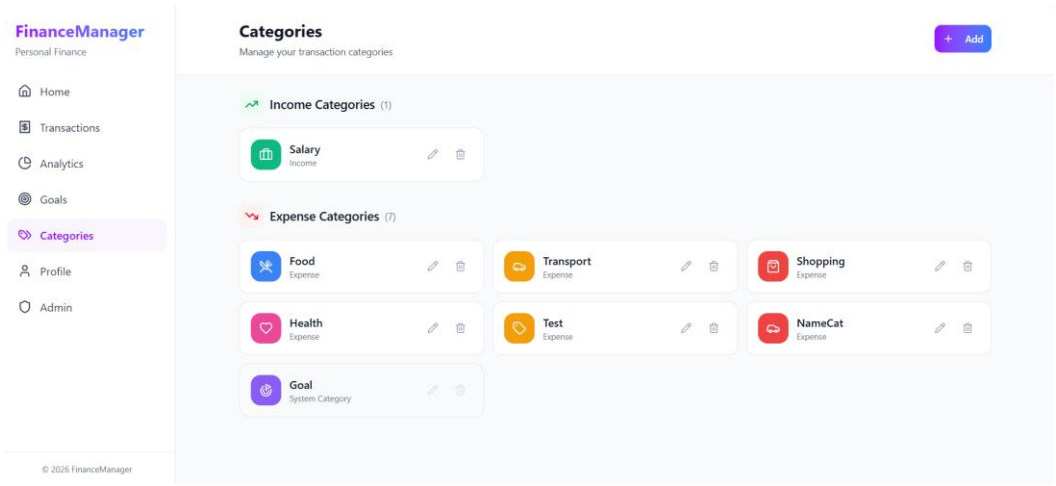

Рисунок 4.10 – Сторінка управління категоріями

Категорії використовуються не тільки для зручного відображення транзакцій, а й для подальшого аналізу фінансової активності. Наприклад, витрати на транспорт, харчування або покупки можна переглядати окремо, що допомагає краще оцінювати структуру особистого бюджету.

Сторінка фінансових цілей призначена для планування накопичень. Користувач може створити ціль, вказати необхідну суму та кінцеву дату. У процесі використання системи поточний прогрес цілі оновлюється відповідно до пов’язаних транзакцій.

Інтерфейс сторінки фінансових цілей наведено на рисунку 4.11.

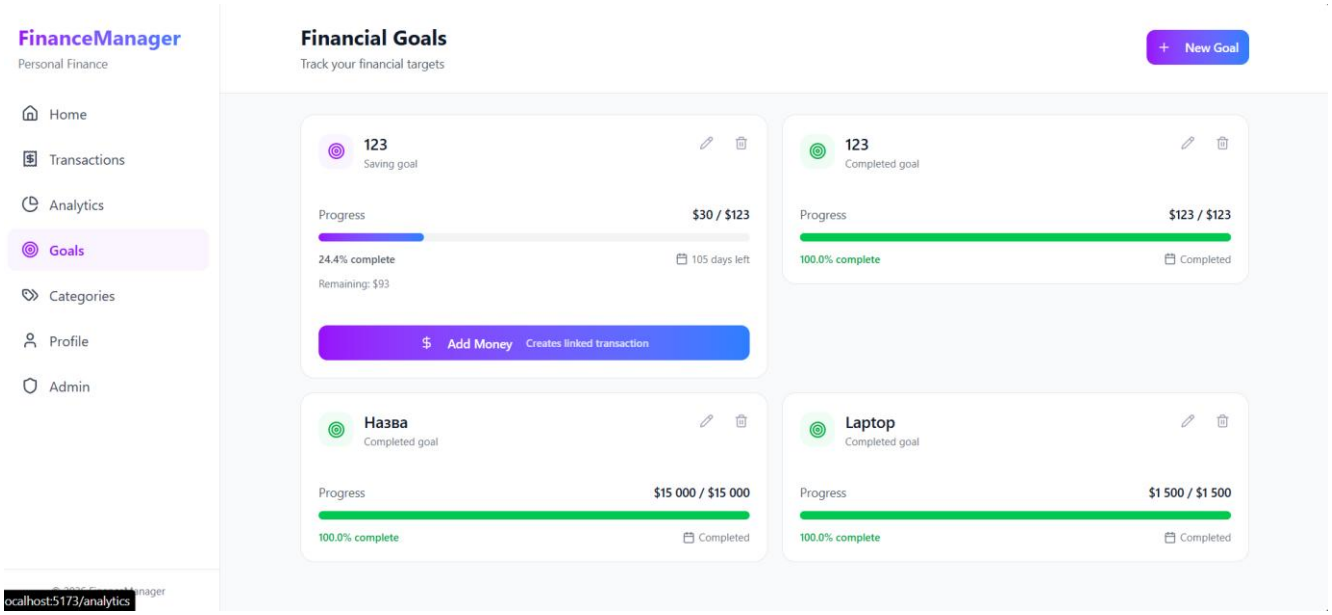

Рисунок 4.11 – Сторінка управління фінансовими цілями

Наявність фінансових цілей робить систему більш корисною для користувача, оскільки дозволяє не лише фіксувати вже виконані операції, а й планувати майбутні витрати або накопичення. Це розширює функціональність застосунку порівняно зі звичайним журналом транзакцій.

Для аналізу фінансових даних реалізовано сторінку статистики. Вона формує узагальнену інформацію про доходи, витрати та баланс користувача на основі транзакцій, збережених у базі даних.

Інтерфейс сторінки статистики наведено на рисунку 4.12.

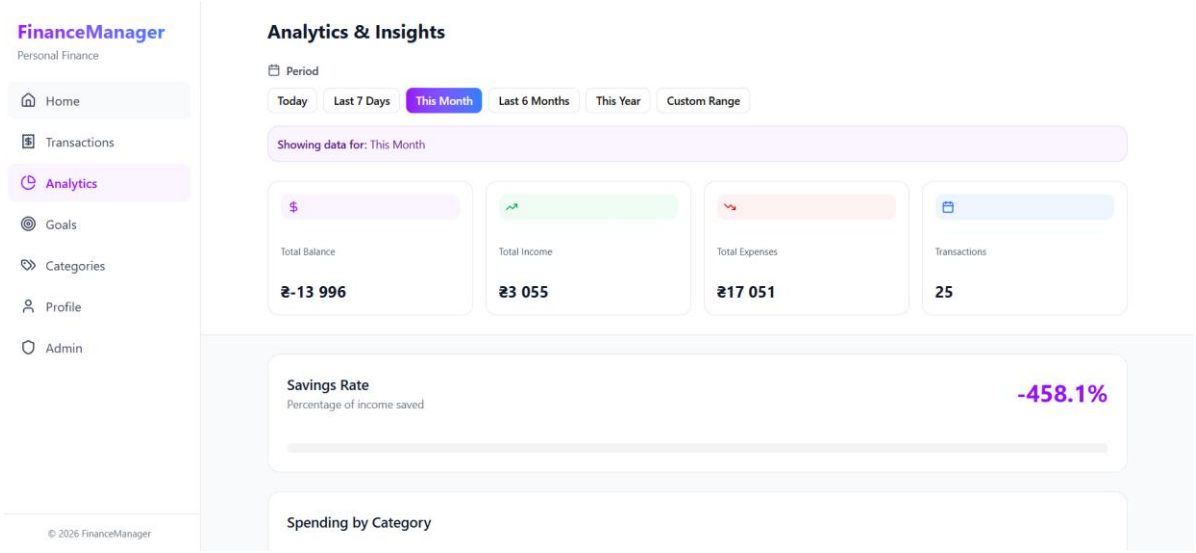

Рисунок 4.12 – Сторінка статистики

Статистика дозволяє користувачу швидко оцінити фінансовий стан та побачити загальну картину руху коштів. Виведення інформації у вигляді числових показників і графічних елементів робить аналіз більш наочним і зручним.

