

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа управління особистими фінансами»

Виконав студент групи ВТ-22-1
ЧЕБЕРА Вікторія Андріївна

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
НЕРОДА Сергій Іванович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЧЕБЕРА Вікторія Андріївна**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Вебплатформа управління особистими фінансами»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92**

Вихідні дані для роботи: **об'єктом дослідження** є процес управління особистими фінансами за допомогою інформаційно-цифрових систем, а **предметом дослідження** є архітектура клієнтської та серверної частин вебплатформи для обліку доходів і витрат, планування бюджету користувача та формування фінансової аналітики.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	01.03.2026р.	01.03.2026р.
2	Власенко О.В.	01.03.2026р.	01.03.2026р.
3	Власенко О.В.	01.03.2026р.	01.03.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебплатформи управління особистими фінансами та пояснювальної записки. Пояснювальна записка до випускної роботи містить 86 сторінок, 37 ілюстрацій та 21 таблицю.

Метою випускної роботи є розробка та впровадження вебплатформи для планування та контролю особистих фінансів і підвищення фінансової грамотності користувачів.

Архітектура вебплатформи управління особистими фінансами побудована на основі клієнт-серверної моделі з використанням фреймворку Next.js, що забезпечує гібридний рендеринг та динамічну роботу інтерфейсу без повного перезавантаження сторінок. Компоненти системи взаємодіють за принципом REST API, завдяки чому дані передаються асинхронно у форматі JSON. Такий підхід відокремлює серверну логіку від того, як дані відображаються користувачу — в результаті система працює швидше і інтерфейс краще адаптується. Вся інформація про транзакції, бюджети та рахунки користувачів зберігається в централізованій базі даних, тому в майбутньому можна буде без проблем додавати нові функції. Вебплатформа має простий у використанні інтерфейс, що дозволяє легко вносити дані про доходи та витрати, переглядати історію транзакцій і керувати особистими даними користувача.

КЛЮЧОВІ СЛОВА: АВТОРИЗАЦІЯ, ВЕБПЛАТФОРМА, ОСОБИСТІ ФІНАНСИ, СИСТЕМА ПЛАНУВАННЯ ТА УПРАВЛІННЯ, ГІБРИДНИЙ РЕНДЕРИНГ.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Чебера В.А.			Вебплатформа управління особистими фінансами	Літ.	Арк.	Аркуші
Керівник		Власенко О.В.					3	86
						ФІКТ Гр. ВТ-22-1		
Рецензент		Нерода С.І.						
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis consists of a web platform for the control, planning, and management of personal finances, as well as an explanatory note. The explanatory note of the thesis contains 86 pages, 37 illustrations, and 21 tables.

The bachelor's thesis aims to develop and implement a web platform for planning and controlling personal finances in order to improve users' financial literacy.

The architecture of the personal finance management web platform is built on a client-server model using the Next.js framework, which provides hybrid rendering and dynamic interface interactions without full page reloads. The system components interact based on REST API principles, ensuring asynchronous data transmission in JSON format. This approach separates the server-side logic from the data presentation layer, resulting in faster system performance and a more adaptive user interface. All information regarding transactions, budgets, and user accounts is stored in a centralized database, allowing for the seamless integration of new features in the future. The web platform features a user-friendly interface that allows for easy entry of income and expense data, viewing of transaction history, and management of personal user data.

KEYWORDS: AUTHORIZATION, WEB PLATFORM, PERSONAL FINANCES, PLANNING AND MANAGEMENT SYSTEM, HYBRID RENDERING.

		Чебера В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	4
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ ПЛАНУВАННЯ ТА КОНТРОЛЮ ВЕБПЛАТФОРМИ ОСОБИСТИХ ФІНАНСІВ	10
1.1 Постановка задачі.....	10
1.2 Аналіз аналогів програмного продукту	13
1.3 Вибір архітектури вебплатформи.....	18
Висновки до першого розділу.....	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ОСОБИСТИХ ФІНАНСІВ	23
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	23
2.2 Опис бази даних системи	33
2.3 Алгоритм роботи та стани програмної системи	45
2.4 Реалізація модуля інтеграції з банківською системою Monobank ...	48
Висновки до другого розділу.....	52
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ	54
3.1 Порядок встановлення та налаштування параметрів вебплатформи ..	54
3.2 Структура інтерфейсу системи управління особистими фінансами ...	57
3.3 Тестування роботи програмного комплексу	73
Висновки до третього розділу.....	77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	80
ДОДАТКИ.....	83

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

СУБД – система управління базами даних

API – набір правил, протоколів та інструментів, який дозволяє різним комп'ютерним програмам та застосункам спілкуватися і обмінюватися даними між собою (Application Programming Interface)

JSON – формат даних, який використовується для обміну даними між різними додатками (JavaScript Object Notation)

CRUD – основні операції, що виконуються для управління даними (Create, Read, Update, Delete)

JWT – безпечний інтернет-стандарт для передачі інформації у вигляді об'єктів JSON між двома сторонами (JSON Web Token)

MPA – підхід до створення вебсайтів і застосунків, де кожна дія або перехід на нову сторінку запитує нову HTML-сторінку з сервера (Multi Page Application)

SQL – стандартна мова програмування, яка використовується для керування, редагування та отримання даних у реляційних базах даних (Structured Query Language)

ORM – підхід до роботи з базами даних, який дозволяє взаємодіяти з даними як з об'єктами мови програмування замість написання сирого SQL-коду (Object-Relational Mapping)

MCC – чотирицифровий код, який використовується банками для класифікації типів фінансових операцій (Merchant Category Code)

SSR – метод рендерингу вебсторінки на сервері під час кожного запиту користувача (Server-Side Rendering)

SSG – метод генерації статичних HTML-сторінок під час збірки проєкту (Static Site Generation)

OAuth – відкритий протокол авторизації, який дозволяє надавати стороннім сервісам обмежений доступ до даних користувача без передачі пароля (Open Authorization)

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	6
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми розробки вебплатформи управління особистими фінансами зумовлена потребою користувачів у зручному інструменті для контролю доходів, витрат, рахунків і бюджетів. У сучасних умовах фінансові операції здійснюються через банківські картки, онлайн-сервіси та різні платіжні системи, тому користувачам стає складніше самостійно відстежувати рух коштів і об'єктивно оцінювати власний фінансовий стан. Відсутність системного обліку може призводити до непередбачуваних витрат, нераціонального використання коштів і труднощів у плануванні бюджету.

Існуючі сервіси для фінансового обліку не завжди повністю відповідають потребам українського користувача. Частина з них орієнтована на іноземний ринок, має обмежену підтримку українських банків або надає важливі функції, такі як бюджети, розширена статистика чи банківська синхронізація, лише за платною підпискою. Також окремі аналоги мають перевантажений інтерфейс, що ускладнює регулярне ведення особистих фінансів.

Розроблювана вебплатформа спрямована на вирішення цих проблем шляхом об'єднання основних процесів фінансового обліку в одній системі. Вона передбачає роботу з транзакціями, рахунками, категоріями, бюджетами та статистикою, що дозволяє користувачеві аналізувати доходи й витрати та приймати обґрунтовані фінансові рішення.

Особливістю системи є адаптація під український ринок: підтримка Monobank, робота з гривнею та іншими валютами, а також автоматичний імпорт банківських операцій. Це зменшує кількість ручних дій і підвищує точність обліку. Додатково реалізована панель адміністратора для керування користувачами, категоріями та журналом подій.

Таким чином, розробка вебплатформи управління особистими фінансами є актуальною, оскільки вона поєднує зручність, автоматизацію, локальну адаптацію та практичну користь для покращення фінансової дисципліни користувачів.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	7
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Метою випускної кваліфікаційної роботи бакалавра є розробка вебплатформи для автоматизації, планування та аналізу обліку особистих фінансів користувача, з можливістю адміністрування системи, контролю користувачів, перегляду журналу подій та забезпечення безпеки доступу.

Щоб реалізувати мету, потрібно виконати ряд завдань. По-перше, необхідно проаналізувати особливості роботи систем обліку особистих фінансів та задля безпеки вебплатформи також дослідити всі вимоги. Наступним потрібно обрати засоби проєктування та спроектувати архітектуру вебплатформи. Не менш важливо розробити базу даних для зберігання інформації користувачів, їх транзакцій, бюджетів та рахунків, а також інформації про категорії. Обов'язково реалізувати реєстрацію та авторизацію користувачів із розмеженими правами доступу, а також розробити можливість входу до системи через сервіс Google.

Для звичайного користувача розробити функціонал створення, редагування та видалення фінансових транзакцій та рахунків. Забезпечити підтримку мультивалютності під час роботи з рахунками, транзакціями та бюджетами. Реалізувати підключення акаунту Monobank за допомогою токена. Забезпечити перегляд транзакцій у вигляді списку та візуалізацію фінансової статистики за допомогою діаграм. Реалізувати функціонал планування бюджету за категоріями та рахунками, а саме: створення, видалення та перегляд бюджетів. Реалізувати сповіщення користувача при перевищенні лімітів бюджету, а також читання та видалення сповіщень.

А для адміністратора необхідно насамперед створити адміністративну сторінку для перегляду кількості користувачів, транзакцій, синхронізацій з банківськими системами та помилок безпеки. Важливо передбачити функціонал пошуку, блокування, видалення користувачів та зміни їхніх ролей. Для відстеження подій безпеки також необхідно створити сторінку, яка буде фіксувати основні дії користувача, зокрема спроби переходу на сторінку /admin без відповідних прав доступу. Реалізувати адміністративне керування категоріями: створення, пошук, редагування та видалення категорій.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	8
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Насамкінець, не менш важливо створити зручний адаптивний інтерфейс для використання вебплатформи на різних пристроях. Також потрібно провести тестування системи для перевірки стабільності роботи вебплатформи.

Об'єктом дослідження є процес управління особистими фінансами, що включає планування бюджету, облік доходів та витрат, контроль заощаджень і аналіз статистичних даних.

Предметом дослідження є архітектура клієнтської та серверної частин вебплатформи для обліку доходів і витрат, планування бюджету та формування фінансової аналітики.

Методи дослідження. У роботі було використано методи системного аналізу для дослідження предметної області управління особистими фінансами та виявлено основні функціональні можливості. Також використовувався об'єктно-орієнтований метод проєктування під час побудови структури програмного забезпечення, моделей та сутностей. Застосовано метод вебпрограмування при реалізації клієнтської та серверної частини вебплатформи. Для створення адаптивного та зручного інтерфейсу користувача було використано методи дизайну.

Інформаційна база досліджень складається з технічної документації до обраних технологій. Важливе місце займають навчально-методичні матеріали зі спеціальності, які були фундаментом вибору архітектурних рішень. А також використано результати аналізу наявних вебплатформ для управління особистими фінансами, а також відкриті джерела інформації, що дозволило визначити сучасні підходи до розробки подібних систем.

Практична цінність полягає у використанні віддаленої бази даних Railway, завдяки якій можливе розгортання вебплатформи на Vercel, яка надає доступ до використання системи для будь-якого користувача через мережу Інтернет. Підвищуючи практичну цінність, бо програмний продукт може бути експлуатований у реальних умовах.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	9
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ ПЛАНУВАННЯ ТА КОНТРОЛЮ ВЕБПЛАТФОРМИ ОСОБИСТИХ ФІНАНСІВ

1.1 Постановка задачі

Метою кваліфікаційної роботи бакалавра є розроблення вебплатформи для управління особистими фінансами з використанням сучасних програмних технологій, які забезпечують зручність роботи користувача, надійне збереження даних і можливість подальшого масштабування системи. Розроблюваний програмний продукт має допомогти користувачам вести облік доходів і витрат, контролювати фінансові операції, аналізувати власну платіжну поведінку та приймати більш обґрунтовані рішення щодо використання коштів.

У сучасних умовах особисті фінанси дедалі частіше розподіляються між кількома джерелами: банківськими рахунками, картками, готівкою, електронними сервісами та іншими фінансовими інструментами. Через це користувачеві складно швидко оцінити реальний стан власного бюджету. Доходи можуть надходити з різних джерел, витрати здійснюються щодня, а частина операцій часто залишається непоміченою. У результаті зростає ризик імпульсивних покупок, перевищення запланованих витрат і втрати контролю над загальною фінансовою ситуацією.

Проблема ускладнюється тим, що багато користувачів не ведуть регулярний облік коштів або роблять це вручну. Наприклад, записи у таблицях чи нотатках потребують постійного оновлення, ручного підрахунку та самостійного групування витрат за категоріями. Такий підхід займає багато часу й не завжди дає точну картину, особливо якщо кількість транзакцій велика. Крім того, ручний облік підвищує ймовірність помилок: користувач може забути внести операцію, неправильно вказати суму або віднести витрату до невідповідної категорії.

Застосування інформаційних технологій дає змогу автоматизувати ці процеси та зробити контроль фінансів більш системним. Вебплатформа

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	10
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

управління особистими фінансами може об'єднати основні дії користувача в одному середовищі: створення рахунків, додавання транзакцій, налаштування категорій, формування бюджетів, перегляд статистики та аналіз витрат за певний період. Завдяки цьому користувач отримує не окремі розрізнені записи, а структуровану фінансову інформацію, яку можна швидко переглянути, порівняти й використати для планування.

Важливою перевагою вебплатформи є доступність з різних пристроїв. На відміну від локальних програм, які можуть бути прив'язані до одного комп'ютера, вебзастосунок дозволяє працювати з фінансовими даними через браузер у будь-який зручний час. Це особливо важливо для користувачів, які часто здійснюють операції протягом дня й хочуть одразу фіксувати витрати або перевіряти стан бюджету. Такий формат також спрощує подальше розширення функціоналу, оскільки нові можливості можуть додаватися централізовано без потреби встановлювати оновлення на кожному пристрої.

У межах предметної області автоматизації потребують основні процеси обліку й контролю особистих фінансів. До них належать реєстрація та авторизація користувачів, створення профілю, збереження даних про рахунки, внесення доходів і витрат, категоризація операцій, контроль бюджетних лімітів, перегляд історії транзакцій і формування аналітики. Важливо також подавати фінансову статистику у зрозумілому вигляді, адже візуалізація допомагає швидше помітити перевитрати, регулярні платежі та зміни у фінансовій поведінці.

Під час постановки задачі потрібно врахувати точність обробки фінансових даних. Система має коректно зберігати суми операцій, виконувати розрахунки без втрати точності, підтримувати різні категорії витрат і не допускати змін, що порушують цілісність інформації. Для фінансової платформи це принципово, оскільки навіть невелика помилка в обчисленнях може вплинути на підсумки й знизити довіру користувача.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	11
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Окремим завданням є побудова логічної структури даних. Кожна транзакція повинна бути пов'язана з конкретним користувачем, рахунком і категорією. За потреби вона також може прив'язуватися до бюджету або інших елементів системи. Така організація зменшує дублювання, спрощує пошук і забезпечує коректне формування звітів. Крім того, продумана база даних створює основу для подальшого розвитку платформи, зокрема додавання нових типів операцій, розширеної аналітики чи інтеграції із зовнішніми сервісами.

Аналіз аналогів показує, що стандартний функціонал систем управління особистими фінансами зазвичай охоплює облік доходів і витрат, категоризацію транзакцій, графіки, планування бюджету та перегляд фінансової історії. Водночас не всі такі рішення є зручними або достатньо гнучкими. Частина застосунків має перевантажений інтерфейс, інші пропонують лише базові можливості або погано адаптовані до мобільних пристроїв. Тому під час розроблення власної вебплатформи важливо поєднати необхідний фінансовий функціонал із простим інтерфейсом, надійною архітектурою та можливістю масштабування.

Розроблювана система має забезпечувати зручну роботу користувача з фінансовими даними. Для цього потрібні зрозуміла навігація, прості форми введення, перевірка даних перед збереженням і швидке відображення результатів. Користувач повинен без зайвих дій додавати операції, переглядати витрати за категоріями, оцінювати залишки на рахунках і контролювати виконання бюджету. Чим простіші базові дії, тим вища ймовірність регулярного використання системи.

Особливу увагу слід приділити безпеці, оскільки платформа працює з персональними та фінансовими даними. Необхідно реалізувати захищену авторизацію, обмеження доступу до чужих даних, перевірку прав користувачів і надійне збереження інформації. Адміністративний функціонал також має бути доступний лише відповідним ролям, щоб запобігти несанкціонованому

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	12
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

керуванню користувачами або системними параметрами. Для такої платформи безпека є базовою умовою коректної роботи.

З урахуванням цих особливостей основне завдання роботи полягає у створенні вебплатформи, яка автоматизує облік особистих фінансів і надає інструменти для контролю доходів, витрат і бюджетів. Для цього потрібно визначити функціональні вимоги, спроектувати базу даних, обрати архітектуру застосунку, реалізувати клієнтську та серверну частини, налаштувати авторизацію й перевірку даних, а також протестувати основні сценарії роботи.

