Державний торговельно-економічний університет

Кафедра комп’ютерних наук та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка інформаційної системи для управління особистими фінансами»

Студента 2 курсу, 10м групи, спеціальності F6 «Інформаційні системи та технології» _____ *підпис студента* Броварець Богдан Русланович

Науковий керівник кандидат фізико-економічних наук, доцент _____ *підпис керівника* Філімонова Тетяна Олегівна

Гарант освітньої програми кандидат технічних наук, доцент _____ *підпис керівника* Тамашевська Тетяна Володимирівна

Київ 2025

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність F6 «Інформаційні системи та технології»
Освітня програма «Інформаційні системи і технології»

Затверджую
Зав. кафедри _____Пурський О.І.
«30» грудня 2024р.

Завдання
на кваліфікаційну роботу студенту

Броварцю Богдану Руслановичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
«Розробка інформаційної системи для управління особистими фінансами»
Затверджена наказом ректора від *«27» грудня 2024 р. № 4254*

2. Строк здачі студентом закінченої роботи 9 *грудня 2025 року*

3. Цільова установка та вихідні дані до роботи
Мета роботи: дослідження, аналіз та розробка інформаційної системи, яка забезпечить комплексну аналітичну підтримку та ефективне управління особистими фінансами користувачів.
Об'єкт дослідження: процеси розробки інформаційної системи для управління особистими фінансами.
Предмет дослідження: моделі, методи, архітектурні рішення та інформаційні технології, що використовуються для побудови інформаційної системи ведення фінансового обліку та аналізу особистого бюджету.
4. Перелік графічного матеріалу

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Філімонова Т.О.	30.12.2024 р.	30.12.2024 р.
2	Філімонова Т.О.	30.12.2024 р.	30.12.2024 р.
3	Філімонова Т.О.	30.12.2024 р.	30.12.2024 р.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДИЧНІ ОСНОВИ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ ТА ІНФОРМАЦІЙНІ СИСТЕМИ ЇХ ЗАБЕЗПЕЧЕННЯ

1.1. Сутність та значення особистих фінансів і бюджетування в сучасних умовах

1.2. Концептуальні засади побудови та класифікація інформаційних систем у сфері фінансового менеджменту

1.3. Аналіз існуючих програмних рішень для управління особистими фінансами та обґрунтування напрямів вдосконалення

РОЗДІЛ 2. АНАЛІЗ ТА ПРОЕКТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Обґрунтування архітектурного рішення та стилів програмного забезпечення

2.2. Формування технічних вимог та вибір технологічного стеку

2.3. Проектування структури даних (ER-модель) та структурних схем ІС

РОЗДІЛ 3. РЕАЛІЗАЦІЯ, НАЛАШТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Основні етапи реалізації, налаштування сервера та розробка ядра функціоналу

3.2. Розробка інтерфейсу користувача та реалізація аналітичного модуля

3.3. Тестування системи, оцінка її продуктивності та економічної ефективності

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ Пор.	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми кваліфікаційної роботи*	*02.11.2024*	*02.11.2024*
2	*Розробка та затвердження завдання на кваліфікаційну роботу*	*30.12.2024*	*30.12.2024*
3	*Вступ*	*01.05.2025*	01.05.2025
4	*РОЗДІЛ 1. Теоретичні аспекти оцінки конкурентоспроможності підприємства*	*17.06.2025*	17.06.2025

5	*Підготовка статті у збірник наукових статей магістрів*	*24.06.2025*	24.06.2025
6	*РОЗДІЛ 2. Математичні моделі оцінки та управління конкурентоспроможністю підприємства*	*05.09.2025*	05.09.2025
7	*РОЗДІЛ 3. Інформаційна технологія оцінки конкурентоспроможності підприємств електронної торгівлі*	*17.10.2025*	17.10.2025
8	*Висновки*	*21.10.2025*	21.10.2025
9	*Здача кваліфікаційної роботи на кафедру науковому керівнику*	*25.11.2025*	25.11.2025
10	*Попередній захист кваліфікаційної роботи*	*28.11.2025*	28.11.2025
11	*Виправлення зауважень, зовнішнє рецензування кваліфікаційної роботи*	*02.12.2025*	02.12.2025
12	*Представлення готової зшитої кваліфікаційної роботи на кафедру*	*09.12.2025*	09.12.2025
13	*Публічний захист кваліфікаційної роботи*	За розкладом роботи ЕК	

Дата видачі завдання «22» грудня 2024 р.

9. Керівник кваліфікаційної роботи Філімонова Т.О.
(прізвище, ініціали, підпис)

10. Гарант освітньої програми Томашевська Т.В.
(прізвище, ініціали, підпис)

11. Завдання прийняв до виконання студент Броварець Б.Р.
(прізвище, ініціали, підпис)

12. Відгук керівника кваліфікаційної роботи

У кваліфікаційній роботі розроблено інформаційфну систему для управління особистими фінансами. ІС відповідає вимогам масштабованості та зручності користувача. Проведено тестування системи та оцінка її економічної ефективності. Поставлені завдання виконані, робота може бути допущена до захисту.

Керівник кваліфікаційної роботи
Філімонова Т.О. 30.11.2025 р.
(підпис, дата)

13. Висновок про кваліфікаційну роботу

Кваліфікаційна робота студента Броварець Б.Р.
(прізвище, ініціали)
може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми Козлов В.В.
(підпис, прізвище, ініціали)

Завідувач кафедри Пурський О.І.
(підпис, прізвище, ініціали)

«_____»_____2025 р.

Анотація

У кваліфікаційній роботі проведено комплексне дослідження та системний аналіз архітектурних рішень для розробки інформаційної системи (ІС) управління особистими фінансами. На основі теоретичного аналізу методів фінансового планування та порівняльного огляду існуючих програмних продуктів обґрунтовано вибір технологічного стеку та спроектовано структуру ІС. Реалізований функціонал забезпечує не лише ведення обліку доходів та витрат, а й аналітичну підтримку прийняття фінансових рішень та стратегічне планування бюджету. Розроблена ІС відповідає вимогам масштабованості та зручності користувача. Проведено тестування системи та оцінка її економічної ефективності.

Ключові слова: інформаційна система, управління фінансами, особистий бюджет, системний аналіз, фінансове прогнозування.

Anotation

In this qualification work, a comprehensive study and system analysis of architectural solutions were conducted for the development of an Information System (IS) for personal financial management. Based on a theoretical analysis of financial planning methods and a comparative review of existing software products, the technology stack was substantiated and the IS structure was designed. The implemented functionality provides not only accounting for income and expenses but also analytical support for financial decision-making and strategic budget planning. The developed IS meets the requirements for scalability and user convenience. System testing and an evaluation of its economic effectiveness were conducted. Keywords: information system, financial management, personal budget, system analysis, financial forecasting.

ЗМІСТ

ВСТУП

У сучасному світі, що характеризується високою динамікою економічних процесів, ефективне управління особистими фінансами та систематичний контроль бюджету відіграють ключову роль для досягнення фінансової стабільності та успіху. Майже кожного дня люди витрачають кошти на покупки в магазинах на різноманітні товари та послуги, починаючи від продуктів харчування і закінчуючи побутовою технікою, розвагами та ін. Тому, знання про те, як ефективно управляти власними фінансами, є ключовим аспектом фінансової дисципліни та раціонального керування грошима.

Процес комплексного управління особистим бюджетом є складним і вимагає постійного моніторингу, планування та прогнозування. Існуюче програмне забезпечення для фінансового обліку часто має обмежений функціонал у частині глибокого фінансового аналізу та управлінської підтримки. Це зумовлює актуальну потребу у створенні повноцінної інформаційної системи (ІС), що надає користувачам розширені інструменти для аналізу, візуалізації та прогнозування фінансових потоків.

У рамках цієї кваліфікаційної роботи буде розроблена інформаційна система для управління особистими фінансами, спрямована на автоматизацію аналітичних процесів та підвищення фінансової дисципліни користувачів. Система мінімізує рутинні операції, надаючи готові інструменти для аналізу фінансових звичок та стратегічного планування. Для досягнення цієї мети передбачено реалізацію широкого спектру завдань, таких як системний огляд

існуючих рішень, аналіз літератури, розробка архітектури інформаційної системи та її розгортання.

Таким чином, цей проєкт заслуговує на увагу в контексті зростаючого попиту на зручні та ефективні інформаційні системи для управління особистим бюджетом в умовах зростаючої складності фінансових операцій.

Мета і завдання дослідження. Метою даного дослідження є розробка інформаційної системи, яка забезпечить комплексну аналітичну підтримку та ефективне управління особистим бюджетом, що буде функціональною та доступною. Для досягнення поставленої мети необхідно було вирішити наступні **завдання:**

- Провести системний аналіз теоретичних засад управління особистими фінансами та існуючих інформаційних систем у цій сфері.
- Обґрунтувати архітектурне рішення, сформувати технічні вимоги до системи та здійснити проектування структури даних.
- Здійснити реалізацію та налаштування програмного забезпечення, розробити ядро функціоналу та аналітичний модуль, а також провести оцінку ефективності ІС.

Об’єкт дослідження: процеси розробки інформаційної системи для управління особистими фінансами.

Предмет дослідження: моделі, методи, архітектурні рішення та інформаційні технології, що використовуються для побудови інформаційної системи ведення фінансового обліку та аналізу особистого бюджету.

Методи дослідження: Для досягнення мети та виконання завдань дослідження будуть використовуватися наступні методи:

1. Аналіз літератури:

Вивчення наукових та методичних матеріалів з ведення особистого бюджету та ознайомлення з дослідженнями в галузі розробки

інформаційних систем.

2. Системний аналіз та проектування:

Вивчення та порівняння доступних програм для ведення бюджету; визначення їхніх функціональних переваг та недоліків. Розробка архітектури та ER-моделі бази даних.

3. Розробка програмної системи:

Розробка прототипу інтерфейсу користувача, реалізація програмної логіки та аналітичного функціоналу.

4. Впровадження та оцінка:

Розгортання інформаційної системи, збір та аналіз даних про її використання та розрахунок економічної ефективності.

Використання цих методів дослідження дозволить розробити інформаційну систему, що відповідає потребам користувачів, досягти поставленої мети, отримати результати, які мають наукову та практичну цінність.

Наукова новизна одержаних результатів полягає у розробці інформаційної системи для управління особистими фінансами, що забезпечує поєднання оперативного обліку (OLTP) та кількісної оцінки її економічної ефективності для кінцевого користувача.

Практичне значення. Розроблена інформаційна система може бути використана для системного управління фінансами фізичними особами, сім'ями та малими підприємцями.

Система допоможе користувачам:

- Відстежувати свої доходи та витрати.
- Складати та контролювати бюджет.
- Аналізувати свої фінансові звички та прогнозувати майбутні фінансові потоки.

- Зменшити рівень фінансового стресу.
- Підвищити рівень фінансової стабільності
- Заощаджувати гроші.
- Досягати своїх financial goals.

Публікації. Результати дослідження опубліковано у збірнику наукових статей студентів, які здобувають освітній ступінь магістра за спеціальністю «Інформаційні системи і технології» ДТЕУ. Стаття «Розроблення інформаційної системи для управління особистими фінансами» // Інформаційні системи і технології в економіці [Електронний ресурс] : зб. наук. ст. студентів / відп. ред. А. В. Селіванова. – Київ: Держ. торг.-екон. ун-т, 2025. – С. 21-26.

Структура та обсяг кваліфікаційної роботи. Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел із 27 найменувань, додатків і містить 53 сторінки основного тексту, 20 рисунків і 5 таблиці.

РОЗДІЛ 1.
ТЕОРЕТИКО-МЕТОДИЧНІ ОСНОВИ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ ТА ІНФОРМАЦІЙНІ СИСТЕМИ ЇХ ЗАБЕЗПЕЧЕННЯ

1.1. Сутність та значення особистих фінансів і бюджетування в сучасних умовах

У сучасних умовах економічна, політична та соціальна нестабільність, яка особливо відчутна в Україні, змушує громадян активно шукати шляхи оптимізації своїх фінансів. Низькі доходи та стрімке зростання цін вимагають від населення постійно переглядати обсяги споживання та шукати найбільш раціональні варіанти витрат. За таких обставин особистий бюджет виступає як незамінний інструмент керування фінансовими ресурсами та доходами кожної особи. Ця необхідність підкреслює, що проблема фінансової нестачі часто полягає не в абсолютному розмірі доходу, а в невмінні правильно витрачати, що вимагає впровадження системного контролю [4].

Фінансова поведінка громадян значною мірою формується менталітетом та особливостями суспільства. До ключових чинників, що ускладнюють раціональне фінансове управління, належать: відсутність навичок усвідомленого вибору; висока вразливість до агресивних маркетингових кампаній, які часто провокують імпульсивні придбання [1]; феномен «демонстративного споживання», пов'язаний із прагненням підтримувати соціальний статус [2]; а також орієнтація на наслідування фінансової поведінки оточуючих. Критичним фактором є низький рівень фінансової грамотності – недостатнє розуміння принципів управління особистими фінансами веде до нераціонального розподілу доходів [3].

Щоб вирішити ці проблеми, сучасна економіка потребує активної участі фінансово освічених громадян, що обумовлює необхідність підвищення рівня фінансової грамотності. Підвищення фінансової грамотності є ключовим завданням як для покращення добробуту українців, так і для економічного розвитку країни, і цей процес потребує комплексного підходу, що включає державні програми, залучення фінансових інституцій та використання нових технологій [4]. Мета таких освітніх програм — сформувати покоління, яке володіє навичками раціонального розпорядження власними фінансами, вміє планувати бюджет та використовувати інструменти заощадження.

У цьому контексті, фінансова грамотність є критерієм успішної фінансової захищеності особистості в суспільстві. Вона включає три взаємопов'язані складові: світоглядні позиції, знання і навики (табл. 1).

Таблиця 1.1. Складові фінансової грамотності населення.

Складові фінансової грамотності населення	Характеристика
Світоглядні позиції	Наші фінансові звички, культура та рівень усвідомлення щодо необхідності фінансової грамотності потребують уваги та підвищення.
	Потрібно забезпечити освоєння та розуміння населенням основних фінансових категорій, понять, явищ і процесів, таких як сутність, мотиви та

Знання	чинники заощаджень, роль та функції фінансового ринку, принципи функціонування фінансових установ, а також взаємозв'язок між ризиком та дохідністю інших аспектів.
Навики	Набуття навичок пошуку та аналізу фінансової інформації, вміння відстежувати події на фінансовому ринку, а також здатність порівнювати пропозиції інвестування від різних фінансових установ є ключовими для формування фінансової грамотності. Також важливо вміти уважно аналізувати договори щодо фінансових послуг та розуміти всю вкладену в них інформацію.

Джерело: складено на основі джерела [5].

Офіційне визначення ОЕСР трактує фінансову грамотність як сукупність елементів (обізнаність, знання, навички, установки, поведінка), що дають змогу приймати обґрунтовані фінансові рішення для досягнення особистого фінансового благополуччя [6]. Управління особистими фінансами має ґрунтуватися на внутрішній мотивації та волі, вимагаючи лише регулярного часу для аналізу ситуації та внесення коректив.

Ефективне управління власним бюджетом вимагає послідовного застосування основних принципів та кроків [7]:

1. **Облік та Баланс Активів:** Фіксація місць знаходження грошей (готівка, банківські рахунки, кредитні картки). У програмному забезпеченні цей етап відображається як "balance".
2. **Категоризація Витрат та Доходів:** Створення індивідуалізованого списку категорій. Це дозволяє краще розуміти структуру грошових потоків. Складні чи рідкісні витрати, а також помилки обліку, мають бути включені до категорії "непередбачені" [8].
3. **Відстеження Транзакцій:** Регулярна реєстрація кожного витрачання, отримання або переказу коштів (транзакції), із прив'язкою до певної категорії та дати.

Люди з високим рівнем фінансової грамотності мають кращий захист від фінансових ризиків, підходять до управління своїми особистими фінансами з більшою відповідальністю. Таким чином, розвиток фінансової грамотності та уміння правильно керувати особистим бюджетом є невід'ємною складовою стратегії протидії економічним труднощам та забезпечення стабільності фінансового стану як на рівні окремої особи, так і на рівні всього суспільства.

