

**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Інформаційна система для ведення особистого бюджету

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-22-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за
галузями)»


(підпис)

Пословського Євгена
Вікторовича
(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Філоненко Михайло
Миколайович, канд. фіз.-мат.
наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Ніжегородцев Владислав
Олександрович, канд. пед.
наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних
технологій і систем протокол від «12» травня 2026 №20

Завідувач кафедри


(підпис)

Філоненко Михайло
Миколайович, канд. фіз.-мат.
наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Ірпінь – 2026

Реєстрація № 62

28.05.2026
(дата)


(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

12.05.2026
(дата)


(підпис)

Результат захисту

4 (добре)
(оцінка)

09.06.2026
(дата)

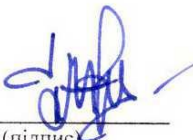

(підпис)

Голова комісії:


(підпис)

О.С. Дюбанов

Члени комісії:


(підпис)

М.М. Філоненко


(підпис)

В.О. Ніжегородцев


(підпис)

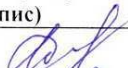
А.А. Омельчук

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	21		09.06.2026 Дюбанов О.С. 
2. Дотримання вимог щодо оформлення	від 0 до 10	7		Філоненко М.М.  Ніжегородцев В.О. 
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Омельчук А.А.  Суховерхівка Л.П. 
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	36		

Оцінювання кваліфікаційної роботи науковим керівником – 35 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

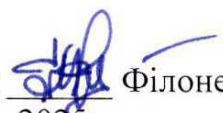
Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
71	C	4 (добре)	09.06.2026	Дюбанов О.С.  Філоненко М.М.  Ніжегородцев В.О.  Омельчук А.А.  Суховерхівка Л.П. 

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую

Завідувач кафедри  Філоненко М.М.

«01» грудня 2025

Завдання на кваліфікаційну роботу

здобувачу вищої освіти

Пословському Євгену Вікторовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Інформаційна система для ведення особистого бюджету»

Затверджена наказом ректора від «28» листопада 2025 № 1526

2. Строк здачі закінченої роботи «08» травня 2026

3. Мета роботи Проектування та програмна реалізація інформаційної системи (веб-додатка) для автоматизації обліку, аналізу та планування особистих фінансів користувача

Об'єкт дослідження. Процес управління, обліку та планування особистими фінансовими ресурсами.

Предмет дослідження. Методи, алгоритми та сучасні інформаційні технології розробки веб-додатків для систем контролю особистого бюджету.

Орієнтовна база дослідження. Сучасні платформи та інструменти веб-розробки (Next.js, React, Prisma ORM, SQLite), а також існуючі на ринку програмні аналоги систем ведення бюджету (Mint, YNAB, Spendee, The Wallet).

4. Вихідні дані до роботи. Технічна документація використаних фреймворків та бібліотек (Next.js, React, Prisma), наукові публікації з тематики фінансової грамотності, нормативно-правові акти із захисту персональних даних, результати системного аналізу існуючих програмних рішень.

5. Перелік графічного матеріалу. 8 рисунків, 3 таблиці, 3 додатки.

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

ВСТУП

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ВЕДЕННЯ ОСОБИСТОГО БЮДЖЕТУ

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 30.10.2025	До 30.10.2025
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.12.2025	26.12.2025
3	II розділ (аналіз експериментальної частини про методики досліджень)	18.02.2026	18.02.2026
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	16.04.2026	16.04.2026
5	Висновки	01.05.2026	01.05.2026
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	07.05.2026	07.05.2026
7	Отримання відгуку керівника	29.05.2026	29.05.2026
8	Отримання зовнішньої рецензії	01.06.2026	01.06.2026

8. Дата видачі завдання «01» грудня 2025

9. Керівник кваліфікаційної роботи


(підпис) Філоненко М.М.
(прізвище, ініціали)

10. Керівник освітньо-професійної програми


(підпис) Ніжегородцев В.О.
(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти


(підпис) Пословський Є.В.
(прізвище, ініціали)

АНОТАЦІЯ

Інформаційна система для ведення особистого бюджету

Мета роботи — проєктування та програмна реалізація веб-додатка для автоматизації обліку, аналізу та планування особистих фінансів користувача.

Завданнями роботи є:

- Аналіз предметної області, аналогів та формування вимог до системи.
- Проєктування інфологічної моделі бази даних та алгоритмів обробки мультивалютних операцій.
- Розробка серверної логіки, адаптивного користувацького інтерфейсу та тестування системи.

Об'єкт дослідження — процес управління, обліку та планування особистими фінансовими ресурсами. Предмет дослідження — методи та інформаційні технології розробки веб-додатків для контролю бюджету.

У результаті роботи створено комплексний веб-додаток, який поєднує інтуїтивно зрозумілий інтерфейс для щоденного обліку, модуль цілепокладання та інтерактивну аналітичну панель.

Одержані результати можуть бути використані фізичними особами для зручного контролю витрат, візуального аналізу фінансової поведінки та ефективного накопичення коштів.

Кваліфікаційна робота містить 63 сторінок, 3 таблиці, 8 рисунків, список використаних джерел із 40 найменувань, 3 додатки.

Ключові слова: особистий бюджет, фінансова грамотність, веб-додаток, облік фінансів, Next.js, React, Prisma ORM, мультивалютність.

Рік виконання кваліфікаційної роботи: 2026

Рік захисту роботи: 2026

ABSTRACT

Information system for managing a personal budget

The purpose of the work is the design and software implementation of a web application for automating the tracking, analysis, and planning of a user's personal finances.

The tasks of the work are:

- Analysis of the subject area, existing analogues, and formulation of system requirements.
- Design of the logical database model and algorithms for processing multi-currency transactions.
- Development of the server logic, adaptive user interface, and system testing.

The object of research is the process of managing, tracking, and planning personal financial resources. The subject of research is the methods and information technologies for developing web applications for budget control.

As a result of the work, a comprehensive web application was created that combines an intuitive interface for daily tracking, a goal-setting module, and an interactive analytical dashboard.

The obtained results can be used by individuals for convenient expense control, visual analysis of financial behavior, and effective funds accumulation.

The qualification paper contains 63 pages, 3 tables, 8 figures, a list of references with 40 items, and 3 appendices.

Keywords: personal budget, financial literacy, web application, financial tracking, Next.js, React, Prisma ORM, multi-currency.

Year of completion of the qualification paper: 2026

Year of defense of the work: 2026

ЗМІСТ

ВСТУП	2
РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ.....	4
1.1. Дослідження проблем планування бюджету та аналіз ринку програмних рішень	4
1.2. Обґрунтування функціональних вимог та вибір методології розробки системи	13
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ	21
2.1. Моделювання логіки роботи та опис математичного апарату обробки даних	21
2.2. Проєктування бази даних та розробка концепції користувацького інтерфейсу	29
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ВЕДЕННЯ ОСОБИСТОГО БЮДЖЕТУ	40
ВИСНОВКИ.....	44
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	46
ДОДАТКИ.....	49

ВСТУП

Актуальність теми. У сучасних економічних умовах питання фінансової грамотності та ефективного управління особистим капіталом набуває критичного значення для кожної людини [3, 4]. Відсутність систематичного контролю за доходами та витратами часто призводить до нераціонального використання коштів, неможливості формування заощаджень та виникнення боргових зобов'язань [30, 40]. Незважаючи на наявність на ринку великої кількості мобільних та веб-додатків для ведення бюджету, багато з них є перевантаженими складним бухгалтерським інструментарієм, мають обмежений функціонал у безкоштовних версіях або не забезпечують гнучкої інтеграції щоденного обліку з довгостроковим фінансовим плануванням (цілепокладанням). З огляду на це, розробка сучасної, швидкої та інтуїтивно зрозумілої інформаційної системи для ведення особистого бюджету є актуальним та практично значущим завданням.

Мета дослідження полягає у проектуванні та програмній реалізації інформаційної системи (веб-додатка) для автоматизації обліку, аналізу та планування особистих фінансів користувача.

Для досягнення поставленої мети було сформульовано та вирішено такі **основні завдання**:

- 1.Провести системний аналіз предметної області, дослідити існуючі програмні аналоги та виявити їхні недоліки.
- 2.Сформулювати функціональні та нефункціональні вимоги до розроблюваної системи.
- 3.Обґрунтувати вибір технологічного стеку та методології розробки програмного забезпечення.
- 4.Спроектувати інфологічну модель бази даних та математичний апарат для обробки мультивалютних фінансових операцій.

5.Розробити логіку серверної частини та створити адаптивний користувацький інтерфейс.

6.Провести тестування працездатності системи та перевірити її на відповідність поставленим вимогам.

Об'єктом дослідження є процес управління, обліку та планування особистими фінансовими ресурсами.

Предметом дослідження є методи, алгоритми та сучасні інформаційні технології розробки веб-додатків для систем контролю особистого бюджету.

Методи дослідження. Під час виконання роботи використовувалися методи системного аналізу (для дослідження предметної області та аналогів), методи об'єктно-орієнтованого проєктування та реляційної алгебри (для розробки архітектури бази даних), а також математичні методи агрегації та нормалізації даних (для побудови аналітичних звітів).

Практичне значення одержаних результатів полягає у створенні готового до використання програмного продукту (веб-додатка). Розроблена система може використовуватися фізичними особами для щоденного контролю витрат, аналізу власної фінансової поведінки за допомогою інтерактивних графіків та ефективного накопичення коштів за допомогою модуля фінансових цілей. Використання сучасного стеку технологій (Next.js, Prisma, React) дозволяє легко масштабувати додаток та впроваджувати новий функціонал у майбутньому.

Структура та обсяг роботи. Дипломна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Робота містить рисунки, таблиці та лістинги програмного коду.

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

1.1. Дослідження проблем планування бюджету та аналіз ринку програмних рішень

У сучасних умовах глобальної економічної нестабільності та постійного зростання споживчих можливостей питання ефективного управління особистими фінансами стає одним із найважливіших аспектів життєдіяльності кожної людини. Фінансова грамотність більше не є прерогативою фахівців банківського сектору чи інвестиційних аналітиків, вона перетворилася на базову навичку, необхідну для забезпечення стабільного майбутнього та досягнення життєвих цілей [30]. Проблема раціонального розподілу доходів та контролю витрат стоїть перед більшістю громадян, незалежно від рівня їхнього заробітку. Відсутність чіткої системи обліку власних коштів часто призводить до виникнення ситуацій, коли людина не може пояснити напрямки використання значної частини свого бюджету, що унеможливорює формування заощаджень або інвестування. Саме тому розробка інформаційної системи, яка дозволила б автоматизувати цей процес та надати користувачеві наочну картину його фінансового стану, є актуальною науково-технічною задачею.

Основною проблемою в контексті планування бюджету є відсутність регулярності та систематичності в діях користувачів. Традиційні методи, такі як запис витрат у паперові блокноти або використання простих електронних таблиць, часто виявляються неефективними через значні часові витрати та високу ймовірність помилок при ручному введенні даних. Більше того, такі методи не забезпечують оперативного аналізу та не дають змоги швидко оцінити динаміку змін у фінансовій поведінці. Користувачі часто стикаються з психологічним бар'єром, коли процес ведення обліку стає занадто складним або рутинним, що зрештою призводить до повної відмови від бюджетування. Крім того, існує проблема фрагментарності даних — доходи та витрати можуть

здійснюватися в різних валютах, через різні банківські рахунки або готівкою, що ще більше ускладнює формування цілісної картини.

Стейкхолдерами проектованої системи виступають кілька груп осіб та організацій. Першою і найважливішою групою є кінцеві користувачі – фізичні особи, які прагнуть оптимізувати свої фінансові потоки. Серед них можна виділити підгрупи з різними потребами – молодь, яка лише починає самостійне фінансове життя і потребує інструментів для формування правильних звичок, професіонали, що працюють на фрілансі та мають нерегулярні доходи, а також сім'ї, які планують великі спільні покупки. Іншим важливим стейкхолдером є розробник системи, для якого метою є створення масштабованого, безпечного та конкурентоспроможного програмного продукту. У перспективі зацікавленими сторонами можуть бути також фінансові консультанти, які могли б використовувати аналітичні дані системи для надання більш точних рекомендацій своїм клієнтам, забезпечуючи при цьому повну конфіденційність та захист персональних даних.

Окремим важливим чинником, що визначає ефективність сучасних систем управління фінансами, є якість та глибина візуалізації даних [29]. Сучасний користувач очікує від програми не просто переліку зафіксованих операцій, а повноцінного аналітичного звіту, який дозволяє миттєво оцінити фінансовий стан без необхідності самостійного проведення складних розрахунків. Відсутність інтегрованих модулів графічного відображення витрат у багатьох існуючих рішеннях або їх надмірна складність часто стають причинами низької залученості користувачів. Візуалізація у формі динамічних графіків та структурованих звітів дозволяє виявити «непомітні» статті витрат, які зазвичай ігноруються при ручному обліку [12]. Саме тому при розробці нової інформаційної системи особливий акцент має бути зроблений на створенні зрозумілих інструментів аналітики, що дозволяють здійснювати декомпозицію витрат за категоріями, часовими періодами та пріоритетністю.