Сторінка профілю використовується для керування персональними налаштуваннями користувача. У цьому розділі можна змінити ім’я, валюту обліку та пароль.

Інтерфейс сторінки профілю наведено на рисунку 4.13.

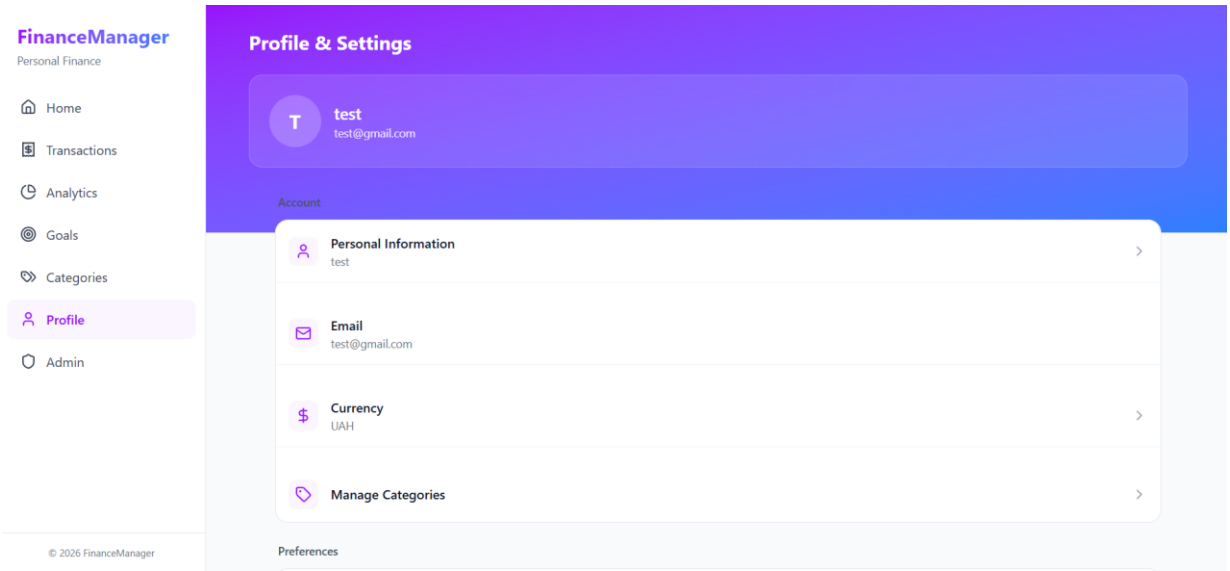

Рисунок 4.13 – Сторінка профілю користувача

Можливість зміни валюти є важливою для персоналізованого фінансового менеджера, оскільки користувачі можуть вести облік у різних грошових одиницях.

Після зміни валюти вона використовується під час відображення сум у різних частинах системи.

Окремо було реалізовано адміністративну панель. Вона призначена для користувачів із роллю адміністратора та дозволяє виконувати базове керування обліковими записами.

Інтерфейс адміністративної панелі наведено на рисунку 4.14.

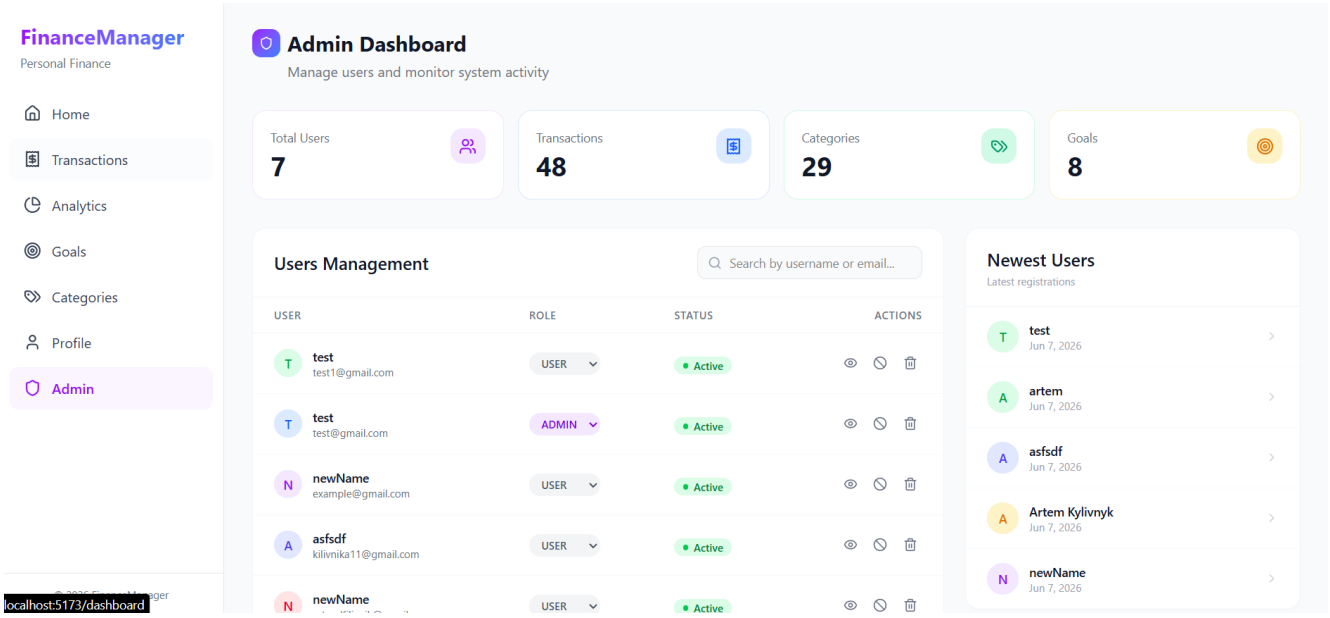

Рисунок 4.14 – Адміністративна панель системи

Адміністратор може переглядати список користувачів, змінювати їхній статус, блокувати або розблоковувати акаунти та переглядати загальні статистичні показники системи. Це дозволяє виконувати базове адміністрування без прямого доступу до бази даних.

Таким чином, web-інтерфейс системи «FinanceManager» охоплює всі основні функції персонального фінансового менеджера. Він забезпечує роботу з транзакціями, категоріями, фінансовими цілями, статистикою, профілем користувача та адміністративною панеллю. Завдяки використанню React, TypeScript, Tailwind CSS та shadcn/ui було створено зручний і адаптивний інтерфейс, який може використовуватися як основний клієнтський застосунок системи.