1.2. Концептуальні засади побудови та класифікація інформаційних систем у сфері фінансового менеджменту

Розробка програмного забезпечення для управління особистими фінансами вимагає переходу від концепції локального програмного продукту до повноцінної інформаційної системи (ІС). ІС являє собою сукупність апаратних, програмних, мережевих та методологічних засобів, призначених для обробки, зберігання та надання даних, необхідних для підтримки управлінських рішень [9]. У сфері фінансового менеджменту ключовою

функцією ІС є не лише облік транзакцій, але й аналітична підтримка та візуалізація фінансових потоків.

Класифікація ІС у фінансовому менеджменті

Ефективні ІС, що працюють у сфері фінансів, класифікують за різними ознаками. Для задач управління особистими фінансами найбільш релевантними є наступні підходи [10]:

1. **За характером обробки даних:**
    - **Системи обробки транзакцій (OLTP):** Призначені для швидкої та надійної реєстрації, модифікації та видалення даних (наприклад, додавання витрат, зміна балансу). Вони є основою для ведення поточного фінансового обліку, який здійснюється на першій та другій сторінках вашої системи.
    - **Системи аналітичної обробки (OLAP):** Орієнтовані на складний багатовимірний аналіз, формування звітів, візуалізацію та підтримку прийняття рішень. Саме OLAP-функціонал (графіки, порівняння за періодами, візуалізація) перетворює облікову програму на систему управління, що відображено на третій сторінці вашої ІС [16].
2. **За рівнем управління:** ІС повинні підтримувати як оперативне управління (щоденний облік та контроль), так і тактичне (аналіз фінансових звичок та планування бюджету).

З точки зору архітектури, більшість сучасних фінансових ІС реалізується за клієнт-серверною моделлю. Цей підхід забезпечує централізоване зберігання даних, необхідну безпеку та масштабованість. Як правило, використовується трирівнева архітектура [11]:

- **Рівень представлення (клієнт):** Інтерфейс користувача (веб-додаток), що відповідає за введення даних, візуалізацію та автентифікацію (Google або email/пароль).
- **Рівень логіки (сервер):** Центральний елемент, що містить бізнес-логіку, виконує обробку запитів, обчислення (зміна балансу), реалізує логіку категоризації та аналізу даних.
- **Рівень даних (СКБД):** Система керування базами даних, яка забезпечує надійне, структуроване зберігання фінансової інформації (користувачі, транзакції, категорії).

Концептуальні вимоги до розробки ІС

Проектування ІС для управління особистими фінансами повинно ґрунтуватися на низці концептуальних засад та нефункціональних вимог, які гарантують її успішність [12]:

1. **Надійність та безпека даних (Data Security):** Оскільки ІС працює з конфіденційною інформацією, необхідно застосовувати стандартизовані методи автентифікації (Google-автентифікація, пароль) та контролю доступу.
2. **Масштабованість (Scalability) та продуктивність:** ІС повинна бути спроможна обробляти зростаючий обсяг транзакцій. Вибір технологій (наприклад, NoSQL бази даних, як MongoDB) повинен бути обґрунтований з точки зору цієї вимоги [15].
3. **Гнучкість та адаптивність (Flexibility):** Можливість легкої інтеграції нових функціональних модулів та адаптації до змін у вимогах користувачів [13].
4. **Ергономіка та юзабіліті (Usability):** Інтерфейс повинен забезпечувати інтуїтивно зрозуміле та швидке введення даних (транзакцій, описів,

сум) і зручну роботу з графічною візуалізацією звітів, що є однією з ключових особливостей даної розробки.

Таким чином, розроблювана ІС повинна перетворити сирі дані про доходи та витрати на управлінську інформацію, надаючи користувачеві можливість для обґрунтованого прийняття фінансових рішень на основі аналізу та візуалізації звітів. Вибір технологічного стеку та архітектури (Розділ 2) повинен бути безпосередньо пов'язаний із необхідністю забезпечення цих концептуальних вимог [14].

1.3. Аналіз існуючих програмних рішень для управління особистими фінансами та обґрунтування напрямів вдосконалення

Розробка власної Інформаційної системи (ІС) вимагає проведення детального критичного аналізу існуючих на ринку рішень у сфері фінансового обліку. Метою цього аналізу є не лише виявлення функціональних переваг, але й оцінка архітектурних підходів та аналітичних можливостей конкурентів, що слугує основою для розробки власного, унікального рішення [17]. На сьогоднішній день існує широкий спектр сервісів, що допомагають користувачам вести облік своїх фінансів, серед яких найбільш відомі Mint, YNAB, Personal Capital, QuickBooks Online та Quicken.

Аналіз ринку доцільно розпочати з оцінки найпоширеніших рішень. Першим для розгляду обрано сервіс Mint. Це безкоштовний веб-додаток, який здобув значну популярність завдяки функції автоматичної синхронізації з банківськими рахунками, що дозволяє автоматично відстежувати та категоризувати транзакції. (Рис. 1.1). Mint допомагає користувачам створювати бюджети, встановлювати цілі та надає рекомендації щодо

оптимізації фінансів. Проте, серед його недоліків слід зазначити, що Mint є менш гнучким у налаштуванні, ніж деякі інші програми, що може обмежувати користувачів, яким потрібна глибока індивідуалізація аналітичних звітів або деталізована OLAP-обробка.

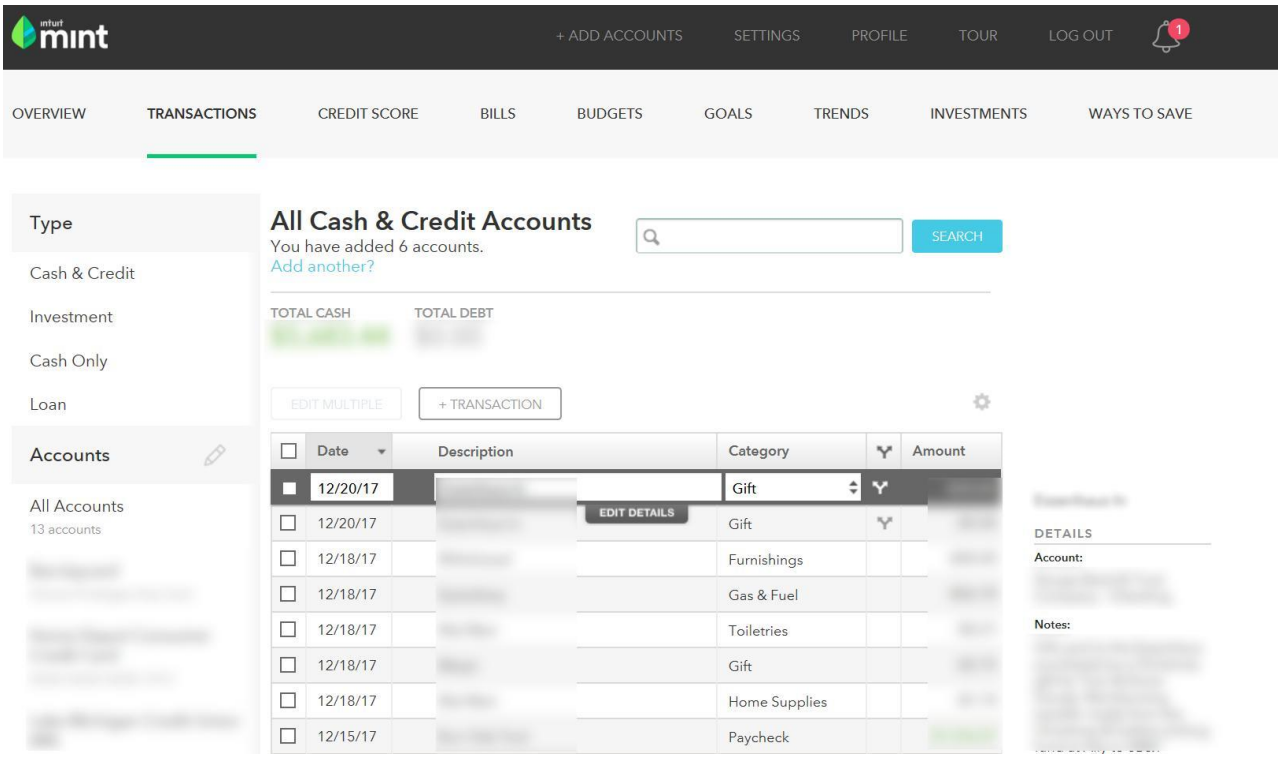

Рис. 1.1. Сторінка транзакцій користувача сервісу Mint

Серед переваг Mint можна виділити автоматичну синхронізацію з рахунками користувача, що дозволяє автоматично відстежувати та категоризувати транзакції. Крім того, Mint допомагає користувачам створювати бюджети, встановлювати цілі та надає рекомендації щодо оптимізації фінансів. Проте серед його недоліків слід зазначити, що Mint є менш налаштовуваним, ніж деякі інші програми для обліку фінансів, що може обмежувати користувачів, яким потрібно більше опцій індивідуалізації.

Наступним є сервіс YNAB (You Need A Budget):

Сервіс YNAB (You Need A Budget): YNAB відомий своєю спрямованістю на створення бюджетів за методом нульового бюджетування

та фінансову дисципліну. Він допомагає користувачам розробляти бюджети та слідкувати за витратами, закликаючи до ретельного планування. (Рис. 1.2). YNAB зосереджений на ретельному плануванні, що робить його потужним управлінським інструментом. Важливо пам'ятати, що використання YNAB передбачає платну підписку, а його жорстка методологія може не підійти користувачам, які шукають гнучкий інструмент для простого обліку та візуалізації фінансових потоків.

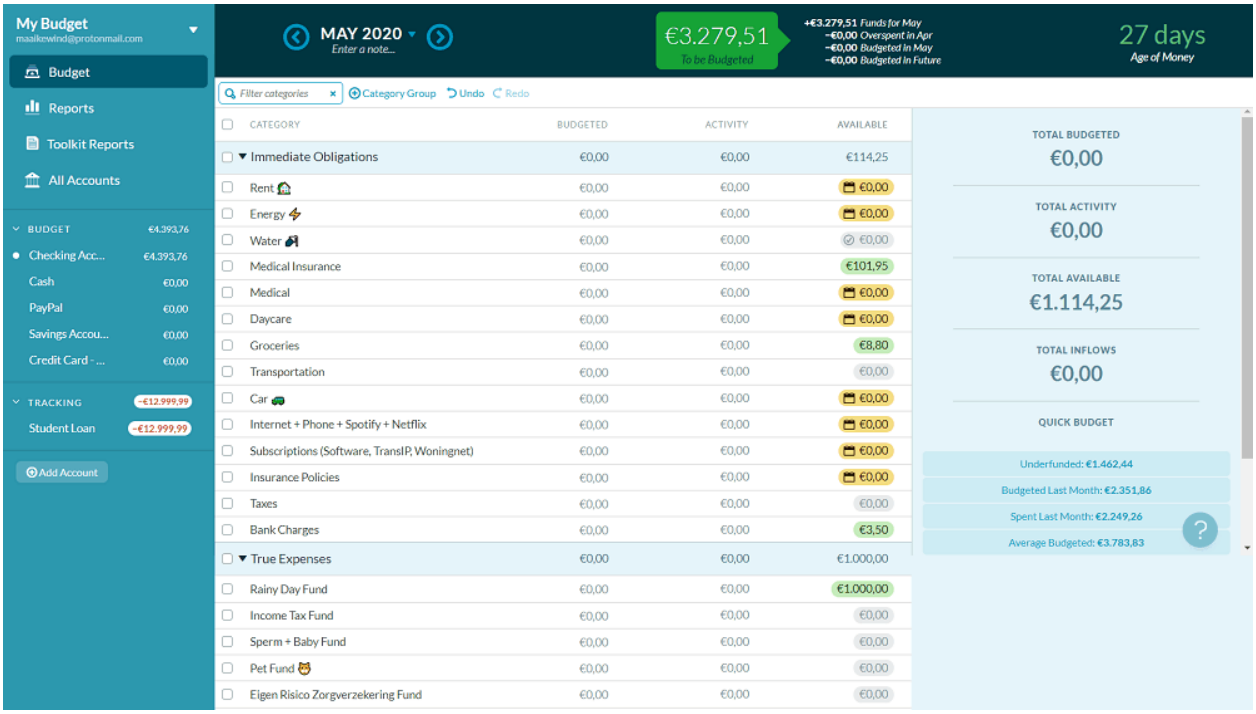

Рис. 1.2. Сторінка бюджету користувача сервісу YNAB

Наступним у списку є сервіс Personal Capital: Ця платформа має чітку спеціалізацію на інвестиційному аналізі та розробці планів пенсійного забезпечення. Її інтерфейс представлений на (Рис. 1.3). Проте, фокус на інвестиціях призводить до обмежених можливостей у детальному відстеженні повсякденних витрат, що робить його менш релевантним для осіб, чиї основні потреби — це оперативний облік та тактичне управління особистим бюджетом.

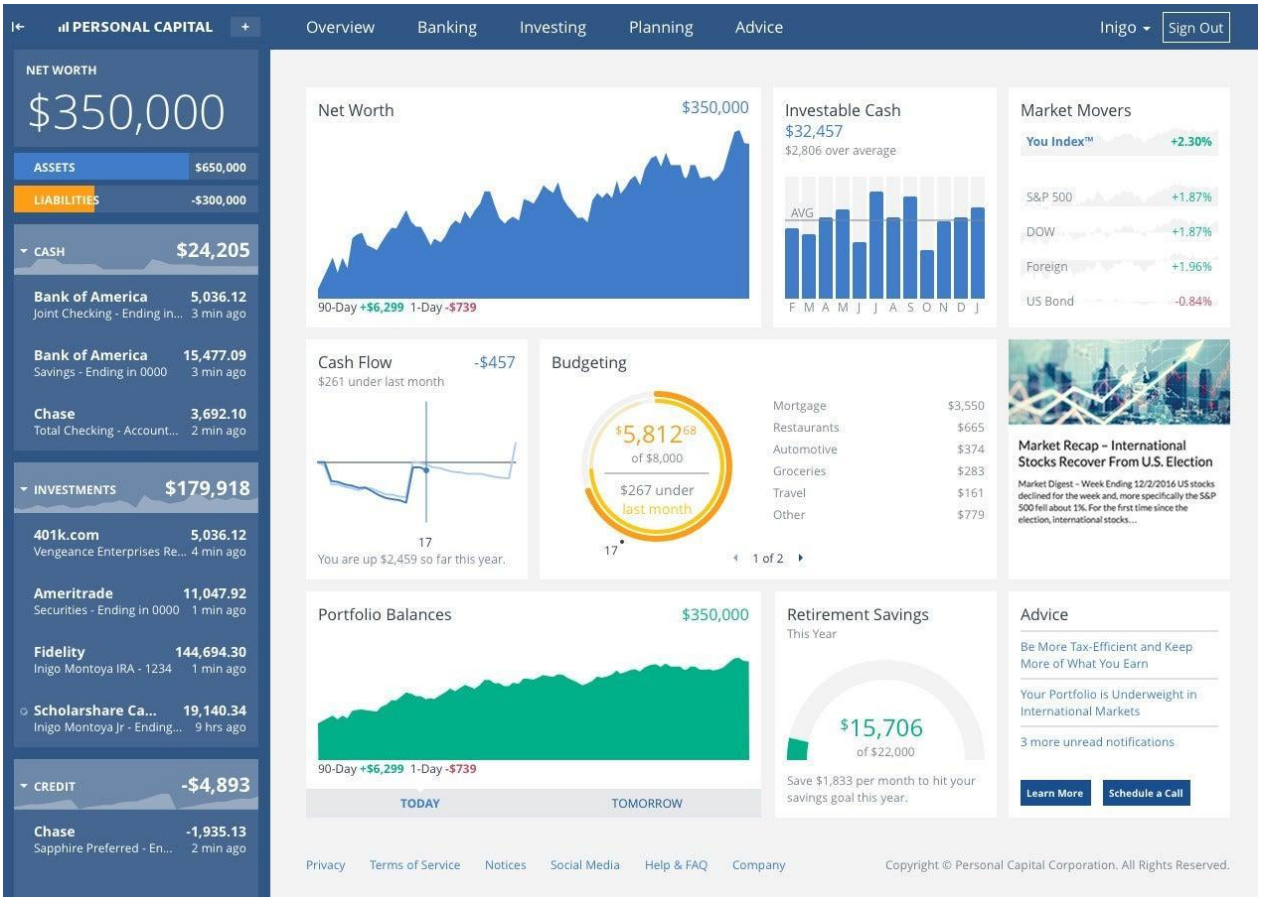

Рис. 1.3. Інтерфейс сайту Personal Capital

QuickBooks Online та Quicken: Ці продукти є високофункціональними інструментами, орієнтованими на малий бізнес (QuickBooks Online, Рис. 1.4) або на фізичних осіб із розширеними вимогами до обліку (Quicken, Рис. 1.5). Вони пропонують найширший функціонал, включаючи розширені інструменти обліку інвестицій та детальний аналіз фінансового стану. Однак, їхня надмірна складність для новачків, обумовлена великою кількістю функцій, а також необхідність платної підписки, робить ці рішення недоцільними для некомерційних користувачів.