Отже, постановка задачі не обмежується створенням вебзастосунку для введення фінансових операцій. Йдеться про цілісну систему, яка поєднує облік, аналіз, планування та захист даних. Така платформа має допомогти користувачеві краще розуміти власні фінансові звички, вчасно реагувати на перевитрати й планувати використання коштів на основі достовірної інформації.

1.2 Аналіз аналогів програмного продукту

Розробка вебплатформи управління особистими фінансами потребує аналізу конкурентів на ринку задля вирішення проблематики застосунків та покращення оптимальних архітектурних рішень для майбутньої платформи. Було розглянуто наступні аналоги вже наявних систем. Були обрані саме найбільш показові та популярні програмні продукти на ринку, такі як : Wallet by BudgetBakers (рис.1.1), YNAB (рис.1.2) та Spendee (рис.1.3).

Wallet by BudgetBakers —один з популярних застосунків для управління особистими фінансами, який дозволяє відстежувати доходи та витрати, планувати бюджет і синхронізувати дані з банківськими рахунками.

Під час аналізу було виявлено головну перевагу — це інтеграційні можливості з банківськими системами. Wallet підтримує автоматичну синхронізацію з понад п'ятнадцятьма тисячами фінансових установ у всьому світі, включаючи також українські банки. Це дозволяє транзакціям з'являтися в додатку миттєво та без витрати часу під час ручного введення обліку фінансів. Крім того, додаток має дуже гнучку систему розподілу категорій та аналітики.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	13
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Можна створювати складні структури підкатегорій, використовувати мітки для окремих проєктів та отримувати детальні візуальні звіти про грошовий потік. Сервіс є хорошим прикладом для фінансового обліку.

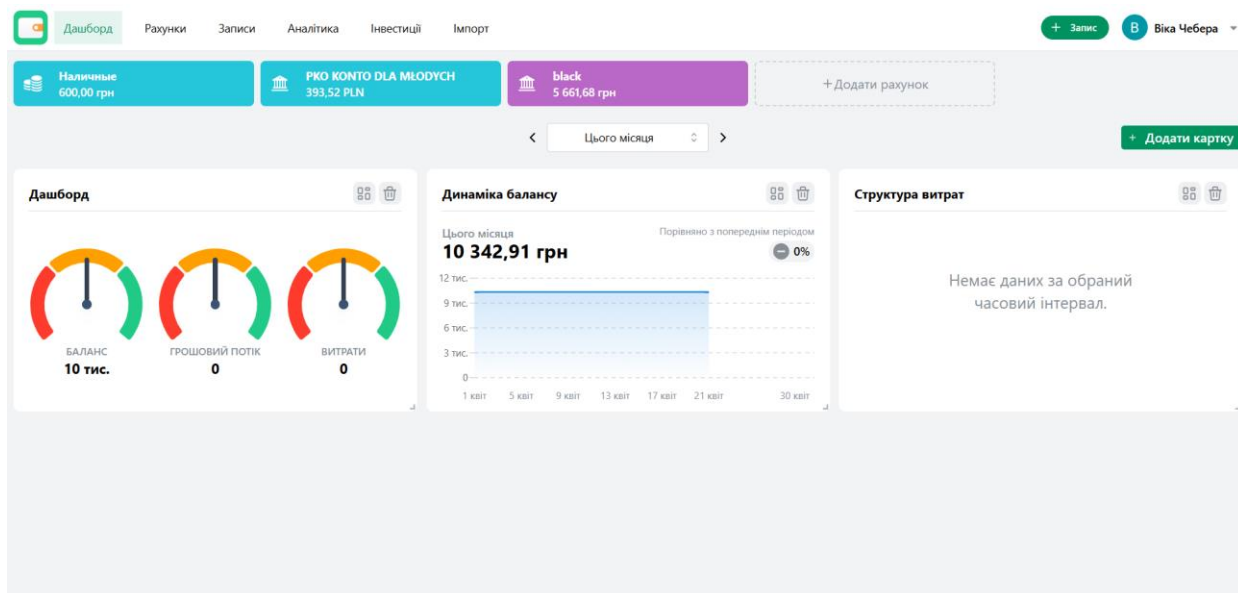


Рисунок 1.1 – Сторінка додатка Wallet

Водночас аналіз сервісу Wallet від BudgetBakers (рис.1.1) виявив чимало проблем, що заважають комфортному контролю грошей. Перше, що може перешкоджати користувачеві в роботі, — це те, що автосинхронізація банків є Premium-функцією. Також було виявлено ряд проблем з некоректним балансом. Користувач змушений самотійно вичищати дублікати транзакцій та вручну перепідключати банки, що замість економії часу лише додає роздратування. Безкоштовна версія досить обмежена, що нагадує звичайний блокнот, а повна функціональність доступна тільки з Premium-підпискою, яка не гарантує стабільної роботи без багів.

На противагу, розроблена вебплатформа розрахована на потреби українського користувача та передбачає інтеграцію з Monobank через персональний токен. Додатково, система має адміністративну панель, яка забезпечує керування ролями користувачів, аудит-логами, які відстежують підозрілу активність користувача, та реалізує управління категоріями для фінансових операцій. Роблячи управління доступнішим як користувачу, так і адміністратору системи.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	14
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

YNAB (You Need A Budget) — ще один аналог на ринку серед систем управління особистими фінансами, який має принцип робити — заздалегідь розподіл коштів за категоріями. На відміну від інших програм, які просто показують, скільки ви витратили, YNAB змушує користувача планувати витрати. Сервіс базується на методології «нульового бюджетування» — кожна зароблена гривня/долар повинна мати призначення ще до моменту її витрати.

Під час аналізу цієї вебплатформи було виявлено низку недоліків. Найголовніший мінус — це відсутність української локалізації. Користувач в Україні змушений використовувати англomовний інтерфейс та адаптуватися до західних фінансових термінів, що ускладнює ведення обліку для людей, які знають тільки українську мову. Крім того, незважаючи на свою функціональність, візуальна складова YNAB виглядає застарілою на тлі сучасних вебстандартів. Його дизайн є досить статичним і консервативним.

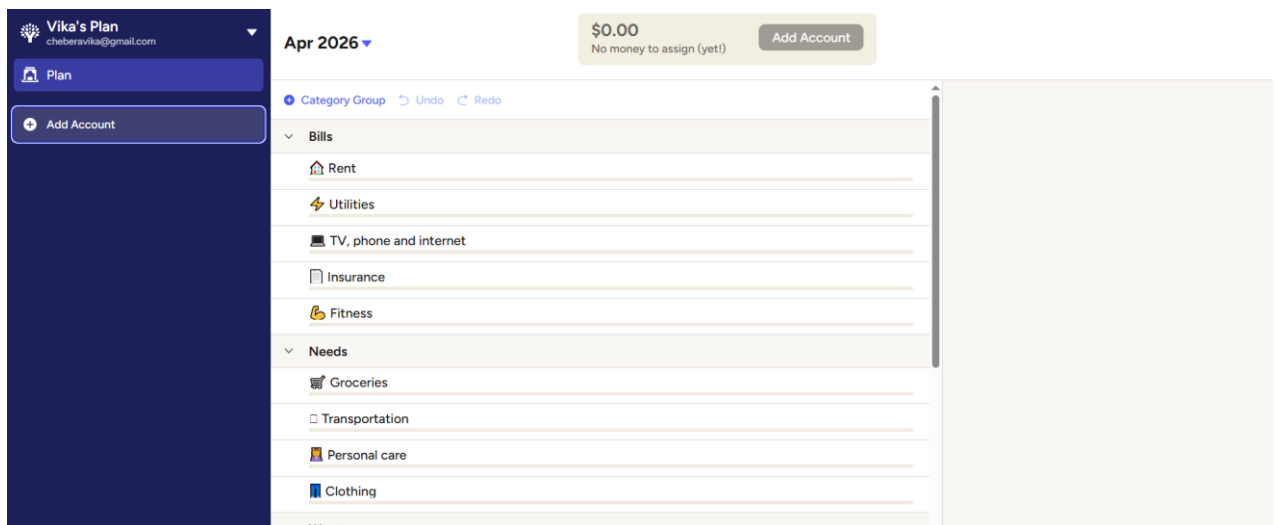


Рисунок 1.2 – Головний екран платформи YNAB

На відміну від YNAB, проєкт розроблено з урахуванням локалізації української мови, що робить фінансовий інструмент максимально доступним для українського користувача. А завдяки використанню сучасного стеку технологій, зокрема повностековий вебдодаток на Next.js, вебплатформа має сучасний, динамічний інтерфейс, який забезпечує кращий досвід взаємодії.

Наступним аналогом є Spendee - мобільний та вебдодаток для обліку фінансів, який дозволяє користувачам створювати власні категорії витрат,

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	15
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

відстежувати доходи та аналізувати фінансові показники за допомогою графіків і діаграм.

На головному екрані (рис.1.3) можна спостерігати досить сучасний та зрозумілий інтерфейс, з яким буде легко працювати користувачеві без досвіду.

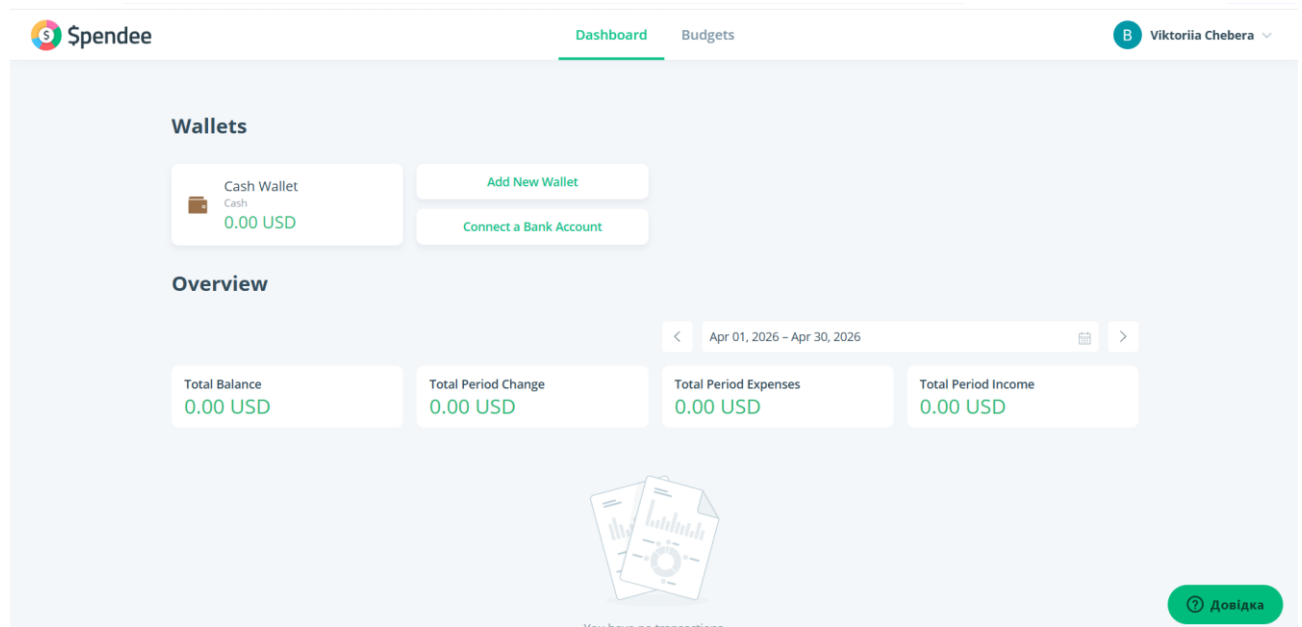


Рисунок 1.3 – Головний екран Spendee

Серед ключових переваг платформи варто виділити функцію створення спільних гаманців (Shared Wallets). Це зручне рішення для сімей або пар, яке дозволяє кільком людям спільно вести бюджет в режимі реального часу. Також величезним плюсом Spendee є потужна візуалізація: яскраві інтерактивні діаграми перетворюють суху статистику на зрозумілу картину, яка дійсно мотивує слідкувати за коштами. Додаток також пропонує готові шаблони категорій, що економить час на початковому етапі налаштування.

Проте незважаючи на привабливий дизайн, Spendee має кілька суттєвих недоліків. По-перше, безкоштовна версія досить сильно урізана: найкорисніші інструменти, зокрема автоматична синхронізація з банками, доступні лише в дорогій преміум-підписці. І як і у випадку з багатьма закордонними сервісами, для українського користувача тут відсутня повноцінна локалізація, а підтримка українських банків обмежена.

Аналіз додатка Spendee дозволив дійти висновку, що багатьом користувачам недостатньо функціоналу, який входить до безкоштовної підписки. Саме тому розроблена вебплатформа зосереджена на створенні повноцінної, масштабованої вебплатформи з першого дня, яка виглядатиме органічно на будь-якому екрані. Вона не ділитиме функціонал на "платний" та "безкоштовний" у рамках базового обліку.

Для підсумкового порівняння вище наведених вебплатформ було створено порівняльну таблицю (табл.1.1).

Таблиця 1.1

Порівняння аналогів вебплатформи

Критерій порівняння	YNAB	Spendee	Wallet
Основний функціонал	Нульове бюджетування, жорсткий контроль кожної гривні	Категоризація витрат, створення спільних гаманців	Ручний облік, планування, підтримка різних валют
Синхронізація з банками	Так (працює лише з закордонними банками)	Так (тільки у платній Premium версії)	Так (основна фішка, але обмежена в безкоштовній версії)
Планування бюджету	Так (розподіл доходів до їх отримання)	Так (встановлення лімітів за категоріями)	Так (встановлення цільових бюджетів)
Аналітика та звіти	Потужні звіти, графіки тенденцій	Яскраві інтерактивні діаграми, візуалізація	Зручні графіки та кругові діаграми
Локалізація	Відсутня	Відсутня	Наявна
Технології платформи	Класичний веб-сайт (МРА), додаток	Мобільний додаток, обмежена веб-версія	Класичний веб-сайт (МРА), додаток
Переваги	Найкраща методологія фінансової дисципліни, надійність	Сучасний мобільний дизайн, зручна візуалізація	Наявність української мови, зручна робота з валютами

продовж. табл. 1.1

Недоліки	Висока ціна, застарілий UI, не працює в Україні	Слабка веб-версія, обмежений безкоштовний тариф	Веб-версія сильно поступається мобільній, працює повільно
----------	---	---	---

Таким чином, проведений огляд аналогів показав, що жоден із наявних на ринку сервісів не вирішує проблему комплексно. З одного боку, такі системи, як YNAB та Spendee, пропонують потужний функціонал, але повністю ігнорують українського користувача, не маючи української локалізації та вимагаючи плати за базові речі. З іншого боку, Wallet, хоча й має українську мову та гарний мобільний додаток, але всі функції додатка — тільки з преміум-підпискою.

Основним завданням для запропонованої системи є створення безкоштовного багатофункціонального застосунку. На основі сучасного фреймворку Next.js (з використанням React та TypeScript), що дозволяє реалізувати швидку повностекову вебплатформу. Тому в результаті користувач отримає повноцінний, багатофункціональний та сучасний інструмент для управління фінансами без купівлі Premium-підписок. А адміністратори платформи — зручне відстеження безпеки системи, управління контентом та керування користувацьким доступом.

1.3 Вибір архітектури вебплатформи

Вебплатформу побудовано на клієнт-серверній архітектурі, де серверна частина реалізована у вигляді REST API. Для розробки обрано фреймворк **Next.js**, який поєднує функціонал для створення клієнтської та серверної частин застосунку в єдиному проєкті. Фреймворк підтримує Server-Side Rendering (SSR), що збільшує швидкість завантаження сторінок та покращує їх SEO-оптимізацію, а також Static Site Generation (SSG) — метод генерації статичного HTML-контенту під час збірки додатка, що зменшує навантаження на сервер [2,11]. Крім того, Next.js підтримує клієнтський рендеринг для створення динамічного та інтерактивного контенту. Важливою перевагою є вбудована

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	18
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

підтримка TypeScript, що збільшує надійність розробки завдяки строгій типізації коду [3,9,28].

Для розробки користувацького інтерфейсу обрано бібліотеку **React**. Вона дозволяє створювати багаторазові компоненти та зручно керувати станами платформи. Бібліотека підтримує принцип DRY (Don't Repeat Yourself), що уникає дублювання коду, прискорює створення інтерфейсів та полегшує подальшу підтримку проєкту. Механізм Virtual DOM мінімізує кількість операцій з оновлення реальної сторінки в браузері, що підвищує швидкість роботи вебзастосунку [4,26]. Під час розробки інтерфейсу враховано принципи зручності та зрозумілості взаємодії користувача з системою. Основні елементи керування розміщені так, щоб користувач міг швидко знайти потрібну дію: додавання транзакції, перегляд бюджету, зміну рахунку або перехід до статистики. Такий підхід зменшує когнітивне навантаження та робить роботу з вебплатформою більш інтуїтивною [20].