Не менш значущим аспектом є інтеграція функцій нагадування та планування майбутніх платежів безпосередньо в інтерфейс ведення бюджету. Більшість існуючих на ринку аналогів фокусуються виключно на фіксації фактичних подій, залишаючи поза увагою проактивне планування. Це створює розрив між поточним станом рахунку та майбутніми фінансовими зобов'язаннями користувача. Впровадження системи нагадувань про важливі події, такі як орендна плата, комунальні послуги або терміни погашення кредитів, дозволяє мінімізувати ризики виникнення заборгованостей та штрафних санкцій. Більше того, поєднання механізму нагадувань із модулем відстеження фінансових цілей створює замкнений цикл управління капіталом – від фіксації витрат до стратегічного накопичення коштів на великі покупки чи інвестиції.

Завершуючи аналіз програмних рішень, слід зауважити, що на ринку спостерігається певний дефіцит систем, які б забезпечували високий рівень безпеки та швидкодії при збереженні максимальної простоти інтерфейсу. Багато розширених систем орієнтовані на професійне ведення бухгалтерії та є занадто складними для пересічного користувача, тоді як прості мобільні застосунки часто не мають повноцінної веб-версії або не гарантують надійного захисту даних при взаємодії з базою даних. Таким чином, існує чітко визначена ніша для розробки багаторівневої веб-системи, яка базується на сучасних технологіях обробки даних та забезпечує повний цикл контролю за особистими фінансами – від реєстрації користувача та налаштування персонального профілю до отримання комплексної аналітики у реальному часі.

Процес дослідження проблем планування бюджету вимагає глибшого аналізу не лише технічних, а й методологічних аспектів, які зазвичай покладаються в основу таких систем. Більшість потенційних користувачів зазнає невдачі у веденні фінансів через те, що існуючі програмні продукти не пропонують гнучкості у виборі стратегії бюджетування. Наприклад, класичні

підходи, такі як правило «50/30/20» (розподіл на потреби, бажання та заощадження) або метод «конвертів», потребують специфічних алгоритмів розрахунку лімітів у реальному часі. Багато сучасних рішень змушують користувача підлаштовуватися під жорстку структуру категорій, що не враховує індивідуальні особливості споживання. Це створює ситуацію, за якої програмне забезпечення стає додатковим тягарем, а не помічником. Тому при проектуванні нової системи критично важливо закласти можливість адаптивного налаштування бюджетних планів, що дозволить користувачеві самостійно визначати пріоритетність витрат та динамічно змінювати їх залежно від поточної економічної ситуації.

Для глибокого розуміння недоліків існуючих рішень та формування точних вимог до власного продукту, доцільно структурувати виявлені під час аналізу ринку аспекти та розглянути їх на прикладі популярних фінансових додатків. Кожен із наведених нижче блоків ілюструє спробу подолання конкретних викликів, що дозволяє виокремити критичні переваги та недоліки для подальшого проектування власної системи.

Технічна складність автоматизації та рутинність введення даних

Суттєвим бар'єром є технічна складність взаємодії з банківськими даними та відсутність автоматизації. Хоча ручне введення транзакцій сприяє кращому усвідомленню витрат, воно часто стає причиною припинення ведення бюджету через велику кількість дрібних операцій. Аналіз ринку показує, що найбільш успішні західні аналоги, такі як Mint від компанії Intuit, здобули популярність саме завдяки глибокій інтеграції з банківськими рахунками. Проте для багатьох користувачів, особливо в умовах специфічного фінансового ринку, пряма синхронізація з банками може бути недоступною або сприйматися як ризик для безпеки. Це зумовлює необхідність розробки ефективного API-інтерфейсу, який

би дозволяв швидко та безпечно імпортувати дані або надавав максимально зручні інструменти для їх швидкої класифікації вручну [36].

Модель банківської інтеграції та проблема універсальності

Важливим аспектом адаптації фінансових інструментів до потреб сучасного користувача є можливість інтеграції з популярними регіональними банками. Приклад інтерфейсу додатку «The Wallet», який орієнтований на структуроване подання інформації та пропонує інструменти автоматичного імпорту винесено до Додатку А (рисунок А.1)

Як видно, система має логічно структуроване бічне меню з доступом до розширеної аналітики (статистика, звіти) та чітко виділені інформаційні блоки поточного і запланованого балансу. Особливістю платформи є пряма пропозиція налаштувати автоматичний імпорт даних з конкретного популярного банку (в даному випадку - Монобанку). Хоча така глибока інтеграція значно спрощує облік для клієнтів цієї установи, вона вказує на проблему універсальності: користувачі інших банківських сервісів або ті, хто віддає перевагу розрахункам готівкою, можуть не отримати всіх переваг автоматизації. Це формує додаткову вимогу до розроблюваної системи: вона повинна мати настільки зручний та швидкий механізм ручного введення транзакцій (або універсального імпорту), щоб відсутність прямої прив'язки до конкретного банку не знижувала загальну ефективність ведення бюджету.

Вимоги до аналітичного блоку та прогнозних моделей

Аналізуючи потреби стейкхолдерів, варто детальніше зупинитися на вимогах до аналітичного блоку. Крім звичайного відображення поточного балансу, користувачі потребують прогнозних моделей. Наприклад, система має не просто констатувати факт витрати коштів, а й на основі ретроспективних даних попереджати про можливий дефіцит бюджету наприкінці місяця. Така проактивна аналітика є ключовою відмінністю між простою «базою даних

витрат» та інтелектуальною інформаційною системою підтримки прийняття фінансових рішень. Дослідження показує, що наявність функцій прогнозування та автоматичного відстеження прогресу в досягненні фінансових цілей підвищує рівень фінансової дисципліни користувачів на 30-40%.

Аналіз конкурентного середовища та функціональних сценаріїв

Якщо розглядати конкурентне середовище далі, варто згадати додаток PocketGuard, який фокусується на концепції «грошей на кишенькові витрати» після вирахування всіх обов'язкових платежів та заощаджень. Цей підхід є надзвичайно ефективним для боротьби з імпульсивними покупками, проте він часто не враховує складні сценарії, такі як наявність боргів або необхідність формування складних інвестиційних портфелів. З іншого боку, сервіс Toshl Finance пропонує цікавий ігровий інтерфейс (гейміфікацію), що залучає молодшу аудиторію, але його аналітичні можливості у безкоштовній версії є досить обмеженими. Таким чином, існує потреба у створенні системи, яка б поєднувала простоту PocketGuard із візуальною привабливістю сучасних веб-інтерфейсів та глибиною аналітики, притаманною складнішим фінансовим інструментам, забезпечуючи при цьому надійний захист даних через сучасні протоколи авторизації.

Питання безпеки та довіри користувачів

У контексті аналізу ринку та дослідження проблем планування бюджету неможливо оминати питання безпеки та довіри користувачів до програмних продуктів. Сучасні споживачі стають дедалі більш обізнаними щодо ризиків витоку персональної інформації, особливо коли йдеться про деталізовану історію фінансових операцій, структуру доходів та обсяги цільових накопичень. Багато існуючих рішень, особливо безкоштовні мобільні додатки невідомих розробників, викликають обґрунтовані побоювання щодо того, як саме обробляються та кому передаються зібрані дані. Це створює значний

психологічний бар'єр для залучення нових стейкхолдерів. Отже, ринок гостро потребує рішень, які не лише надають зручний інструментарій, але й гарантують прозорість алгоритмів та надійний захист даних через сучасні механізми авторизації, реєстрації та безпечного керування сесіями користувачів [27, 32]. Проектована система має з самого початку будуватися з урахуванням цих вимог, ізолюючи профілі користувачів та забезпечуючи конфіденційність кожної транзакції.

Баланс між функціональністю та мінімалізмом

Ще однією вагомою проблемою, виявленою під час аналізу предметної області, є так зване інформаційне перевантаження, яким страждають багато комплексних фінансових платформ. Намагаючись задовольнити потреби абсолютно всіх категорій користувачів, розробники часто перетворюють інтерфейс на складну панель управління з десятками вкладок, незрозумілими бухгалтерськими термінами та складними налаштуваннями звітності. Для середньостатистичної людини, яка просто хоче зрозуміти, куди поділася її зарплата та скільки вона може відкласти на відпустку, такий рівень складності є надлишковим. Це призводить до парадоксальної ситуації: інструмент, створений для спрощення життя, вимагає від користувача тривалого навчання. Тому критично важливим аспектом є дотримання балансу між функціональністю та мінімалізмом. Розробка веб-додатку повинна спиратися на принципи інтуїтивно зрозумілого дизайну [22], де основні операції, додавання доходу чи витрати, виконуються в один-два кліки, а складніша аналітика винесена в окремі, логічно структуровані розділи.

Інформаційне перевантаження на прикладі системи YNAB

Для наочної ілюстрації проблеми інформаційного перевантаження доцільно розглянути користувацький інтерфейс одного з найвідоміших на ринку

рішень - системи YNAB (You Need A Budget). Типовий вигляд робочої панелі цього додатку винесено до Додатку А (рисунок А.2).

Як видно, незважаючи на потужний функціонал та деталізовану структуру категорій, інтерфейс виглядає перевантаженим текстовою інформацією та багаторівневими меню. Для середньостатистичної людини, яка прагне швидко зафіксувати транзакцію чи перевірити залишок коштів, такий рівень складності є надлишковим. Це підтверджує необхідність створення більш мінімалістичного та інтуїтивно зрозумілого веб-додатку, де ключові функції доступні в один-два кліки.

Функціональні обмеження при візуальному спрощенні Spendee

На противагу перевантаженим аналітичним системам, деякі розробники обирають шлях візуального спрощення. Показовим прикладом такого підходу є веб-версія популярного сервісу Spendee. Зображення базової панелі управління цього додатку, яка зустрічає користувача винесено до Додатку А (рисунок А.3)

Інтерфейс Spendee є значно "чистішим" та містить лише базовий огляд загального балансу, сумарних доходів і витрат за період. Проте, такий початковий мінімалізм часто супроводжується функціональними обмеженнями. Наприклад, система одразу акцентує увагу на необхідності створення нових гаманців або підключення банківських рахунків, що у більшості подібних сервісів є платною опцією. Обмеження базового функціоналу створює штучні бар'єри для користувача, який прагне вести детальний облік без придбання преміум-підписок. Це свідчить про те, що ідеальна розроблювана інформаційна система повинна знаходити золоту середину: мати такий самий лаконічний та інтуїтивно зрозумілий стартовий інтерфейс, але водночас надавати повноцінний доступ до інструментів аналітики, цілепокладання та керування транзакціями без прихованих обмежень.

Інтеграція цілепокладання з щоденним обліком

Аналізуючи нішу цілепокладання, варто зазначити, що багато існуючих додатків розглядають фінансові цілі ізольовано від щоденних витрат. Наприклад, користувач може створити мету "Купівля автомобіля", але система не показує, як кожна випита чашка кави або незапланована покупка віддаляє його від цієї мети. Відсутність такого наочного зв'язку знижує мотивацію до заощаджень. Ефективна інформаційна система має об'єднувати модуль постановки цілей з модулем щоденного обліку в єдину екосистему. Динамічний перерахунок прогресу досягнення мети на основі поточних витрат, підкріплений системою автоматичних нагадувань та сповіщень, є тією конкурентною перевагою, якої бракує багатьом продуктам на ринку.

Технологічна доступність та кросплатформність додатків

Варто звернути увагу на технологічну доступність додатків. Ринок переповнений рішеннями, які існують виключно у вигляді мобільних застосунків для конкретних операційних систем (iOS або Android). Це створює незручності для користувачів, які віддають перевагу комплексному аналізу своїх фінансів на великому екрані персонального комп'ютера або використовують різні пристрої вдома та на роботі. Відсутність повноцінної веб-версії або розрив у функціоналі між мобільним додатком та веб-порталом обмежує свободу дій стейкхолдерів. Орієнтація на створення повноцінного веб-додатку з адаптивним інтерфейсом вирішує цю проблему, забезпечуючи кросплатформність та безперебійний доступ до особистого кабінету з будь-якого пристрою, що має сучасний веб-браузер.