4.3 Реалізація мобільного застосунку

4.3.1 Програмна реалізація мобільного клієнта

Для забезпечення кросплатформної підтримки системи «FinanceManager» було реалізовано мобільний клієнт. Мобільний застосунок створено з використанням React Native, Expo та TypeScript. Обраний стек дозволяє розробляти застосунок одночасно для Android та iOS, використовуючи спільну кодову базу.

Основним завданням мобільного застосунку є надання користувачу доступу до функцій персонального фінансового менеджера зі смартфона. Мобільний клієнт взаємодіє з тією самою серверною частиною, що й web-застосунок, через REST API. Завдяки цьому бізнес-логіка системи залишається централізованою на backend-рівні, а мобільний застосунок виконує роль окремого клієнтського інтерфейсу.

Структура мобільного застосунку побудована за модульним принципом. Основні файли згруповано за призначенням: API-модулі винесено в директорію api, глобальний стан застосунку – у context, навігацію – у navigation, екрани користувацького інтерфейсу – у screens, а допоміжні функції – у utils.

Фрагмент структури мобільного проєкту наведено на рисунку 4.15.

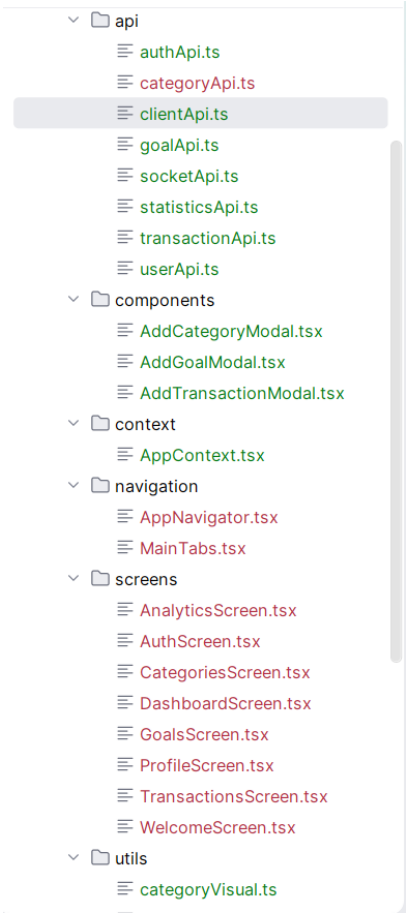

Рисунок 4.15 – Структура мобільного застосунку

Для взаємодії із серверною частиною у мобільному застосунку реалізовано окремий API-модуль. Він містить базову адресу backend-сервера та універсальну функцію виконання HTTP-запитів. Передавання даних між мобільним клієнтом і сервером здійснюється у форматі JSON.

Фрагмент реалізації клієнтського API мобільного застосунку наведено на рисунку 4.16.

```ts
export const API_BASE_URL = 'c';

export async function apiRequest<T>(
  endpoint: string,
  options: RequestInit = {}
): Promise<T> {
  const response = await fetch(`${API_BASE_URL}${endpoint}`, {
    ...options,
    headers: {
      'Content-Type': 'application/json',
      ...(options.headers || {}),
    },
  });

  const data = await response.json().catch(() => null);

  if (!response.ok) {
    throw new Error(data?.message || 'Request failed');
  }

  return data as T;
}
```

Рисунок 4.16 – Реалізація API-модуля мобільного застосунку

У наведеному фрагменті коду показано підключення мобільного застосунку до REST API серверної частини. Для роботи в локальній мережі використовується IP-адреса комп’ютера, на якому запущено backend-сервер WildFly. Це дозволяє тестувати мобільний клієнт на реальному пристрої або емуляторі.

Навігація у мобільному застосунку реалізована за допомогою React Navigation. Для побудови переходів між екранами використано стекову навігацію та нижню панель вкладок. Стекова навігація відповідає за перехід між стартовим екраном, екраном авторизації та основною частиною застосунку. Після входу користувач потрапляє до MainTabs, де доступні основні розділи системи.

Фрагмент реалізації навігації мобільного застосунку наведено на рисунку 4.17.

```tsx
export type RootStackParamList = {
  Welcome: undefined;
  Auth: { mode?: 'login' | 'register' };
  Main: undefined;
};

const Stack = createNativeStackNavigator<RootStackParamList>();

export function AppNavigator() {
  const { isAuthenticated, isLoadingSession } = useApp();

  if (isLoadingSession) {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <ActivityIndicator size="large" color="#8B5CF6" />
      </View>
    );
  }

  return (
    <Stack.Navigator screenOptions={{ headerShown: false }}>
      {isAuthenticated ? (
        <Stack.Screen name="Main" component={MainTabs} />
      ) : (
        <>
          <Stack.Screen name="Welcome" component={WelcomeScreen} />
          <Stack.Screen name="Auth" component={AuthScreen} />
        </>
      )}
    </Stack.Navigator>
  );
}
```

Рисунок 4.17 – Реалізація навігації мобільного застосунку

Основними екранами мобільного застосунку є DashboardScreen, TransactionsScreen, AnalyticsScreen, GoalsScreen, CategoriesScreen та ProfileScreen. Така структура відповідає web-версії системи, що забезпечує однакову логіку роботи на різних платформах.

Для зберігання інформації про поточного користувача та основних даних системи у мобільному застосунку використовується глобальний контекст. Він виконує подібну роль до AppContext у web-клієнті: зберігає дані авторизованого користувача, список транзакцій, категорій, фінансових цілей та статистичні показники.

Фрагмент реалізації мобільного AppContext наведено на рисунку 4.18.

```tsx
const STORAGE_USER_KEY = '@finance_manager_user';

const AppContext = createContext<AppContextType | undefined>(undefined);

export function AppProvider({ children }: { children: ReactNode }) {
  const [user, setUser] = useState<User | null>(null);
  const [isLoadingSession, setIsLoadingSession] = useState(true);

  const [categories, setCategories] = useState<Category[]>([]);
  const [transactions, setTransactions] = useState<Transaction[]>([]);
  const [goals, setGoals] = useState<Goal[]>([]);

  const [balance, setBalance] = useState(0);
  const [totalIncome, setTotalIncome] = useState(0);
  const [totalExpense, setTotalExpense] = useState(0);

  const saveUserToStorage = async (mappedUser: User) => {
    await AsyncStorage.setItem(
      STORAGE_USER_KEY,
      JSON.stringify(mappedUser)
    );
  };

  const loadUserData = async (userId: number) => {
    const [categoriesData, transactionsData, goalsData, statisticsData] =
      await Promise.all([
        getCategories(userId),
        getTransactions(userId),
        getGoals(userId),
        getStatistics(userId),
      ]);

    setCategories(categoriesData.map(mapCategory));
    setTransactions(transactionsData.map(mapTransaction));
    setGoals(goalsData.map(mapGoal));

    setBalance(statisticsData.balance);
    setTotalIncome(statisticsData.income);
    setTotalExpense(statisticsData.expense);
  };

  const login = async (email: string, password: string) => {
    const apiUser = await loginUser({ email, password });
    const mappedUser = mapUser(apiUser);

    setUser(mappedUser);
    await saveUserToStorage(mappedUser);
    await loadUserData(mappedUser.id);
  };
}
```

Рисунок 4.18 – Реалізація глобального контексту мобільного застосунку

Використання глобального контексту спрощує обмін даними між екранами мобільного застосунку. Наприклад, після авторизації дані користувача стають доступними для всіх вкладок, а після створення транзакції список фінансових операцій може бути оновлений без повторного входу в систему.