Для стилізації інтерфейсу та створення адаптивного дизайну обрано фреймворк **Tailwind CSS**. Це сучасний інструмент класу Utility-First, який пропонує набір готових низькорівневих класів для стилізації елементів безпосередньо в HTML або JSX-кодi. Такий підхід значно прискорює процес верстки, оскільки зникає потреба в написанні окремих CSS-файлів та придумуванні назв для класів [17].

За механізм автентифікації та керування сесіями відповідає бібліотека **NextAuth.js**. Вона забезпечує підтримку різних способів входу в систему, зокрема за допомогою електронної пошти та сервісу Google OAuth 2.0, що суттєво спрощує реалізацію безпечної автентифікації [7].

Для безпечної автентифікації та авторизації використано **JWT** (JSON Web Token) — стандарт для створення компактних автономних токенів безпеки. Уся інформація про користувача зберігається всередині токена, що дозволяє виконувати фонові запити та оновлювати сторінки без повторної перевірки сесії в базі даних. Сервер не потребує зберігання стану сесії, що підвищує

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	19
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

продуктивність системи [10]. Для безпечного зберігання паролів користувачів у базі даних застосовано алгоритм гешування Bcrypt, який забезпечує високий рівень захисту шляхом використання солі та множинного хешування.

Сервіс **Google OAuth 2.0** — це протокол авторизації, який дозволяє автентифікувати користувача через його Google-акаунт. Перевагами є автоматичне підтвердження електронної пошти користувача та відсутність ризику витоку пароля, оскільки вебплатформа його не отримує. Крім того, використання цього сервісу збільшує конверсію, оскільки вхід займає лише кілька секунд і не потребує заповнення реєстраційної форми [8].

Вибір СУБД **MySQL** обумовлений її надійністю та високою ефективністю роботи з великими обсягами даних, що критично для вебплатформи, де кожен користувач може мати тисячі транзакцій. Підтримка ACID-транзакцій гарантує коректність при одночасному додаванні кількох операцій без втрат або пошкодження даних. Індексація полів пришвидшує пошук потрібних записів без перебирання всіх таблиць [1,18,25].

Для взаємодії між застосунком та базою даних використано **Prisma ORM**. Цей інструмент має глибоку інтеграцію з Next.js та повну підтримку TypeScript, забезпечуючи автоматичну генерацію типів на основі структури бази даних. Prisma пропонує зручні методи для створення, читання та оновлення даних без написання сирого SQL-коду [5].

Для розгортання бази даних на хостингу використано хмарну платформу **Railway**, що дозволило швидко налаштувати середовище та автоматизувати процеси деплою. Платформа пропонує вбудовані бази даних, зокрема MySQL, які можна підключити в один клік [12].

Хостинг вебплатформи реалізовано на **Vercel** — сучасній хмарній платформі, оптимізованій для застосунків на базі Next.js. Vercel забезпечує автоматичне розгортання проєкту з репозиторію GitHub, швидку доставку контенту завдяки глобальній мережі CDN, високу доступність сервісу та можливість масштабування відповідно до потреб користувачів [13].

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	20
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Контроль версій здійснювався за допомогою **Git** — розподіленої системи керування версіями, що дозволило ефективно відстежувати зміни в проєкті, зберігати історію розробки та мати можливість швидкого повернення до попередніх станів системи [22].

Висновки до першого розділу

У першому розділі кваліфікаційної роботи було комплексно досліджено предметну область та обґрунтовано необхідність цифровізації процесів управління особистими фінансами. Аналіз наявних рішень на ринку, зокрема YNAB, Spendee та Wallet, допоміг виявити їхні сильні сторони, а також суттєві недоліки. Більшість популярних світових сервісів має надто узагальнений характер, ігнорує українського користувача (відсутня локалізація) та часто вимагає плати за базовий функціонал. Відсутність зручного та локалізованого інструменту для ведення фінансової аналітики стала головною передумовою для розробки системи.

З урахуванням виявлених недоліків конкурентних продуктів було визначено функціональні можливості, що повністю відповідають сформованому технічному завданню. Система має забезпечувати зручну авторизацію через Google-сервіс і вести надійний журнал дій для контролю безпеки. Також передбачено гнучкий облік різних типів рахунків із підтримкою мультивалютності. Окрему роль відіграє автоматична категоризація імпортованих банківських транзакцій за кодами МСС, оскільки вона скорочує час, який користувач витрачає на ручне впорядкування операцій. Крім того, система повинна підтримувати детальне планування бюджету з можливістю встановлення лімітів для окремих категорій.

Для реалізації цих функцій було обґрунтовано вибір сучасного технологічного стеку. Для побудови інтерфейсу обрано Next.js, React та TypeScript, для роботи з даними — Prisma ORM та MySQL, а для безпеки — Google OAuth 2.0, JWT та Bcrypt.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	21
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Практична цінність результатів, отриманих у цьому розділі, полягає в тому, що спроектована логічна й фізична архітектура формує надійну основу для подальшої розробки. Завдяки продуманій структурі бази даних система зможе швидко й без зайвих труднощів масштабуватися зі зростанням кількості користувачів.

Подальша робота над проєктом передбачатиме написання серверної логіки та створення клієнтського інтерфейсу. Також планується підключити додаток до реальних банківських API, щоб автоматично завантажувати виписки. Ще одним важливим кроком стане додавання інтерактивних графіків статистики, які зроблять аналітику більш наочною та корисною для користувача.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	22
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ОСОБИСТИХ ФІНАНСІВ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

З метою створення якісної та надійної вебплатформи управління особистими фінансами було визначено узагальнені вимоги до програмного продукту, що відповідають сучасним підходам до проєктування веборієнтованих систем [19,21]. Основна увага приділялася забезпеченню коректної роботи механізмів авторизації, захисту персональних даних, розмежування прав доступу, а також зручному обліку фінансових операцій і аналізу даних користувача.

Розроблений перелік функціональних і нефункціональних вимог систематизовано у каталозі вимог.

На підставі аналізу потреб користувачів було сформовано набір базових сценаріїв взаємодії з платформою. Сукупність цих сценаріїв відображена на діаграмі варіантів використання, яка демонструє функціональні можливості системи та взаємодію між користувачами і реалізованими сервісами [6,15].

У структурі вебплатформи передбачено дві основні ролі: адміністратор і користувач.

Адміністратор здійснює контроль та підтримку функціонування платформи, включаючи управління користувачами, блокування та видалення облікових записів, зміну ролей, тим самим змінюючи права доступу. Також адміністратор має можливість створювати, редагувати та видаляти категорії доходів і витрат, контролювати коректність даних і відстежувати події системи за допомогою журналу аудиту.

Користувач є основним учасником системи, тому йому необхідно мати можливість реєструватися та авторизуватися за допомогою логіна і пароля або через обліковий запис Google. Користувач може керувати власним профілем, додавати, редагувати та видаляти фінансові операції, переглядати історію

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	23
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

транзакцій, виконувати пошук і фільтрацію даних. Також система забезпечує ведення особистих рахунків, контроль залишків коштів, створення та налаштування бюджетів, перегляд статистики й аналітики фінансової діяльності. Для автоматичної синхронізації транзакцій передбачена інтеграція з банківським сервісом, а у разі перевищення ліміту бюджету користувач отримує відповідне повідомлення. Крім того, користувач може обрати основну валюту системи, вийти з облікового запису або видалити власний акаунт.

Узагальнену характеристику ролей та їх функціональних можливостей наведено в (таблиці 2.1).

Таблиця 2.1

Виявлення акторів

Актор	Короткий опис
Адміністратор	Здійснює управління користувачами системи, включаючи перегляд, зміну ролей, блокування та видалення облікових записів. Виконує керування категоріями доходів і витрат. Має доступ до журналу аудиту для моніторингу подій у системі та контролює коректність функціонування вебплатформи.
Користувач	Виконує облік особистих фінансів шляхом додавання, редагування та видалення транзакцій доходів і витрат. Створює та управляє власними рахунками, планує бюджети, контролює витрати за категоріями та аналізує фінансовий стан за допомогою статистики і графічної аналітики. Має можливість підключати банківський сервіс для автоматичної синхронізації транзакцій, переглядати історію фінансових операцій та редагувати власний профіль.

На основі цих характеристик сформовано детальний перелік варіантів використання для кожної з ролей. Зокрема, функціональні обов'язки адміністратора, спрямовані на управління платформою та контроль безпеки, систематизовано у (таблиці 2.2).

Таблиця 2.2

Виявлення варіантів використання для адміністратора

Основний актор	Найменування	Формулювання
Адміністратор	Керування користувачами	Адміністратор може переглядати список користувачів, блокувати або видаляти акаунти у разі порушень, а також контролювати їх активність

продовж. табл. 2.2

	Моніторинг системи	Адміністратор контролює стабільність роботи вебплатформи, відстежує помилки та забезпечує коректне функціонування основних модулів системи
	Керування категоріями	Адміністратор має можливість створювати, редагувати та видаляти категорії доходів і витрат, які використовуються користувачами під час обліку фінансових операцій
	Керування МСС	Адміністратор може налаштовувати відповідність МСС категоріям для автоматичної класифікації банківських транзакцій
	Перегляд журналу аудиту	Адміністратор здійснює моніторинг подій у системі, переглядає інформацію про дії користувачів та результати виконання операцій
	Керування правами доступу	Адміністратор визначає рівні доступу користувачів до функціональних можливостей системи шляхом призначення та зміни ролей

Основним актором системи є звичайний користувач, для якого реалізовано найширший набір функцій, пов'язаних безпосередньо з фінансовим обліком. Перелік варіантів використання, доступних користувачеві для роботи з платформою, наведено у (таблиці 2.3).

Таблиця 2.3

Виявлення варіантів використання для користувача

Користувач	Реєстрація та вхід	Користувач може створити обліковий запис або увійти в систему, щоб отримати доступ до функціоналу платформи
	Управління транзакціями	Користувач може додавати фінансові операції (доходи та витрати), вказуючи суму, категорію, рахунок, дату та опис операції. А також змінювати або видаляти раніше створені фінансові операції
	Перегляд історії транзакцій	Дає можливість переглядати всі фінансові операції у хронологічному порядку, використовуючи пошук та фільтрацію даних.
	Пошук та фільтрація даних	Користувач може здійснювати пошук транзакцій та відбирати їх за критерієм типу операції.
	Управління рахунками	Користувач може створювати, редагувати та видаляти рахунки для обліку коштів (банківські картки, рахунки, готівка тощо).
	Інтеграція з банківськими сервісами	Користувач може підключати банківські акаунти та автоматично синхронізувати транзакції з системою.
	Планування бюджету	Користувач встановлює бюджетні обмеження для окремих категорій витрат та контролює їх виконання.

	Перегляд аналітики	Користувач переглядає статистику, графіки та баланс для аналізу фінансового стану.
	Перегляд особистих даних	Користувач має можливість переглядати інформацію власного профілю та персональні налаштування.
	Керування профілем	Користувач може переглядати та редагувати особисті дані (ім'я, email, налаштування).
	Отримання сповіщень	Користувач отримує повідомлення про перевищення бюджету.

Для наочної взаємодії між елементами додатка було створено діаграму варіантів використання адміністратора (рис.2.1) та діаграму варіантів використання користувача (рис.2.2).[6]