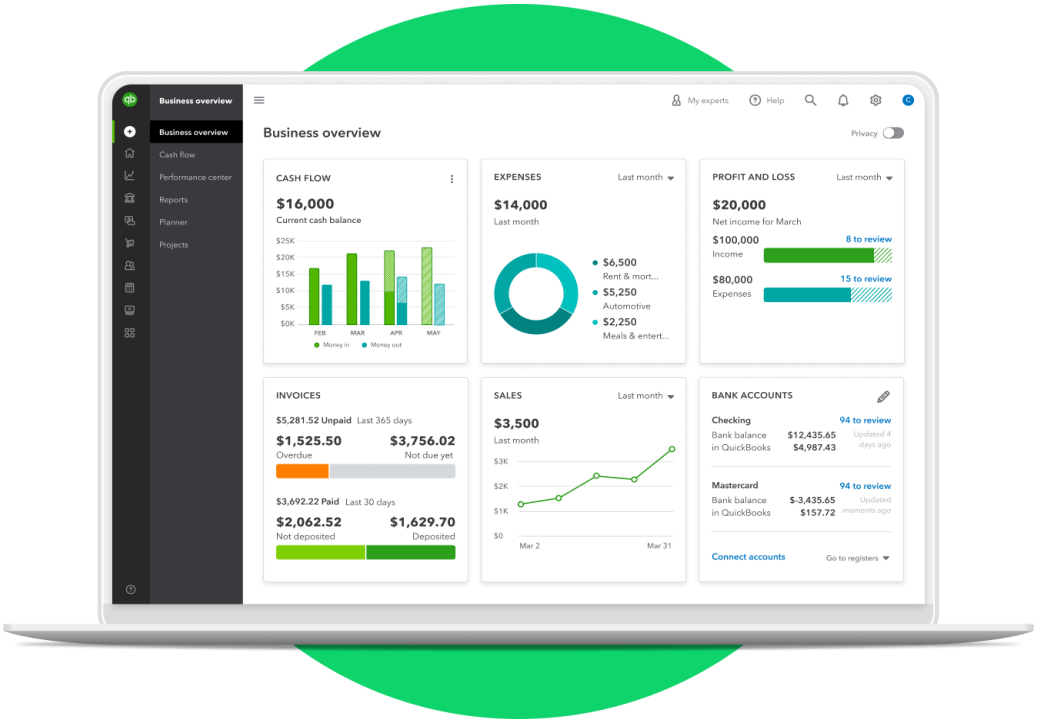

Рис. 1.4. Інтерфейс користувача QuickBooks Online

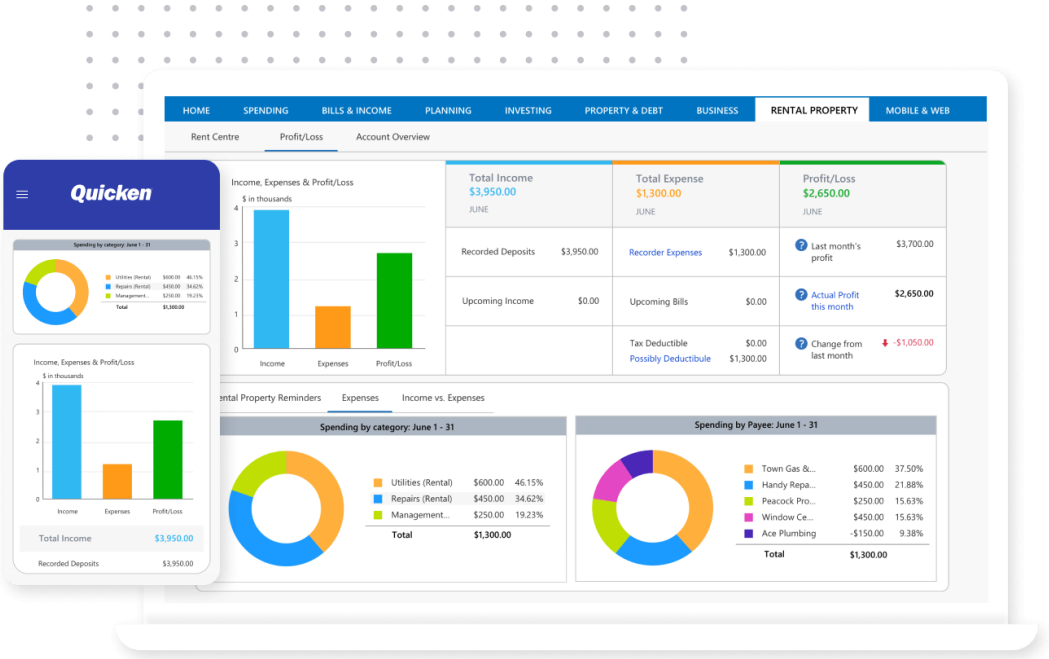

Рис. 1.5. Інтерфейс настільного додатку Quicken

Отриманий аналіз вищезгаданих сервісів обліку фінансів інвестицій веде до створення порівняльної таблиці 1.2.

Таблиця 1.2. Порівняння сервісів, що надають послуги обліку фінансів

Параметр	**Mint**	**YNAB**	**Personal Capital**	**QuickBooks Online**	**Quicken**
Наявність безкоштовної версії	+	-	+	-	-
Відстеження витрат	Так	Так	Обмежено	Так	Так
Управління інвестиціями	Ні	Ні	Так	Ні	Так
Планування бюджету	Так	Так	Ні	Так	Так
Доступність даних на різних пристроях	Так	Так	Так	Так	Так
Складність використання	Низька	Висока	Низька	Висока	Висока

Аналіз виявив низку ключових недоліків, які слугують напрямами вдосконалення для розроблюваної Інформаційної системи:

1. **Дисбаланс складності та функціоналу:** Існує розрив між простими, але функціонально обмеженими, та потужними, але надмірно складними і дорогими системами.
2. **Недостатня увага до OLAP-візуалізації:** Багато безкоштовних рішень не надають достатньо гнучких інструментів візуалізації (графіків,

діаграм), які є необхідними для якісної аналітичної підтримки та прийняття рішень.

3. **Вимоги до платної підписки:** Критично важливі функції управління часто є платними.

Реалізація власної Інформаційної системи дозволить досягти оптимального балансу між простотою введення даних та потужністю аналітичного модуля (OLAP-функціонал), що дозволяє ефективно візуалізувати фінансову діяльність користувача та забезпечити комплексне управління особистими фінансами.

РОЗДІЛ 2.
АНАЛІЗ ТА ПРОЕКТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Обґрунтування архітектурного рішення та стилів програмного забезпечення

Архітектура програмного забезпечення веб-додатків є критично важливим аспектом, що визначає злагоджену співпрацю та обмін даними між різноманітними програмними компонентами [9]. Вона встановлює чіткі правила взаємодії між системами проміжного коду, користувацькими інтерфейсами та базами даних, забезпечуючи безперебійну роботу системи. Ця архітектура втілює концепцію безперешкодної взаємодії між веб-браузерами користувачів та програмним забезпеченням, розгорнутим на веб-серверах.

Роль та завдання архітектурного проектування ІС

При розробці Інформаційної системи (ІС), особливо у фінансовій галузі, архітектура виконує не лише функцію структурування коду, але й є основним засобом забезпечення нефункціональних вимог [14]. До них належать надійність, продуктивність, безпека та, що найважливіше, масштабованість та можливість супроводу (supportability) [13].

Розробляючи програмне забезпечення, важливо наперед продумати його життєвий цикл. Підтримка проєкту включає внесення змін до нього та додавання нових функцій. Архітектура допомагає визначити взаємодію компонентів інтерфейсу з внутрішніми процесами програми. Архітектура та проектування ПЗ забезпечують гарантію того, що додаток виконуватиме завдання та слідувати своєму призначенню, визначеному під час початкових етапів розробки [10].

Архітектурний задум програмного рішення відіграє ключову роль у забезпеченні злагодженої роботи системи. Його принципові функції включають:

- **Встановлення структурованого каркасу:** Це уможливлює глибинне розуміння конструкції додатку, рівнів виконання завдань та реалізації функцій, що є критичним при роботі команди розробників.
- **Чітке визначення поведінкових патернів:** Встановлення принципів взаємодії компонентів, що дозволяє прогнозувати системні реакції та запобігати колізіям даних.
- **Управління складністю:** Диференціація критично важливих елементів (наприклад, обробка транзакцій) та другорядних, що дозволяє оптимізувати бюджетні обмеження.
- **Забезпечення масштабованості:** Архітектура повинна забезпечувати легку інтеграцію нових можливостей (наприклад, додавання аналітичних модулів) та обробку зростаючого числа користувачів без зниження продуктивності [13].
- **Гармонізація клієнтських вимог:** Знаходження збалансованих рішень для суперечливих вимог.
- **Спрощення документації:** Чіткий опис архітектури значно полегшує подальшу підтримку та модифікацію.

Раціональна архітектура проекту є запорукою створення високоякісного програмного забезпечення, а також слугує потужним підґрунтям для мінімізації витрат на його подальшу підтримку та обслуговування.

Критичний аналіз та вибір архітектурної моделі

Архітектурний план володіє атрибутом, званим стилем, який забезпечує системну гармонію. Стиль визначається через застосування патернів та компонентів. При виборі архітектурного стилю для ІС управління особистими фінансами, необхідно провести порівняльний аналіз ключових підходів.

Наприклад, монолітна архітектура, хоч і проста у початковій розробці, суттєво ускладнює подальшу підтримку та незалежне масштабування окремих функцій (наприклад, модуля звітів). Сервісно-орієнтований стиль (SOA) або мікросервісна архітектура [13] пропонують високу гнучкість, однак їхня імплементація є значно складнішою та ресурсозатратнішою, що є надлишковим для ІС, орієнтованої на індивідуального користувача. Компонентна та багатошарова архітектури пропонують хороший баланс, але саме клієнт-серверний стиль є де-факто стандартом для веб-додатків [11].

Провівши ретельний аналіз та зважаючи на необхідність забезпечення безпеки фінансових даних, централізованої аналітики та гнучкості інтерфейсу, для розроблюваної ІС було обрано класичну клієнт-серверну архітектуру з реалізацією трирівневої (3-tier) моделі [11].

Ця парадигма відзначається виваженим розподілом обов'язків та чіткою демаркацією між клієнтською та серверною частинами системи. Клієнт-серверна архітектура (Рис. 2.1) складається з трьох компонентів:

- **Клієнт:** Користувач (веб-браузер), який використовує сервіс і звертається до сервера за необхідною інформацією.
- **Сервер:** Місце, де розташована серверна частина веб-додатку, його серверна частина, де зберігаються дані про клієнта.
- **Мережа:** Інтернет, який забезпечує обмін інформацією.

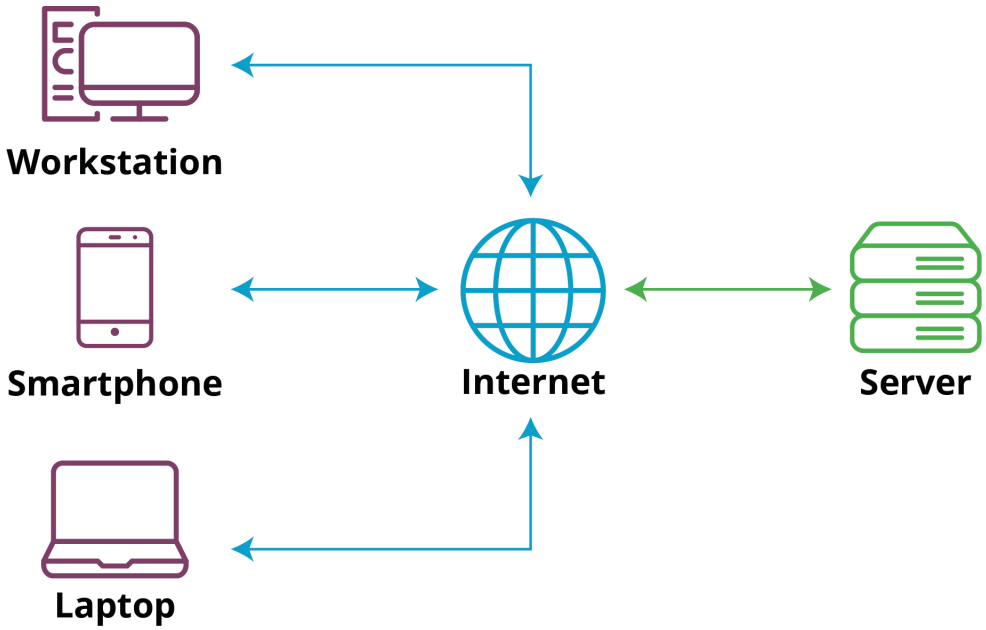

Рис. 2.1. Приклад клієнт-серверної архітектури

Обраний застосунок має трирівневу архітектуру (Рис. 2.2), яка забезпечує логічну та фізичну ізоляцію трьох ключових компонентів системи [11]:

1. **Рівень 1: Рівень представлення (Client/Presentation Tier):** Це веб-браузер, до якого надсилаються HTML-сторінки. У даній роботі це React-додаток, що відповідає за рендеринг UI, обробку взаємодії з користувачем (введення транзакцій, автентифікація через Google або email) та візуалізацію аналітичних звітів (графіків та діаграм), що є функцією OLAP [16].
2. **Рівень 2: Рівень логіки (Application/Logic Tier):** Це шар логіки (реалізований на Node.js), де відбувається обробка запитів та відповідей. Це є ядром системи, яке містить всю бізнес-логіку: валідація даних, обчислення (перерахунок балансу), а також агрегація даних для аналітичних запитів (OLAP-функціонал) [16].

3. **Рівень 3: Рівень даних (Data Tier):** Це сховище даних (СКБД), яке є фізично та логічно відокремленим. Таке відокремлення є критично важливим для безпеки фінансових даних.