Одним із ключових екранів мобільного застосунку є екран транзакцій. Він дозволяє користувачу переглядати список фінансових операцій та додавати нові транзакції. Під час створення транзакції користувач вказує суму, тип операції, категорію та опис, після чого дані надсилаються на сервер через REST API.

Фрагмент реалізації екрана транзакцій наведено на рисунку 4.19.

```tsx
export function TransactionsScreen() {
  const { user, transactions, categories, deleteTransaction } = useApp();

  const [showAddTransaction, setShowAddTransaction] = useState(false);
  const [searchQuery, setSearchQuery] = useState('');
  const [filterType, setFilterType] =
    useState<'all' | 'income' | 'expense'>('all');
  const [filterCategory, setFilterCategory] = useState('all');

  const currencySymbol = getCurrencySymbol(user?.currency);

  const filteredTransactions = useMemo(() => {
    return transactions.filter((transaction) => {
      const query = searchQuery.toLowerCase();

      const matchesSearch =
        transaction.description.toLowerCase().includes(query) ||
        transaction.categoryName.toLowerCase().includes(query);

      const matchesType =
        filterType === 'all' || transaction.type === filterType;

      const matchesCategory =
        filterCategory === 'all' ||
        transaction.categoryName === filterCategory;

      return matchesSearch && matchesType && matchesCategory;
    });
  }, [transactions, searchQuery, filterType, filterCategory]);

  const totalFiltered = useMemo(() => {
    const income = filteredTransactions
      .filter((transaction) => transaction.type === 'income')
      .reduce((sum, transaction) => sum + transaction.amount, 0);

    const expense = filteredTransactions
      .filter((transaction) => transaction.type === 'expense')
      .reduce((sum, transaction) => sum + transaction.amount, 0);

    return { income, expense };
  }, [filteredTransactions]);
}
```

Рисунок 4.19 – Фрагмент реалізації екрана транзакцій

Реалізація мобільного екрана транзакцій демонструє загальний принцип роботи мобільного клієнта: екран отримує дані з глобального контексту, відображає їх користувачу та викликає відповідні методи для створення або оновлення інформації. Такий підхід дозволяє зберегти єдину логіку роботи з даними та спростити подальше розширення застосунку.

4.3.2 Інтерфейс користувача мобільного застосунку

Інтерфейс мобільного застосунку «FinanceManager» було розроблено з урахуванням особливостей використання системи на смартфонах. На відміну від web-клієнта, мобільна версія має обмежений розмір екрана, тому основна увага приділялася компактному розміщенню інформації, простій навігації та швидкому

доступу до основних фінансових функцій. Користувач повинен мати змогу швидко переглянути баланс, додати транзакцію, перевірити статистику або змінити налаштування профілю без зайвих переходів між екранами.

Після запуску мобільного застосунку користувач потрапляє на стартовий екран, який виконує роль початкової точки взаємодії із системою. На цьому екрані користувачу пропонується перейти до авторизації або створення нового облікового запису. Такий підхід дозволяє розділити перше знайомство із застосунком та безпосередню роботу з фінансовими даними.

Стартовий екран також виконує навігаційну функцію, оскільки саме з нього користувач обирає подальший сценарій роботи. Якщо користувач уже має обліковий запис, він переходить до входу в систему. Якщо облікового запису ще немає, користувач може перейти до реєстрації та створити новий профіль.

Стартовий екран мобільного застосунку наведено на рисунку 4.20.

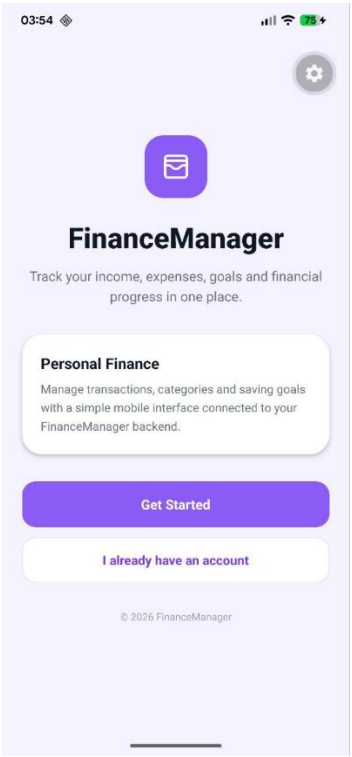

Рисунок 4.20 – Стартовий екран мобільного застосунку

Екран авторизації призначений для входу користувача до системи. На ньому користувач вводить електронну пошту та пароль, після чого мобільний застосунок надсилає запит до серверної частини. Сервер перевіряє введені дані та повертає інформацію про користувача у разі успішної авторизації.

Після успішного входу дані користувача зберігаються у глобальному контексті мобільного застосунку та локальному сховищі AsyncStorage. Це дозволяє зберігати сесію користувача між запусками застосунку. Під час наступного відкриття програми система перевіряє наявність збереженого користувача і, якщо дані коректні, одразу відкриває основну частину застосунку без повторного введення логіна та пароля.

Екран авторизації наведено на рисунку 4.21.

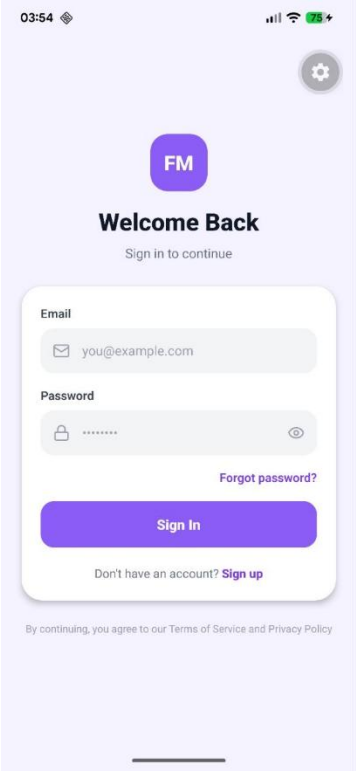

Рисунок 4.21 – Екран авторизації користувача

Після авторизації користувач потрапляє на головний екран мобільного застосунку. Він виконує роль інформаційної панелі та дозволяє швидко оцінити поточний фінансовий стан. На цьому екрані відображаються основні показники: поточний баланс, сума доходів та сума витрат. Таке розміщення інформації є зручним для щоденного використання, коли користувачу потрібно швидко перевірити загальну ситуацію з особистими фінансами.

Дані для головного екрана завантажуються із серверної частини після авторизації користувача. Для цього мобільний застосунок звертається до API статистики та отримує узагальнені показники на основі збережених транзакцій.