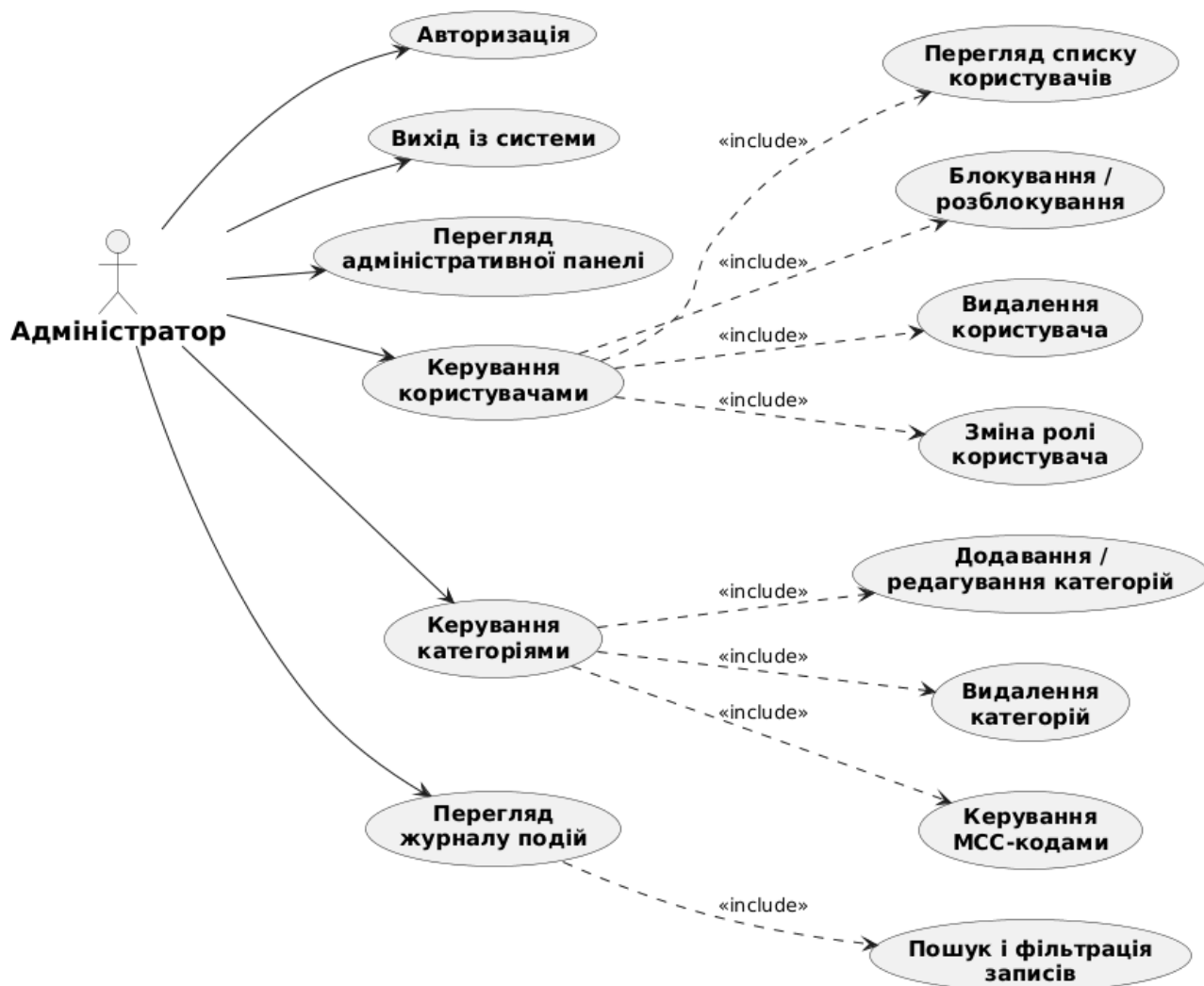


Рисунок 2.1 – Діаграма варіантів використання адміністратора

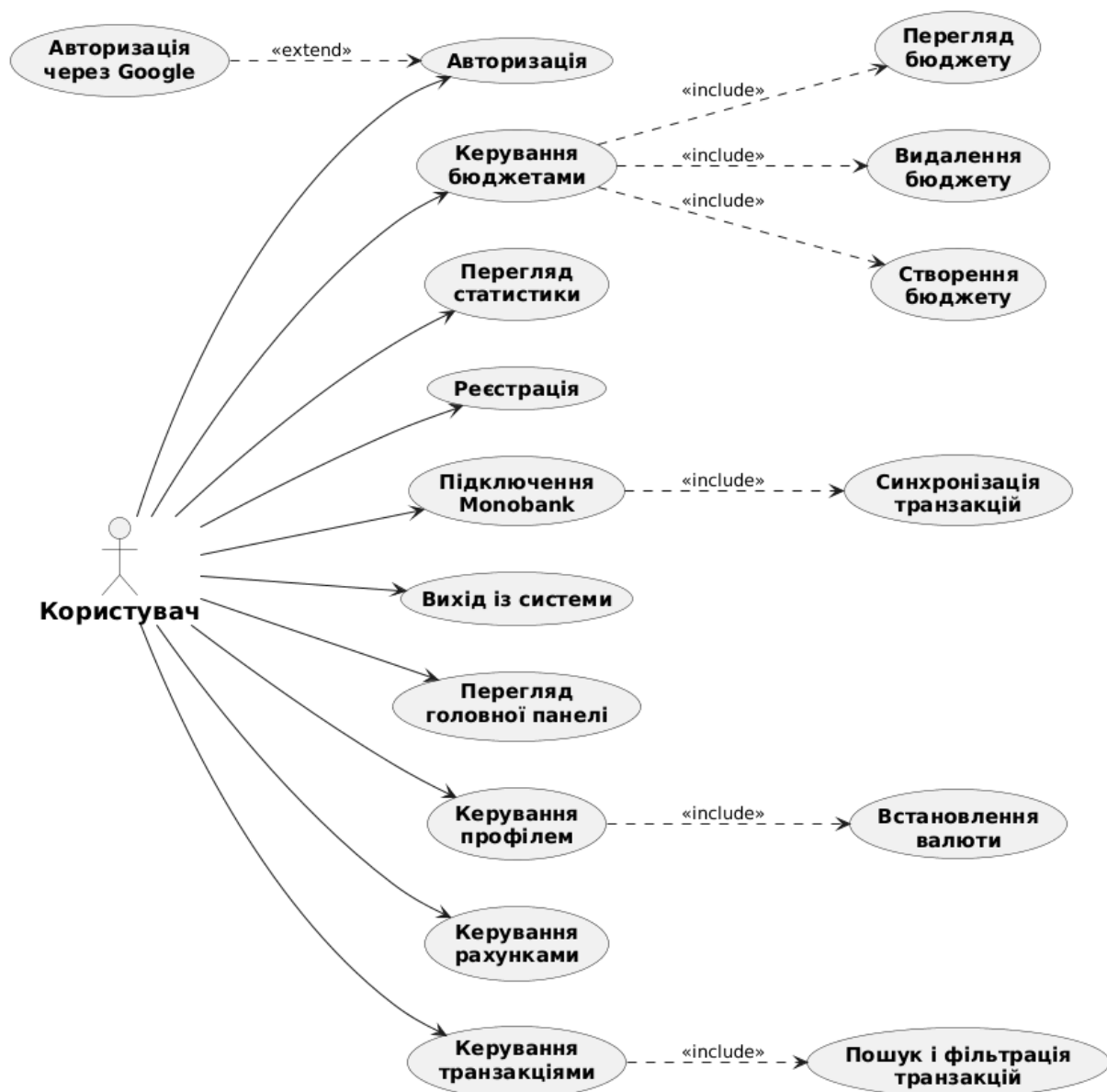


Рисунок 2.2 – Діаграма варіантів використання користувача

Розглянемо функціональні вимоги, які ставляться до користувачів системи. По-перше, мають бути передбачені реєстрація та авторизація користувачів, а також можливість входу через обліковий запис Google. Користувач повинен мати доступ до власного профілю, переглядати його дані та за потреби редагувати їх. Також система має забезпечувати роботу з фінансовими операціями: додавання, редагування та видалення доходів і витрат.

Окремо має бути реалізовано ведення обліку фінансових рахунків користувача, створення бюджетів і керування ними. Транзакції повинні класифікуватися за категоріями, що дозволяє спростити аналіз фінансової

активності користувача. Для зручності система має підтримувати пошук і фільтрацію фінансових операцій, а також відображати статистику, графіки та узагальнені показники фінансового стану.

Платформа повинна підтримувати інтеграцію з банківськими сервісами для автоматичного завантаження рахунків і транзакцій. Крім того, у системі необхідно реалізувати розмежування прав доступу між звичайним користувачем і адміністратором. Адміністратор має отримати можливість керувати користувачами, категоріями та кодами МСС. Також система повинна вести журнал аудиту, у якому фіксуються важливі дії користувачів і адміністраторів.

Не функціональні вимоги вебплатформи також не менш важливі. Вебплатформа має бути простою для сприйняття та швидкою в освоєнні. Для нового користувача час ознайомлення з основними можливостями системи повинен становити від 3 до 5 хвилин, а для досвідченого користувача — менше 2 хвилин. Інтерфейс має бути інтуїтивно зрозумілим, зручним і не вимагати додаткової підготовки.

Час відповіді вебплатформи на типові дії користувача не повинен перевищувати 5 секунд. Для більш складних серверних операцій, зокрема синхронізації з банківським сервісом та обробки імпортованих транзакцій, час виконання може бути більшим, однак система повинна надавати користувачеві зрозуміле повідомлення про стан виконання операції.

Система повинна бути надійною та забезпечувати стабільну роботу вебплатформи. Також необхідно передбачити достатню продуктивність, щоб декілька користувачів могли одночасно працювати з вебресурсом без помітного погіршення швидкодії. Важливою вимогою є захист даних користувачів, зокрема безпечна автентифікація, розмежування прав доступу та фіксація важливих дій у журналі аудиту.

Щодо системних вимог, то платформа має коректно працювати на пристроях із мінімальними технічними характеристиками. Для використання вебплатформи потрібно не менше 2 Гб оперативної пам'яті, процесор із тактовою

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	28
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

частотою не нижче 1.4 GHz, доступ до мережі Інтернет і сучасний веббраузер із підтримкою JavaScript.

Серверна частина інформаційної системи повинна підтримувати виконання вебдодатка на основі Next.js, використання системи керування базами даних MySQL та Prisma ORM для взаємодії з базою даних. Також має бути передбачена можливість розгортання системи на хмарних платформах, зокрема Vercel для вебдодатка та Railway для бази даних.

Архітектура системи була реалізована за принципом єдиного повностекового моноліту (Full-Stack Monolith) з використанням фреймворку Next.js. Вона містить у собі як клієнтську частину, так і серверну логіку через механізм Route Handlers. Фронтенд взаємодіє з бекендом асинхронно через внутрішні REST-запити у форматі JSON. Такий вибір архітектури зумовлений кількома факторами. По-перше, щоб не розділяти її на фронтенд та окремий бекенд: єдина типізація TypeScript-моделі бази даних використовується як на фронтенді, так і в API-роутах, що виключає помилки десеріалізації. Така архітектура також спрощує деплоймент, оскільки весь додаток розгортається як єдиний екземпляр на платформі Vercel за допомогою Git-інтеграції. Та наостанок для обсягу логіки управління особистими фінансами створення мікросервісної архітектури або роздільних репозиторіїв є надмірним ускладненням, яке несе лише додаткові витрати на DevOps без очевидної вигоди в продуктивності.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	29
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

для реалізації обліку фінансових операцій, керування рахунками, планування бюджетів, класифікації транзакцій, інтеграції з банківськими сервісами та контролю дій користувачів [6,15]. Діаграма дозволяє наочно представити внутрішню організацію даних системи і показати, як окремі сутності взаємодіють у межах єдиного монолітного веб-додатка (рис. 2.3).

На діаграмі класів представлено логічну структуру вебплатформи управління особистими фінансами, яка відображає основні сутності предметної області, їхні атрибути та взаємозв'язки. Центральним класом системи є User, оскільки більшість функціональних можливостей платформи пов'язана саме з обліковим записом користувача. Користувач може мати одну роль, що визначає його права доступу в системі, а одна роль може бути призначена багатьом користувачам. Це забезпечує розмежування можливостей між звичайним користувачем та адміністратором.

Клас User пов'язаний з фінансовими рахунками, транзакціями, бюджетами, сповіщеннями, інтеграціями та журналом аудиту. Така структура дозволяє зберігати всі фінансові дані окремо для кожного користувача та забезпечує персоналізовану роботу з платформою. Фінансовий облік у системі базується на взаємодії класів BankAccount, Transaction, Category і Budget. Клас BankAccount описує рахунок користувача, на якому зберігається баланс у визначеній валюті. З кожним рахунком можуть бути пов'язані фінансові операції, що представлені класом Transaction.

Клас Transaction є однією з ключових складових системи, оскільки він відображає окрему фінансову операцію користувача. Транзакція обов'язково належить певному користувачу та категорії, а також може бути пов'язана з рахунком і бюджетом. Такий підхід дозволяє не лише зберігати інформацію про доходи й витрати, а й використовувати ці дані для подальшого аналізу, побудови статистики та контролю виконання бюджетів [18,27].

Для класифікації фінансових операцій використовується клас Category. Він має ієрархічну структуру, оскільки категорія може містити підкатегорії. Це дає

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	31
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

зможу деталізувати витрати та доходи, наприклад, поділяти загальну категорію на більш конкретні напрямки. Додатково, з категоріями пов'язаний клас MscMapping, який використовується для автоматичного визначення категорії банківської транзакції за МСС. Завдяки цьому імпортовані з банківського сервісу операції можуть класифікуватися без ручного вибору категорії користувачем.

Планування фінансів реалізовано за допомогою класів Budget і BudgetCategory. Клас Budget описує бюджет користувача за певний період часу та може бути прив'язаний до конкретного фінансового рахунку. Оскільки один бюджет може охоплювати декілька категорій витрат, використовується проміжний клас BudgetCategory. Він зберігає заплановану суму для певної категорії та фактичні витрати, що дозволяє порівнювати планові й реальні фінансові показники [18].

Для підтримки автоматизованого завантаження банківських даних у системі передбачено клас Integration. Він зберігає інформацію про підключені зовнішні сервіси, зокрема токени доступу, стан інтеграції та параметри вебхуків. Через цей клас реалізується зв'язок між обліковим записом користувача та банківським API. У результаті система може автоматично отримувати інформацію про рахунки й транзакції.

Клас Notification використовується для інформування користувача про важливі події, наприклад перевищення встановлених бюджетних обмежень. Клас AuditLog виконує службову роль і забезпечує фіксацію важливих дій користувачів у системі. Збереження інформації про дію, статус виконання, IP-адресу та додаткові деталі дозволяє підвищити контроль за роботою платформи й покращити безпеку [19,27].

Окремо на діаграмі відображено класи Account, Session і VerificationToken, які використовуються для підтримки механізму автентифікації. Вони забезпечують роботу сесій користувача, зовнішніх облікових записів та перевірочних токенів. Ці класи мають допоміжний характер, однак є важливими для реалізації входу в систему та підтримки авторизованого доступу.

Таким чином, діаграма класів демонструє не лише перелік основних сутностей системи, а й логіку їхньої взаємодії. Вона показує, як користувач пов'язаний із фінансовими рахунками, транзакціями, категоріями, бюджетами, інтеграціями та службовими механізмами системи. Така модель є основою для реалізації функціональних можливостей вебплатформи і забезпечує цілісну організацію даних у межах монолітного веб-додатка.

2.2 Опис бази даних системи

Реляційну базу даних було спроектовано на основі СУБД MySQL і реалізовано з використанням ORM (Object-Relational Mapping) інструменту Prisma Client, який подає кожну таблицю у вигляді типізованої моделі.[1,5,18]

Загальна логіка бази даних побудована навколо головної сутності — користувача. Саме з нею пов'язані фінансові дані, персональні налаштування та системні логи. Візуально структура бази даних, основні сутності та зв'язки між ними наведені на (рисунку 2.4).

Фізична схема бази даних передбачає поділ інформації на взаємопов'язані таблиці, кожна з яких відповідає за конкретний тип даних. Основними сутностями системи є користувачі, рахунки, транзакції, категорії, бюджети, банківські підключення та журнали подій. Така структура дає змогу впорядковано зберігати інформацію, уникати дублювання та підтримувати цілісність даних.

Оновлення й використання даних виконуються через Prisma Client, який забезпечує зв'язок між серверною логікою та базою даних. Перед створенням або зміною записів система перевіряє коректність даних, права доступу користувача та наявність пов'язаних сутностей. Збережена інформація використовується для фінансової статистики, розрахунку залишків, контролю бюджетних лімітів, перегляду історії операцій і ведення журналів подій.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	33
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

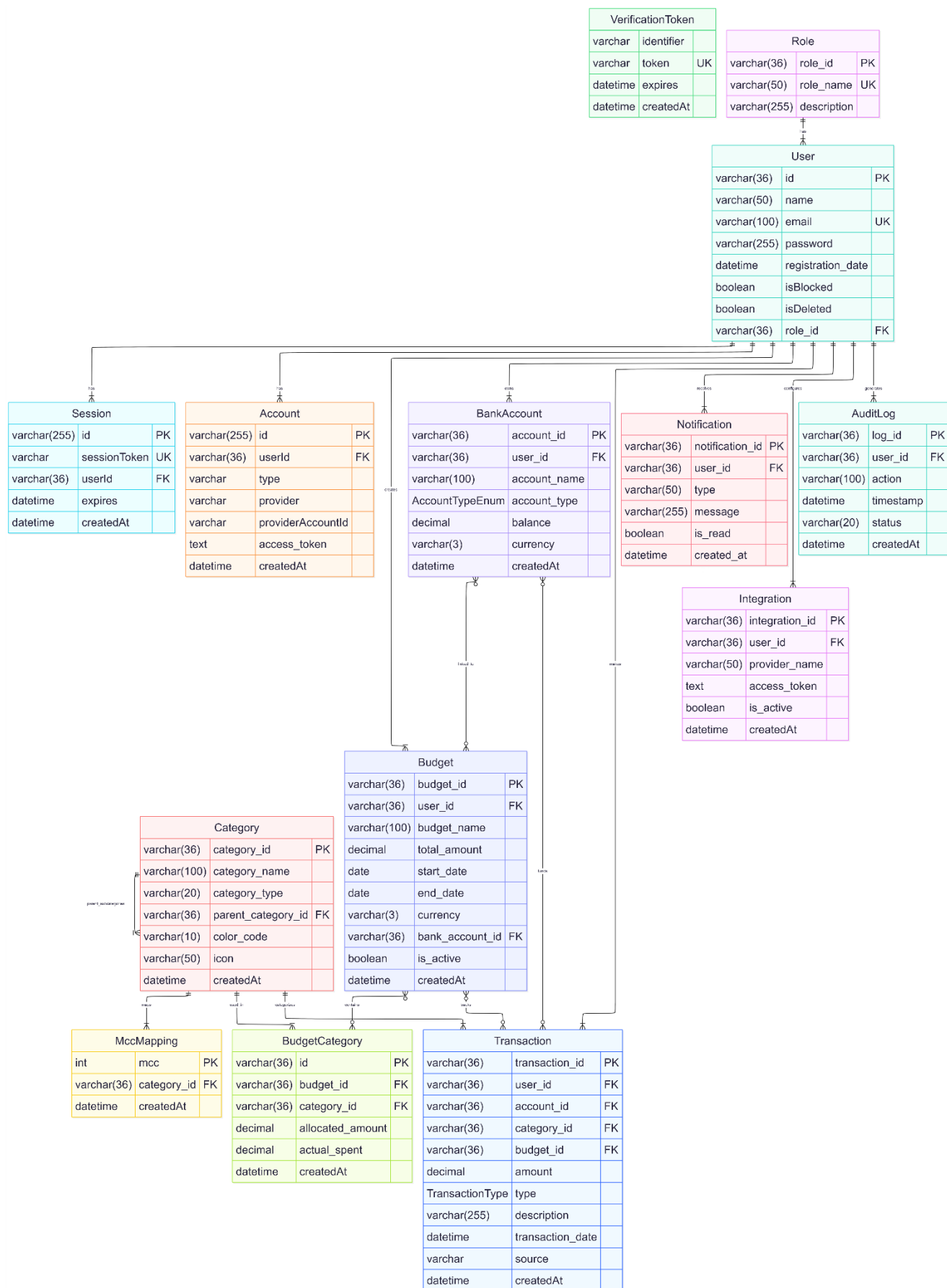


Рисунок 2.4 – Діаграма бази даних

База даних системи складається з чотирнадцяти таблиць. Розглянемо першу таблицю **User** (табл. 2.4). Таблиця зберігає облікові дані користувачів із застосуванням механізмів безпеки (ідентифікатори UUID, м'яке видалення через прапорець isDeleted, підтримка як парольного, так і OAuth-входу).

Через те що користувач є центром системи, таблиця формує низку логічних залежностей. А саме строгий зв'язок «багато-до-одного» з таблицею Role, який забороняє видалення ролі, якщо на неї посилаються користувачі. Також присутні зв'язки «один до багатьох» з технічними таблицями Account і Session для роботи автентифікації. Крім того, саме від цього запису залежать всі фінансові дані — його банківські рахунки, транзакції та бюджети, а також сервісні сутності. Наприклад, підключені банківські інтеграції, індивідуальні сповіщення при перевищенні бюджету та журнал аудиту його дій.