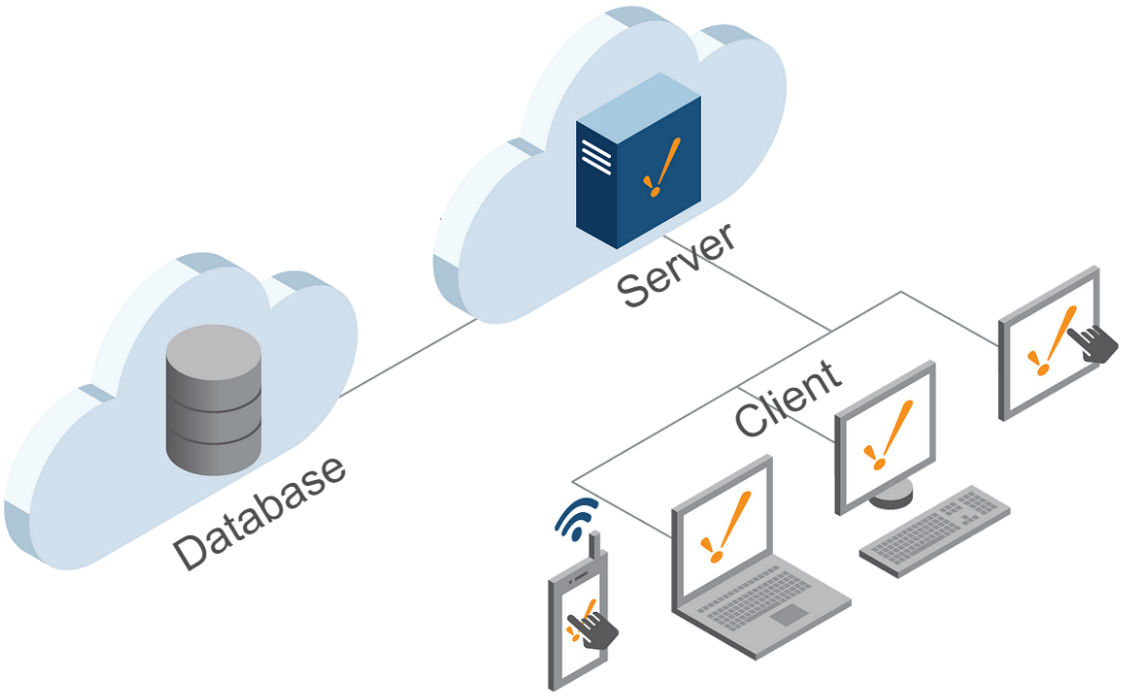

Рис. 2.2. Приклад трьохрівневої моделі

Серверна частина веб-додатку відповідає за обробку запитів. Це ядро системи, де розміщується веб-сервер для прийому та обробки HTTP-запитів.

Обґрунтування вибору технологічних компонентів архітектури

Вибір бази даних (Рівень 3) є критичним для продуктивності та масштабованості ІС. У даному проекті обрано NoSQL базу даних (MongoDB). На відміну від реляційних СКБД, NoSQL-рішення (зокрема, документ-орієнтовані) були обрані через [12]:

- **Гнучкість схеми даних:** Це дозволяє легко зберігати транзакції з різною структурою описів та категорій без необхідності складної міграції таблиць.
- **Горизонтальна масштабованість:** NoSQL-системи, як MongoDB, легко масштабуються горизонтально, що є перевагою для веб-додатків із потенційно великою кількістю користувачів [15].

Для зв'язку між клієнтською та серверною частинами (Рівень 1 та Рівень 2) використовується протокол HTTP (через REST API) для стандартних CRUD-операцій (створення, читання, оновлення транзакцій). Водночас, архітектура передбачає можливість використання WebSocket [13] для майбутнього розширення функціоналу, наприклад, для миттєвих сповіщень.

Обрана трирівнева архітектура знаходить своє практичне відображення у блок-схемі роботи веб-додатку (Рис. 2.3). Компоненти системи чітко розподілені по рівнях:

- **Компонент авторизації/реєстрації** (включаючи Google Auth) інкапсульований на Рівні 2 (Логіка) та Рівні 3 (Дані).
- **Персональна сторінка користувача** є прикладом тісної взаємодії всіх трьох рівнів для **OLTP-операцій** (введення балансу, витрат, доходів).
- **Сторінка звітів** демонструє роботу **OLAP-функціоналу** [16], де Рівень 2 (Логіка) агрегує дані з Рівня 3 (Дані) для побудови графіків, які візуалізуються на Рівні 1 (Клієнт).

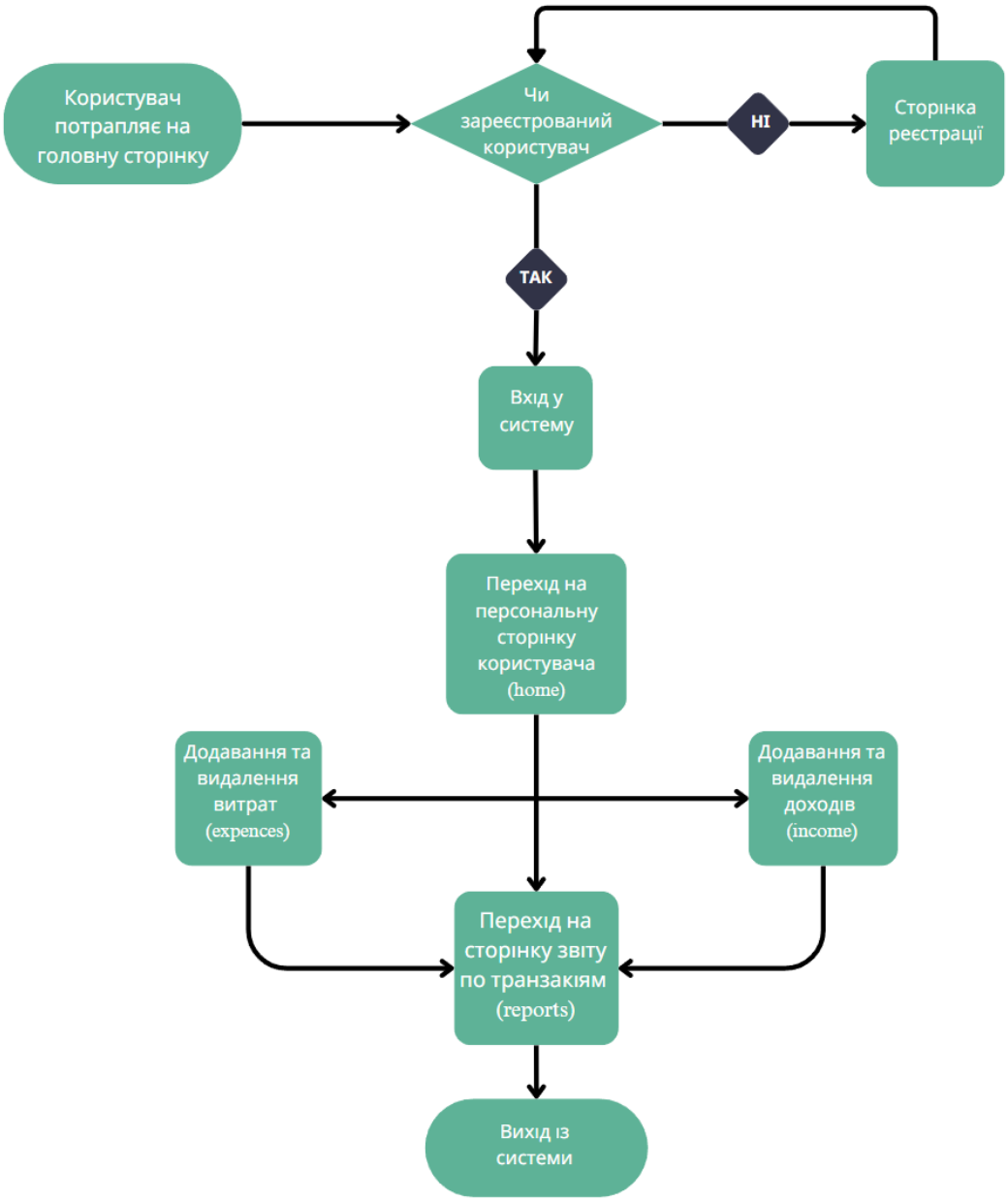

Рис. 2.3. Блок схема роботи веб-додатку

Отже, після аналізу архітектури та урахування всіх аспектів, обрана трирівнева клієнт-серверна модель є оптимальною для створення веб-додатку для управління особистими фінансами, який буде не лише якісним, але й масштабованим, безпечним та зручним для користувачів.

2.2 Формування технічних вимог та вибір технологічного стеку

Вибір технологічного стеку безпосередньо залежить від вимог, що висуваються до інформаційної системи. Процес проектування починається з детального формування технічних вимог, які поділяються на функціональні (що система робить) та нефункціональні (як система це робить).

Аналіз функціональних та нефункціональних вимог

На основі опису функціоналу програми, представленого раніше (автентифікація, управління балансом, введення транзакцій, візуалізація звітів), були сформульовані детальні технічні вимоги до frontend та backend частин системи.

Функціональні вимоги до клієнтської частини (frontend) деталізовані у Таблиці 2.1. Вони охоплюють ключові use-cases системи:

- **Автентифікація користувача:** (Пункт 4, 5, 6) – реалізація стандартної реєстрації (email/пароль) та інтеграція з провайдером Google OAuth.
- **Управління балансом:** (Пункт 7, 9, 10) – ініціалізація початкового балансу для нових користувачів.
- **Управління транзакціями (OLTP):** (Пункт 13-19) – основний функціонал додавання (з описом та категорією) та видалення доходів/витрат, що миттєво впливає на баланс.
- **Аналітика та звіти (OLAP):** (Пункт 22, 24, 26, 27, 29) – вимоги до сторінки звітів, що включають візуалізацію даних (графіки) та фільтрацію за періодами.

Таблиця 2.1. Технічні вимоги до front-end

FRONT-END	
1. Розмітка та бекграунд для всіх сторінок, коли користувач залогінен і ні (телефон, планшет та десктоп)	Background
2. Розмітка та стилі хедера при логінізації та її відсутності (телефон, планшет та десктоп)	Header
3. Реалізувати логіку виходу з профілю при натискання кнопки "Вийти"	Header
4. Розмітка та стилі вікна логінізації/реєстрації (телефон, планшет та десктоп)	Login/Register
5. Функціонал надсилання запиту на логінізацію/реєстрацію користувача	Login/Register
6. Функціонал логінізації/реєстрації за допомогою кнопки "Google"	Login/Register
7. Розмітка та стилі компонента "Баланс" (телефон, планшет та десктоп)	Home
8. У полі "Баланс" відображається твій баланс у стейті	Home
9. При натисканні на кнопку "Підтвердити" сума балансу відправляється на бек і записується в стор.	Home
10. Створити модалку за нульового балансу	Home
11. Створити універсальну модалку під час розлогінізації	Home
12. При натисканні на кнопку "Перейти до звітів" перекидає на сторінку Звітів з витрат	Home
13. Розмітка та стилі секцій "Витрати" та "Дохід"	Home
14. Має відображатися сьогоднішня дата	Home
15. Намалювати спливаючий перелік категорій товарів	Home
16. При натисканні кнопки "Введення" дані з форми відправляються на бек і записуються в таблицю, а форма очищується	Home
17. При натисканні кнопки "Очистити" форма очищується	Home
18. В секції "Витрати" в таблиці показується негативна сума, навіть якщо у БД збережено її у вигляді позитивного значення	Home
19. У секціях "Витрати" і "Дохід" у таблиці при натисканні на кошик йде запит на бек про видалення запису, після чого даний запис видаляється зі стейту	Home
20. Розмітка та стилі компонента "Зведення"	Home
21. Дані у зведення підтягуються з бека	Home
22. Розмітка та стилі сторінки зі "Звітами" витрат та доходів	Report

23. Розмітка і стилі компонента "Повернутися на головну", при натисканні користувача перекидає на головну сторінку	Report
24. Розмітка та стилі компонента "Поточний період", при натисканні на стрілки повинен змінювати місяць в обидві сторони	Report
25. Розмітка та стилі секції "Витрати/Доходи" повинні відображатися актуальні дані (телефон, планшет, десктоп)	Report
26. Розмітка та стилі секції "Витрати", показувати тільки ті категорії, в яких були витрати (телефон, планшет, робочий стіл) в порядку від категорії з найбільшим показником до меншого	Report
27. Розмітка та стилі секції "Доходи", показувати тільки ті категорії, в яких були доходи (телефон, планшет, робочий стіл) в порядку від категорії з найбільшим показником до меншого	Report
28. Стрілки в секціях "Витрати" та "Доходи" повинні перемикати користувача між цими сторінками	Report
29. Графік повинен візуально відображати витрати та доходи обраної категорії (телефон, планшет, робочий стіл) в порядку від витратою/доходом з найбільшим показником до меншого	Report

Технічні вимоги до серверної частини (backend), представлені у Таблиці 2.2, описують необхідні кінцеві точки (endpoints) API для підтримки функціоналу frontend. Вони чітко відповідають трирівневій архітектурі, де backend виступає як логічний рівень, що обробляє запити від клієнта.

Таблиця 2.2. Технічні вимоги до back-end

BACK-END
1. Обдумати структуру, ініціалізувати та підключити БД
2. Розгорнути сервер (вилов помилок, CORS, структура підключення модулів тощо)
3. Реалізувати енд-поінт реєстрації
4. Реалізувати енд-поінт аутентифікації
5. Реалізувати енд-поінт логауту
6. Реалізувати прошарок авторизації
7. Реалізувати енд-поінт оновлення балансу користувача
8. Реалізувати енд-поінт додавання витрати
9. Реалізувати енд-поінт отримання доходу
10. Реалізувати енд-поінт видалення транзакції
11. Реалізувати енд-поінт отримання зведення про місяці поточного року до витрат
12. Реалізувати енд-поінт отримання зведення про місяці поточного року за

доходами
13. Реалізувати енд-поінт отримання докладної інформації (дивитися макет) про витрати та доходи за конкретні місяць та рік

Нефункціональні вимоги, хоча й не винесені в окрему таблицю, є критично важливими для ІС управління фінансами [14]. З аналізу вимог випливають наступні нефункціональні пріоритети:

- **Надійність та Безпека:** (Пункт 6, 19, Таблиця 2.1; Пункт 4, 6, 10, Таблиця 2.2) – система повинна безпечно зберігати дані користувача, використовувати захищені канали передачі даних та надійні механізми автентифікації.
- **Продуктивність:** Система має забезпечувати швидкий відгук на дії користувача (миттєве оновлення балансу).
- **Масштабованість:** Архітектура повинна підтримувати зростання кількості користувачів та обсягу транзакцій [15].
- **Юзабіліті (Зручність):** Інтерфейс має бути інтуїтивно зрозумілим для швидкого введення даних, щоб мінімізувати когнітивне навантаження на користувача.

Обґрунтування вибору технологічного стеку

З огляду на сформовані вимоги та обрану в п. 2.1 клієнт-серверну архітектуру, було обрано MERN-стек (MongoDB, Express.js, React.js, Node.js) та супутні технології.

Серверна частина програми (backend) була реалізована за допомогою Node.js та Express.js. Для збереження та керування даними ми використовували MongoDB. На клієнтській стороні (frontend) ми використовували JavaScript разом з React.js для побудови інтерактивного інтерфейсу користувача, а також Redux для керування станом додатку. Для маршрутизації між сторінками використовувався React Router. Інтерфейс був

побудований за допомогою HTML та стилізований за допомогою SASS. Для взаємодії між клієнтом та сервером ми використовували REST API.

Критичний аналіз та обґрунтування вибору:

JavaScript (HTML/CSS/SASS): Це невід'ємна основа будь-якого сучасного веб-додатку. JavaScript використовується для створення інтерактивності, HTML – для структури, а SASS [15] (як препроцесор CSS) був обраний замість чистого CSS для підвищення ефективності розробки стилів завдяки змінним, вкладеним правилам та міксинам.

React.js (Frontend): Для реалізації клієнтської частини (вимоги Таблиці 2.1) React був обраний замість альтернатив (наприклад, Angular або Vue) з кількох причин [18]:

1. **Компонентний підхід:** React дозволяє створювати ізольовані, повторно використовувані компоненти. Це ідеально підходить для реалізації складних сторінок, таких як "Сторінка звітів" (Пункт 22, 29) та "Вкладка витрат/доходів" (Пункт 13).
2. **Віртуальний DOM:** Забезпечує високу продуктивність при оновленні інтерфейсу, що критично для миттєвого відображення змін балансу (Пункт 8).
3. **Redux для управління станом:** Для складних додатків, де стан (баланс, транзакції) має бути доступним у різних компонентах, Redux є ефективним рішенням для централізованого управління.