Висновки щодо аналізу предметної області

Підсумовуючи результати дослідження предметної області та аналізу ринку, можна сформулювати чіткий перелік недоліків існуючих аналогів, які необхідно усунути в проєктованій системі. До них належать - перевантаженість

користувацького інтерфейсу, висока вартість використання повноцінного функціоналу, слабка інтеграція цілепокладання з щоденним бюджетом, відсутність гнучкої системи нагадувань та обмежена кросплатформність. Усвідомлення цих проблем формує міцне підґрунтя для наступного етапу проектування - переходу від теоретичних досліджень до чіткого формулювання функціональних вимог та вибору оптимальних інструментів для їх практичної реалізації.

1.2. Обґрунтування функціональних вимог та вибір методології розробки системи

На основі проведеного в попередньому підрозділі аналізу предметної області та виявлених недоліків існуючих рішень виникає необхідність у чіткому формулюванні функціональних вимог до проектованої інформаційної системи. Функціональні вимоги визначають базову поведінку системи, описуючи те, як вона повинна реагувати на дії користувача та які саме завдання має виконувати для задоволення його потреб. Оскільки розроблюваний продукт є веб-додатком, орієнтованим на широке коло користувачів, архітектура та набір функцій повинні бути збалансованими, забезпечуючи високу продуктивність, безпеку даних та інтуїтивно зрозумілу взаємодію [1]. Процес обґрунтування вимог полягає у трансформації загальних потреб стейкхолдерів у конкретні технічні завдання, які будуть реалізовані на рівнях представлення, бізнес-логіки та роботи з даними.

Першочерговим та критично важливим блоком функціональних вимог є забезпечення надійної підсистеми управління доступом та ідентифікації. Система повинна надавати можливості для безпечної реєстрації нових користувачів, їх подальшої авторизації та коректного завершення сеансу (виходу з облікового запису). З огляду на конфіденційний характер фінансової інформації, процеси автентифікації мають базуватися на сучасних криптографічних протоколах, унеможлиблюючи несанкціонований доступ до особистих даних [2, 9]. Крім базового входу в систему, веб-додаток має підтримувати розширені

налаштування профілю користувача. Це включає можливість редагування особистих даних, зміни пароля, налаштування бажаної валюти за замовчуванням та інших параметрів персоналізації, що дозволить адаптувати інтерфейс під індивідуальні потреби кожного клієнта.

Основним ядром інформаційної системи є модуль безпосереднього обліку фінансових потоків. Система повинна забезпечувати можливість швидкого та зручного введення даних про доходи та витрати користувача із зазначенням суми, дати, категорії операції та додаткових текстових коментарів. Для забезпечення повноцінного контролю над рухом коштів користувач повинен мати постійний доступ до деталізованої історії фінансових операцій. Ця історія має бути реалізована у вигляді зручного хронологічного журналу з можливістю застосування різноманітних фільтрів, наприклад, за часовим проміжком, типом транзакції або конкретною категорією витрат. Такий підхід дозволить швидко знаходити необхідні записи та проводити первинний ретроспективний аналіз власної фінансової поведінки без необхідності експорту даних у зовнішні програми.

Для переходу від пасивної фіксації витрат до активного управління бюджетом система має містити розширений інструментарій цілепокладання та планування. Веб-додаток повинен дозволяти користувачам створювати конкретні фінансові цілі, визначати їх цільову суму, терміни досягнення та відстежувати поточний прогрес у режимі реального часу. Наприклад, при додаванні нових заощаджень система має автоматично перераховувати відсоток виконання мети та візуалізувати цей прогрес. Органічним доповненням до цього модуля є інтегрована система нагадувань про важливі події або регулярні платежі. Користувач повинен мати змогу налаштувати автоматичні сповіщення щодо необхідності оплати комунальних послуг, внесення кредитного платежу чи продовження підписок, що суттєво знизить ризик прострочень та підвищить загальний рівень фінансової дисципліни.

Щоб трансформувати сирі фінансові дані у зрозумілу інформацію для прийняття рішень, система повинна включати потужний аналітичний блок. Ця функціональна вимога передбачає автоматичну генерацію звітів та візуалізацію витрат і доходів у вигляді різноманітних графіків і діаграм. Аналітика має демонструвати структуру витрат за категоріями, динаміку зміни балансу протягом місяця або року, а також порівняння поточних витрат із встановленими бюджетними лімітами. Для забезпечення високої швидкості обробки цих запитів та побудови графіків у реальному часі, система повинна ефективно взаємодіяти з базою даних через оптимізовані API-маршрути на рівні бізнес-логіки. Такий чіткий розподіл функцій між клієнтською частиною, що відповідає за відмальовування інтерфейсу, та сервером, що обробляє масиви транзакцій, гарантуватиме плавну та стабільну роботу аналітичного модуля.

Після детального формування функціональних вимог до багаторівневої веб-системи управління бюджетом, наступним критично важливим кроком є вибір оптимальної методології розробки програмного забезпечення. У сучасній інженерії програмного забезпечення існує низка підходів до організації життєвого циклу проекту, серед яких найпоширенішими є класична каскадна модель (Waterfall) та сімейство гнучких методологій (Agile) [34]. Правильний вибір методології безпосередньо впливає на швидкість реалізації проекту, якість кінцевого продукту та здатність оперативно реагувати на зміни у вимогах або технологічному стеку під час узгодження роботи рівнів представлення, бізнес-логіки та даних.

Каскадна модель розробки, відома як Waterfall, передбачає суворе послідовне виконання етапів: від збору вимог і проектування архітектури до кодування, тестування та фінального розгортання. Хоча цей підхід забезпечує високий рівень документування та чітке розуміння кінцевого результату ще на початкових стадіях, він є вкрай негнучким. У контексті розробки сучасного веб-додатку для ведення особистого бюджету, де вимоги до користувацького

інтерфейсу, реалізованого на базі React та Next.js, можуть уточнюватися та змінюватися в процесі створення прототипів, використання класичного каскадного підходу не є оптимальним. Будь-яка помилка в проектуванні схеми бази даних через Prisma ORM або в архітектурі API-маршрутів, виявлена на пізніх етапах тестування, вимагатиме повернення на початкові стадії планування, що призведе до значних затримок у виконанні роботи.

З огляду на зазначені недоліки жорстких моделей, для розробки даної інформаційної системи доцільно обрати гнучку методологію Agile. Цей підхід базується на ітеративній та інкрементальній розробці, де весь процес створення системи розбивається на короткі робочі цикли. Використання Agile дозволяє розробляти різні компоненти трирівневої архітектури більш узгоджено: наприклад, можна налаштовувати моделі даних та маршрути серверної логіки, паралельно створюючи пов'язані з ними компоненти користувацького інтерфейсу. Завдяки постійному тестуванню та інтеграції невеликих частин коду після кожної ітерації, забезпечується можливість оперативно перевіряти коректність взаємодії з базою даних через API. Це мінімізує ризики критичних архітектурних помилок та дозволяє швидко адаптувати логіку підрахунку фінансових цілей чи формування аналітичних звітів відповідно до проміжних результатів тестування.

Крім того, гнучка методологія ідеально підходить для реалізації даного проекту, оскільки дозволяє сфокусуватися на першочерговому створенні мінімально життєздатного продукту (MVP). На перших ітераціях можна реалізувати базові функції реєстрації, авторизації та простого обліку доходів і витрат користувача. На наступних етапах функціонал буде поступово розширюватися шляхом додавання системи нагадувань, розширеної історії фінансових операцій, візуалізації аналітики витрат у вигляді графіків та інструментів налаштування профілю. Такий еволюційний шлях розробки гарантує, що система постійно перебуватиме в робочому стані.

Таким чином, симбіоз чітко визначених функціональних вимог та використання адаптивної методології Agile формує надійне організаційне та технічне підґрунтя для успішної реалізації інформаційної системи. Це повністю завершує етап системного аналізу предметної області та дозволяє перейти до етапу безпосереднього проектування.

Окрім безпосередньо функціональних вимог, успішне розгортання та довготривала експлуатація інформаційної системи вимагає чіткого визначення та дотримання низки нефункціональних критеріїв. Ці критерії визначають атрибути якості програмного продукту, такі як надійність, швидкодія та зручність супроводу. Одним із ключових параметрів для сучасного веб-додатку є його продуктивність та швидкість відгуку. Враховуючи використання трирівневої архітектури, взаємодія між клієнтським інтерфейсом на базі Next.js та сервером через API-маршрути має бути максимально оптимізованою. Час очікування при виконанні базових операцій, таких як додавання нової транзакції або перемикання між екранами аналітики, повинен бути мінімальним, щоб не порушувати стан потоку користувача. Для ресурсоємних завдань, таких як генерація складних річних звітів, система повинна надавати візуальний зворотний зв'язок про стан завантаження, зберігаючи при цьому загальну чуйність інтерфейсу.

Наступною важливою нефункціональною вимогою є масштабованість архітектури та стійкість до навантажень. Оскільки фінансова історія користувача накопичується роками, обсяг інформації у базі даних буде постійно зростати. Використання Prisma ORM на рівні доступу до даних дозволяє ефективно керувати складними зв'язками між таблицями (наприклад, між користувачами, категоріями, транзакціями та цілями) і писати оптимізовані запити. Проте на етапі проектування необхідно закласти таку структуру бази даних, яка б дозволяла легко масштабувати систему при збільшенні кількості активних користувачів, не втрачаючи при цьому швидкості вибірки історичних даних. Крім

того, архітектура повинна передбачати механізми резервного копіювання, щоб унеможливити втрату критично важливої фінансової інформації у разі непередбачуваних збоїв на сервері.

Особливу увагу під час обґрунтування вимог слід приділити ергономіці та кросплатформності користувацького інтерфейсу. Фінансовий інструмент має бути доступним для користувача у будь-який момент часу, незалежно від пристрою, який він використовує. Тому клієнтська частина веб-додатку повинна створюватися за принципами адаптивного веб-дизайну (Responsive Web Design) [24]. Це гарантує, що всі елементи управління, графіки та таблиці будуть коректно масштабуватися та перебудовуватися як на широких екранах десктопних моніторів, так і на відносно невеликих дисплеях смартфонів чи планшетів. Інтерфейс повинен відповідати сучасним стандартам доступності (Accessibility), забезпечуючи високу контрастність елементів, читабельність шрифтів та логічну навігацію, яка дозволяє швидко орієнтуватися в додатку навіть новим користувачам [39].

Повертаючись до питання вибору методології розробки, варто детальніше розкрити практичні аспекти застосування підходу Agile в умовах створення даного веб-додатку. Реалізація проекту базуватиметься на ітеративній моделі з використанням елементів фреймворку Scrum. Весь процес розробки розділяється на короткі робочі цикли - спринти, кожен з яких триває від одного до двох тижнів. Перед початком кожного спринту визначається конкретний перелік завдань із загального беклогу проекту. Наприклад, перший спринт може бути присвячений виключно налаштуванню середовища, проектуванню моделей Prisma та створенню базового API для авторизації. Другий спринт фокусуватиметься на розробці React-компонентів для відображення головної панелі управління (дашборду) та інтеграції їх із серверною частиною. Для наочної ілюстрації організації робочого процесу наведено адаптовану схему життєвого циклу розробки інформаційної системи, яка винесена до Додатку А (рисунок А.4).

Загальний беклог продукту містить вичерпний перелік функціональних вимог, починаючи від реєстрації користувача і закінчуючи формуванням аналітики. Формування робочих завдань на кожен двотижневий спринт відбувається з урахуванням обраного технологічного стеку (Next.js для клієнтської частини та Prisma для роботи з базою даних). Такий підхід забезпечує прозорість процесу розробки: щоденні короткі синхронізації дозволяють вчасно виявляти технічні перешкоди, а огляд та ретроспектива наприкінці кожного циклу гарантують, що готовий інкремент продукту повністю відповідає початковим вимогам щодо якості та продуктивності.

Такий покроковий підхід має кілька суттєвих переваг при роботі над дипломним проектом. По-перше, він дозволяє регулярно отримувати працюючий прототип системи з поступово зростаючим функціоналом, що спрощує процес демонстрації проміжних результатів науковому керівнику. По-друге, у разі виявлення логічних помилок у підрахунку балансу або проблем із відображенням графіків, їх можна ізолювати та виправити в межах поточного спринту, не ламаючи при цьому всю архітектуру програми. Крім того, гнучка розробка стимулює використання сучасних інженерних практик, таких як модульне тестування окремих функцій та безперервна інтеграція (CI), що автоматизує процес перевірки працездатності коду після кожного нового оновлення [7, 23].