Завдяки цьому користувач бачить актуальну інформацію, яка відповідає даним у базі.

Головний екран мобільного застосунку наведено на рисунку 4.22.

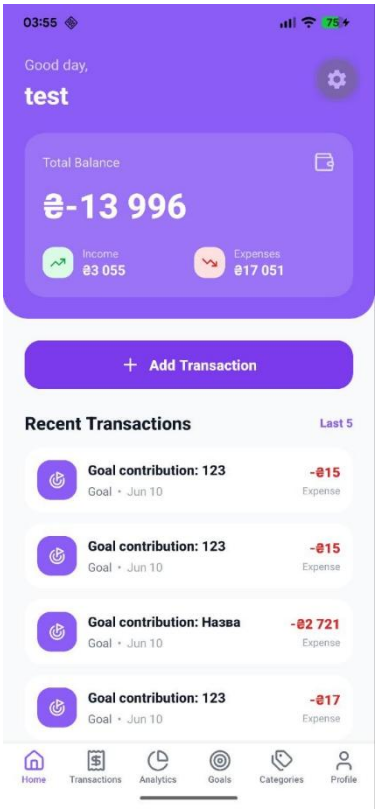

Рисунок 4.22 – Головний екран мобільного застосунку

Екран управління транзакціями є одним із ключових компонентів мобільного застосунку, оскільки саме через нього здійснюється основна взаємодія користувача із системою фінансового обліку. Після відкриття даного розділу застосунок отримує перелік фінансових операцій поточного користувача та відображає їх у вигляді структурованого списку.

Для кожної транзакції відображається її опис, категорія, сума та тип операції. Доходи та витрати мають різне візуальне оформлення, що дозволяє швидко відрізняти надходження коштів від витрат. Також для покращення зручності використання реалізовано відображення категорій із відповідними іконками та кольорами.

На сторінці транзакцій реалізовано пошук і фільтрацію фінансових операцій. Користувач може знайти потрібну транзакцію за описом або назвою категорії, переглянути тільки доходи чи тільки витрати, а також обмежити список операцій

конкретною категорією. Це особливо важливо у випадку великої кількості записів, коли ручний перегляд усієї історії є незручним.

У верхній частині екрана також відображаються підсумкові значення доходів і витрат відповідно до поточних параметрів фільтрації. Завдяки цьому користувач може не лише переглядати окремі операції, а й швидко оцінювати фінансову картину за вибраними умовами.

Для додавання нової транзакції використовується окрема кнопка, яка відкриває форму введення даних. У формі користувач вказує суму, тип операції, категорію та опис. Після збереження дані надсилаються до серверної частини через REST API, а список транзакцій оновлюється. Якщо транзакція пов’язана з фінансовою ціллю, система також враховує її під час оновлення прогресу відповідної цілі.

Екран транзакцій наведено на рисунку 4.23.

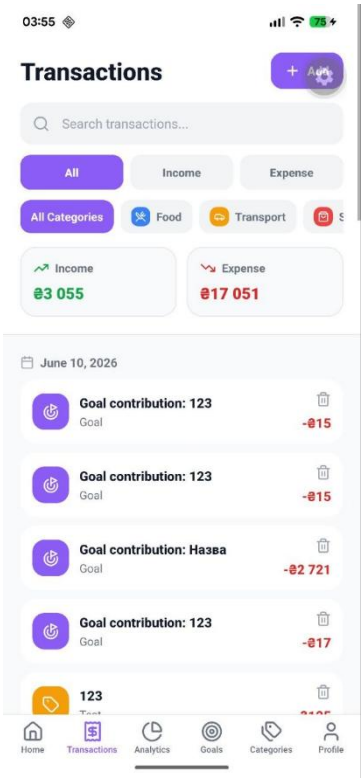

Рисунок 4.23 – Екран управління транзакціями

Для контролю довгострокових фінансових планів у мобільному застосунку реалізовано екран фінансових цілей. Даний модуль дозволяє користувачу створювати цілі накопичення коштів, вказувати бажану суму, поточний прогрес та

кінцеву дату виконання. Такий функціонал робить систему корисною не лише для обліку вже виконаних операцій, а й для планування майбутніх витрат або накопичень.

Кожна фінансова ціль містить назву, цільову суму, поточну накопичену суму та дату завершення. На екрані користувач може оцінити, наскільки він наблизився до досягнення поставленої мети. Візуальне подання прогресу дозволяє швидше сприймати інформацію та мотивує користувача продовжувати накопичення.

Після створення або оновлення фінансової цілі дані синхронізуються із серверною частиною. Це забезпечує однаковий стан інформації у web- та mobile-клієнтах, оскільки обидві версії застосунку працюють із єдиною базою даних.

Екран фінансових цілей наведено на рисунку 4.24.

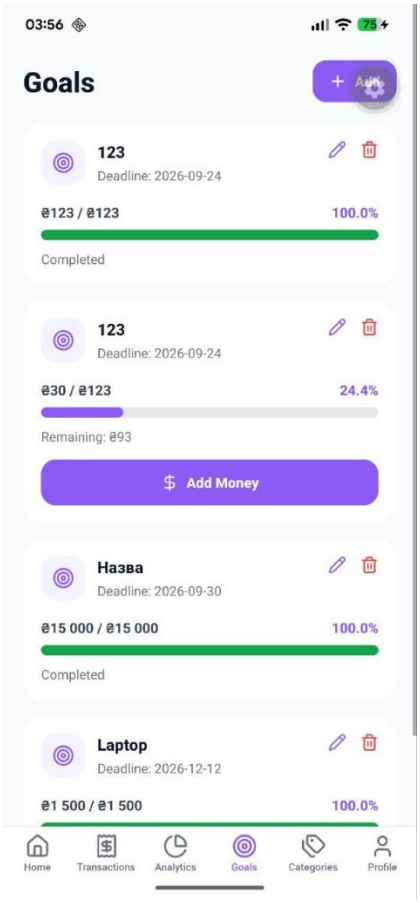

Рисунок 4.24 – Екран фінансових цілей

Екран категорій використовується для керування групами доходів і витрат. Категорії є важливим елементом системи, оскільки вони дозволяють структурувати фінансові операції користувача. Наприклад, витрати можуть бути поділені на

харчування, транспорт, покупки, здоров’я або інші напрями, а доходи – на зарплату, подарунки чи інші джерела надходжень.

У мобільному інтерфейсі категорії відображаються у вигляді окремих елементів із назвою, кольором та іконкою. Такий спосіб представлення робить список більш зрозумілим і зручним для використання на невеликому екрані смартфона. Користувач може створювати нові категорії, редагувати існуючі або видаляти ті, які більше не використовуються.

Категорії також використовуються під час створення транзакцій. Завдяки цьому кожна операція має чітку прив’язку до певного напряму витрат або доходів, що надалі використовується для формування статистики та аналізу фінансової активності.

Екран категорій наведено на рисунку 4.25.