Таблиця 2.4

Опис таблиці User

Поле	Тип	Призначення
id	VarChar(36)	Унікальний ідентифікатор користувача (UUID)
name	VarChar(50)	Ім'я користувача
email	VarChar(100)	Електронна пошта (унікальне значення)
password	VarChar(255)	Хеш пароля (необов'язкове, якщо вхід через OAuth)
emailVerified	DateTime	Дата підтвердження електронної пошти
image	VarChar(255)	Посилання на аватар користувача
registration_date	Timestamp	Дата реєстрації в системі
last_login	Timestamp	Дата останнього входу в систему
preferred_currency	VarChar(10)	Обрана валюта за замовчуванням (напр., UAH)
isBlocked	Boolean	Прапорець блокування облікового запису
isDeleted	Boolean	Прапорець "м'якого видалення" облікового запису
role_id	VarChar(36)	Зовнішній ключ, що посилається на таблицю Roles

Наступною таблицею є Role. Ця таблиця відповідає за розмежування прав доступу користувача. Вона містить в собі такі ролі як «Адміністратор» та

«Користувач». Як зазначалося раніше при описі таблиці **User**, залежність між цими двома сутностями є асоціацією: роль є повністю незалежною сутністю, яка існує в системі незалежно від того, скільки користувачів її зараз мають. Назва ролі `role_name` має строге обмеження унікальності, щоб уникнути дублювання повноважень, тоді як поле `description` дозволяє додатково пояснити суть прав доступу для зручності адміністрування (табл. 2.5).

Таблиця 2.5

Опис таблиці Role

Поле	Тип	Призначення
<code>role_id</code>	VarChar(36)	Унікальний ідентифікатор ролі (UUID)
<code>role_name</code>	VarChar(50)	Назва ролі (унікальне значення)
<code>description</code>	VarChar(255)	Текстовий опис ролі (необов'язкове поле)

Таблиця **BankAccount** відображає фінансові ємності користувача для обліку коштів на різних типах рахунків. Зв'язок із таблицею **User** має кратність «один до багатьох»: один користувач може мати кілька рахунків, а кожен рахунок належить одному користувачу. Для цього використовується зовнішній ключ `user_id`. Зв'язок із таблицею **Transaction** також має кратність «один до багатьох», однак при видаленні рахунку транзакції не знищуються, а лише втрачають прив'язку до нього. Це дозволяє зберегти історичні фінансові дані та коректність статистики (табл. 2.6).

Таблиця 2.6

Опис таблиці BankAccount

Поле	Тип	Призначення
<code>account_id</code>	VarChar(36)	Унікальний ідентифікатор рахунку (UUID)
<code>user_id</code>	VarChar(36)	Зовнішній ключ, що посилається на таблицю Users
<code>account_name</code>	VarChar(100)	Назва рахунку, задана користувачем
<code>account type</code>	Enum	Тип рахунку (CASH, CARD, BANK_ACCOUNT, INVESTMENT)
<code>balance</code>	Decimal(15,2)	Поточний залишок коштів на рахунку
<code>currency</code>	VarChar(3)	Код валюти рахунку (наприклад, "UAH")

продовж. табл. 2.6

createdAt	Timestamp	Дата створення рахунку
updatedAt	Timestamp	Дата останнього оновлення даних рахунку

Дані про категорії містяться в таблиці **Category**. Особливістю цієї таблиці є наявність рекурсивного зв'язку (`parent_category_id`), що дозволяє будувати ієрархію. Багато підкатегорій можуть належати одній батьківській категорії. Цей зв'язок реалізований як агрегація — при видаленні батьківської категорії дочірні не знищуються, а стають самостійними. Зв'язки з таблицями `MscMapping` та `BudgetCategory` є композицією. Оскільки дані таблиці не мають сенсу без категорії. Крім того, зв'язок з таблицею `Transaction` «один-до-багатьох» реалізовано як строгу асоціацію, що забороняє видаляти категорію, якщо з нею вже пов'язані проведені транзакції, щоб не порушити фінансову звітність (табл. 2.7).

Таблиця 2.7

Опис таблиці Categories

Поле	Тип	Призначення
<code>category_id</code>	VarChar(36)	Унікальний ідентифікатор категорії (UUID)
<code>category_name</code>	VarChar(100)	Назва категорії
<code>category_type</code>	VarChar(20)	Тип категорії (наприклад, "INCOME" або "EXPENSE")
<code>parent_category_id</code>	VarChar(36)	Зовнішній ключ на цю ж таблицю (для ієрархії підкатегорій)
<code>color_code</code>	VarChar(10)	Код кольору для відображення в інтерфейсі
<code>icon</code>	VarChar(50)	Назва іконки категорії
<code>createdAt</code>	Timestamp	Дата створення
<code>updatedAt</code>	Timestamp	Дата останнього оновлення

Таблиця **MscMapping** забезпечує автоматичну категоризацію транзакцій, зіставляючи банківські МСС з внутрішніми категоріями системи. Зв'язок із таблицею `Category` має кратність «багато-до-одного» і є композицією. Тому якщо категорія видаляється, усі прив'язані до неї правила мапінгу МСС також автоматично видаляються (табл. 2.8).

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	37
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 2.8

Опис таблиці MccMapping

Поле	Тип	Призначення
mcc	Int	Код Merchant Category Code (використовується як РК)
category_id	VarChar(36)	Зовнішній ключ, що посилається на таблицю Categories
createdAt	Timestamp	Дата створення запису
updatedAt	Timestamp	Дата останнього оновлення

Таблиця **Budget** зберігає інформацію про фінансові плани користувача на певний період. Зв'язок з User є композицією, оскільки бюджети належать виключно їхньому власнику. Наявність зовнішнього ключа до BankAccount дозволяє опціонально прив'язати план до конкретного рахунку. Найважливіший зв'язок «один-до-багатьох» з таблицею BudgetCategory також реалізований як композиція. Цей підхід було використано, щоб при видаленні бюджету всі його розбивки по категоріях автоматично знищувалися (табл. 2.9).

Таблиця 2.9

Опис таблиці Budgets

Поле	Тип	Призначення
budget_id	VarChar(36)	Унікальний ідентифікатор бюджету (UUID)
user_id	VarChar(36)	Зовнішній ключ, що посилається на таблицю Users
budget_name	VarChar(100)	Назва бюджету (унікальна для кожного користувача)
total_amount	Decimal(15,2)	Загальна запланована сума бюджету
start_date	Date	Дата початку бюджетного періоду
end_date	Date	Дата закінчення бюджетного періоду
currency	VarChar(3)	Валюта бюджету
bank_account_id	VarChar(36)	Зовнішній ключ до BankAccounts (необов'язковий)
is_active	Boolean	Прапорець активності бюджету
createdAt	Timestamp	Дата створення
updatedAt	Timestamp	Дата останнього оновлення

Таблиця **BudgetCategory** є проміжною сутністю, яка реалізує зв'язок «багато-до-багатьох» між бюджетами та категоріями. Створення окремої таблиці замість простого зовнішнього ключа зумовлено тим, що одна й та сама категорія може використовуватися в різних бюджетах (на різні місяці), що вимагає індивідуального підрахунку лімітів для кожного випадку. На відміну від звичайного зв'язку, ця таблиця містить власні атрибути для кожної такої пари — запланований ліміт (`allocated_amount`) та фактичні витрати (`actual_spent`). Зв'язки з обома батьківськими таблицями (`Budget` та `Category`) налаштовані як композиція, тому видалення будь-якого з батьківських записів автоматично очищає відповідні рядки у цій таблиці (табл. 2.10).

Таблиця 2.10

Опис таблиці BudgetCategories

Поле	Тип	Призначення
<code>id</code>	<code>VarChar(36)</code>	Унікальний ідентифікатор запису (UUID)
<code>budget_id</code>	<code>VarChar(36)</code>	Зовнішній ключ, що посилається на таблицю <code>Budgets</code>
<code>category_id</code>	<code>VarChar(36)</code>	Зовнішній ключ, що посилається на таблиці <code>Categories</code>
<code>allocated_amount</code>	<code>Decimal(15,2)</code>	Запланована сума на цю категорію в бюджеті
<code>actual_spent</code>	<code>Decimal(15,2)</code>	Фактично витрачена сума в цій категорії
<code>createdAt</code>	<code>Timestamp</code>	Дата створення
<code>updatedAt</code>	<code>Timestamp</code>	Дата останнього оновлення

Дані про транзакції містяться в таблиці **Transaction**. Саме транзакції є ядром системи, що фіксують кожну операцію (дохід чи витрату). Як зазначалося раніше, зв'язок з таблицею `User` є композицією, проте зв'язки з рахунком та бюджетом реалізовані як агрегація. Це гарантує збереження історичної достовірності: якщо рахунок або бюджет видаляється, транзакція залишається в базі, просто втрачаючи ці прив'язки. А зв'язок з `Category` є строгою асоціацією — системі заборонено видаляти категорію, якщо по ній вже є проведені транзакції, щоб забезпечити достовірність статистики (табл. 2.11).

Таблиця 2.11

Опис таблиці Transaction

Поле	Тип	Призначення
transaction_id	VarChar(36)	Унікальний ідентифікатор транзакції (UUID)
user_id	VarChar(36)	Зовнішній ключ, що посилається на таблицю Users
account_id	VarChar(36)	Зовнішній ключ до BankAccounts (необов'язковий)
category_id	VarChar(36)	Зовнішній ключ, посилається на таблицю Categories
budget_id	VarChar(36)	Зовнішній ключ до Budgets (необов'язковий)
amount	Decimal(15,2)	Сума фінансової операції
type	Enum	Тип транзакції (INCOME - дохід, EXPENSE - витрата)
description	VarChar(255)	Опис транзакції (коментар користувача)
transaction_date	Timestamp	Дата та час здійснення операції
external_id	VarChar(100)	Зовнішній ID (наприклад, ID транзакції з банку)
source	VarChar(30)	Джерело створення (наприклад, "MANUAL")
createdAt	Timestamp	Дата створення запису в системі
updatedAt	Timestamp	Дата останнього оновлення запису

Таблиця **Notification** призначена для зберігання повідомлень для користувача. Було використано зв'язок із таблицею User, який є прикладом композиції з кратністю «один до багатьох». Для того, щоб повідомлення створювалися виключно для конкретного користувача і автоматично видалялися разом із його обліковим записом (табл. 2.12).

Таблиця 2.12

Опис таблиці Notifications

Поле	Тип	Призначення
notification_id	VarChar(36)	Унікальний ідентифікатор сповіщення (UUID)
user_id	VarChar(36)	Зовнішній ключ, що посилається на таблицю Users
type	VarChar(50)	Тип сповіщення
message	VarChar(255)	Текст повідомлення сповіщення
is_read	Boolean	Прапорець прочитання сповіщення користувачем
created_at	Timestamp	Дата створення сповіщення

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	40
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

За журнал безпеки відповідає таблиця **AuditLog**. Ця таблиця фіксує критичні дії користувачів разом з IP-адресою та часом. Зв'язок з таблицею User має кратність «один до багатьох» та реалізований через композицію. Для того, щоб історія дій була невіддільною частиною облікового запису, при видаленні користувача всі його логи автоматично очищаються з бази даних (табл. 2.13).

Таблиця 2.13

Опис таблиці AuditLogs

Поле	Тип	Призначення
log_id	VarChar(36)	Унікальний ідентифікатор запису журналу (UUID)
user_id	VarChar(36)	Зовнішній ключ, що посилається на таблицю Users
action	VarChar(100)	Назва виконаної дії (наприклад, «LOGIN»)
timestamp	Timestamp	Дата та час виконання дії
ip_address	VarChar(45)	IP-адреса, з якої була виконана дія
status	VarChar(20)	Статус виконання дії (наприклад, "SUCCESS")
details	Text	Деталі дії (зазвичай у форматі JSON)

Таблиця **Integration** зберігає конфігурації та токени доступу для підключення до сторонніх банківських сервісів. Зв'язок з User реалізований як композиція («один до багатьох»). Щоб відповідна інтеграція належала лише одному користувачу і при видаленні акаунту користувача автоматично видалялися дані про інтеграцію в цілях безпеки (табл. 2.14).

Таблиця 2.14

Опис таблиці Integrations

Поле	Тип	Призначення
integration_id	VarChar(36)	Унікальний ідентифікатор інтеграції (UUID)
user_id	VarChar(36)	Зовнішній ключ, що посилається на таблицю Users
provider_name	VarChar(50)	Назва зовнішнього сервісу (унікальна для користувача)
access_token	Text	Токен доступу до API зовнішнього сервісу
refresh_token	Text	Токен оновлення доступу до API
expires_at	Timestamp	Час закінчення дії токена доступу

продовж. табл. 2.14

is_active	Boolean	Прапорець активності інтеграції
createdAt	Timestamp	Дата створення
updatedAt	Timestamp	Дата останнього оновлення

Таблиця **Accounts** є технічною сутністю бібліотеки NextAuth, призначеною для зберігання даних облікових записів сторонніх провайдерів, прив'язаних до користувача, таких як Google-сервіс. Зв'язок з таблицею User реалізований як «багато до одного» і є композицією. Щоб при видаленні користувача всі його прив'язані сторонні акаунти автоматично видалялися з бази даних для запобігання завислим даним (табл. 2.15).[7]

Таблиця 2.15

Опис таблиці Accounts

Поле	Тип	Призначення
id	VarChar(255)	Унікальний ідентифікатор запису (CUID)
userId	VarChar(36)	Зовнішній ключ, що посилається на таблицю Users
type	VarChar(255)	Тип аутентифікації (наприклад, "oauth")
provider	VarChar(255)	Назва провайдера (наприклад, "google")
providerAccountId	VarChar(255)	ID користувача на боці провайдера
refresh_token	Text	Токен оновлення доступу провайдера
access_token	Text	Токен доступу провайдера
expires_at	Int	Час закінчення дії токена
token_type	VarChar(255)	Тип токена (наприклад, "Bearer")
scope	VarChar(255)	Область доступу, надана токеном
id_token	Text	ID-токен провайдера
session_state	VarChar(255)	Стан сесії провайдера
createdAt	Timestamp	Дата створення запису

Таблиця **Session** призначена для фіксації активних сесій авторизації (наприклад, при вході з різних пристроїв). Залежність від таблиці User формується

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	42
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

за типом «багато до одного» і є строгою композицією. Життєвий цикл сесії повністю залежить від існування власника, тому на рівні бази даних налаштовано каскадне видалення. Тобто при анулюванні облікового запису користувача автоматично припиняються та видаляються всі його активні сесії (табл. 2.16).

Таблиця 2.16

Опис таблиці Sessions

Поле	Тип	Призначення
id	VarChar(255)	Унікальний ідентифікатор сесії (CUID)
sessionToken	VarChar(255)	Унікальний токен сесії
userId	VarChar(36)	Зовнішній ключ, що посилається на таблицю Users
expires	DateTime	Дата та час закінчення дії сесії
createdAt	Timestamp	Дата створення сесії

Таблиця **VerificationToken** є технічною сутністю для тимчасового зберігання одноразових токенів (наприклад, для підтвердження електронної пошти). Ця таблиця повністю ізольована і не має зовнішніх ключів до інших таблиць, оскільки ідентифікує користувача виключно за текстовим полем identifier (його email-адресою). Записи з цієї таблиці мають короткий життєвий цикл і видаляються після використання (табл. 2.17).

Таблиця 2.17

Опис таблиці VerificationTokens

Поле	Тип	Призначення
identifier	VarChar(255)	Ідентифікатор цілі (зазвичай email користувача)
token	VarChar(255)	Унікальний токен для підтвердження (напр., email)
expires	DateTime	Час закінчення дії токена
createdAt	Timestamp	Дата створення токена

Сам процес збереження та читання даних реалізований за клієнт-серверною архітектурою. Під час ініціації дії в інтерфейсі, клієнтська частина формує запит та надсилає його на сервер. Завдяки Prisma, серверна логіка автоматично перетворює

отримані об'єкти на SQL-інструкції та передає їх до СУБД MySQL. Запит фізично стає збереженим на серверах платформи Railway лише після його успішного виконання та підтвердження з боку бази даних. Подальша робота з накопиченими даними (фільтрація транзакцій, агрегація бюджетів) здійснюється через згенеровані методи ORM, що повністю виключає необхідність написання сирих SQL-запитів вручну та гарантує безпеку на рівні звернень до бази.

Після опису всіх таблиць також не менш важливо обґрунтувати вибір типів даних для фінансових операцій. Для зберігання грошових сум важливо дотримуватися точності всіх розрахунків. Тому для полів `balance` (рахунки), `amount` (транзакції), `total_amount` (бюджети) та `allocated_amount` (категорії бюджету) було обрано саме тип **Decimal(15,2)**. На відміну від типів з рухомою комою (Float або Double), які є приблизними і можуть призводити до помилок округлення, тип Decimal забезпечує точне збереження значень з фіксованою точкою. Для фінансової системи це критична вимога, оскільки втрата навіть копійки при масштабуванні системи на тисячі користувачів призведе до некоректної фінансової звітності та порушення цілісності даних.