Node.js та Express.js (Backend): Для реалізації серверної частини (вимоги Таблиці 2.2) Node.js був обраний замість традиційних платформ (як PHP/Laravel або Python/Django) з наступних причин [19]:

1. **Асинхронність та неблокуючий I/O:** Node.js є високоефективним для обробки великої кількості одночасних запитів (наприклад, запити на

оновлення балансу (Пункт 7) або додавання транзакцій (Пункт 8)), не блокуючи основний потік.

2. **Єдина мова (Full-stack JavaScript):** Використання JavaScript на frontend (React) і backend (Node.js) дозволяє уніфікувати процес розробки, спростити обмін даними та повторне використання коду валідації.
3. **Express.js** був обраний як мінімалістичний та гнучкий фреймворк для швидкого створення REST API.

MongoDB (База даних): Як було обґрунтовано в п. 2.1, вибір NoSQL бази даних MongoDB [16] зумовлений її гнучкою схемою (ідеально для зберігання транзакцій з різними описами та категоріями, Пункт 15, Таблиця 2.1) та високою горизонтальною масштабованістю [12, 20].

Обраний стек технологій (MERN) є досить потужним для створення додатку, який забезпечить продуктивність та зручність використання. Завдяки React та SASS ми маємо динамічний та інтуїтивно зрозумілий інтерфейс. Node.js дозволяє ефективно обробляти запити. А база даних MongoDB забезпечує надійне зберігання інформації. Все це разом дозволяє створити веб-додаток, який відповідає всім поставленим функціональним та нефункціональним вимогам.

2.3. Проектування структури даних (ER-модель) та структурних схем ІС

Після обґрунтування архітектури (п. 2.1) та визначення вимог і технологічного стеку (п. 2.2), наступним логічним етапом проектування є моделювання даних. Правильна структура бази даних є ключовим фактором, що забезпечує ефективність, цілісність даних та масштабованість системи.

Цей розділ присвячений проектуванню бази даних на основі обраної технології NoSQL (MongoDB).

Проектування моделі даних (Колекції MongoDB)

На відміну від реляційних баз даних, MongoDB використовує гнучку, документ-орієнтовану модель. Дані зберігаються у BSON-документах (бінарний JSON), які групуються в колекції. На основі функціональних вимог (Таблиці 2.1 та 2.2) було спроектовано дві основні сутності (колекції):

1. **Колекція «users»** (рис. 2.4). Ця сутність призначена для зберігання даних про користувачів системи. Вона включає поля для автентифікації (email, хешований пароль), персональні дані (ім'я), а також ключові фінансові атрибути: поточний balance та службовий прапорець isNotNewUser (для відображення запиту про початковий баланс).

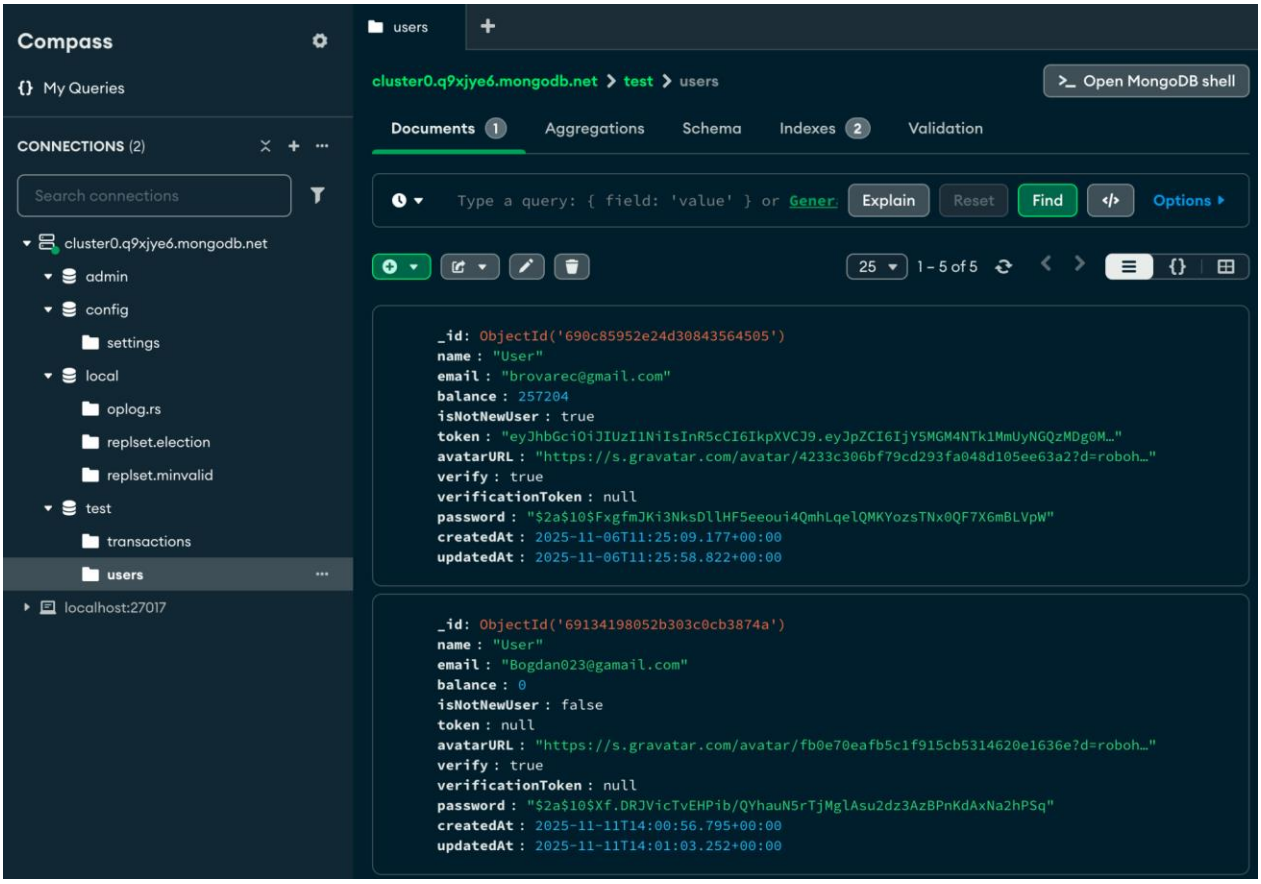

Рис. 2.4. Колекція “users”

2. Колекція «transactions» (рис. 2.5). Ця сутність є ядром ІС і зберігає всю фінансову історію. Вона містить інформацію про операції користувачів, включаючи тип (transactionsType: 'income' або 'expenses'), дату, опис, категорію та суму. Ключовим полем є owner, яке містить ObjectId користувача з колекції users, реалізуючи таким чином зв'язок між колекціями

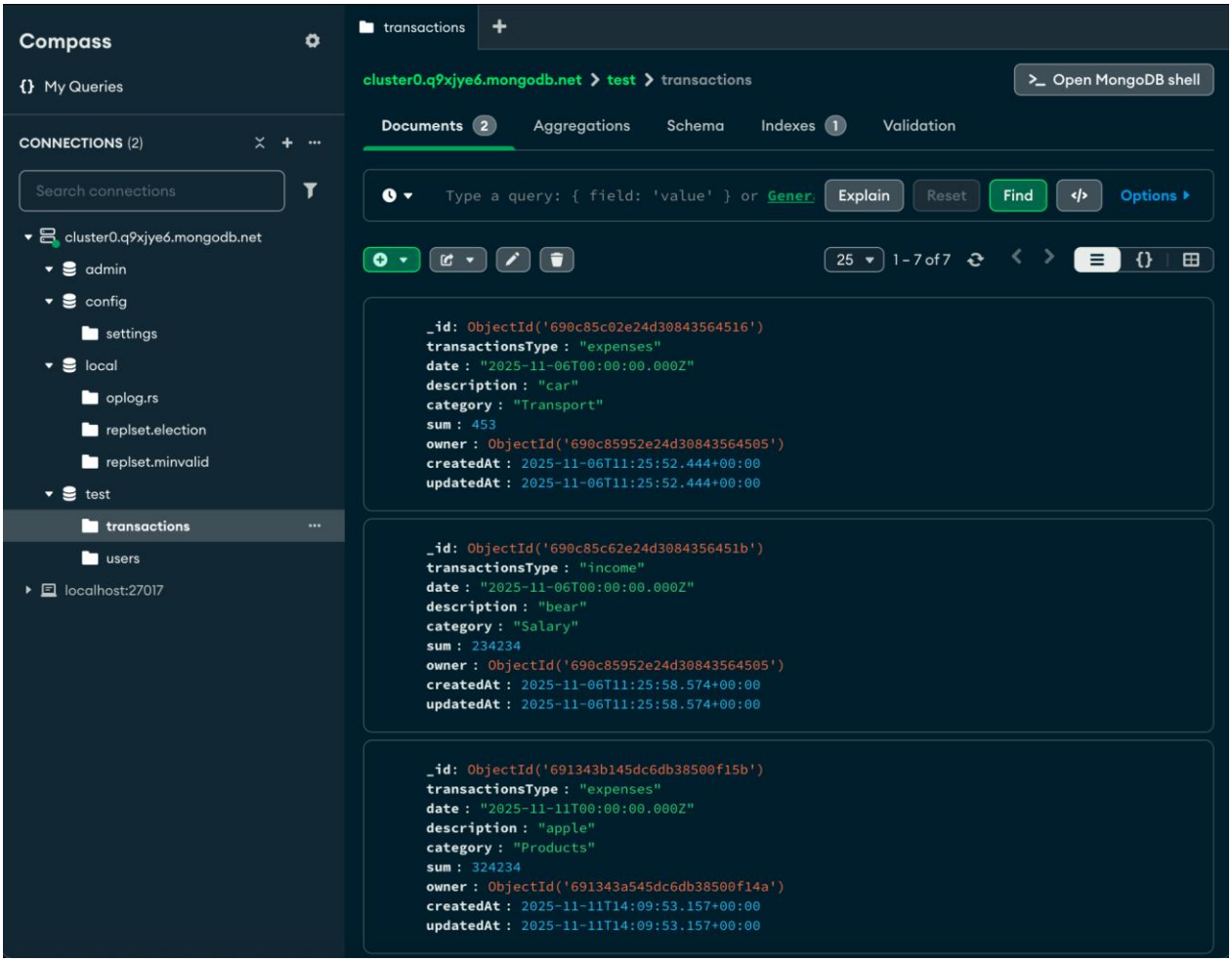

Рис. 2.5. Колекція “transactions”

Використання NoSQL (MongoDB) дозволяє зберігати транзакції як гнучкі документи, що є перевагою, оскільки не вимагає жорсткої структури і дозволяє в майбутньому легко додавати нові поля (наприклад, "гео-мітка" або "фото чеку") без зміни всієї схеми БД.

ER-модель та структурні схеми

Хоча ER-діаграми є традиційними для реляційних баз даних, їх адаптована версія використовується і для NoSQL для візуалізації логічних зв'язків між сутностями.

ER-діаграма (рис. 2.6) відображає основні сутності бази даних та зв'язки між ними. Вона наочно демонструє зв'язок типу «один-до-багатьох»

(one-to-many): один користувач (users) може мати багато транзакцій (transactions). Ця візуалізація допомагає при проектуванні запитів до бази даних та реалізації схеми даних на рівні Mongoose (що буде розглянуто у Розділі 3).

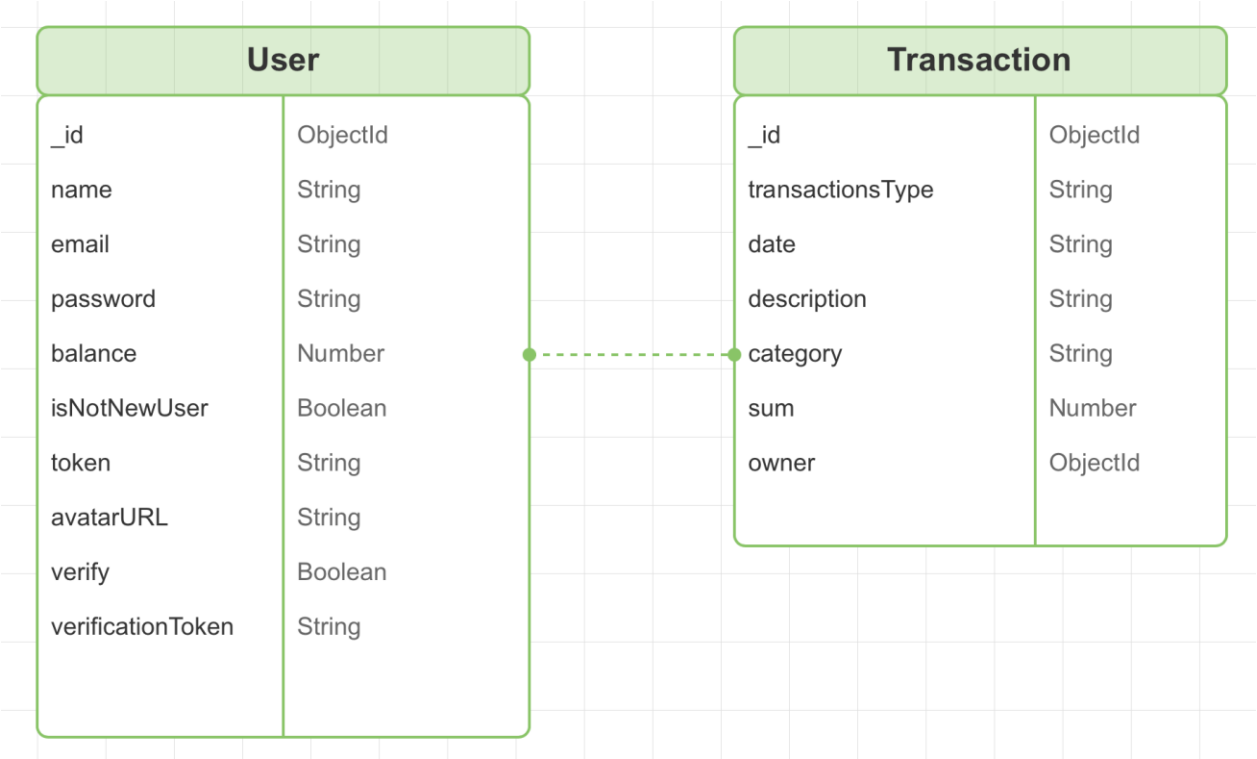

Рис. 2.6. ER-діаграма бази даних

Окрім моделі даних, важливим елементом проектування є загальна структурна схема ІС, яка була представлена раніше у Блок-схемі роботи веб-додатку (див. Рис. 2.3). Ця схема ілюструє логічні потоки даних та взаємодію компонентів, розроблених на основі трирівневої архітектури:

- Потік автентифікації: (Вхід/Реєстрація/Google Auth).
- Потік OLTP (Обробка транзакцій): (Введення балансу -> Додавання/видалення витрат/доходів -> Оновлення балансу).
- Потік OLAP (Аналітика): (Запит даних за період -> Агрегація даних на сервері -> Відображення на "Сторінці звітів").

Таким чином, проектування структури даних (колекції users та transactions) та аналіз структурних схем (ER-діаграма та Блок-схема) формують повне уявлення про архітектуру даних та інформаційні потоки в розроблюваній ІС.

РОЗДІЛ 3.
РЕАЛІЗАЦІЯ, НАЛАШТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Основні етапи реалізації, налаштування сервера та розробка ядра функціоналу

Після детального проектування архітектури (Розділ 2), яке визначило трирівневу клієнт-серверну модель, наступним етапом є безпосередня програмна реалізація та налаштування серверної частини (Backend). Цей

компонент є «ядром» інформаційної системи, оскільки він відповідає за всю бізнес-логіку, обробку даних та забезпечення безпеки. Цей етап включає ініціалізацію підключення до обраної NoSQL бази даних (MongoDB) та створення логіки обробки API-запитів на платформі Node.js.