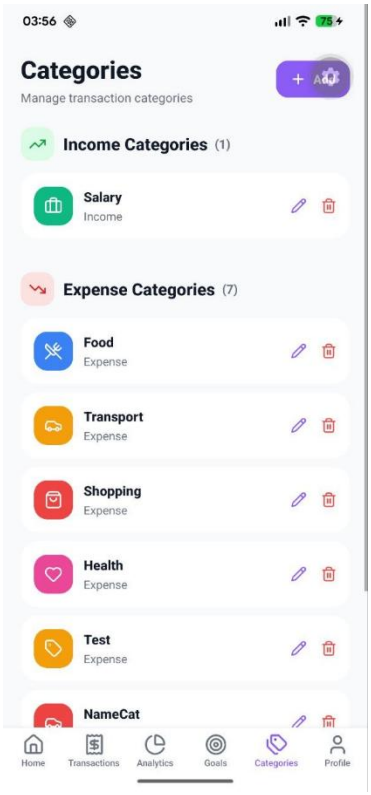

Рисунок 4.25 – Екран управління категоріями

Для аналізу фінансової активності користувача у мобільному застосунку реалізовано окремий екран аналітики. Його основним призначенням є перетворення набору фінансових операцій у зрозумілі статистичні показники, які допомагають оцінити поточний стан особистого бюджету.

Після відкриття екрана аналітики застосунок використовує дані, отримані із серверної частини. На основі транзакцій користувача система розраховує загальну суму доходів, витрат та поточний баланс. Також можуть відображатися додаткові показники, пов’язані з активністю користувача та структурою його фінансових операцій.

Важливість цього екрана полягає в тому, що користувач отримує не просто список окремих транзакцій, а узагальнену картину власних фінансів. Це дозволяє швидше помічати перевищення витрат, аналізувати основні напрями витрачання коштів та оцінювати, чи відповідають поточні фінансові дії запланованим цілям.

Екран аналітики наведено на рисунку 4.26.

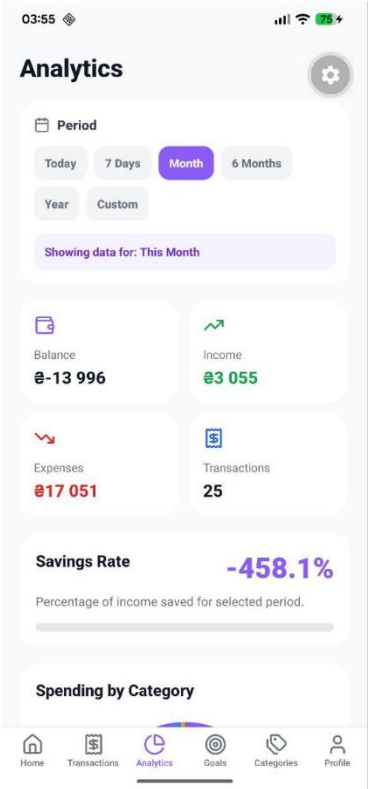

Рисунок 4.26 – Екран аналітики мобільного застосунку

Екран профілю використовується для перегляду та зміни персональних налаштувань користувача. У цьому розділі відображається основна інформація облікового запису: ім’я користувача, електронна пошта та обрана валюта обліку. Наявність такого екрана є важливою, оскільки користувач повинен мати можливість змінювати власні налаштування без звернення до адміністратора або прямої роботи з базою даних.

Через екран профілю користувач може змінити ім’я, валюту та пароль. Зміна валюти впливає на відображення фінансових сум у застосунку, тому цей параметр використовується в різних екранах системи. Після оновлення дані надсилаються на сервер, а мобільний застосунок оновлює локальний стан користувача.

Також на екрані профілю передбачено можливість виходу із системи. Після виходу дані користувача видаляються з локального сховища, очищується глобальний контекст, а застосунок повертається до стартового або авторизаційного екрана. Це забезпечує базову безпеку використання застосунку на мобільному пристрої.

Екран профілю наведено на рисунку 4.27.

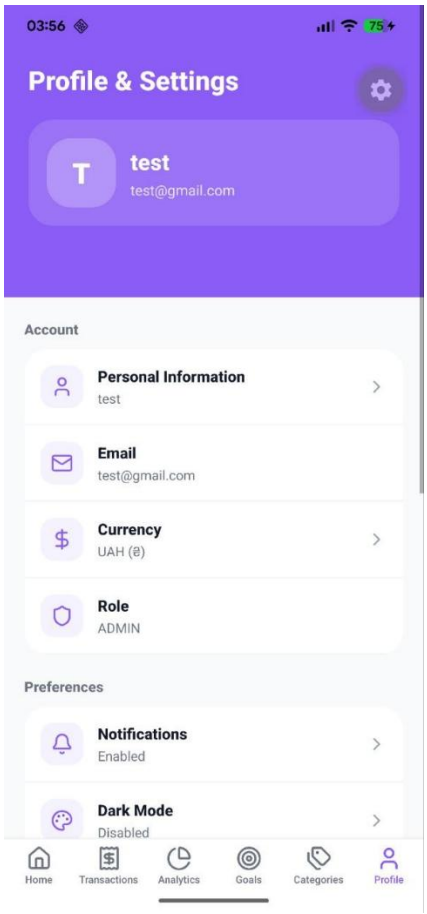

Рисунок 4.27 – Екран профілю користувача

Таким чином, інтерфейс мобільного застосунку «FinanceManager» охоплює всі основні сценарії роботи користувача: авторизацію, перегляд фінансового стану, управління транзакціями, роботу з категоріями, фінансовими цілями, аналітикою та профілем. Реалізація мобільного клієнта на React Native та Expo дозволила

створити застосунок, який працює з тією самою серверною частиною, що й web-клієнт, і забезпечує користувачу доступ до персонального фінансового менеджера з мобільного пристрою.

4.4 Тестування програмного продукту

Після реалізації основних функціональних модулів системи «FinanceManager» було проведено тестування програмного продукту. Метою тестування була перевірка коректності роботи основних сценаріїв взаємодії користувача із системою, стабільності web-інтерфейсу, правильності обміну даними між клієнтською та серверною частинами, а також перевірка роботи основних CRUD-операцій.

Тестування виконувалося для web-версії застосунку, оскільки саме вона містить повний набір основних функцій системи та безпосередньо взаємодіє із серверною частиною через REST API. Під час тестування перевірялися сценарії реєстрації користувача, створення категорій, додавання фінансових операцій, фільтрації транзакцій, створення фінансової цілі, перегляду аналітики, роботи з профілем користувача та виходу із системи.

Для проведення автоматизованого тестування було використано Selenium WebDriver. Даний інструмент дозволяє імітувати дії реального користувача у браузері: відкриття сторінок, натискання кнопок, введення даних у форми, вибір значень зі списків та перевірку наявності очікуваних елементів інтерфейсу. У межах роботи було створено тестовий клас FinanceManagerTest, який послідовно виконує набір E2E-сценаріїв.

Перед запуском тесту серверна частина застосунку була розгорнута на сервері застосунків WildFly, база даних PostgreSQL була запущена у Docker-контейнері, а web-клієнт React працював локально за адресою http://localhost:5173. Таким чином, тестування проводилося в умовах, наближених до реального використання системи.