Окрім вибору типів даних, велике значення має коректне налаштування зв'язків між таблицями, адже саме вони підтримують цілісність фінансової інформації. Наприклад, кожна транзакція прив'язується до конкретного користувача, рахунку, категорії, а за потреби — і до бюджету. Завдяки цьому можна уникнути дублювання даних і побудувати зрозумілу логіку збереження інформації. Коли користувач переглядає статистику або аналізує витрати за вибраний період, система працює не з окремими розрізненими записами, а з пов'язаними даними, що дає змогу формувати правильні підсумки. Також застосування зовнішніх ключів у базі даних знижує ймовірність появи некоректних або «загублених» записів, зокрема транзакцій без відповідного користувача чи категорії. Для фінансової вебплатформи це особливо суттєво, бо точність, узгодженість і надійність збережених даних напряму впливають на правильність розрахунків та рівень довіри користувача до системи.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	44
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

2.3 Алгоритм роботи та стани програмної системи

У цій роботі використано діаграми активності для відображення процесу авторизації та реєстрації в системі, від введення логіну користувача та пароля, а також реєстраційної форми до успішного входу в систему (рис.2.5).

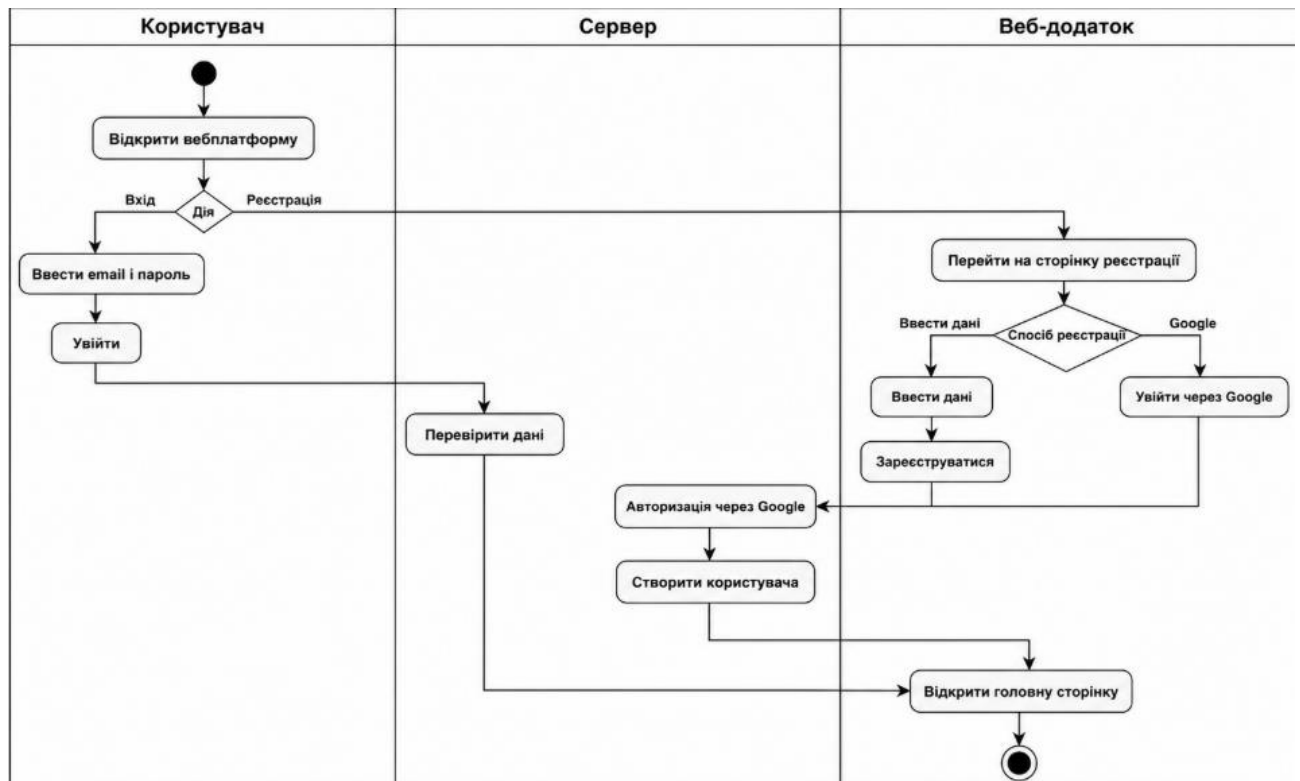


Рисунок 2.5 – Діаграма активності авторизації та реєстрації

Під час авторизації забезпечено можливість обрати один із варіантів «Вхід в систему» чи «Реєстрація». Під час входу в систему користувачу буде виведено сторінку з формою для входу, де він має ввести свої облікові дані. Якщо введені дані є невірними, то користувач залишиться на сторінці входу для повторного введення свого логіна та пароля. Якщо ж користувач ввів коректні дані, то він отримує доступ до функціоналу системи відповідно до його ролі. Якщо було обрано варіант «Реєстрація», то відобразиться реєстраційна форма, де користувач має заповнити всі обов'язкові поля так, щоб було пройдено валідацію. Після валідації даних на клієнті вони відправляються на сервер, і там також відбувається перевірка на правильність даних, наприклад, на відсутність дублювання даних тощо.

Наступна діаграма активності демонструє процес додавання нової фінансової транзакції користувачем (рис.2.6). Користувач відкриває спеціальну форму "Додати транзакцію", де заповнює обов'язкові поля: вводить суму, обирає категорію (наприклад, "Продукти", "Транспорт", "Зарплата") та вказує тип операції — дохід або витрата. Після натискання кнопки "Зберегти" вебдодаток спочатку перевіряє коректність введених даних (наприклад, чи сума є додатним числом, чи обрано категорію). Якщо дані невалідні, користувачеві показується повідомлення про помилку для виправлення. Якщо дані коректні, вони надсилаються на сервер. Сервер, отримавши запит, спочатку перевіряє авторизацію користувача (чи має він право виконувати цю дію), а потім зберігає дані транзакції в базу даних. Після успішного збереження сервер повертає статус успіху веб-додатку, який, у свою чергу, оновлює список транзакцій на сторінці користувача та показує підтвердження про успішне додавання запису.

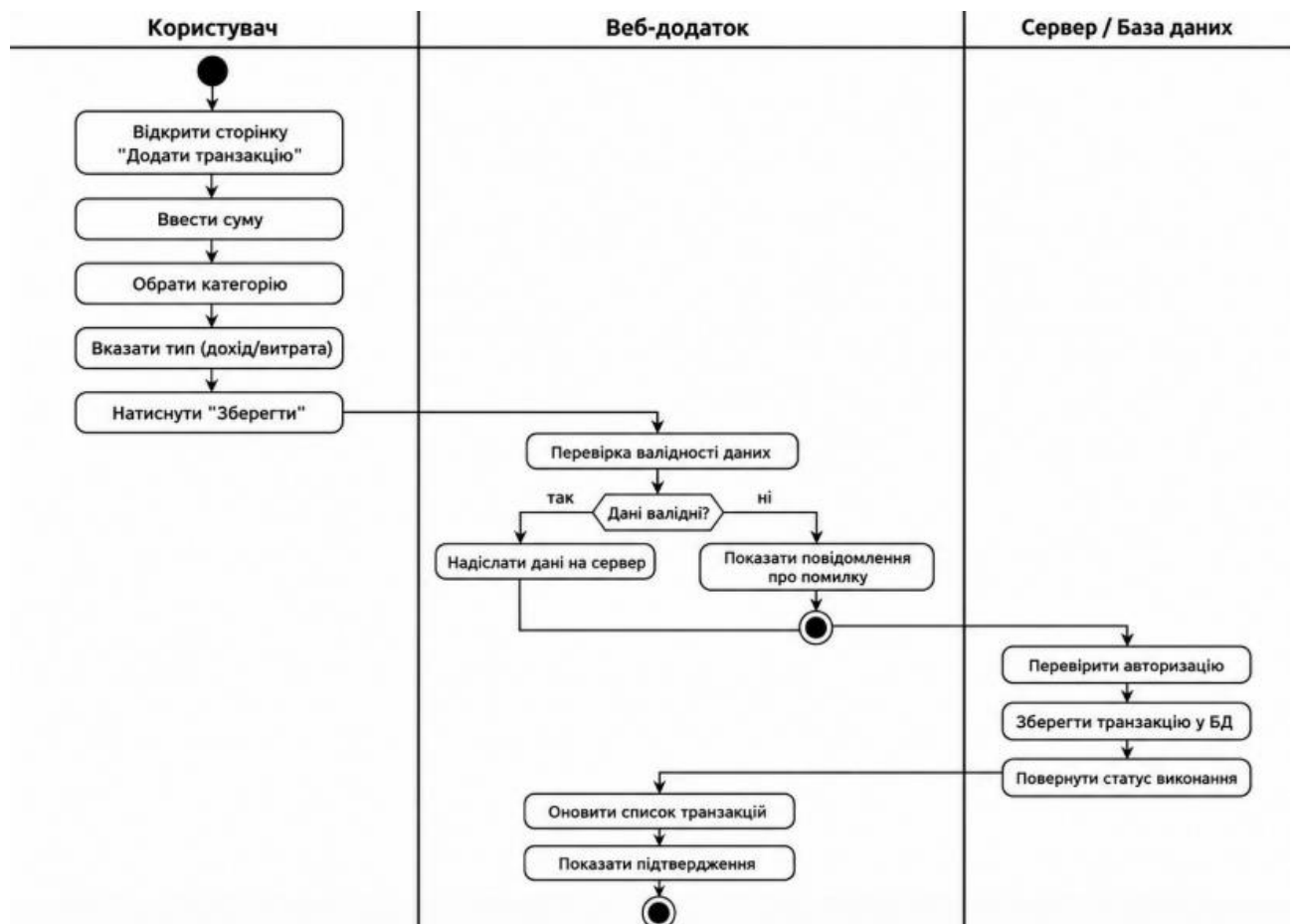


Рисунок 2.6 – Діаграма додавання нової фінансової транзакції

Зображено діаграму активності, що моделює процес редагування даних користувачем (рис.2.7). Процес починається, коли користувач обирає запис зі списку транзакцій та натискає іконку редагування. Після цього він може внести необхідні зміни у значення полів та підтвердити дію, натиснувши «Зберегти».

Вебдодаток отримує оновлені дані та здійснює їхню валідацію (перевіряє коректність формату, обов'язковість заповнення тощо). Якщо дані не відповідають вимогам, користувачеві показується повідомлення про помилку, що дозволяє виправити неточності, не відправляючи запит на сервер. Якщо дані коректні, вони надсилаються на сервер.

На сервері відбувається додаткова перевірка: система підтверджує, що користувач має необхідні права на редагування саме цього запису. Якщо права підтверджено, сервер оновлює відповідні дані в базі даних. Після успішного оновлення сервер повертає позитивний результат вебплатформи. Вебдодаток, отримавши підтвердження, оновлює інформацію на сторінці користувача, щоб відобразити зміни. Процес завершується успішним оновленням даних.

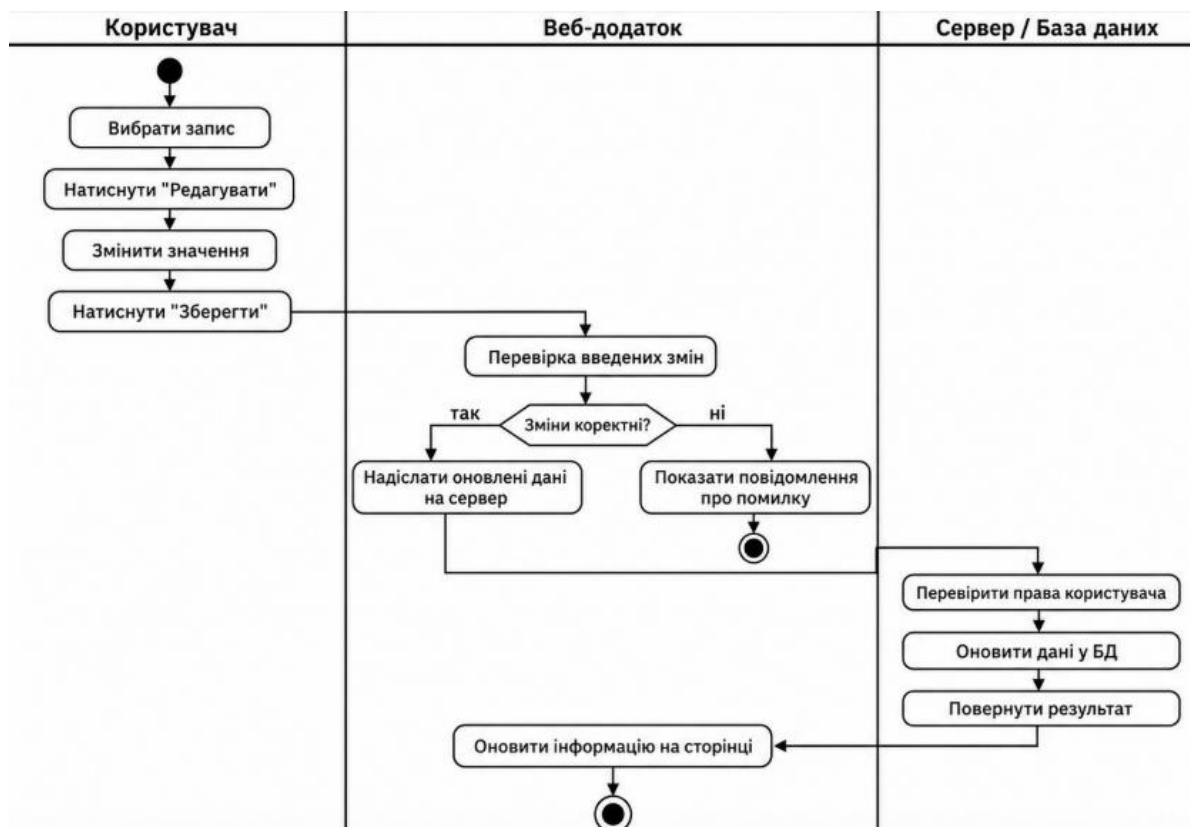


Рисунок 2.7 – Діаграма активності для редагування даних

2.4 Реалізація модуля інтеграції з банківською системою Monobank

Щоб забезпечити актуальність фінансових даних у системі було реалізовано двоступеневий модуль інтеграції з API Monobank [14]. Архітектурно модуль складається з трьох основних компонентів: серверного маршруту синхронізації (sync/route.ts), обробника вебхуків для отримання транзакцій у реальному часі (webhook/[secret]/route.ts) та клієнтського хука для керування станом запитів (useBankSync.ts).

Серверна синхронізація та підключення реалізовані у файлі (sync/route.ts). Цей файл реалізує REST API ендпоінти (GET, POST, DELETE) для керування життєвим циклом інтеграції [2,29]. Найскладнішою з точки зору алгоритміки є метод GET, який відповідає за примусову синхронізацію історії транзакцій.

Для початку розглянемо принцип роботи методу **GET**, першочергово авторизація та валідація перевіряються через сесію користувача через Next.js Auth [7]. За її наявності з БД витягується запис користувача та перевіряється активність токена до Monobank (integration.is_active). За допомогою токена виконується запит до API Monobank для отримання масиву банківських рахунків [14]. Для імпорту транзакцій обчислюється часовий проміжок (останні 31 день) у форматі Unix timestamp. З БД завантажуються існуючі транзакції цього користувача з джерелом MONOBANK. Їхні external_id зберігаються у колекції Set для швидкого пошуку та запобігання дублюванню.

Для автоматичної категоризації транзакцій за кодами МСС словник відповідностей завантажується з БД у пам'ять як структура даних Map. Це виключає надлишкові звернення до бази під час обробки великого масиву даних. На фінальному етапі замість ресурсомісткого послідовного створення записів формується масив, який записується в БД одним пакетним запитом prisma.transaction.createMany зі встановленим прапорцем skipDuplicates: true [5].

Такий підхід зменшує кількість окремих операцій запису та допомагає підтримувати цілісність даних під час масового імпорту [5,27].

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	48
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Важливо зазначити, що під час синхронізації історії транзакцій з API Monobank існує ризик отримання HTTP-помилки 429 Too Many Requests при надходженні великої кількості запитів від користувачів [14]. Щоб не заблокувати потік виконання (Event Loop) заборонено використання синхронних затримок. Тому було вирішено застосувати патерн Exponential Backoff (експоненційна затримка) з обмеженою кількістю повторних спроб (максимум 3). При отриманні помилки 429 система не блокується, а робить паузу, яка подвоюється з кожною спробою (1с, 2с, 4с). Якщо після 3 спроб доступ не відновлено, запит припиняється, а помилка фіксується у журналі аудиту (AuditLog) для подальшого аналізу адміністратором.

Фрагмент логіки мапінгу категорій та підготовки масиву для пакетного вставлення наведено у (Додатку А).