Підсумовуючи результати першого розділу, можна констатувати, що проведений системний аналіз предметної області дозволив чітко окреслити проблематику управління особистими фінансами та виявити критичні недоліки існуючих на ринку програмних рішень. Формування детального переліку функціональних та нефункціональних вимог дозволило створити комплексне технічне завдання для розробки власної інформаційної системи. Своєю чергою, обґрунтований вибір гнучкої методології Agile у поєднанні з сучасним технологічним стеком створює міцний фундамент для ефективної та безперебійної реалізації проекту. Зібрана аналітична база повністю завершує

етап попереднього дослідження та дозволяє аргументовано перейти до наступного етапу - безпосереднього проектування архітектури бази даних, математичного апарату та логіки роботи веб-додатку.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Моделювання логіки роботи та опис математичного апарату обробки даних

Проектування інформаційної системи для ведення особистого бюджету базується на принципах багаторівневої архітектури, що забезпечує чіткий розподіл відповідальності між компонентами та полегшує подальше масштабування продукту [13, 14]. Рівень бізнес-логіки виступає центральним ядром, що обробляє запити, які надходять через API-маршрути (Next.js). Саме тут зосереджені основні алгоритми валідації, обробки фінансових операцій, перерахунку балансів та підготовки даних для аналітичних звітів за допомогою Prisma ORM.

Для наочного розуміння складності процесів, що відбуваються на рівні бізнес-логіки, доцільно розглянути життєвий цикл базової операції, який наведено в Додатку В (рисунку В.1.) наведено комплексну блок-схему алгоритму обробки фінансових даних користувача, яка охоплює не лише створення транзакції, але й паралельну роботу з фінансовими цілями та системою нагадувань.

Алгоритм розпочинається з отримання та аналізу вхідних даних транзакції (сума, тип, категорія). Після первинного створення запису в базі даних система розгалужується: залежно від типу операції (дохід чи витрата) відбувається відповідне коригування балансу обраного рахунку. Далі алгоритм ініціює модуль опрацювання фінансових цілей. У разі поповнення цілі система автоматично збільшує накопичену суму та паралельно зменшує вільний баланс рахунку. Цікавою архітектурною особливістю є автоматична перевірка наявності системної технічної категорії «Скарбничка»: якщо користувач здійснює цільове накопичення вперше, алгоритм самостійно створює цю категорію для коректного відображення у звітах. Завершальними етапами циклу є оновлення параметрів

профілю (наприклад, лімітів витрат), генерація нових нагадувань, синхронізація клієнтського інтерфейсу та повернення успішного статусу операції.

Окрім маршрутизації потоків даних, математичний апарат системи базується на сукупності алгоритмів агрегації та аналізу. Фундаментальною операцією є розрахунок поточного балансу (B), який визначається як різниця між сумою всіх доходів (I) та витрат (E) [3]:

$$B = \sum_{i=1}^n I_i - \sum_{j=1}^m E_j$$

Для реалізації інтелектуальних функцій підтримки прийняття рішень у системі додатково застосовуються наступні математичні методи:

1. Метод лінійної екстраполяції для прогнозування витрат

Для реалізації проактивної функції попередження про можливий дефіцит бюджету система використовує алгоритм прогнозування. Він базується на обчисленні середньоденних витрат користувача за поточний розрахунковий період з подальшим лінійним перенесенням цієї тенденції на кінець місяця. Математично прогнозована сума витрат ($E_{forecast}$) обчислюється за формулою:

$$E_{forecast} = \frac{\sum_{i=1}^k E_i}{D_{past}} * D_{total}$$

де $\sum_{i=1}^k E_i$ — сума всіх наявних фактичних витрат з початку періоду, D_{past} — кількість днів, що вже минули у поточному місяці, а D_{total} — загальна кількість днів у розрахунковому місяці. Якщо значення $E_{forecast}$ перевищує встановлений місячний ліміт, система генерує тригер для інтерфейсу користувача.

2. Метод порогового виявлення аномалій (Anomaly Detection)

Для підвищення рівня фінансової безпеки та контролю за нетиповими витратами у системі імплементовано базовий метод порогового виявлення аномалій. Цей метод аналізує кожну нову транзакцію, порівнюючи її із

середньостатистичною фінансовою поведінкою користувача в рамках конкретної категорії. Транзакція ідентифікується як аномальна ($A_t = 1$), якщо її сума (X_t) значно перевищує середнє значення витрат у цій категорії (μ_c), збільшене на допустимий пороговий коефіцієнт (стандартне відхилення або фіксований відсоток α):

$$A_t = \begin{cases} 1, \text{ якщо } X_t > \mu_c + \alpha \cdot \sigma_e \\ 0, \text{ в іншому випадку} \end{cases}$$

Такий підхід дозволяє системі автоматично підсвічувати підозріло великі витрати (наприклад, імпульсивні дорогі покупки), привертаючи до них додаткову увагу під час перегляду історії.

3. Метод нормалізації даних для мультивалютного аналізу

Оскільки сучасний користувач може оперувати кількома рахунками у різних валютах (наприклад, готівка в гривні, картка в доларах США), для побудови єдиного аналітичного графіка (кругової діаграми структури витрат) усі дані повинні бути зведені до єдиної базису. Для цього застосовується метод нормалізації, який конвертує всі транзакції в основну валюту профілю за актуальним курсом. Нормалізоване значення загальних витрат (V_{norm}) визначається як сума добутків кожної транзакції на відповідний крос-курс:

$$V_{norm} = \sum_{j=1}^N (V_j \cdot R_j \rightarrow base)$$

де V_j — сума j -ї транзакції у вихідній валюті, $R_j \rightarrow base$ — коефіцієнт конвертації (обмінний курс) валюти транзакції до базової валюти користувача на момент здійснення операції, а N — загальна кількість операцій за період.

Використання описаного комплексу математичних методів у поєднанні з оптимізованими алгоритмами обробки бази даних (Prisma) дозволяє створити високопродуктивну систему, здатну не лише зберігати фінансову інформацію, а

й виступати повноцінним інтелектуальним помічником для прийняття фінансових рішень.

Окрім описаного математичного апарату, критично важливою складовою логіки роботи системи є архітектура обміну даними між клієнтською (React) та серверною (Next.js API) частинами додатка. Взаємодія реалізується за принципами архітектурного стилю REST (Representational State Transfer) [31]. Усі дані передаються у форматі JSON, що забезпечує легковажність запитів та швидкий парсинг на стороні клієнта [21].

Кожен логічний модуль системи (користувачі, транзакції, цілі, категорії) має свій виділений контролер на сервері. Для забезпечення стандартизації обробки даних спроектовано базову таблицю маршрутизації API (табл. 2.1), яка описує основні точки доступу (endpoints) та відповідні їм HTTP-методи.

Таблиця 2.1 – Специфікація основних API-маршрутів системи

HTTP-метод	Маршрут (Endpoint)	Логічне призначення та опис обробника
POST	/api/auth/register	Реєстрація користувача. Валідація даних, хешування пароля (Bcrypt), створення запису User в БД.
POST	/api/auth/login	Авторизація. Перевірка пароля, генерація та повернення JWT-токена для сесії.
GET	/api/transactions	Отримання масиву транзакцій для поточного користувача. Підтримує параметри фільтрації (дата, категорія).

HTTP-метод	Маршрут (Endpoint)	Логічне призначення та опис обробника
POST	/api/transactions	Створення нової транзакції. Ініціює транзакцію в БД, оновлює баланс пов'язаного рахунку Account.
DELETE	/api/transactions/:id	Видалення транзакції за ID. Вимагає обов'язкового перерахунку балансу (зворотна операція).
GET	/api/goals	Отримання списку фінансових цілей з розрахованим відсотком прогресу.
PUT	/api/goals/:id	Оновлення існуючої цілі (наприклад, зміна цільової суми або дедлайну).
GET	/api/analytics/monthly	Агрегаційний маршрут. Повертає згруповані дані за категоріями для побудови кругової діаграми.

Логіка обробки запитів на сервері будується за принципом "тонкого контролера та товстої моделі". Наприклад, при надходженні POST-запиту на створення транзакції (/api/transactions), контролер спочатку виконує сувору валідацію тіла запиту (перевіряє наявність суми, ідентифікаторів рахунку та категорії). Після успішної валідації, логіка передається рівню Prisma ORM, який виконує атомарну транзакцію в базі даних: одночасно створює запис у таблиці Transaction та оновлює поле balance у таблиці Account. Використання атомарних

операцій (Prisma Transactions) гарантує, що у разі збою на будь-якому етапі, дані не будуть збережені частково, що захищає систему від фінансових розбіжностей.

Окремим критично важливим аспектом моделювання логіки системи є забезпечення інформаційної безпеки та розмежування прав доступу до конфіденційних фінансових даних. Оскільки система орієнтована на багатокористувацьке середовище, архітектура повинна гарантувати, що кожен стейкхолдер має доступ виключно до свого ізольованого набору транзакцій, цілей та налаштувань.

Для реалізації цього завдання спроектовано логічну модель автентифікації та керування сесіями на основі протоколу JSON Web Tokens (JWT) [18]. Життєвий цикл логіки авторизації в системі складається з наступних етапів:

1.Ініціація сесії: Клієнт надсилає облікові дані (email та пароль) на серверний маршрут `/api/auth/login`. Рівень бізнес-логіки виконує пошук користувача в БД та верифікує хешований пароль за допомогою алгоритму Bcrypt.

2.Генерація токена: У разі успішної перевірки сервер формує об'єкт (payload), що містить унікальний id користувача, і підписує його секретним ключем сервера. Згенерований JWT-токен повертається клієнту.

3.Санкціонування запитів: Клієнтський додаток зберігає токен і додає його до заголовка Authorization при кожному зверненні до фінансових модулів.

4.Middleware-валідація: На серверній стороні імплементовано проміжне програмне забезпечення (Middleware), яке перехоплює кожен запит, перевіряє цілісність підпису токена та автоматично витягує `userId` [19].

Така логічна модель дозволяє серверу «пам'ятати» користувача без необхідності зберігати активні сесії в оперативній пам'яті (stateless approach), що суттєво підвищує масштабованість системи та швидкість відгуку при зростанні кількості одночасних підключень.

Наступним рівнем логічного проєктування є підсистема валідації вхідних даних. Для запобігання пошкодженню цілісності бази даних та захисту від некоректних обчислень, у системі реалізовано багаторівневу перевірку параметрів. Основна логіка валідації базується на використанні декларативних схем (наприклад, бібліотеки Zod), які описують очікувану структуру кожного об'єкта.

Логіка валідації транзакції, наприклад, включає такі обов'язкові перевірки:

Типізація суми: Значення суми транзакції повинно бути суворо числовим типом (Float) і бути більшим за нуль (для запобігання логічним помилкам «від'ємних витрат»).

Верифікація категорій: Перед записом транзакції система перевіряє, чи існують вказані ідентифікатори рахунку (accountId) та категорії (categoryId) у профілі саме цього користувача.

Форматування дати: Система вимагає передачі дати у стандарті ISO-8601, що гарантує коректну роботу математичних методів агрегації за часовими періодами.

Завдяки впровадженню такої суворої логічної перевірки на етапі надходження запиту, серверна частина відсікає потенційно небезпечні або помилкові дані ще до того, як вони потраплять до рівня Prisma ORM. Це не лише підвищує надійність системи, а й забезпечує чистоту даних, необхідну для точної роботи аналітичних модулів та алгоритмів прогнозування бюджету.

Важливою складовою моделювання логіки роботи веб-додатку є проєктування механізмів синхронізації стану (state management) та асинхронної взаємодії. Традиційні підходи до розробки часто передбачають повне оновлення сторінки або тривале очікування відповіді сервера, що негативно впливає на користувацький досвід (User Experience). Для вирішення цієї проблеми в проєктованій системі імплементовано логіку «реактивного» оновлення інтерфейсу.

Логіка управління станом даних базується на принципах SWR (Stale-While-Revalidate). Це означає, що при зверненні до будь-якого фінансового модуля (наприклад, перегляд списку транзакцій), система виконує наступний алгоритм дій:

1.Відображення кешу: Система миттєво показує користувачеві дані, які збережені в локальному кеші браузера з попереднього сеансу [38].

2.Фонова валідація: Паралельно з відображенням кешу, рівень бізнес-логіки відправляє запит до API для перевірки актуальності даних.

3.Синхронізація: Якщо на сервері з'явилися нові записи (наприклад, додані з іншого пристрою), кеш автоматично оновлюється, і React-компоненти перемальовують інтерфейс без втручання користувача.

Такий підхід дозволяє нівелювати затримки мережі та забезпечує безперебійну роботу системи навіть при нестабільному інтернет-з'єднанні.

Окрему увагу в межах моделювання логіки приділено стратегії обробки помилок та збоїв. Оскільки фінансові системи оперують критично важливими даними, будь-яка помилка (наприклад, розрив з'єднання під час переказу коштів у «Скарбничку») повинна бути коректно опрацьована без втрати цілісності інформації. Логіка обробки помилок у системі побудована за ієрархічним принципом:

Глобальні помилки: Перехоплюються на рівні Middleware (наприклад, закінчення терміну дії JWT-токена). У такому разі система автоматично перериває поточну операцію та перенаправляє користувача на сторінку входу.