Автоматизований тест починається з відкриття стартової сторінки застосунку та перевірки наявності назви FinanceManager. Після цього тест переходить до

форми реєстрації, створює нового користувача з унікальною електронною поштою та перевіряє, що після успішної реєстрації користувач потрапляє на головну сторінку системи. Для уникнення дублювання облікових записів email формується автоматично на основі поточного часу.

Фрагмент реалізації автоматизованого Selenium-тесту наведено на рисунку 4.28.

```
run("Створення витрати", () -> {
    clickNav("Transactions");
    waitText("Transactions");

    clickButton("Add");
    waitText("Add Transaction");

    typeById("amount", "150");

    openSelectNearLabel("Category");
    clickOption("Test Coffee");

    typeById("description", "Coffee expense test");

    clickButton("Add Expense");

    waitText("Coffee expense test");
    waitText("Test Coffee");
});
```

Рисунок 4.28 – Фрагмент реалізації автоматизованого Selenium-тесту

Після перевірки авторизаційного сценарію тест переходить до основних функціональних модулів системи. На сторінці категорій створюються дві тестові категорії: категорія витрат Test Coffee та категорія доходів Test Salary. Це дозволяє надалі перевірити коректність створення різних типів фінансових операцій.

Далі тест відкриває модуль транзакцій та створює витрату з сумою 150, категорією Test Coffee і описом Coffee expense test. Після збереження перевіряється, що створена операція відображається у списку транзакцій. Аналогічно створюється дохід із сумою 1000, категорією Test Salary і описом Salary income test. Після цього виконується перевірка фільтрації транзакцій за типом доходу.

Окремо тестується модуль фінансових цілей. Selenium переходить на сторінку Goals, відкриває форму створення нової цілі, заповнює назву, цільову

суму та дату завершення, після чого перевіряє появу створеної цілі у списку. Такий сценарій підтверджує працездатність модуля планування накопичень.

Також було перевірено сторінку аналітики. Тест переходить до розділу Analytics та перевіряє наявність основних статистичних показників: Total Balance, Total Income та Total Expenses. Додатково перевіряється зміна періоду аналітики на Today, після чого система повинна відобразити повідомлення про показ даних за поточний день.

На завершальному етапі тест відкриває сторінку профілю користувача, перевіряє наявність основних пунктів налаштувань, зокрема Personal Information, Currency та Change Password. Після цього виконується перевірка відкриття форми редагування імені користувача. Завершується тест виходом із системи та перевіркою повернення користувача на екран авторизації. Основні автоматизовані тестові сценарії наведено у таблиці 4.4.

Таблиця 4.4 – Автоматизовані тестові сценарії системи FinanceManager

№	**Тестовий сценарій**	**Дія користувача / вхідні дані**	**Очікуваний результат**	**Результат**
1	Відкриття стартової сторінки	Перехід на http://localhost:5173	Відображається стартова сторінка FinanceManager	Успішно
2	Перехід до реєстрації	Натискання кнопки Get Started	Відкривається форма Create Account	Успішно
3	Реєстрація користувача	Введення імені, email та пароля	Створюється акаунт, відкривається Dashboard	Успішно
4	Перевірка Dashboard	Перевірка елементів Income, Expenses, Recent Transactions	Головна сторінка відображає фінансові показники	Успішно
5	Створення категорії витрат	Створення категорії Test Coffee	Категорія відображається у списку витрат	Успішно
6	Створення категорії доходів	Створення категорії Test Salary	Категорія відображається у списку доходів	Успішно
7	Створення витрати	Сума 150, категорія Test Coffee, опис Coffee expense test	Витрата зберігається та відображається у списку	Успішно

Кінець таблиці 4.4

8	Перевірка створеної витрати	Перехід до списку транзакцій	Відображається Coffee expense test і Test Coffee	Успішно
9	Створення доходу	Сума 1000, категорія Test Salary, опис Salary income test	Дохід зберігається та відображається у списку	Успішно
10	Фільтрація транзакцій	Вибір типу Income	У списку відображається створений дохід	Успішно
11	Створення фінансової цілі	Назва Test Laptop, сума 30000, дата завершення	Ціль створюється та відображається у списку	Успішно
12	Перевірка сторінки аналітики	Перехід до Analytics	Відображаються Total Balance, Total Income, Total Expenses	Успішно
13	Зміна періоду аналітики	Натискання кнопки Today	Відображається інформація за поточний день	Успішно
14	Перевірка профілю	Перехід до Profile	Відображаються Personal Information, Currency, Change Password	Успішно
15	Відкриття форми редагування імені	Натискання Personal Information	Відкривається форма Edit Name	Успішно
16	Вихід із системи	Натискання кнопки Log Out	Користувач повертається на екран авторизації	Успішно

Аналіз результатів, наведених у таблиці 4.4, показує, що всі розроблені автоматизовані тестові сценарії були виконані успішно. Під час тестування було перевірено основні функціональні можливості системи, включаючи реєстрацію користувача, управління категоріями доходів і витрат, створення фінансових операцій, формування фінансових цілей, роботу модуля аналітики, налаштування профілю користувача та завершення сеансу роботи із системою.

Особливу увагу було приділено перевірці коректності взаємодії між клієнтською та серверною частинами застосунку. Результати тестування підтвердили правильність обробки даних, коректне виконання CRUD-операцій та

стабільність роботи інтерфейсу користувача. У процесі виконання тестів помилок або критичних збоїв виявлено не було.

Результат виконання автоматизованого тесту наведено на рисунку 4.29.

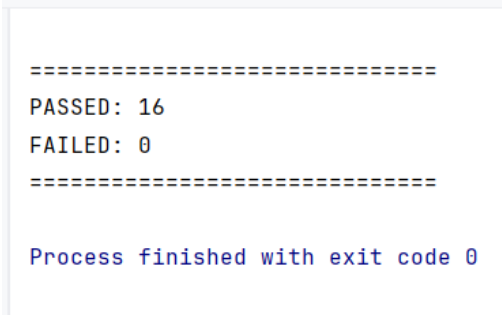

Рисунок 4.29 – Результат виконання автоматизованого тесту

Окрім автоматизованого тестування, під час розробки виконувалася ручна перевірка роботи основних модулів web- та mobile-застосунків. Ручне тестування використовувалося для перевірки зовнішнього вигляду інтерфейсу, адаптивності елементів, правильності відображення даних після оновлення сторінки, а також для перевірки сценаріїв, які потребують візуальної оцінки користувацького досвіду.

У процесі ручного тестування перевірялося створення та видалення транзакцій, зміна валюти користувача, оновлення імені профілю, робота сторінок категорій, цілей та аналітики. Також перевірялася синхронізація даних між клієнтською частиною та сервером після виконання CRUD-операцій.

За результатами тестування встановлено, що основні функціональні можливості програмного продукту працюють коректно. Система успішно виконує реєстрацію користувача, створення категорій, додавання доходів і витрат, створення фінансових цілей, побудову аналітики та вихід із системи. Результати автоматизованого та ручного тестування підтверджують працездатність програмного продукту «FinanceManager» і його готовність до подальшого використання та розвитку.