Метод **POST** призначений для підключення інтеграції з “Monobank”. Після отримання токена від клієнта, верифікує його зверненням до API “Monobank”. Якщо токен валідний, генерується унікальний webhook_secret (використовуючи crypto.randomUUID()), формується URL для вебхуків на основі змінної середовища APP_URL, і дані зберігаються через метод upsert. Після цього відправляється запит до “Monobank” для встановлення вебхука. Одночасно відбувається первинне створення карткових рахунків користувача.

Видалення інтеграції реалізоване через метод **DELETE** з використанням транзакцій prisma.\$transaction, що забезпечує цілісність даних. В межах однієї транзакції послідовно видаляються записи інтеграції, пов'язані транзакції та самі банківські рахунки, назви яких починаються на “Monobank”. Такий підхід унеможливорює виникнення ситуації, коли частина даних була видалена, а частина залишилася в системі [27].

Для отримання інформації про нові транзакції в реальному часі у системі реалізовано обробник вебхуків у реальному часі (**webhook/[secret]/route.ts**). Його використання дозволяє уникнути постійного фетчингу API, який в результаті призведе до блокування доступу до токена, якщо кількість запитів

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	49
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

буде більша ніж 1 на хвилину [14]. Замість цього система очікує POST-запити від серверів Monobank, які надсилаються автоматично після появи нової операції.

Особливістю маршрутизації Next.js є динамічний параметр [secret]. Коли Monobank надсилає вебхук, він звертається за адресою, яка містить секретний ключ, згенерований при підключенні. Це дозволяє ідентифікувати користувача без передачі додаткових заголовків авторизації.[14]

Після отримання **POST** запиту система виконує пошук інтеграції в базі даних за значенням webhook_secret. Після успішної автентифікації з тіла запиту витягується об'єкт statementItem, що містить інформацію про операцію. Для забезпечення цілісності даних виконується перевірка на наявність дублікатів за унікальним ідентифікатором external_id. У разі виявлення нової операції, запускається механізм визначення категорії за МСС, який працює аналогічно до логіки повної синхронізації. На фінальному етапі система створює запис транзакції в базі даних та оновлює поточний залишок відповідного банківського рахунку актуальним значенням із поля statementItem.balance. Важливим фрагментом реалізації є обробка помилок, наведено у (Додатку Б).

В документації Monobank зазначено, що при поверненні сервером помилку більше 3–5 разів поспіль банк автоматично відключає вебхук. Тому у блоці catch навіть при критичних помилках система завжди відповідає 200 ОК. А помилка логується на сервері для подальшого ручного або автоматичного розгляду, але інтеграція з банком не розривається.[14]

У клієнтській частині для ініціалізації процесу завантаження даних застосовано кастомний React-хук useBankSync. Оскільки в інтерфейсі відсутня окрема кнопка примусової синхронізації, цей хук викликається виключно на етапі первинного підключення банку (всередині функції connectAndSync). Це зумовлено обмеженнями API Monobank: вебхуки фіксують лише нові операції, тоді як для формування повноцінного дашборду користувачу необхідно миттєво відобразити історію за останні 31 день.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	50
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Важливою особливістю реалізації хука є вбудований механізм локального тротлінгу (cooldown) з інтервалом у 90 секунд. Це запобіжний захист від перевищення лімітів (Rate Limiting) банківського API, яке загрожує автоматичним блокуванням токена при частоті запитів більш ніж один раз на хвилину. У разі спроби повторного виклику раніше вказаного часу користувач отримує зрозуміле повідомлення з таймером. Після успішного завершення пакетного імпорту хук генерує глобальну подію `finance-data-updated`, що дозволяє реагувати та оновити пов'язані компоненти інтерфейсу (залишки на рахунках, графіки) без перезавантаження сторінки [24].

На (рисунку 2.8) продемонстровано динаміку обробки вхідного вебхука від банківської системи в межах монолітної архітектури Next.js.

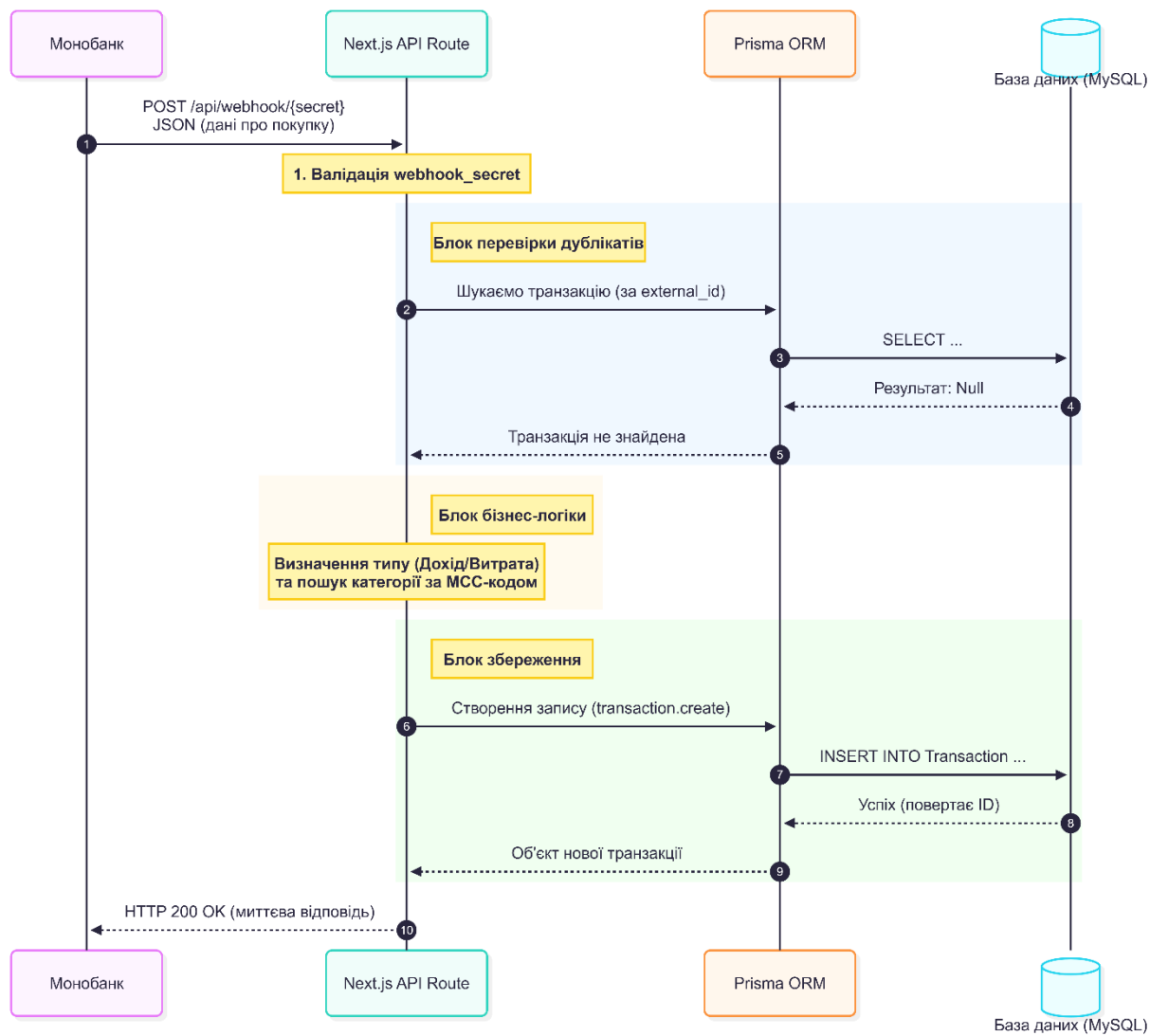


Рисунок 2.8 – Діаграма послідовності обробки вебхука Monobank

Після здійснення фінансової операції зовнішня система Monobank надсилає POST-запит у форматі JSON на спеціальний endpoint вебдодатка [29]. Цей запит обробляється серверною функцією Next.js Route Handler, розміщеною в API-маршруті інтеграції з банківським сервісом.

На першому етапі Route Handler виконує перевірку секретного параметра `webhook_secret`, що дозволяє підтвердити коректність отриманого запиту та визначити відповідну інтеграцію користувача. Після цього за допомогою Prisma Client здійснюється звернення до бази даних MySQL для перевірки наявності транзакції з таким самим зовнішнім ідентифікатором. Така перевірка потрібна для уникнення дублювання фінансових операцій у системі.

Якщо транзакція ще не була збережена, серверна логіка визначає її тип, суму, рахунок і категорію. Для автоматичної класифікації використовується код МСС, який порівнюється із записами таблиці відповідностей категорій. Після визначення необхідних даних Route Handler через Prisma Client створює новий запис транзакції в базі даних та оновлює пов'язаний банківський рахунок[5].

Після успішного виконання операцій з базою даних сервер повертає Monobank відповідь HTTP 200 OK [14,29]. Це підтверджує зовнішній системі, що вебхук був успішно прийнятий і оброблений. Такий механізм дозволяє автоматично отримувати банківські транзакції без ручного введення користувачем і підтримувати актуальність фінансових даних у системі.

Висновки до другого розділу

Підсумовуючи викладене вище, у процесі проєктування вебплатформи управління особистими фінансами були проаналізовані всі ключові сценарії використання програмного продукту відповідно до сформульованих вимог. На основі отриманих результатів була спроектована архітектура системи, що включає моделювання структури проєкту та взаємодії між його складовими компонентами.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	52
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Окрему увагу було приділено проєктуванню бази даних. Запропонована схема забезпечує цілісність даних та передбачає використання складених індексів, що є критично важливим для швидкої обробки запитів .

Також були описані алгоритми роботи додатка та послідовність дій користувача під час взаємодії з системою. Основний функціонал платформи зосереджений на автоматизованому обліку фінансів: це інтеграція з банківськими системами (на прикладі Monobank) для автоматичного імпорту транзакцій, їхня інтелектуальна категоризація, а також управління бюджетами та відстеження залишків на рахунках у реальному часі.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	53
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

3.1 Порядок встановлення та налаштування параметрів вебплатформи

Встановлення вебплатформи управління особистими фінансами охоплює розгортання компонентів, підготовку середовища, підключення MySQL, автентифікацію та зовнішні інтеграції. Система побудована як повностековий вебдодаток на Next.js, тому клієнтська й серверна частини працюють разом в одному проєкті. Це спрощує підтримку, оновлення й адміністрування платформи.

Архітектуру розгортання показано на діаграмі (рис. 3.1). Вона включає пристрій користувача, середовище Vercel, базу даних MySQL на Railway, Google OAuth 2.0 і Monobank API [8,12,13,14]. Користувач відкриває платформу у браузері, а запити надходять до додатка, розміщеного на Vercel.

Ключова особливість архітектури — використання Next.js як основи повностекового моноліту. В одному проєкті поєднано React-компоненти, сторінки інтерфейсу, серверні API-маршрути, авторизацію, транзакції, бюджети, категорії, рахунки та інтеграції. Тому не потрібно створювати окремий бекенд чи додатково налаштовувати зв'язок між сервісами. Уся система розгортається як один вебдодаток, а клієнтська й серверна частини обмінюються даними через внутрішні JSON-запити [2,29].

Клієнтська частина відповідає за інтерфейс, форми, списки транзакцій, бюджети, рахунки, статистику та адміністративні сторінки. Вона створена на React-компонентах і використовує маршрутизацію Next.js. Серверна логіка працює через Route Handlers, які обробляють API-запити для створення, редагування, видалення й отримання фінансових даних, категорій, бюджетів, журналу аудиту та профілю користувача.

Дані зберігаються в реляційній базі MySQL, розгорнутій на Railway. Окреме хмарне середовище відділяє зберігання інформації від виконання

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	54
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

вебдодатка, що спрощує адміністрування й оновлення системи. Зв'язок між Next.js і MySQL забезпечує Prisma ORM, яка надає типізований доступ до таблиць, виконує запити, створює міграції та синхронізує моделі з базою.

Розгортання починається з підготовки репозиторію вихідного коду. Проект зберігається у віддаленому Git-репозиторії та підключається до Vercel [13,22]. Після цього Vercel визначає структуру Next.js-додатка й налаштовує збірку. Коли основна гілка оновлюється, платформа автоматично запускає нову збірку та повторне розгортання[13].

Далі створюється екземпляр MySQL на Railway. Після цього сервіс надає рядок підключення з адресою сервера, портом, користувачем, паролем і назвою бази. Його додають до Vercel через змінну середовища DATABASE_URL. Це дає змогу не зберігати конфіденційні дані у вихідному коді й захищає доступ до бази, ключі та параметри сервісів.

Для роботи у Vercel налаштовуються основні змінні середовища. DATABASE_URL відповідає за підключення до MySQL, NEXTAUTH_SECRET — за захист сесій, NEXTAUTH_URL — за адресу розгорнутого додатка, а GOOGLE_CLIENT_ID і GOOGLE_CLIENT_SECRET — за Google OAuth 2.0. Також задається APP_URL, яка використовується для формування публічної адреси вебхука Monobank.

Після підключення бази створюється її структура за допомогою Prisma Migrate. Моделі у schema.prisma описують сутності, поля, типи даних, зв'язки, індекси та унікальні обмеження. Після запуску міграцій Prisma створює таблиці користувачів, ролей, рахунків, транзакцій, категорій, бюджетів, інтеграцій, сповіщень, журналу аудиту та інших сутностей.

Потім виконується первинне наповнення бази. Для роботи платформи додаються базові ролі, зокрема адміністратор і звичайний користувач, а також стартові категорії доходів і витрат, підкатегорії та відповідності кодів МСС. Ці записи потрібні для класифікації транзакцій і автоматичного визначення

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	55
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

категорій під час імпорту банківських операцій. Їх можна внести через seed-скрипти Prisma.

Окремо налаштовується автентифікація. У системі використовується NextAuth, який підтримує вхід за email і паролем, а також Google OAuth 2.0. Для Google-авторизації створюються облікові дані в Google Cloud Console, отримуються Client ID та Client Secret і задаються дозволені URI перенаправлення. Після входу сервер перевіряє дані користувача, створює або оновлює запис і формує сесію. Доступ розмежовується за ролями [7].

Інтеграція з Monobank використовується для автоматичного отримання рахунків і банківських транзакцій. Після введення токена система перевіряє його через Monobank API. Якщо токен чинний, у базі створюється або оновлюється інтеграція та зберігаються пов'язані рахунки. Для нових операцій формується webhook-адреса, яку система передає до Monobank, після чого сервіс надсилає POST-запити до API-маршруту додатка.

Вебхуки обробляє серверна логіка Next.js. Коли Monobank повідомляє про нову операцію, Route Handler перевіряє секретний параметр, визначає користувача й інтеграцію, шукає транзакцію з таким самим зовнішнім ідентифікатором, класифікує її за МСС і створює запис у базі. Це запобігає дублюванню транзакцій і забезпечує автоматичне оновлення фінансових даних [5,14].

Після налаштування змінних середовища, бази даних, авторизації та банківської інтеграції створюється продакшн-збірка. Під час цього Vercel перевіряє структуру проєкту, встановлює залежності, генерує Prisma Client і формує оптимізований вебдодаток. Після успішної збірки система стає доступною за публічною адресою, де перевіряються сторінки, реєстрація, вхід, профіль, рахунки, транзакції, бюджети, статистика, адміністративна панель і зв'язок із Monobank.

Фінальна перевірка охоплює клієнтські та серверні сценарії. З боку користувача тестуються реєстрація, авторизація, редагування профілю,

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	56
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

додавання й видалення операцій, створення бюджетів і перегляд статистики. З боку адміністратора перевіряються адміністративні сторінки, користувачі, категорії та журнал аудиту. Також контролюються запис даних у MySQL, зв’язки між таблицями й стабільність API-запитів [1,19,29,30].