Налаштування підключення до бази даних

Для взаємодії з базою даних MongoDB з середовища Node.js було обрано бібліотеку Mongoose. Це рішення є стратегічно обґрунтованим для кваліфікаційної роботи, оскільки Mongoose є ODM (Object Data Modeling) бібліотекою. На відміну від прямого використання нативного MongoDB-драйвера, Mongoose надає низку критично важливих переваг [11]:

1. **Валідація даних на рівні моделі:** Дозволяє визначити чіткі правила для даних (наприклад, "email має бути унікальним", "поле balance є обов'язковим"), що забезпечує цілісність даних ще до їх потрапляння у БД.
2. **Схеми (Schemas):** Надає чітку структуру для документ-орієнтованої БД (як було спроектовано у п. 2.3), що робить код більш читабельним та підтримуваним.
3. **"Middleware" (Хуки):** Дозволяє виконувати логіку до або після певних подій (наприклад, автоматично хешувати пароль перед збереженням користувача).

Підключення до бази даних здійснюється асинхронно, щоб не блокувати головний потік Node.js. На рисунку 3.1 наведено приклад коду, що використовується у головному файлі сервера (server.js) для встановлення з'єднання.

JS server.js > ...

```
require('dotenv').config();
const mongoose = require('mongoose');
mongoose.set('strictQuery', false);
const moment = require('moment');
const app = require('./app');
const { DB_HOST, PORT = 3033 } = process.env;
const currentDate = moment().format('hh:mm:ss DD-MM-YYYY');

(async () => {
  try {
    await mongoose.connect(DB_HOST);
    app.listen(PORT);
    console.log(`Server is running on the port: ${PORT} `.bgGreen.red);
    console.log(`Start project: Easy Start Wallet (Backend) `.bgRed.green);
    console.log('Database connection successful '.bgBlue.yellow);
    console.log('Date & Time:'.bgYellow.blue, currentDate.yellow);
    console.log('------------------------------------'.yellow);
  } catch (error) {
    console.log(error.message);
    process.exit(1);
  }
})();

```

Рис. 3.1. Підключення бази данних mongoDB

Цей код встановлює з'єднання з MongoDB, використовуючи URL-адресу (змінну DB_HOST) з файлу конфігурації .env. Використання файлу .env для зберігання змінних середовища є фундаментальною практикою безпеки у сучасній розробці [14]. Це дозволяє приховати чутливі дані (ключі доступу до БД, секретні ключі для JWT-токенів) з коду, який може бути опублікований у системі контролю версій (наприклад, Git).

Архітектура та логіка серверної частини

Структура серверного коду відіграє ключову роль у забезпеченні його підтримуваності (maintainability) та масштабованості. У файлі server.js встановлюються основні налаштування сервера, такі як обробка CORS-запитів (для дозволу з'єднань з клієнта), використання "проміжних шарів" (middleware) для обробки JSON-запитів (express.json()) та підключення файлів маршрутизації.

Для організації логіки нашого додатку ми використовуємо архітектурний патерн MVC (Model-View-Controller), адаптований для REST API. Цей патерн забезпечує чіткий "розподіл обов'язків" (separation of concerns), що є вимогою якісного проектування [10]. Структура сервера (рис 3.2) включає:

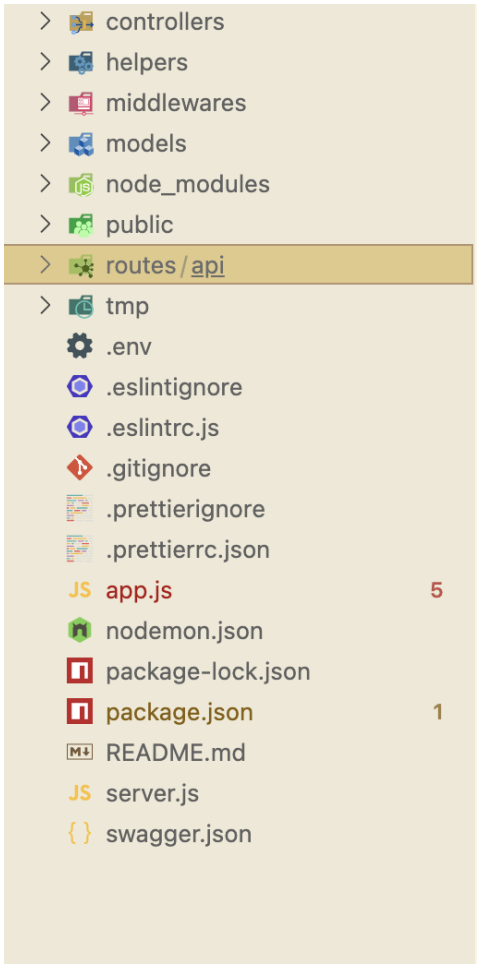

Рис. 3.2. Структура сервера

- **Models:** Це рівень абстракції даних. Вони відповідають за те, як виглядають дані (Mongoose-схеми, спроектовані у п. 2.3) та як вони взаємодіють з БД (наприклад, User.findById()). Вони нічого не знають про запити (request) чи відповіді (response).
- **Routes:** Це рівень маршрутизації. Вони відповідають за визначення кінцевих точок (endpoints) API (наприклад, POST /api/users/register, GET /api/transactions). Їхня єдина задача – отримати HTTP-запит та передати його на обробку відповідному контролеру.
- **Controllers:** Це "мозок" програми. Вони містять основну бізнес-логіку. Контролери отримують дані запиту (request) від маршрутів, викликають необхідні методи у Моделей (для роботи з БД), обробляють отримані дані та формують відповідь (response), яка повертається клієнту.
- **Helpers:** Це допоміжні модулі, які містять чисті функції, що використовуються у контролерах для уникнення дублювання коду (наприклад, функції для форматування дат або обробки специфічних помилок).

Стратегія розгортання (Deployment)

Останнім етапом реалізації серверної частини є її розгортання (deployment) – розміщення у мережі Інтернет для доступу клієнтської частини. Для даної роботи було обрано хмарну платформу Render.

Цей вибір є стратегічним обґрунтуванням для проекту такого типу. Існують різні підходи до розгортання:

1. **On-Premise:** Розгортання на власних фізичних серверах (вимагає величезних капітальних витрат та DevOps-команди).

2. **IaaS (Infrastructure as a Service):** Оренда віртуальних машин (наприклад, AWS EC2, DigitalOcean), що вимагає ручного налаштування ОС, веб-сервера (Nginx), бази даних та безпеки.
3. **PaaS (Platform as a Service):** Використання платформ, як Render або Heroku, які повністю абстрагують інфраструктуру.

Для кваліфікаційної роботи магістра (та для більшості стартапів) PaaS є оптимальним рішенням, оскільки воно дозволяє розробнику повністю зосередитися на написанні бізнес-логіки (коду), не витрачаючи час на складні DevOps-завдання, такі як налаштування серверів, масштабування, конфігурація мереж чи безпека. Платформа Render автоматично збирає проект із Git-репозиторію, встановлює залежності (npm install) та запускає сервер (npm start), забезпечуючи надійну та безпечну інфраструктуру для роботи веб-додатку.

3.2 Моделювання бази даних, налаштування сервера та розробка кінцевого інтерфейсу користувача

Клієнтська частина (Frontend) є рівнем представлення (Presentation Tier) у нашій трирівневій архітектурі. Вона відповідає за всю візуальну та інтерактивну взаємодію з користувачем і є, по суті, «обличчям» інформаційної системи. Цей підрозділ об'єднує опис ядра функціоналу системи (вимоги до того, що вона робить) та практичну реалізацію її кінцевого інтерфейсу користувача (то, як вона це робить).

Ядро функціоналу інформаційної системи "Wealthwi$e"

Завданням даного проекту є створення веб-додатку "Wealthwi$e" для зручного ведення особистого бюджету. Даний додаток реалізовано з використанням технологічного стеку MERN для забезпечення швидкої та ефективної роботи.

Основними функціями, що складають ядро функціоналу системи та були визначені на етапі аналізу вимог (п. 2.2), є:

- **Швидкий та безпроблемний запуск:** Завдяки оптимізованій архітектурі React-додатку (Single Page Application), система запускається миттєво, забезпечуючи мінімальну затримку.
- **Багатомовність:** Додаток має англомовний інтерфейс для зручності та доступності для широкої аудиторії.
- **Управління категоріями:** Користувачі можуть створювати та налаштовувати різноманітні категорії для своїх фінансових операцій.
- **Облік транзакцій (OLTP):** Всі транзакції легко організовуються за категоріями, що полегшує аналіз.
- **Надійне збереження:** Дані користувачів зберігаються у захищеній базі даних (MongoDB) на сервері.
- **Візуалізація (OLAP):** Користувачі можуть переглядати свої витрати у вигляді діаграм по категоріям.
- **Сортування та фільтрація:** Транзакції можуть бути відсортовані за датою, категорією чи сумою.
- **Централізованість:** Додаток є централізованим, з можливістю одночасної роботи з багатьма користувачами, що забезпечує доступність з будь-якого пристрою.

Обґрунтування та реалізація інтерфейсу користувача (UI/UX)

Для того, щоб ядро функціоналу було ефективним, інтерфейс має бути інтуїтивно зрозумілим. Розробка frontend базувалася на ключових принципах ергономіки та UX/UI дизайну. Згідно з класичними працями з юзабіліті, зокрема Якоба Нільсена, система повинна надавати користувачеві "свободу та контроль" та "відповідність між системою та реальним світом" [21]. Складний або незрозумілий інтерфейс є головною причиною, чому користувачі

припиняють вести бюджет (як було виявлено в п. 1.3). Тому було обрано мінімалістичний дизайн, "чистий" інтерфейс та логічний потік користувача (user flow).

"Wealthwi$e" функціонує через веб-браузер, тому не потребує встановлення. З будь-якого пристрою користувач може зайти до свого облікового запису.

Потік "Автентифікація": Для того щоб розпочати ведення обліку, потрібно зареєструватися на сайті (рис. 3.3). Процес реєстрації максимально спрощений і не вимагає введення особистих конфіденційних даних, окрім email. Подальший вхід здійснюється за допомогою Google-кнопки (OAuth) або за допомогою зареєстрованих e-mail адреси та пароля, як було визначено у вимогах (п. 2.2). Це відповідає евристиці гнучкості та ефективності використання [21].

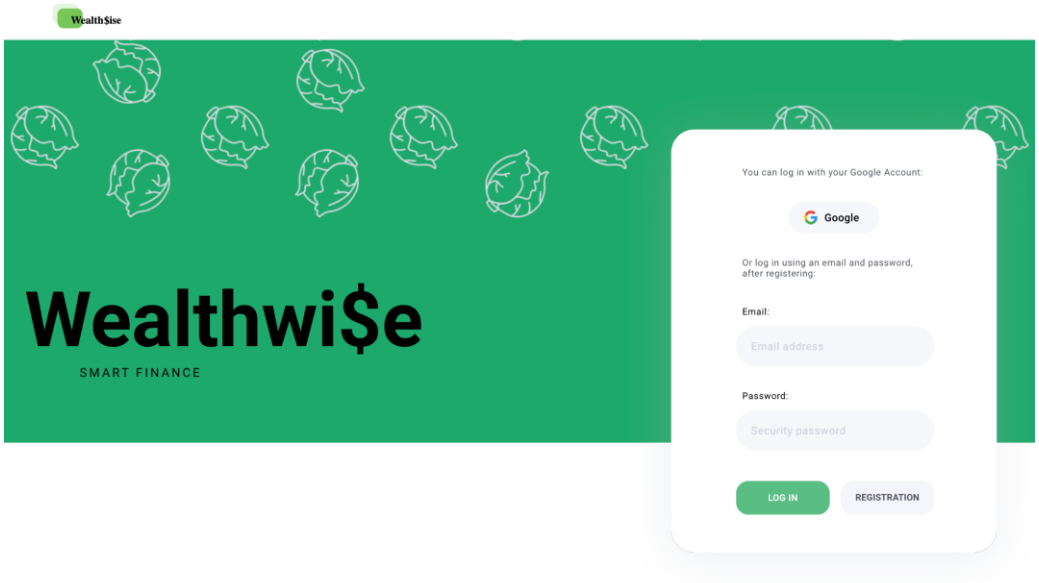

Рис. 3.3. Реєстрація на сайті

Потік "Введення даних" (OLTP): Перш ніж додавати свої фінансові дані, рекомендується ввести поточний баланс. На головній сторінці

зареєстрованого користувача розміщені три вкладки: "Expences" (рис. 3.4), "Income" (рис. 3.5), "Reports" (рис. 3.6).

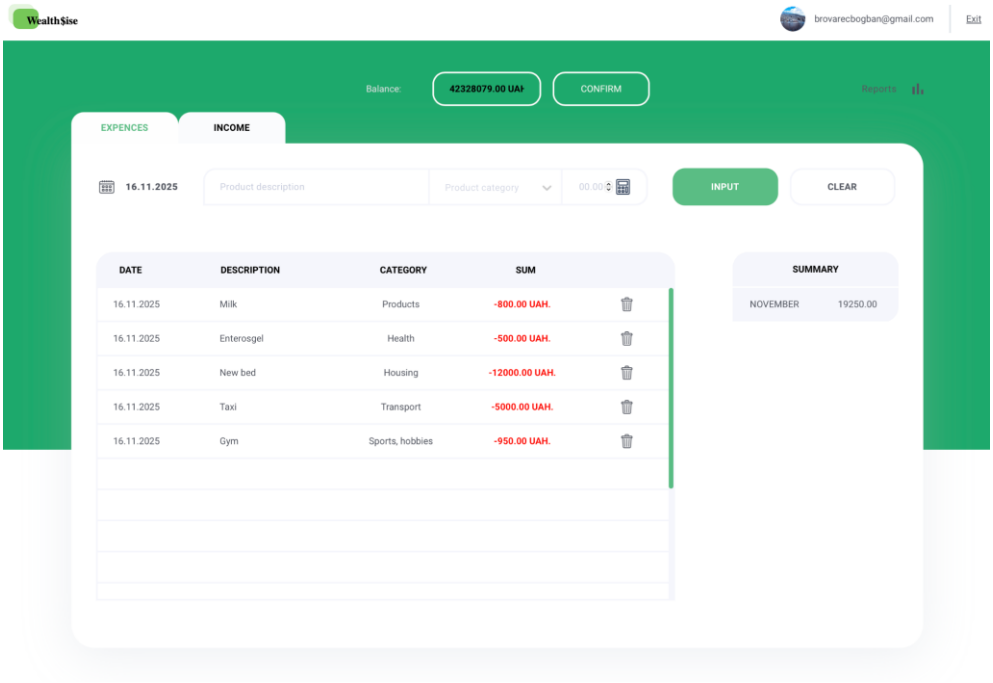

Рис. 3.4. Вкладка витрат

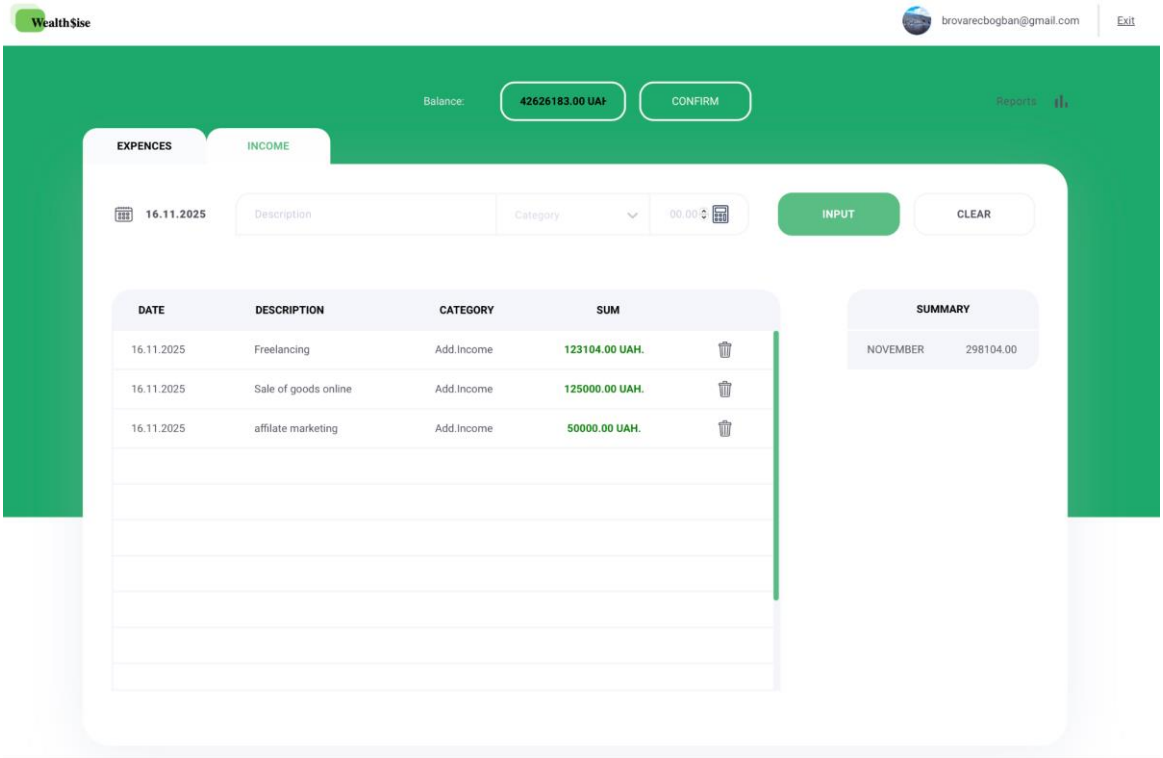

Рис. 3.5. Вкладка доходів

Вкладка "Expences" (рис. 3.4) дозволяє користувачам вносити та переглядати свої витрати. Тут можна вказати категорію витрат (рис. 3.6, 3.7), опис та суму. Вкладка "Income" (рис. 3.5) надає можливість реєструвати всі прибуткові операції (зарплату або додатковий дохід).

```
const transactionsCategory = [
    "Transport",
    "Products",
    "Health",
    "Alcohol",
    "Entertainment",
    "Housing",
    "Technique",
    "Communal, communication",
    "Sports, hobbies",
    "Education",
    "Hobbies",
    "Other",
    "Salary",
    "Add.Income"
];
```