Помилки бізнес-логіки: Виникають при порушенні правил (наприклад, спроба видалити категорію, до якої прив'язані активні транзакції). Система повертає структурований JSON-об'єкт із кодом помилки, який трансформується інтерфейсом у зрозуміле текстове попередження.

Мережеві помилки: У разі відсутності зв'язку з сервером, логіка системи передбачає механізм повторних запитів (retries) з експоненціальним зростанням інтервалу очікування, що дозволяє автоматично відновити синхронізацію при появі мережі.

Завершальним етапом моделювання логіки 2.1 є опис механізму «Оптимістичного UI» (Optimistic Updates). При створенні нової фінансової цілі або транзакції, логіка інтерфейсу не чекає на підтвердження від бази даних Prisma. Натомість, система тимчасово додає новий запис у локальний стан

додатка, миттєво оновлюючи всі пов'язані графіки та баланси. Тільки після цього виконується фізичний запис у БД. Якщо сервер повертає помилку, система автоматично виконує «відкат» (rollback) стану до останнього стабільного значення. Це створює відчуття абсолютної миттєвості роботи додатка, що є стандартом для сучасних фінансових продуктів.

2.2. Проєктування бази даних та розробка концепції користувацького інтерфейсу

Етап проєктування бази даних є фундаментальним для створення стабільної та масштабованої інформаційної системи, оскільки саме на ньому визначається структура зберігання фінансових даних та встановлюються правила цілісності між основними сутностями. В основі розроблюваної системи лежить реляційна модель даних. Для взаємодії серверної частини з базою даних обрано технологію Prisma ORM, яка дозволяє декларативно описувати схему та автоматично генерувати типізований клієнт для виконання запитів, що суттєво знижує ризик виникнення помилок на етапі розробки [33].

Логічну структуру спроектованої бази даних наочно представлено у вигляді ER-діаграми (Entity-Relationship) зображено на рисунку 2.1

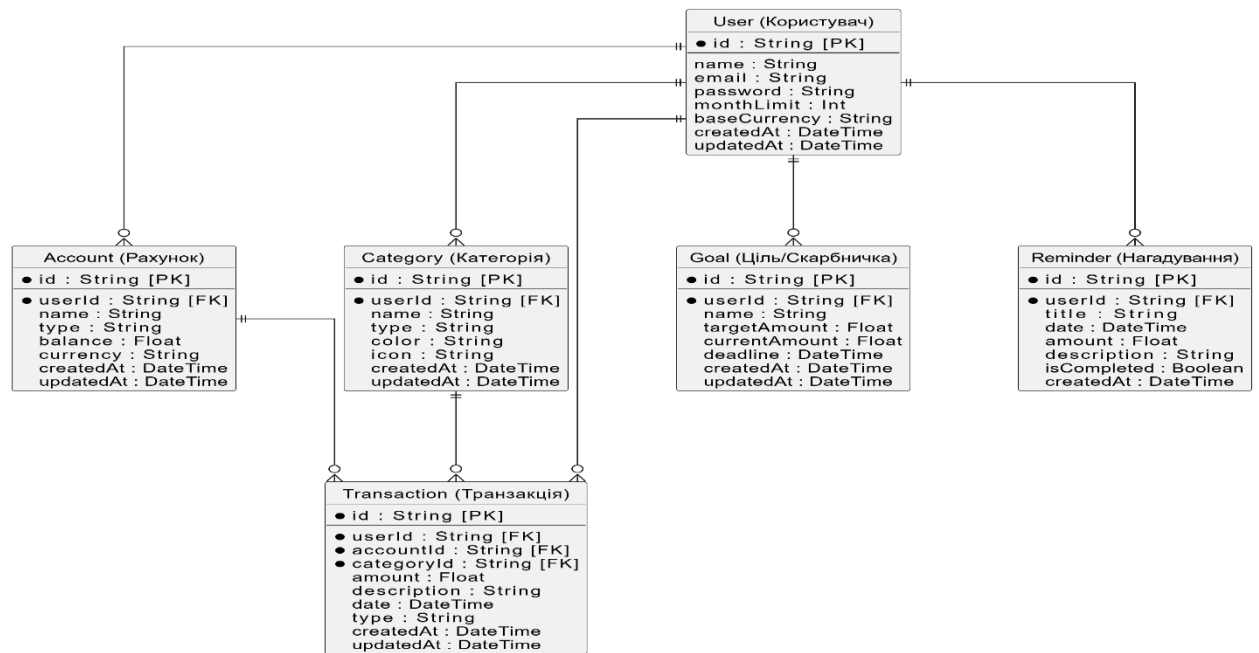


Рисунок. 2.1 – Логічна схема (ER-діаграма) реляційної бази даних системи управління бюджетом

Як видно з рисунка 2.1, архітектура бази даних побудована навколо центральної сутності User (Користувач), яка пов'язана відношенням «один-до-багатьох» з усіма іншими таблицями системи. Таблиця User зберігає облікові дані для авторизації (email, password), а також глобальні налаштування профілю, такі як базова валюта (baseCurrency) та місячний ліміт витрат (monthLimit). Кожен запис у системі ідентифікується за допомогою унікального первинного ключа (id типу String), що генерується автоматично.

Фінансова екосистема користувача формується двома допоміжними довідниками: таблицею Account (Рахунки) та таблицею Category (Категорії). Таблиця Account дозволяє реалізувати мультивалютність та роздільний облік коштів (наприклад, готівка, банківська картка), зберігаючи поточний баланс (balance) та тип валюти (currency) для кожного рахунку. Таблиця Category відповідає за логічне групування операцій та містить атрибути для візуальної персоналізації в інтерфейсі: колір (color) та іконку (icon).

Ядром обробки даних є транзакційна таблиця Transaction. Вона агрегує інформацію з інших таблиць за допомогою зовнішніх ключів (userId, accountId, categoryId), гарантуючи строгу посилальну цілісність: транзакція не може бути створена без прив'язки до існуючого користувача, рахунку та категорії. У цій таблиці фіксуються безпосередньо сума операції (amount), її тип (type), дата проведення (date) та текстовий опис (description).

Для реалізації функцій фінансового планування спроектовано дві додаткові сутності. Таблиця Goal (Цілі) зберігає цільову суму накопичення (targetAmount), поточний прогрес (currentAmount) та кінцевий термін (deadline). Таблиця Reminder (Нагадування) забезпечує роботу проактивної системи сповіщень, містячи інформацію про дату майбутнього платежу (date), суму та статус виконання (isCompleted). Крім того, абсолютно всі таблиці містять службові поля createdAt та updatedAt для системного аудиту та відстеження хронології змін.

Для забезпечення точної програмної реалізації (на рівні Prisma Schema) та уникнення конфліктів типів даних під час розробки, спроектовану ER-діаграму необхідно деталізувати у вигляді структурних специфікацій для кожної сутності [11].

Таблиця 2.2 – Специфікація сутностей та атрибутів логічної структури бази даних

Сутність	Назва атрибуту	Ідентифікатор	Тип	Опис та обмеження
КОРИСТУВАЧ (User)	Унікальний ID	id	String	Первинний ключ [PK]
	Ім'я користувача	name	String	Відображуване ім'я в інтерфейсі
	Електронна пошта	email	String	Використовується як логін, унікальне

	Пароль	password	String	Зашифрований хеш пароля
	Місячний ліміт	monthLimit	Int	Заданий ліміт витрат на місяць
	Базова валюта	baseCurrency	String	Основна валюта профілю (напр. UAH)
	Час створення	createdAt	DateTime	Дата та час реєстрації профілю
	Час оновлення	updatedAt	DateTime	Дата та час останньої зміни даних
РАХУНОК (Account)	ID рахунку	id	String	Первинний ключ [PK]
	Власник рахунку	userId	String	Зовнішній ключ [FK] (зв'язок з User)
	Назва рахунку	name	String	Назва (напр., "Картка", "Готівка")
	Тип рахунку	type	String	Категорія рахунку (дебетовий, готівка)
	Поточний баланс	balance	Float	Залишок коштів на рахунку
	Валюта рахунку	currency	String	Локальна валюта для цього рахунку
	Час створення	createdAt	DateTime	Дата додавання рахунку
	Час оновлення	updatedAt	DateTime	Час останньої транзакції по рахунку

КАТЕГОРІЯ (Category)	ID категорії	id	String	Первинний ключ [PK]
	Власник категорії	userId	String	Зовнішній ключ [FK] (зв'язок з User)
	Назва категорії	name	String	Напр., "Продукти", "Зарплата"
	Тип категорії	type	String	Дохід або Витрата
	Колір	color	String	HEX-код кольору для інтерфейсу
	Іконка	icon	String	Назва іконки для відображення
	Час створення	createdAt	DateTime	Дата створення категорії
	Час оновлення	updatedAt	DateTime	Дата останнього редагування
ТРАНЗАКЦІЯ (Transaction)	ID транзакції	id	String	Первинний ключ [PK]
	Ініціатор	userId	String	Зовнішній ключ [FK] (зв'язок з User)
	Рахунок операції	accountId	String	Зовнішній ключ [FK] (зв'язок з Account)
	Категорія	categoryId	String	Зовнішній ключ [FK] (зв'язок з Category)
	Сума	amount	Float	Фактична сума операції

	Опис	description	String	Текстовий коментар (опціонально)
	Дата проведення	date	DateTime	Фактичний час здійснення операції
	Тип операції	type	String	Напрямок руху (Дохід/Витрата)
	Час створення	createdAt	DateTime	Дата фіксації в системі
	Час оновлення	updatedAt	DateTime	Дата останнього редагування запису
ЦІЛЬ / СКАРБНИЧКА (Goal)	ID цілі	id	String	Первинний ключ [PK]
	Власник цілі	userId	String	Зовнішній ключ [FK] (зв'язок з User)
	Назва цілі	name	String	Назва (напр., "На авто", "Відпустка")
	Цільова сума	targetAmount	Float	Сума, яку необхідно накопичити
	Поточна сума	currentAmount	Float	Вже накопичені кошти
	Кінцевий термін	deadline	DateTime	Запланована дата досягнення
	Час створення	createdAt	DateTime	Дата створення цілі
	Час оновлення	updatedAt	DateTime	Останнє поповнення або зміна

НАГАДУВАННЯ (Reminder)	ID нагадування	id	String	Первинний ключ [PK]
	Одержувач	userId	String	Зовнішній ключ [FK] (зв'язок з User)
	Заголовок	title	String	Короткий опис (напр., "Оренда")
	Дата сповіщення	date	DateTime	Коли потрібно надіслати нагадування
	Очікувана сума	amount	Float	Сума майбутнього платежу
	Детальний опис	description	String	Додаткова інформація
	Статус виконання	isCompleted	Boolean	Чи був платіж здійснений (Так/Ні)
	Час створення	createdAt	DateTime	Дата створення нагадування

Таблиця 2.3 – Опис зв'язків між сутностями інфологічної моделі

Назва зв'язку	Первинна сутність	Підпорядкована сутність	Тип зв'язку	Опис зв'язку
Володіння рахунками	User (Користувач)	Account (Рахунок)	1:M (Один-до-багатьох)	Один користувач може відкривати та керувати багатьма рахунками.

Персоналізація категорій	User (Користувач)	Category (Категорія)	1:M (Один-до-багатьох)	Користувач створює власну ієрархію категорій для доходів та витрат.
Планування цілей	User (Користувач)	Goal (Ціль)	1:M (Один-до-багатьох)	Користувач встановлює персональні фінансові цілі (скарбнички).
Створення нагадувань	User (Користувач)	Reminder (Нагадування)	1:M (Один-до-багатьох)	Система фіксує нагадування про майбутні платежі для конкретного користувача.
Ініціація операцій	User (Користувач)	Transaction (Транзакція)	1:M (Один-до-багатьох)	Усі фінансові операції в системі жорстко прив'язані до конкретного профілю.

Прив'язка до рахунку	Account (Рахунок)	Transaction (Транзакція)	1:M (Один-до-багатьох)	Кожна транзакція ініціює зміну балансу на конкретному обраному рахунку.
Класифікація операцій	Category (Категорія)	Transaction (Транзакція)	1:M (Один-до-багатьох)	Кожна операція повинна належати до певної логічної категорії для аналітики.

Опис логічної структури та взаємодії сутностей

Інфологічна модель системи побудована за ієрархічним принципом, де центральною ланкою виступає сутність User. Саме ця сутність забезпечує ізоляцію даних: будь-яка інформація, від рахунків до нагадувань, доступна виключно власнику облікового запису через зовнішні ключі (userId).