Висновки до розділу 4

У четвертому розділі було розглянуто процес реалізації та тестування програмного продукту «FinanceManager». Було описано структуру програмного

забезпечення, особливості реалізації основних функціональних модулів, а також механізми взаємодії між клієнтською та серверною частинами системи. Реалізований програмний продукт забезпечує можливість ведення особистого бюджету, управління категоріями доходів і витрат, створення фінансових цілей, перегляду статистики та аналітичної інформації щодо фінансової діяльності користувача.

У процесі розробки було використано сучасний стек технологій, що включає Jakarta EE для серверної частини, PostgreSQL для зберігання даних, React для web-клієнта та React Native для мобільного застосунку. Такий підхід дозволив створити кросплатформне рішення з єдиною бізнес-логікою та централізованим зберіганням інформації.

Для перевірки працездатності програмного продукту було проведено комплексне тестування. Автоматизоване тестування за допомогою Selenium WebDriver дозволило перевірити коректність виконання основних користувацьких сценаріїв, включаючи реєстрацію користувача, управління категоріями, створення транзакцій, роботу з фінансовими цілями, перегляд аналітики та налаштування профілю. Додатково виконувалося ручне тестування інтерфейсу та перевірка роботи окремих функціональних модулів.

Результати тестування підтвердили коректність реалізованих функцій, стабільність роботи системи та відповідність програмного продукту поставленим функціональним і нефункціональним вимогам. Усі основні сценарії роботи користувача були успішно виконані без критичних помилок, що свідчить про готовність системи до практичного використання та подальшого розвитку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено програмний продукт «FinanceManager» – кросплатформну систему управління особистими фінансами, призначену для обліку доходів і витрат, планування фінансових цілей та аналізу фінансової активності користувача.

Під час виконання роботи було досліджено сучасні підходи до організації систем персонального фінансового менеджменту та проведено аналіз існуючих програмних рішень. Це дозволило визначити ключові функціональні можливості, які є найбільш важливими для користувачів подібних систем, а також сформувати вимоги до майбутнього програмного продукту.

На основі проведеного аналізу було спроєктовано архітектуру системи та виконано моделювання основних бізнес-процесів із використанням UML-діаграм. Результати проєктування стали основою для подальшої реалізації серверної та клієнтської частин застосунку.

У межах роботи реалізовано серверну частину системи на базі Jakarta EE та PostgreSQL, що забезпечує зберігання, обробку та надання фінансових даних через REST API. Для взаємодії з користувачем було створено вебзастосунок на основі React і TypeScript, а також мобільний застосунок із використанням React Native. Для забезпечення актуальності даних у реальному часі впроваджено механізм обміну повідомленнями на базі WebSocket.

Розроблений програмний продукт підтримує повний цикл роботи з персональними фінансами: створення та класифікацію фінансових операцій, управління категоріями доходів і витрат, формування фінансових цілей, перегляд статистичної інформації та налаштування профілю користувача. Завдяки модульній архітектурі система може бути розширена новими функціональними можливостями без суттєвих змін існуючої структури.

Для перевірки працездатності програмного продукту було проведено комплексне тестування. Автоматизовані сценарії, реалізовані за допомогою Selenium WebDriver, підтвердили коректність роботи основних користувацьких

сценаріїв, а результати ручного тестування засвідчили стабільність роботи інтерфейсу та правильність взаємодії між клієнтською і серверною частинами системи.

Практична цінність роботи полягає у створенні готового програмного рішення, яке може використовуватися як основа для персонального фінансового менеджера або подальшого розвитку в комерційний програмний продукт. Реалізована архітектура дозволяє масштабувати систему, додавати нові модулі та інтегрувати її із зовнішніми сервісами.

Перспективними напрямами розвитку системи є інтеграція з банківськими API для автоматичного отримання фінансових операцій, впровадження механізмів прогнозування витрат, використання інструментів штучного інтелекту для формування персоналізованих рекомендацій та розширення функціональності мобільного застосунку.

Отже, поставлену мету роботи досягнуто повністю. У результаті створено працездатний кросплатформний програмний продукт, який об'єднує сучасні вебтехнології, серверні рішення та засоби мобільної розробки для ефективного управління особистими фінансами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 1Money - Expense Tracker, Money Manager, Budget. Google Play. URL: HTTP://play.google.com/store/apps/details?id=org.pixelrush.moneyiq&hl=uk&pli=1 (Accessed: 13.02.2026).

2. 2025 Stack Overflow Developer Survey. Stack Overflow. URL: HTTP://survey.stackoverflow.co/ (Accessed: 13.02.2026)

3. Docker Documentation. URL: HTTP://docs.docker.com/ (Accessed: 25.05.2026).

4. Digital Payments - Worldwide : Statista Market Forecast / Statista Research. 2026. URL: HTTP://www.statista.com/outlook/dmo/fintech/digital-payments/worldwide (Accessed: 21.05.2026).

5. Fowler M., Scott K. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 2nd ed. Reading, MA : Addison-Wesley, 1999. 224 p.

6. Git Reference Manual. URL: HTTP://git-scm.com/docs (Accessed: 25.05.2026).

7. ISO/IEC 9075-1:2023. Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). URL: HTTP://www.iso.org/standard/76583.html (Accessed: 25.05.2026).

8. Jakarta EE Specifications / Eclipse Foundation. URL: HTTP://jakarta.ee/specifications/ (Accessed: 25.05.2026)

9. Jakarta RESTful Web Services Specification. URL: HTTP://jakarta.ee/specifications/restful-ws/ (Accessed: 25.05.2026).

10. Jakarta Persistence Specification. URL: HTTP://jakarta.ee/specifications/persistence/ (Accessed: 25.05.2026)

11. PocketGuard. Official website. URL: HTTP://pocketguard.com/ (Accessed: 13.02.2026).

12. PostgreSQL 16 Documentation / The PostgreSQL Global Development Group. URL: HTTP://www.postgresql.org/docs/16/ (Accessed: 25.05.2026).

13. Pressman R. S. Software Engineering: A Practitioner's Approach. 7th ed. New York : McGraw-Hill Higher Education, 2010. 976 p.

14. React Documentation. URL: HTTP://react.dev/reference/react (Accessed: 25.05.2026).

15. Shadcn/ui Documentation. URL: HTTP://ui.shadcn.com/docs (Accessed: 25.05.2026)

16. Software Developer Statistics 2024 - Report on the State of the Developer Ecosystem. JetBrains. URL: HTTP://www.jetbrains.com/lp/devecosystem-2024/ (Accessed: 13.02.2026)

17. The TypeScript Handbook. URL: HTTP://www.typescriptlang.org/docs/handbook/ (Accessed: 25.05.2026).

18. Tailwind CSS Documentation. URL: HTTP://tailwindcss.com/docs (Accessed: 25.05.2026).

19. The WebSocket Protocol / IETF RFC 6455. URL: HTTP://datatracker.ietf.org/doc/html/rfc6455 (Accessed: 25.05.2026).

20. What is Personal Finance and Why Does It Matter? / B2Broker Financial News. 2024. URL: HTTP://b2broker.com/news/what-is-personal-finance-and-why-does-it-matter/ (Accessed: 21.05.2026).

21. YNAB. Official website. URL: HTTP://www.ynab.com/ (Accessed: 13.02.2026).