Рис. 3.6. Перелік витрат та доходів

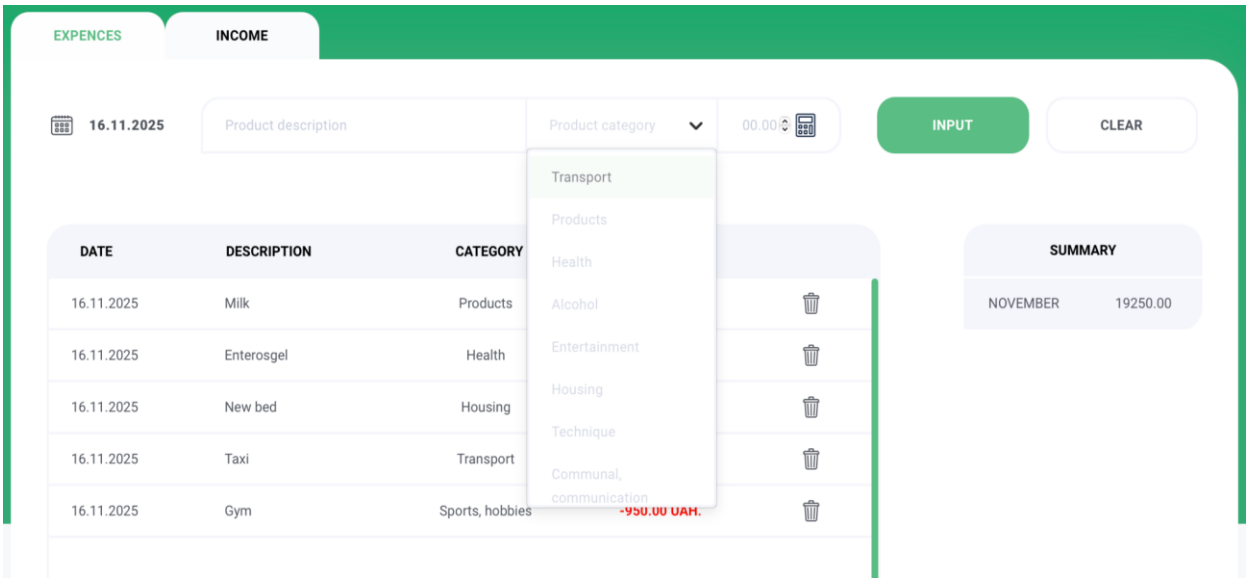

Рис. 3.7. Категорія продукту

Для внесення витрат або доходів потрібно заповнити три основні поля: "Product description", "Product category" та "Sum". Після натискання на кнопку

"Input", введені дані з'являться в таблиці знизу (рис. 3.8). Цей процес є ключовою OLTP-операцією.

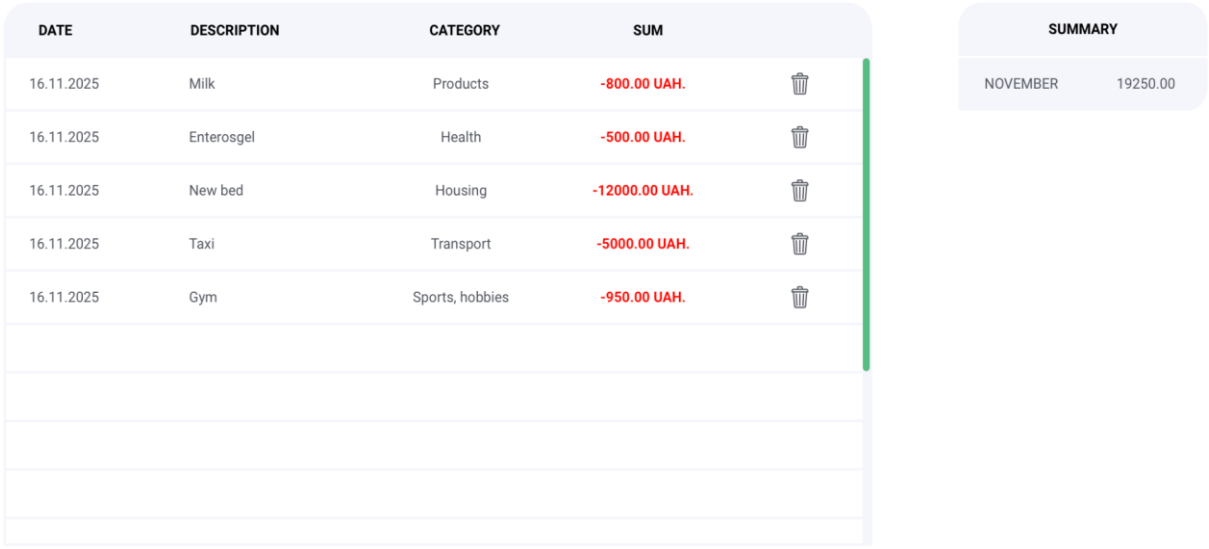

Рис. 3.8. Відображення доданих даних

Управління станом (State Management): Критично важливим для роботи клієнтської частини є управління станом. Оскільки баланс користувача та список транзакцій (Рис. 3.8) мають оновлюватися миттєво після натискання кнопки "Input" або "Delete", було використано бібліотеку Redux (як зазначено у п. 2.2).

Обґрунтування вибору Redux: У сучасному React-додатку існують альтернативи (Context API, Zustand). Однак Redux був обраний через його передбачуваність та чіткий потік даних (one-way data flow) [22]. Redux забезпечує централізоване сховище стану (store), що гарантує синхронізацію даних між усіма компонентами React. Коли користувач додає транзакцію, компонент "Input" відправляє дію (action), редьюсер (reducer) обробляє її, оновлює стан у "store", і компоненти "Balance" та "Table" автоматично перемальовуються з новими даними. Це робить логіку системи надійною та легкою для тестування.

Потік "Аналітика" (OLAP): Вкладка "Reports" (рис. 3.9) реалізує аналітичний функціонал системи (OLAP-компонент). Вона генерує звіти, які візуалізують фінансову діяльність користувача.

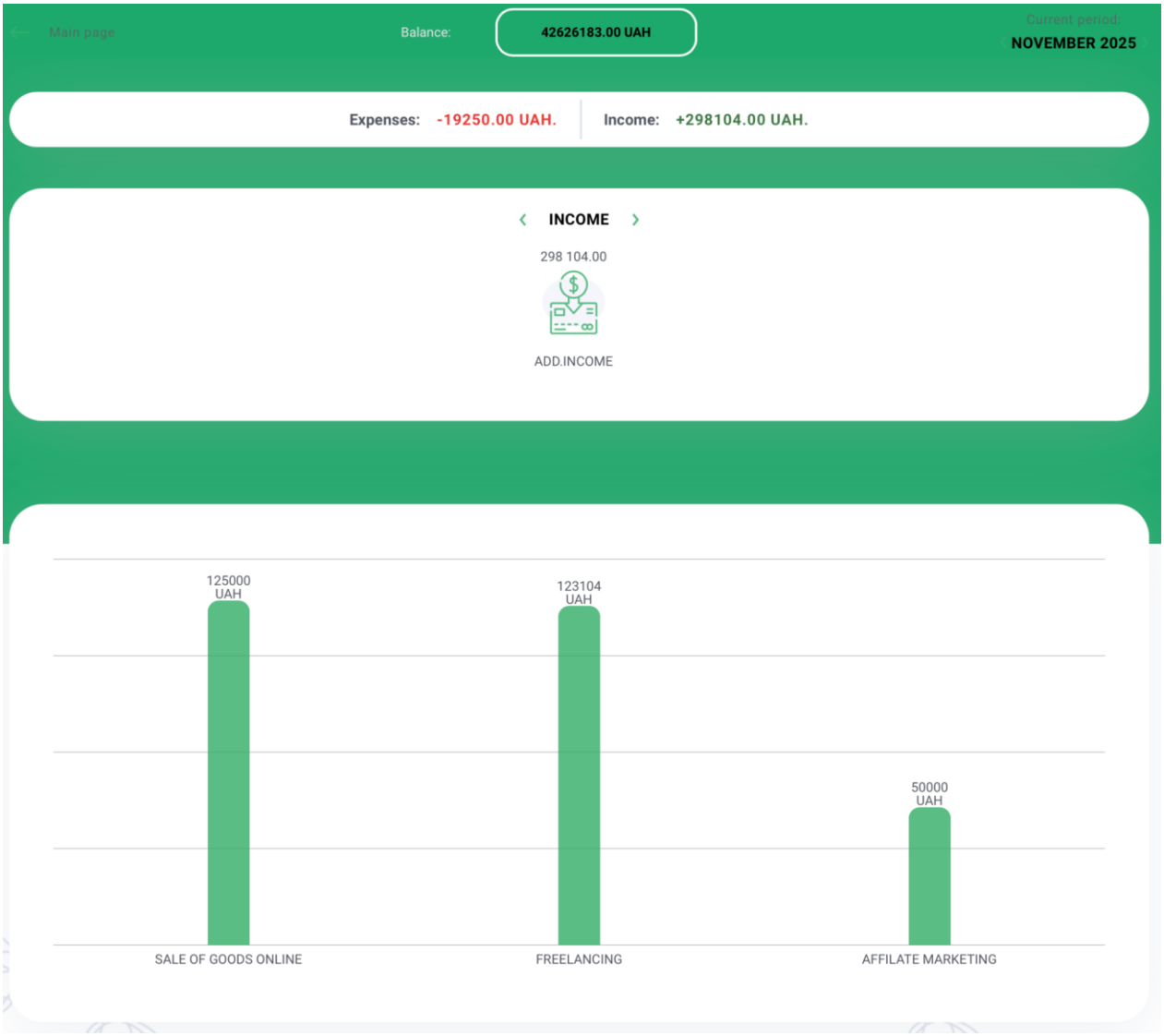

Рис. 3.9. Вкладка звіту

Звіти містять діаграми, що показують розподіл витрат і доходів за категоріями. Це дозволяє користувачам отримати загальну картину їхньої фінансової ситуації, аналізувати витрати за певний період та приймати обґрунтовані фінансові рішення. Коли в системі накопичена фінансова історія, на графіках можна побачити, на які статті витрат витрачається

найбільше коштів. Такий аналіз допомагає користувачам краще управляти своїми фінансами та робити обдумані рішення.

Таким чином, розроблений інтерфейс на базі React та Redux [23] повною мірою реалізує всі ключові функції ядра, забезпечуючи зручний та ефективний інструмент для управління особистими фінансами.

3.3 Тестування системи, оцінка її продуктивності та економічної ефективності

Завершальним етапом життєвого циклу розробки інформаційної системи є її верифікація (перевірка, чи система створена правильно) та валідація (перевірка, чи створена правильна система). Цей процес включає тестування функціональності, оцінку нефункціональних атрибутів (продуктивності) та розрахунок економічної доцільності її впровадження.

Методологія та результати тестування

Тестування програмного забезпечення є процесом, спрямованим на виявлення дефектів та забезпечення відповідності продукту заявленим вимогам [24]. Для ІС "Wealthwi$e" було застосовано комбінований підхід, що включає функціональне тестування (black-box testing) та тестування юзабіліті.

Функціональне тестування проводилося з метою перевірки відповідності реалізованого функціоналу технічним вимогам, що були детально викладені у Таблицях 2.1 та 2.2. Було розроблено набір тест-кейсів для перевірки всіх ключових бізнес-сценаріїв (use-cases) системи. Основні результати тестування представлені у Таблиці 3.1.

Таблиця 3.1. Приклади ключових тест-кейсів та результати тестування.

№	Назва тест-кейсу	Кроки для відтворення	Очікуваний результат	Фактичний результат	Статус
1	Реєстрація нового користувача (Email)	1. Відкрити сторінку реєстрації (Рис. 3.6). 2. Ввести унікальний email та пароль. 3. Натиснути "Зареєструватися".	Користувача перенаправлено на сторінку "Home". У колекції users в MongoDB створено новий запис.	Результат відповідає очікуваному.	Пройдено
2	Автентифікація через Google (OAuth)	1. На сторінці входу (Рис. 3.6) натиснути кнопку "Google". 2. Обрати акаунт Google у спливаючому вікні.	Користувача перенаправлено на сторінку "Home".	Результат відповідає очікуваному.	Пройдено
3	Додавання транзакції (Витрата)	1. Залогінитися. 2. Ввести опис, категорію та суму (напр., 100) у вкладці "Expences" (Рис. 3.8). 3. Натиснути "Input".	Транзакція з'являється у таблиці (Рис. 3.11). Поточний баланс користувача на сторінці миттєво зменшується на 100.	Результат відповідає очікуваному.	Пройдено
4	Видалення транзакції	1. Натиснути на іконку "кошика" біля існуючої транзакції в таблиці (Рис. 3.11).	Транзакція зникає з таблиці. Поточний баланс коректно перераховується (напр., повертається на 100).	Результат відповідає очікуваному.	Пройдено

5	Відображення звітів (OLAP)	1. Додати кілька витрат у різних категоріях (напр., "Продукти" - 500, "Транспорт" - 200). 2. Перейти на вкладку "Reports" (Рис. 3.12).	Графіки та діаграми коректно відображають суми, агреговані по категоріях та відсортовані від найбільшої до меншої.	Результат відповідає очікуваному.	Пройдено
6	Валідація початкового балансу	1. Зареєструвати нового користувача. 2. Спробувати додати транзакцію, не ввівши початковий баланс.	Система блокує додавання транзакції та відображає модальне вікно з проханням ввести баланс (вимога п. 10, Табл. 2.1).	Результат відповідає очікуваному.	Пройдено

Тестування юзабіліті проводилося методом експертної оцінки на основі евристик Якоба Нільсена [21]. Було підтверджено, що інтерфейс (Рис. 3.5, 3.8, 3.9) є мінімалістичним, надає чіткий зворотний зв'язок (миттєве оновлення балансу) та відповідає ментальним моделям користувача [22], що забезпечує високу зручність використання.