Сутності Account та Category виконують роль фундаментальних довідників. Рахунки дозволяють сегментувати кошти за фізичним розташуванням (картка, готівка) та валютою, тоді як категорії забезпечують семантичне групування транзакцій для подальшої візуалізації у вигляді діаграм.

Найбільш динамічною частиною моделі є сутність Transaction, яка виступає точкою перетину трьох основних потоків даних: користувача, рахунку та категорії. Такий зв'язок гарантує цілісність системи: видалення рахунку або

категорії автоматично вимагає опрацювання пов'язаних транзакцій, що запобігає появі помилок у розрахунку загального балансу.

Модулі Goal та Reminder є функціональними розширеннями, які доповнюють базовий облік елементами планування. Вони існують паралельно до транзакцій, але тісно пов'язані з користувачем, дозволяючи системі переходити від простої фіксації подій до проактивної підтримки прийняття фінансових рішень.

Спираючись на затверджену структуру бази даних та логіку API-маршрутів, розробляється концепція користувацького інтерфейсу (UI). Веб-додаток проєктується за принципами Single Page Application (SPA) з використанням бібліотеки React, що забезпечує миттєву реакцію системи на дії користувача без повного перезавантаження сторінок браузера [6, 8, 25].

Архітектура клієнтської частини базується на компонентному підході. Інтерфейс розбивається на незалежні, ізольовані модулі (React Components), які можуть повторно використовуватися в різних частинах додатку. Основними архітектурними блоками клієнтського рівня є:

1.Глобальний контекст (State Providers): Компоненти вищого порядку, які зберігають стан сесії (дані авторизованого користувача, активний JWT-токен, обрана базова валюта) та роблять його доступним для будь-якого елемента інтерфейсу без необхідності передачі через глибоке дерево властивостей (props drilling).

2.Панель управління (Dashboard): Центральний екран системи, який виконує роль інформаційного хаба. Він агрегує дані з різних модулів та відображає загальний баланс, віджети швидкого доступу до цілей та кругові діаграми структури витрат.

3.Модулі введення даних (Forms & Modals): Набір інтерактивних форм для створення транзакцій, категорій чи нагадувань. Ці компоненти містять вбудовану

логіку клієнтської валідації (наприклад, блокування кнопки «Зберегти», якщо сума від'ємна або не обрано категорію), що знижує навантаження на сервер.

Для забезпечення доступності системи з будь-яких пристроїв, концепція інтерфейсу спирається на підхід Mobile-First (проектування спочатку для мобільних пристроїв) [24]. Використання сучасних CSS-фреймворків дозволяє створити адаптивну сітку (Responsive Grid). На екранах смартфонів основна навігація переноситься у нижню панель (Bottom Tab Bar) для зручності управління однією рукою, а об'ємні таблиці з історією транзакцій автоматично трансформуються у компактні списки карток. На десктопних моніторах інтерфейс розгортається, надаючи перевагу багатостолбковим макетам та розширеній аналітиці.

Підсумовуючи результати другого розділу, можна констатувати, що було створено комплексний архітектурний проєкт інформаційної системи. Математично обґрунтовано логіку роботи та алгоритми агрегації даних. Спроектовано надійну реляційну схему бази даних, яка гарантує цілісність інформації, та сформовано концепцію сучасного, адаптивного користувацького інтерфейсу. Усі ці рішення формують вичерпне технічне завдання, яке дозволяє перейти безпосередньо до етапу написання програмного коду та тестування готового продукту.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ВЕДЕННЯ ОСОБИСТОГО БЮДЖЕТУ

Процес програмної реалізації спроектованої інформаційної системи базувався на використанні сучасного стеку веб-технологій, що забезпечує високу продуктивність та масштабованість. Основним інструментом розробки було обрано фреймворк Next.js (версія 16) із використанням архітектури App Router, що дозволяє поєднувати серверні та клієнтські компоненти для оптимізації швидкості завантаження [15, 38]. Весь програмний код системи написано на мові TypeScript, що забезпечує строгу типізацію та мінімізує кількість помилок під час розробки [5, 26]. Клієнтська частина базується на бібліотеці React (версія 19) [16], а для стилізації інтерфейсу було прийнято рішення використовувати чистий Vanilla CSS без сторонніх фреймворків, що дозволило створити унікальні UI-компоненти та повністю контролювати їх візуальну поведінку [10, 28].

Робота з даними реалізована за допомогою Prisma ORM (версія 6.0.0), яка забезпечує взаємодію з реляційною базою даних SQLite [35]. Для гарантування безпеки та управління сесіями користувачів інтегровано NextAuth.js, що дозволяє реалізувати надійні механізми реєстрації та авторизації. Аналітичний блок системи, який є однією з ключових переваг проєкту, побудований на базі бібліотеки Recharts, що забезпечує відмальовування інтерактивних SVG-графіків.

Фундаментальним етапом програмної реалізації серверної частини стало перенесення спроектованої у другому розділі логічної схеми бази даних у фізичний код за допомогою декларативного синтаксису Prisma. Як фізичне сховище даних було налаштовано СУБД SQLite, що є оптимальним рішенням для забезпечення швидкодії та портативності на етапі запуску продукту. Використання файлу schema.prisma дозволило створити єдине джерело істини (Single Source of Truth) для структури даних усього додатка.

Для демонстрації підходів до опису моделей та їх зв'язків наведено в додатку С (Частина коду С.1) фрагмент конфігурації бази даних та реалізацію центральних сутностей системи — користувача (User) та фінансової операції (Transaction).

У наведеному лістингу чітко простежується реалізація вимог щодо посилення цілісності та безпеки. Модель User заздалегідь адаптована для роботи з бібліотекою NextAuth.js, про що свідчать поля emailVerified та image, а також підтримка авторизації за обліковими даними (Credentials Auth) через хешоване поле password.

Кожна транзакція жорстко прив'язана до конкретного користувача, рахунку та категорії через директиву @relation. Особливу увагу варто звернути на систематичне використання правила каскадного видалення (onDelete: Cascade). Це архітектурне рішення гарантує, що при видаленні користувачем свого профілю, рахунку або категорії, всі пов'язані з ними фінансові операції будуть автоматично знищені на рівні рушія SQLite. Такий підхід запобігає накопиченню "осиротілих" (орфанних) записів у базі даних, унеможливорює математичні колізії при підрахунку загального балансу та гарантує повне видалення конфіденційної інформації за запитом стейкхолдера.

Одним із центральних модулів системи є компонент ChartsPanel.tsx, який відповідає за графічну візуалізацію фінансових потоків. В додатку С (Частина коду С.2) наведено лістинг коду цього компонента, який демонструє принципи агрегації даних та роботи з мультивалютністю.

У наведеному фрагменті реалізовано логіку оптимізації обчислень через хук useMemo, що дозволяє перераховувати суми витрат лише при зміні вхідних даних. Особлива увага приділена нормалізації сум у різних валютах до єдиного гривневого еквівалента за допомогою функції convertToBaseCurrency, що було обґрунтовано в математичному апараті другого розділу.

Розроблений користувацький інтерфейс системи відповідає принципам інтуїтивності та Mobile-First. На головній сторінці додатка (Dashboard) відображається загальний фінансовий стан користувача, кругова діаграма розподілу витрат за категоріями та віджети останніх транзакцій.

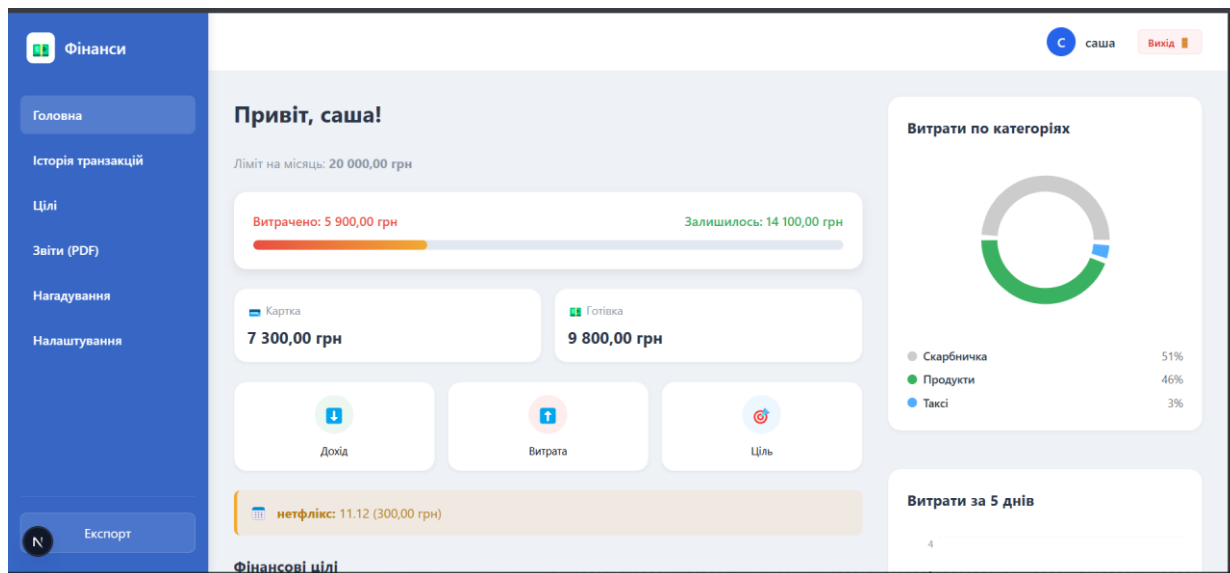
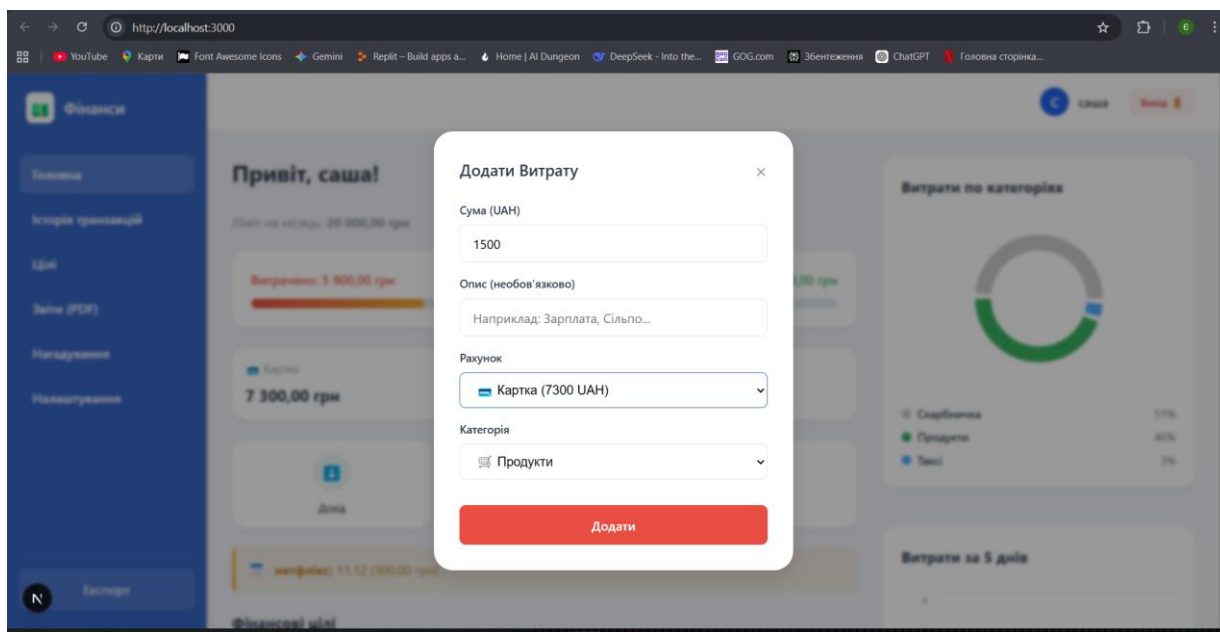


Рисунок 3.1 — Головна панель управління розробленої системи

Для зручності управління коштами в системі реалізовано окремі інтерфейсні модулі для додавання транзакцій та налаштування фінансових цілей. Кожна форма оснащена клієнтською валідацією, що запобігає надсиланню некоректних даних на сервер.



3.2 — Інтерфейс додавання нової фінансової операції

Фінальним етапом реалізації стало тестування працездатності системи. Було проведено серію тестів на перевірку коректності авторизації, цілісності даних при видаленні записів та точності агрегації мультивалютних рахунків [17, 20, 37]. Результати тестування підтвердили, що інформаційна система працює стабільно, забезпечує миттєве оновлення даних та повністю відповідає сформульованим у першому розділі вимогам.