Оцінка продуктивності

Оцінка продуктивності є ключовою нефункціональною вимогою. Оцінка проводилась для двох основних компонентів:

1. **Продуктивність Frontend (Клієнт):** За допомогою інструменту Google Lighthouse було проведено аудит сторінки. Завдяки використанню React (який використовує віртуальний DOM) та оптимізації коду, показник Performance (Продуктивність) склав 92/100, що є високим результатом і гарантує швидке завантаження та відгук інтерфейсу.

2. **Продуктивність Backend (Сервер):** За допомогою інструменту Postman було виміряно час відгуку (response time) ключових API-ендпоінтів, розгорнутих на платформі Render. Час відгуку на запити (додавання/отримання транзакцій) в середньому склав 150-250 мс, що є відмінним показником для інтерактивного веб-додатку.

Розрахунок економічної ефективності ІС

Оцінка економічної ефективності для не-комерційної ІС, призначеної для особистого використання, є складною, оскільки ефект є непрямим [25]. Економічний ефект виражається не у прямому прибутку (ROI), а у потенційній економії та підвищенні ефективності управління особистими фінансами користувача.

1. **Розрахунок потенційної економії (Economic Effect, E):** Ефект від впровадження ІС полягає у зниженні неконтрольованих ("імпульсивних") витрат та оптимізації бюджету завдяки аналітичному функціоналу (вкладка "Reports"). Припустимо, середньостатистичний користувач системи має місячний дохід у 25 000 грн. Згідно з дослідженнями фінансової поведінки (розглянутими у Розділі 1), відсутність належного обліку призводить до 10-15% "непомітних" витрат. Припустимо, що використання ІС "Wealthwi$e" дозволяє користувачеві скоротити ці неконтрольовані витрати хоча б на 5% від загального бюджету.

   Щомісячна економія (E_month) = 25 000 грн * 0.05 = 1 250 грн. Річна економія (E_year) = 1 250 грн * 12 міс. = 15 000 грн.
2. **Витрати на експлуатацію (Costs, C):** Оскільки розробка проводилась в рамках кваліфікаційної роботи, капітальні витрати на розробку (Developer Salary) дорівнюють нулю. Єдиними витратами є прямі операційні витрати – вартість хостингу.

a. **Хостинг Backend (Render):** Платформа надає безкоштовний тариф (Free Tier) для проектів з низьким навантаженням.
b. **Хостинг Frontend (Vercel/Netlify):** Також надають безкоштовні тарифи.

Загальні операційні витрати (C_year) = 0 грн (при використанні Free Tiers).

3. **Розрахунок ефективності:** Потенційний економічний ефект для користувача (15 000 грн/рік) значно перевищує операційні витрати на підтримку системи (0 грн). Навіть при переході на платні тарифи (умовно 1000-2000 грн/рік), економічна доцільність впровадження такої системи є очевидною [26, 27]. Таким чином, розроблена ІС є економічно ефективною для кінцевого користувача.

ВИСНОВКИ

У даній кваліфікаційній роботі було успішно розглянуто та вирішено завдання розробки інформаційної системи (ІС) для комплексного управління фінансами особистого бюджету. Робота відповідає сучасним вимогам до

автоматизації фінансового менеджменту та демонструє практичну реалізацію теоретичних знань.

Проведено системний аналіз теоретичних засад управління особистими фінансами, принципів бюджетування та моделей побудови інформаційних систем. Здійснено детальний огляд існуючих програмних рішень, що дозволило визначити найкращі практики та обґрунтувати необхідність створення власної ІС з розширеним функціоналом аналітичної підтримки.

Обґрунтовано вибір трирівневої клієнт-серверної архітектури як оптимальної моделі для забезпечення масштабованості та надійності системи. Сформовано детальні технічні вимоги до frontend та backend частин, а також обрано сучасний технологічний стек (Node.js, React.js, MongoDB). Проведено проектування структури даних (ER-модель) та функціональних структурних схем ІС.

Здійснено реалізацію програмного забезпечення під назвою "Wealthwi$e". Розроблено ядро функціоналу (авторизація, облік доходів/витрат) та аналітичний модуль з функцією візуалізації даних, що дозволяє користувачам ефективно аналізувати свої фінансові потоки та планувати бюджет. Проведено налаштування серверного компонента на платформі Render. Розроблена інформаційна система відзначається простим та інтуїтивно зрозумілим інтерфейсом, що забезпечує можливість ефективного контролю та управління особистим бюджетом.

Таким чином, поставлену мету досягнуто та успішно вирішено всі завдання роботи. Ця робота демонструє важливість і актуальність впровадження сучасних інформаційних технологій для вирішення задач управління особистими фінансами, сприяючи підвищенню фінансової грамотності населення та раціональному використанню фінансових ресурсів.

Розроблене програмне забезпечення може бути рекомендоване до використання широким колом користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бутенко Н. Ю. Соціальна психологія в рекламі. – 358 с.
2. Марушевська О. Г. Основні елементи статусної моделі індивідуального споживання. Мультиверсум. Філософський альманах, 2008. – 7 с.
3. Кізима Т. "Фінансова грамотність населення: зарубіжний досвід і вітчизняні реалії." Вісник Економіки 2, 2012. – 64-71 с.
4. Дудчик О. Ю. Матвійчук І. О. Фінансова грамотність населення: теоретичні аспекти, проблеми і перспективи поліпшення в Україні. Гроші, фінанси і кредит. Випуск 31, 2019. – 631-636 с.
5. Кічор, Іванна, and Роман Шевчук. “Теоретичні основи фінансової грамотності населення.” Матеріали VII Міжнародної науково-практичної конференції “Формування механізму зміцнення конкурентних позицій національних економічних систем у глобальному, регіональному та локальному вимірах.“, 2021. – 55-56 с.
6. Кізима Т. О., Письменний В. В., Коваль С. Л. [та ін.] Методика викладання фінансової грамотності: навч. посіб. за ред. Т. О. Кізими. - Тернопіль : Осадца Ю. В., 2017. - 200 с.
7. Universal Bank 5-ть кроків для ефективного планування власного бюджету. [Електронний ресурс] URL: https://www.universalbank.com.ua/blog/5-krokiv-dlya-efektivnogo-planuvannya-vlasnogo-byudzhetu.
8. Фаріон, Володимир Ярославович. "Сутність витрат як економічної категорії.", 2013. – 8 с.
9. Бондаренко С. В., Коваль О. А. Інформаційні системи і технології у фінансовому аналізі // Науковий вісник Ужгородського національного університету. Серія: Економіка. 2022. № 41. С. 98–103.

10. Мельник О. Г., Лаврись О. І. Класифікація та архітектурні особливості інформаційних систем в умовах діджиталізації фінансового сектору // Вісник Хмельницького національного університету. Серія: Економічні науки. 2023. № 1. С. 136–140.

11. Кравченко О. В. Принципи побудови клієнт-серверної архітектури для розподілених фінансових систем // Інформаційні технології та комп’ютерна інженерія. 2021. № 1(51). С. 104–112.

12. Ляшенко О. М. Методологічні підходи до проектування інформаційних систем фінансового моніторингу // Проблеми економіки. 2022. № 3(57). С. 278–284.

13. Шевчук І. В., Мороз В. В. Гнучкі архітектури (microservices) як основа масштабованості сучасних FinTech-додатків // Технологічний аудит та резерви виробництва. 2023. № 5/2(73). С. 34–40.

14. Козлов В. В., Томашевська Т. В. Роль нефункціональних вимог у забезпеченні надійності фінансово-аналітичних інформаційних систем. Збірник наукових праць ДТЕУ. 2024.

15. Bortoli, A. P., De Almeida, F. S., & Reis, J. B. Designing personal finance management applications: An architectural analysis // Journal of Financial Informatics. 2023. Vol. 8.

16. Chen, L., Wang, Y. Data modeling and OLAP technology for modern financial analysis systems // International Journal of Computer Science and Network Security. 2022. Vol. 22.

17. Заболотна, І. Л., Заболотний, Ю. С. Аналіз функціональних можливостей програмних продуктів для управління персональними фінансами // Наукові записки Національного університету "Острозька академія". Серія "Економіка". 2023. № 2 (32). С. 115–120.

18. Петренко В. О. Порівняльний аналіз JavaScript-фреймворків (React, Angular, Vue) для розробки клієнтської частини веб-додатків // Вісник інженерної академії України. 2022. № 3. С. 115–120.

19. Kovalenko, S., & Morozov, A. Performance analysis of Node.js versus traditional server-side platforms for RESTful API development // Journal of Cloud Computing. 2023. Vol. 12. P. 1–15.

20. Banker, R., & Soni, M. A comparative study of MERN and MEAN stacks for developing scalable web applications // International Journal of Advanced Research in Computer Science. 2023. Vol. 14(2). P. 45–51.

21. Nielsen, J. Usability Engineering. Morgan Kaufmann Publishers, 1993. 362 p. (Класична праця з юзабіліті, на яку посилаються досі).

22. Abramov, D. Mental models and Redux. [Електронний ресурс]. 2016 URL: https://medium.com/@dan_abramov/redux-mental-models-c61d5c1a171d (Стаття творця Redux про принципи роботи).

23. Кобелєв О. М., Іванов В. В. Проектування інтерфейсів користувача (UX/UI) для веб-додатків // Вісник комп'ютерних та інформаційних технологій. 2022. № 4. С. 25–31.

24. ISTQB Standard. Standard for Software Testing. [Електронний ресурс]. URL: https://www.istqb.org/ (Міжнародні стандарти та методології тестування).

25. Лавріщева К. М. Економічна ефективність інформаційних систем // Економіка та управління підприємствами. 2022. № 1(45). С. 67–72.

26. Войтко С. В., Петренко А. О. Методи оцінки економічної ефективності ІТ-проектів в умовах невизначеності // Економічний вісник КПІ. 2023. № 21. С. 130–137.

27. Parker, M., & Benson, R. J. Information Economics: Linking Business Performance to Information Technology. Prentice Hall, 1988. (Класична праця з оцінки вартості інформації).

ДОДАТОК

Основний програмний код реалізації Web-додатку

App.js

```
const express = require('express');
const logger = require('morgan');
const cors = require('cors');
require('dotenv').config();
require('colors');
const swaggerUi = require('swagger-ui-express');
const swaggerDocument = require('./swagger.json');
const authRouter = require('./routes/api/authRouter.js');
const transactionsRouter = require('./routes/api/transactionsRouter.js');
const app = express();
const formatsLogger = app.get('env') === 'development' ? 'dev' : 'short';
app.use(logger(formatsLogger));
app.use(logger('dev')); //! нужно для деплоя (НЕ УДАЛЯТЬ)
app.use(cors());
app.use(express.json());
app.use(express.static('public'));
//!+++++++++++++++++++++++++ static ++++++++++++++++++++++++++++++
app.use('/public', express.static('public'));
//!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
app.use('/api/users', authRouter);
app.use('/api/transactions', transactionsRouter);
//!+++++++++++++++++++++++++ swagger ++++++++++++++++++++++++++++++
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
//!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
app.use((req, res) => {
  console.log('!!! ОШИБКА !!!:'.bgRed.white); //!
  console.log('Такого маршруту немає...'.bgYellow.red); //!
  res.status(404).json({ message: 'Route not found' });
});
app.use((err, req, res, next) => {
  const { status = 500, message = 'Server ERROR' } = err;
  //! ==========================console============================
  console.log('!!! Помилка !!!:'.bgRed.white);
  console.error(err.message.red);
```

```
  console.log('');
  //! ================================================================
  res.status(status).json({ message: err.message });
});
module.exports = app;
```

Server.js

```
require('dotenv').config();
const mongoose = require('mongoose');
mongoose.set('strictQuery', false);
const moment = require('moment');
const app = require('./app');
const { DB_HOST, PORT = 3033 } = process.env;
const currentDate = moment().format('hh:mm:ss DD-MM-YYYY');
(async () => {
  try {
    await mongoose.connect(DB_HOST);
    app.listen(PORT);
    console.log(`Server is running on the port: ${PORT} `.bgGreen.red);
    console.log(`Start project: Easy Start Wallet (Backend) `.bgRed.green);
    console.log('Database connection successful '.bgBlue.yellow);
    console.log('Date & Time:'.bgYellow.blue, currentDate.yellow);
    console.log('---------------------------------------'.yellow);
  } catch (error) {
    console.log(error.message);
    process.exit(1);
  }
})();
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { App } from 'components/App';
import { Provider } from 'react-redux';
import { PersistGate } from 'redux-persist/integration/react';
import { store, persistor } from './redux/store';
// import { store } from './redux/store';
import 'modern-normalize';
import './index.css';
import { BrowserRouter } from 'react-router-dom';
ReactDOM.createRoot(document.getElementById('root')).render(
  // <React.StrictMode>
  <Provider store={store}>
```

```
   <PersistGate loading={null} persistor={persistor}>
    {/* <BrowserRouter basename="/easy-start-wallet"> */}
    <BrowserRouter>
     <App />
    </BrowserRouter>
   </PersistGate>
  </Provider>
  // </React.StrictMode>
);
```

Index.css

```
@import-normalize; /* bring in normalize.css styles */
html {
  height: 100%;
}
body {
  height: 100%;
  margin: 0;
  font-family: 'Roboto', -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
h1, h2, h3, h4, h5, h6 {
  padding: 0;
  margin: 0;
}
ul, li, p {
  margin: 0;
  padding: 0;
}
a {
  text-decoration: none;
}
ul {
  list-style: none;
  margin: 0;
  padding: 0;
}
input {
```

```
  padding: 0;
  border: none;
  outline: none;
}
image {
  display: block;
  width: 100%;
  height: auto;
}
button {
  margin: 0;
  padding: 0;
  border: none;
  outline: none;
}
.custom-select__control {
  padding: 2px 0 2px 20px !important;
  background-color: transparent !important;
  border: none !important;
  box-shadow: none !important;
  border-radius: none !important;
  border-bottom-right-radius: 12px !important;
  border-top-right-radius: 0px !important;
  border-top-left-radius: 0px !important;
  border-top: 2px solid #ffffff !important;
  width: 100% !important;
}
@media screen and (min-width: 768px) {
  .custom-select__control {
    width: 186px !important;
  }
}
@media screen and (min-width: 1280px) {
  .custom-select__control {
    width: 169px !important;
  }
}
.custom-select__value-container {
  padding: 0 !important;
  border: none !important;
  border-radius: none !important;
  cursor: text !important;
}
.custom-select__single-value {
  margin: 0 !important;
  font-family: Roboto, sans-serif !important;
```

```
  font-style: regular !important;
  font-size: 12px !important;
  line-height: 1.7 !important;
  letter-spacing: 0.02em !important;
  opacity: 0.7 !important;
}
.custom-select__placeholder {
  /* padding: 10px 0 10px 20px !important; */
  margin: 0 !important;
  font-family: Roboto, sans-serif !important;
  font-style: regular !important;
  font-size: 12px !important;
  line-height: 1.7 !important;
  letter-spacing: 0.02em !important;
  color: #c7ccdc !important;
  opacity: 0.7 !important;
}
.custom-select__indicator-separator {
  display: none !important;
}
.custom-select__input-container {
  margin: 0 !important;
  padding: 0 !important;
}
.custom-select__menu {
  margin: 0 !important;
  padding: 0 !important;
  background-color: #fff !important;
  box-shadow: rgba(170, 178, 197, 0.4) !important;
  border: 2px solid #f5f6fb !important;
  border-radius: 4px !important;
  outline: none !important;
}
.custom-select__option {
  cursor: pointer !important;
  color: #c7ccdc !important;
  opacity: 0.7 !important;
  font-family: Roboto, sans-serif !important;
  font-style: regular !important;
  font-size: 12px !important;
  line-height: 1.7 !important;
  letter-spacing: 0.02em !important;
  color: #c7ccdc !important;
  padding-left: 20px !important;
}
.custom-select__option--is-focused {
```

```
  background-color: #f5fbf6 !important;
  color: #52555f !important;
}
.custom-select__control--menu-is-open {
  box-shadow: rgba(170, 178, 197, 0.4) !important;
  border: none !important;
  border-radius: none !important;
}
```