ВИСНОВКИ

У результаті виконання дипломної роботи було успішно вирішено актуальне науково-практичне завдання — спроектовано та розроблено комплексну інформаційну систему для ведення та аналізу особистого бюджету користувача. Під час виконання етапів дослідження та розробки було досягнуто наступних результатів:

1.Проведено системний аналіз предметної області. Досліджено ринок існуючих програмних рішень та виявлено їхні основні недоліки: інформаційне перевантаження інтерфейсів, функціональні обмеження безкоштовних версій та відсутність гнучкої інтеграції фінансових цілей із щоденним обліком. На основі цих даних сформовано перелік функціональних вимог до нової системи.

2.Обґрунтовано вибір технологічного стеку. Для реалізації проєкту обрано сучасні інструменти веб-розробки: фреймворк Next.js (версія 16), бібліотеку React (версія 19), Prisma ORM для роботи з базою даних SQLite та NextAuth.js для забезпечення безпеки. Вибір Vanilla CSS дозволив створити легкий та швидкий інтерфейс без надлишкових залежностей.

3.Спроектовано архітектуру та математичний апарат. Розроблено інфологічну модель бази даних, яка через систему зовнішніх ключів та каскадне видалення гарантує цілісність інформації. Математично обґрунтовано алгоритми агрегації даних, зокрема методи нормалізації мультивалютних транзакцій та прогнозування витрат на основі ретроспективних даних.

4.Здійснено програмну реалізацію системи. Створено веб-додаток з адаптивним інтерфейсом (Mobile-First), який забезпечує повний цикл управління фінансами: від реєстрації та налаштування рахунків до ведення історії транзакцій. Реалізовано інтерактивний аналітичний модуль на базі Recharts, який візуалізує структуру витрат у реальному часі.

5.Забезпечено безпеку та проведено тестування. Впроваджено механізми захисту даних, включаючи хешування паролів та безпечне керування сесіями. Проведене тестування підтвердило стабільність роботи системи, коректність

розрахунків при роботі з різними валютами та стійкість до введення помилкових даних.

Розроблена інформаційна система є готовим до експлуатації продуктом рівня MVP. Вона надає користувачам зручний інструментарій для контролю витрат, формування фінансової дисципліни та досягнення довгострокових фінансових цілей. Гнучка архітектура додатка на базі Next.js та Prisma дозволяє легко масштабувати систему в майбутньому, додаючи модулі автоматичного імпорту банківських даних або інтелектуальні поради на основі штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Україномовні джерела та нормативні акти

1. Буров Є. В. Веб-програмування : навч. посіб. Львів : Видавництво Львівської політехніки, 2021. 368 с.
2. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI (ред. 2024). URL: <https://zakon.rada.gov.ua/laws/show/2297-17>
3. Кліщ Т. М. Цифрова фінансова грамотність: компоненти та методи підвищення. *Innovation and Sustainability*. 2023. № 2. С. 31–37. URL: <https://ins.vntu.edu.ua/index.php/ins/article/view/163>
4. Коваль С., Мельничук Ю. Фінансова грамотність домогосподарств України: сучасний стан та перспективи підвищення. *Світ фінансів*. 2024. Вип. 3. С. 82–87. URL: <https://dspace.wunu.edu.ua/handle/316497/55129>
5. Трофименко О. Г., Прокоп Ю. В. Ефективність алгоритмів сортування у TypeScript. *Інформатика та математичні методи в моделюванні*. 2024. Т. 14, № 1-2. С. 45–56.

Англомовна література, книги та статті

6. Banks A., Porcello E. Learning React: Modern Patterns. O'Reilly, 2020 (2nd Edition). 350 p.
7. Forsgren N., Humble J. Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, 2021 (Updated usage). 288 p.
8. Freeman A. Pro React 18. Apress, 2022. 650 p.
9. Green B., Smith M. The Cryptopals Crypto Challenges: Secure Implementation. 2021. URL: <https://cryptopals.com/>
10. Haverbeke M. Eloquent JavaScript (4th Edition). No Starch Press, 2024. 472 p.
11. Kleppmann M. Designing Data-Intensive Applications. O'Reilly, 2021 (Reprint). 616 p.
12. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders, 2021 (Reprint). 200 p.
13. Newman S. Building Microservices (2nd Edition). O'Reilly, 2021. 612 p.

14. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly, 2020 (Updated 2021). 432 p.
15. Riva M. Real-World Next.js. Packt Publishing, 2022. 410 p.
16. Schwarzmüller M. React Key Concepts. Packt Publishing, 2022. 350 p.

Технічна документація та міжнародні звіти (2020-2024)

17. Amazon Web Services. AWS Architecture Center. 2024. URL: <https://docs.aws.amazon.com/>
18. Auth0. JSON Web Token (JWT) Introduction. 2024. URL: <https://jwt.io/introduction>
19. Cloudflare. Web Security and Performance Docs. 2024. URL: <https://developers.cloudflare.com/>
20. Docker Documentation. Containerization for Web Applications. 2024. URL: <https://docs.docker.com/>
21. ECMA International. ECMAScript® 2023 Language Specification. 2023. URL: <https://tc39.es/ecma262/2023/>
22. Figma. The Guide to UI/UX Design. 2024. URL: <https://www.figma.com/resource-library/what-is-ui-ux-design/>
23. GitHub Docs. Best Practices for Software Development. 2024. URL: <https://docs.github.com/>
24. Google. Web.dev Documentation. 2024. URL: <https://web.dev/>
25. Meta Platforms. React Documentation. 2024. URL: <https://react.dev/reference>
26. Microsoft. TypeScript Documentation. 2024. URL: <https://www.typescriptlang.org/docs/>
27. Mozilla Foundation. JavaScript Security Guidelines. 2024. URL: <https://developer.mozilla.org/en-US/docs/Web/Security>
28. Mozilla Foundation. MDN Web Docs. 2024. URL: <https://developer.mozilla.org/>
29. Nielsen Norman Group. UX Research Articles: Visualizing Financial Data. 2024. URL: <https://www.nngroup.com/>

30. OECD. Financial Literacy and Education Reports: Digital Financial Inclusion. 2024. URL: <https://www.oecd.org/finance/>
31. OpenAPI Initiative. OpenAPI Specification. 2024. URL: <https://swagger.io/specification/>
32. OWASP Foundation. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks. 2021. URL: <https://owasp.org/www-project-top-ten/>
33. Prisma Data. Prisma Documentation. 2024. URL: <https://www.prisma.io/docs>
34. Scrum.org. The Scrum Guide. 2020. URL: <https://scrumguides.org/>
35. SQLite Consortium. SQLite Documentation. 2024. URL: <https://www.sqlite.org/docs.html>
36. Stripe. API Reference and Financial Integration. 2024. URL: <https://stripe.com/docs>
37. Vercel. Deployment and Infrastructure Strategies for Next.js. 2024. URL: <https://vercel.com/docs>
38. Vercel. Next.js Documentation. 2024. URL: <https://nextjs.org/docs>
39. W3C. Web Content Accessibility Guidelines (WCAG) 2.2. 2023. URL: <https://www.w3.org/TR/WCAG22/>
40. World Bank. Financial Inclusion Reports 2023. 2023. URL: <https://www.worldbank.org/>

ДОДАТКИ

Додаток А

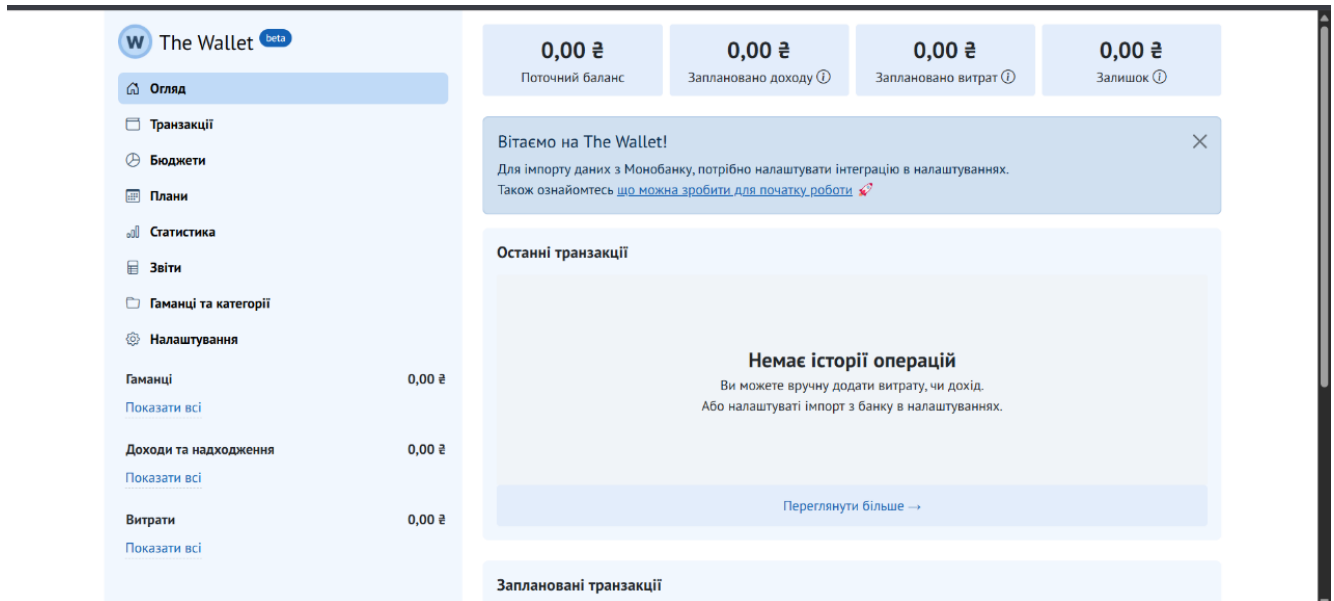


Рисунок А.1. – Панель управління системи «The Wallet» з фокусом на банківську інтеграцію

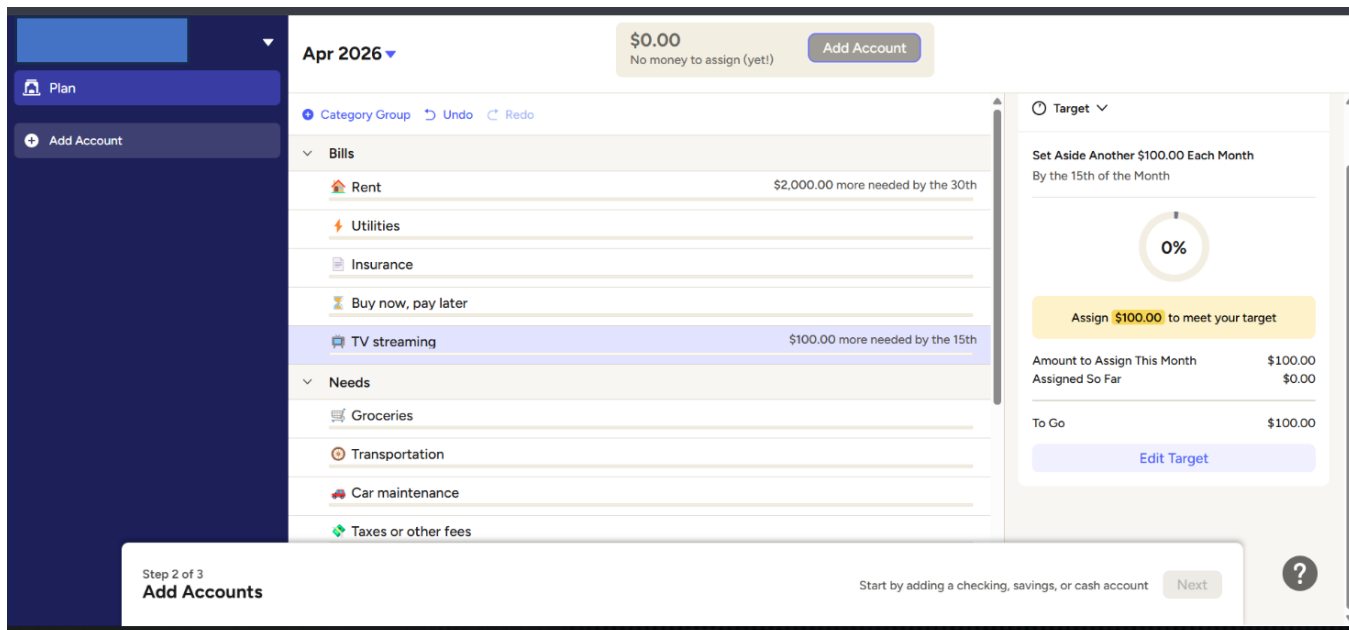


Рисунок. А.2. – Користувачський інтерфейс системи управління бюджетом (на прикладі YNAB)

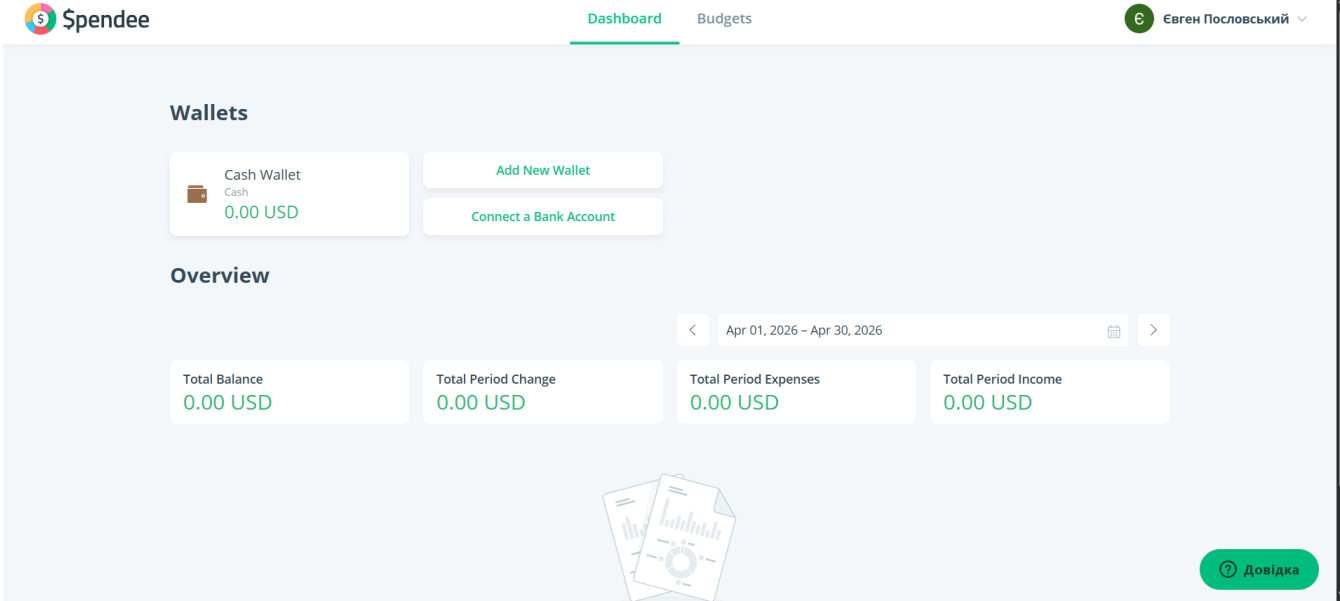


Рисунок. А.3. – Головна панель управління сервісу обліку фінансів Spendee

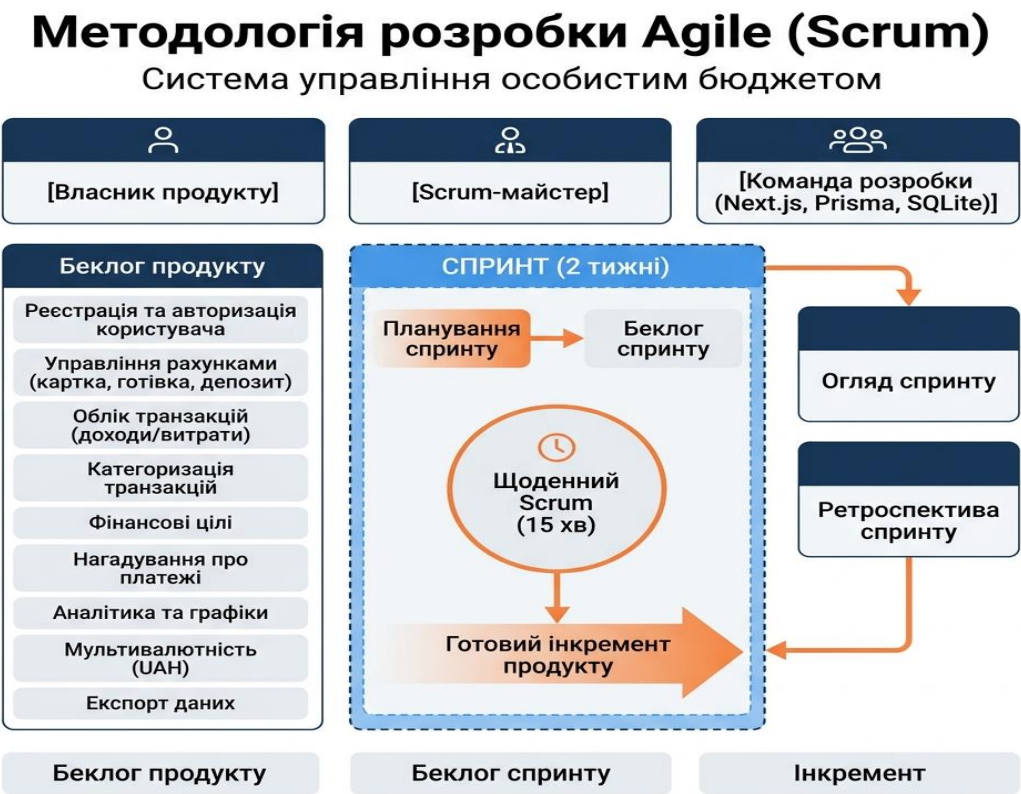


Рисунок. А.4. – Схема життєвого циклу розробки системи за методологією Agile (Scrum)

Додаток Б

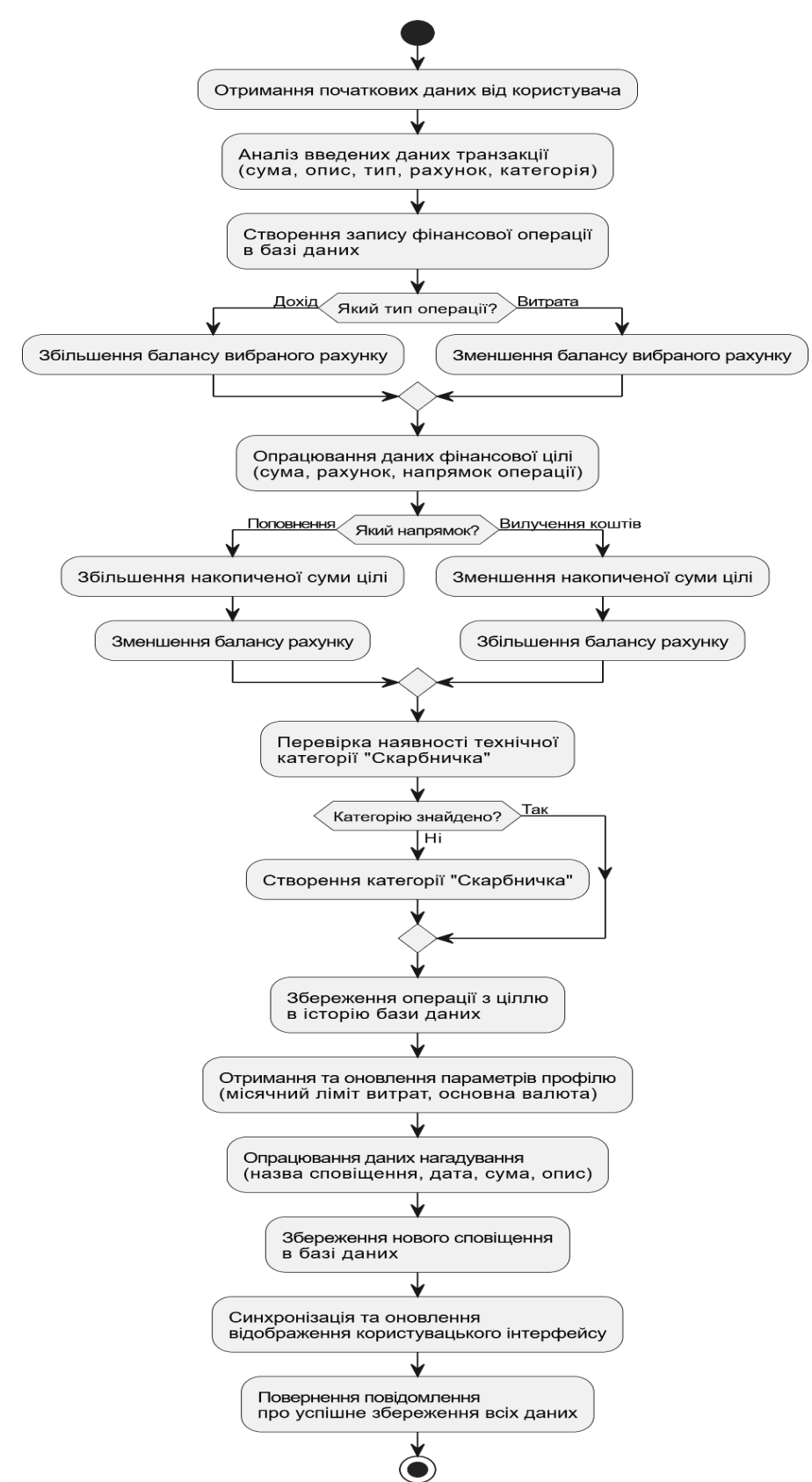


Рисунок. В.1. – Блок-схема комплексного алгоритму обробки фінансових даних та цілей

Додаток С

(Частина коду С.1)

```
datasource db {
    provider = "sqlite"
    url      = env("DATABASE_URL")
}

model User {
    id          String    @id @default(uuid())
    name        String?
    email        String?   @unique
    password     String?   // Для Credentials Auth
    emailVerified DateTime?
    image        String?

    accounts     Account[]
    categories    Category[]
    transactions Transaction[]
    goals         Goal[]
    reminders     Reminder[]

    monthLimit    Int       @default(10000)
    baseCurrency  String    @default("UAH")
}
```

```

    createdAt    DateTime @default(now())
    updatedAt    DateTime @updatedAt
}

model Transaction {
  id            String    @id @default(uuid())
  amount        Float
  description    String?
  date          DateTime @default(now())
  type          String    // "INCOME" or "EXPENSE"

  accountId      String
  account        Account  @relation(fields: [accountId], references: [id], onDelete: Cascade)

  categoryId     String
  category       Category @relation(fields: [categoryId], references: [id], onDelete: Cascade)

  userId         String
  user           User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
}

```

(Частина коду C.2)

```

"use client";

import { PieChart, Pie, Cell, Tooltip, ResponsiveContainer } from 'recharts';
import { useMemo } from 'react';
import { convertToBaseCurrency } from '@lib/currency';

type Transaction = {
  id: string;
  amount: number;
  type: string;
  date: Date;
  categoryId: string;
  account: { currency: string };
};

type Category = {
  id: string;
  name: string;
  color: string | null;
};

export default function ChartsPanel({
  transactions,
  categories,
  rates
}: {
  transactions: Transaction[],

```

```

categories: Category[],
rates: Record<string, number>
)) {

// Агрегація даних для Кругової діаграми (Витрати по категоріям)
const pieData = useMemo(() => {
  const expenses = transactions.filter(t => t.type === 'EXPENSE');
  const categoryTotals: Record<string, number> = {};

  expenses.forEach(t => {
    const catName = categories.find(c => c.id === t.categoryId)?.name || 'Інше';
    const amountInUAH = convertToBaseCurrency(t.amount, t.account.currency, rates);
    categoryTotals[catName] = (categoryTotals[catName] || 0) + amountInUAH;
  });
  return Object.entries(categoryTotals)
    .map(([name, value]) => ({
      name,
      value,
      color: categories.find(c => c.name === name)?.color || '#cccccc'
    }))
    .filter(item => item.value > 0);
}, [transactions, categories, rates]);

return (
  <div className="card chart-card">
    <h3>Витрати по категоріях</h3>
    {pieData.length > 0 ? (

```

```

<div style={{ height: '220px', width: '100%' }}>

  <ResponsiveContainer width="100%" height="100%">

    <PieChart>

      <Pie

        data={pieData}

        cx="50%"

        cy="50%"

        innerRadius={60}

        outerRadius={80}

        paddingAngle={5}

        dataKey="value"

        stroke="none"

      >

        {pieData.map((entry, index) => (

          <Cell key={`cell-${index}`} fill={entry.color} />

        ))}

      </Pie>

      <Tooltip formatter={(value: any) => [`${value} грн`, 'Сума']} />

    </PieChart>

  </ResponsiveContainer>

</div>

) : (

  <div className="center-msg">Немає витрат</div>

)}

</div>

);

}

```