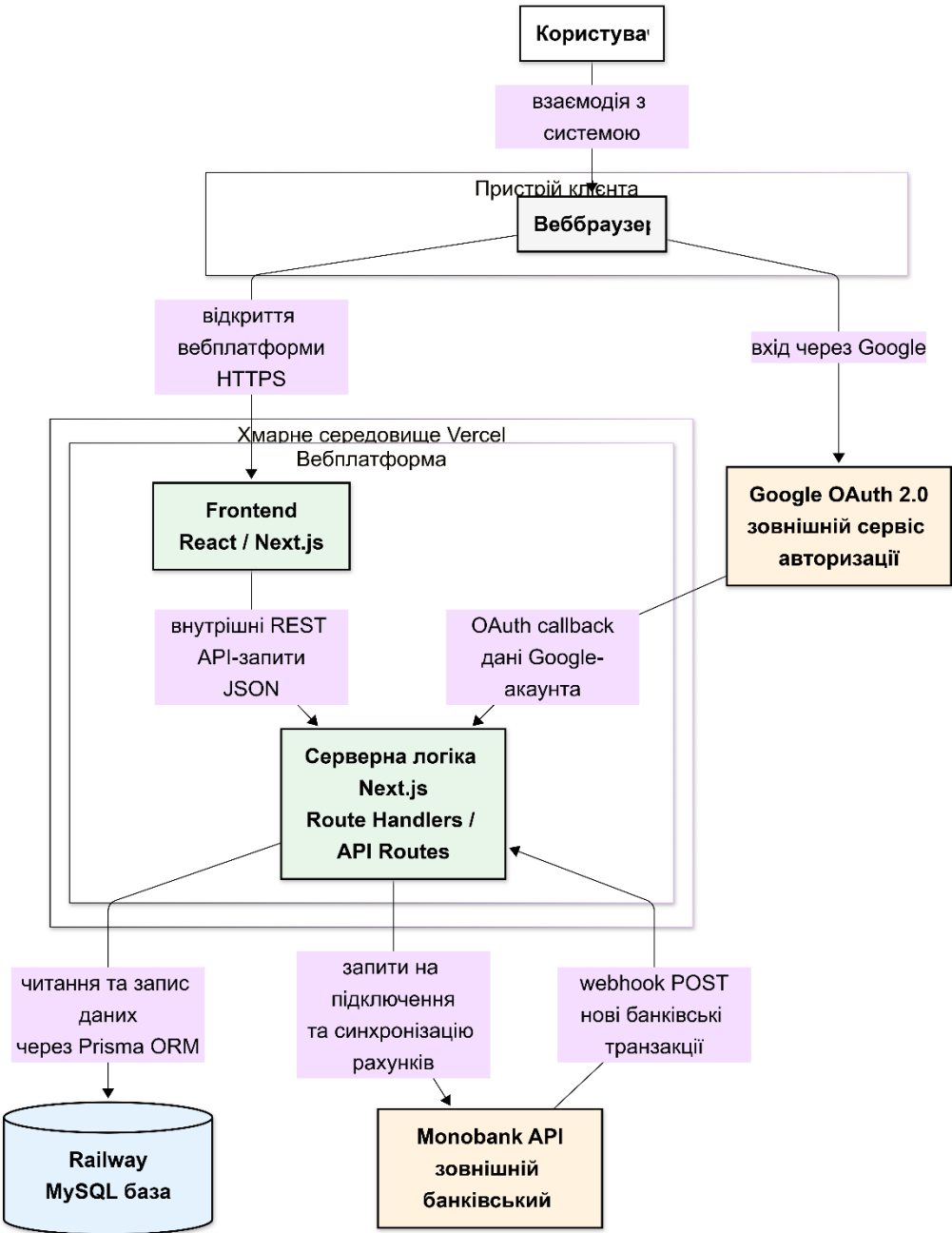


Рисунок 3.1 – Діаграма розгортання

3.2 Структура інтерфейсу системи управління особистими фінансами

Взаємодія з системою здійснюється через веб-інтерфейс, який починається з форми авторизації (рис.3.2). Для входу користувачу необхідно ввести електронну пошту (email) та пароль у відповідні поля і натиснути кнопку

«Увійти». Також для зручності передбачена альтернативна автентифікація через Google-акаунт.

Рисунок 3.2 – Форма входу

Якщо користувач вперше звертається до платформи, йому потрібно натиснути посилання «Зареєструватися», що перенаправить його на відповідну форму (рис.3.3). Для створення облікового запису необхідно заповнити такі поля: «Ім'я», «Email», «Пароль» та «Підтвердження паролю». Після заповнення даних та натискання кнопки «Зареєструватися» профіль створюється, і користувач потрапляє до системи.

Створення акаунту
Заповніть форму для реєстрації

Ім'я *

Ваше ім'я

Email *

your@email.com

Пароль *

Не менше 6 символів

Підтвердження пароля *

Не менше 6 символів

Зареєструватися

Вже маєте акаунт? [Увійти](#)

Рисунок 3.3 – Форма реєстрації

Після успішної авторизації користувач потрапляє на головну сторінку системи (рис.3.4). Інтерфейс складається з лівого меню навігації, центральної області та правої панелі (конвертер валют та останні транзакції). Зверху фіксується загальний баланс. Можна змінити візуальну тему – зі світлої (рис. 3.4) на темну (рис.3.5). Оскільки новий користувач ще не має даних, розділи «Рахунки», «Бюджети» та «Транзакції» порожні. Щоб розпочати повноцінну роботу, користувачу насамперед потрібно створити фінансовий рахунок, натиснувши на кнопку «Додати рахунок» у відповідному блоці. Аналогічним чином, натиснувши «Створити перший бюджет», ініціюється процес налаштування лімітів витрат.

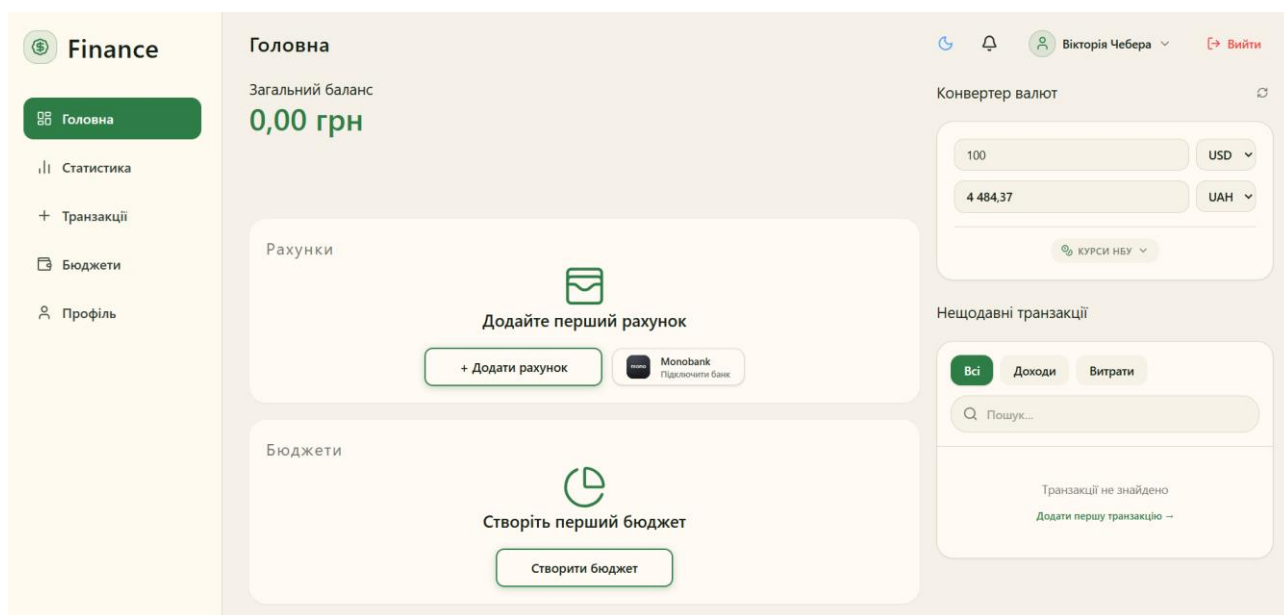


Рисунок 3.4 – Світла тема головної сторінки

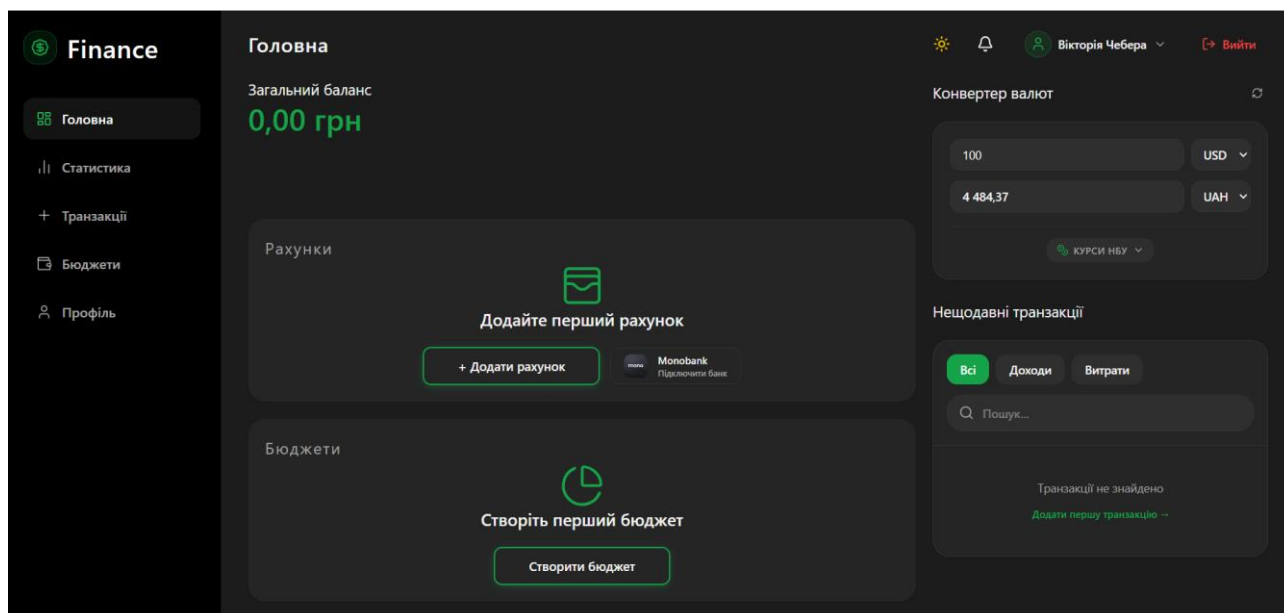


Рисунок 3.5 – Темна тема головної сторінки

Варто зазначити, що вебдодаток має повністю адаптивний дизайн, що забезпечує зручну роботу з системи з мобільних пристроїв (рис.3.6). На екранах смартфонів відбувається реструктуризація інтерфейсу: ліве бокове меню переміщується у нижню частину екрана у вигляді фіксованої панелі навігації, а всі робочі блоки (баланс, рахунки, бюджети) виводяться у вигляді єдиної вертикальної стрічки. При цьому весь основний функціонал зберігається.

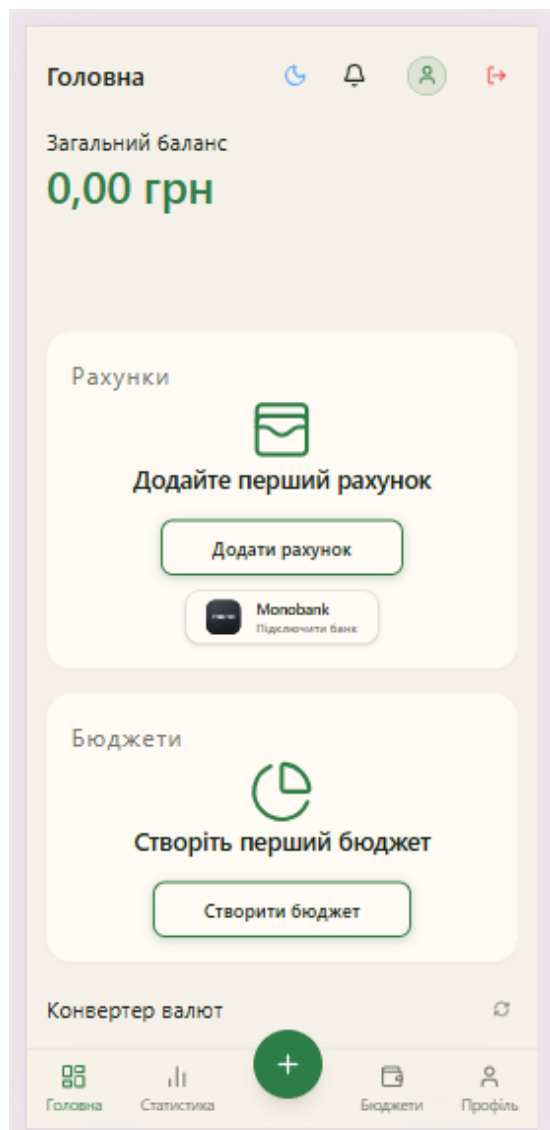


Рисунок 3.6 – Мобільна версія головної сторінки

При натисненні кнопки «Додати рахунок» відкривається модальне вікно для створення нового фінансового інструменту (рис.3.7). Форма містить набір полів для заповнення, з яких обов'язковими є: «Назва рахунку», «Тип рахунку» та «Валюта». Додатково передбачено поле «Початковий баланс» для введення наявних коштів на момент додавання. Для завершення створення користувач натискає кнопку «Створити рахунок», або може скасувати дію за допомогою кнопки «Скасувати». Після додавання рахунку він буде відображати на головній сторінці (рис.3.8).

ДОДАВАННЯ НОВОГО РАХУНКУ

Вкажіть параметри рахунку для ведення коректного обліку.

НАЗВА РАХУНКУ *

Наприклад, Картка ПриватБанк

ТИП РАХУНКУ *

Готівка

ВАЛЮТА *

UA UAH

ПОЧАТКОВИЙ БАЛАНС

0.00

Створити рахунок

Скасувати

Рисунок 3.7 – Модальне вікно створення рахунку

Також можна відредагувати та видалити рахунок за потреби, натиснувши на відповідні іконки конкретного рахунку (рис.3.8).

Finance

Головна

Статистика

Транзакції

Бюджети

Профіль

Головна

Загальний баланс

103 795,20 грн

EUR 2 000,00

Рахунки

Новий

ГОТІВКА

2 000,00 EUR

Евро

Бюджети

Створіть перший бюджет

Вікторія Чебера

Вийти

Конвертер валют

100 USD

4 484,37 UAH

КУРСИ НЕБУ

Нещодавні транзакції

Всі

Доходи

Витрати

Пошук...

Транзакції не знайдено

Додати першу транзакцію --

Рисунок 3.8 – Результат додавання рахунків

Після створення рахунку можна додавати транзакції для відповідного рахунку. Для цього потрібно перейти за посиланням «Транзакції» в сайдбарі (рис.3.9). Щоб створити транзакцію, спочатку користувач визначає тип операції, перемикаючись між вкладками «Дохід» та «Витрата». Далі заповнюються основні поля: сума операції, дата та час її здійснення, а також рахунок, з якого списуються або на який надходять кошти.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	62
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Рисунок 3.9 – Форма додавання транзакції

Обов'язковим кроком є прив'язка транзакції до категорії. При натисканні на відповідне поле відкривається вікно вибору (рис.3.10), де категорії згруповані за тематичними блоками. Також у формі передбачена можливість додавання текстових нотаток. Після заповнення всіх даних та натискання на кнопку «Додати транзакцію» збережеться запис.

Рисунок 3.10 – Список категорій

Після створення операція миттєво відображається у блоці «Недавні транзакції» на головній сторінці (рис.3.11). Для зручності аналізу історії операцій над блоком розташовані фільтри («Всі», «Доходи», «Витрати») та рядок пошуку. Кожен елемент списку містить назву категорії, суму (з автоматичною конвертацією у базову валюту, обрану користувачем), назву рахунку, а також іконки для швидкого редагування або видалення запису.

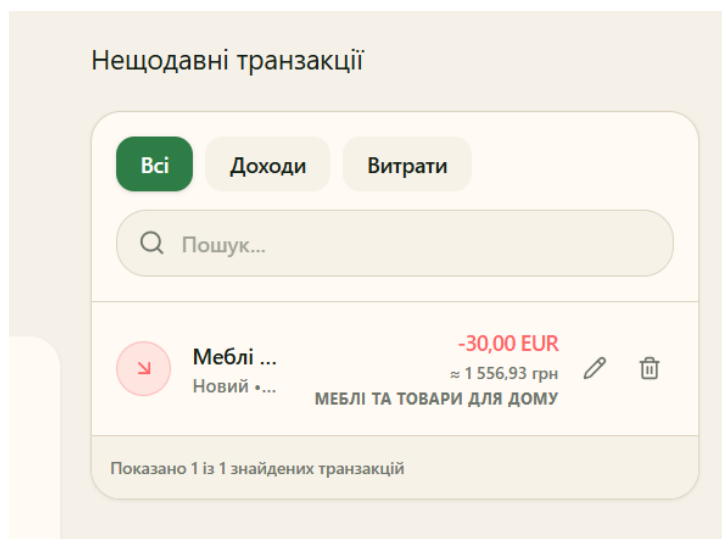


Рисунок 3.11 – Список доданих транзакцій

Користувачеві надана функція бюджетування для встановлення окремих лімітів витрат на певну категорію за певний проміжок часу. Для створення бюджету можна натиснути відповідну кнопку на головній сторінці або перейти до розділу «Бюджети» через бічне меню. На цій сторінці розташована кнопка додавання, натискання на яку відкриває модальне вікно (рис.3.12). У ньому користувач вказує назву, обирає валюту, вказує період дії, вибирає категорію та задає грошовий ліміт, після чого підтверджує дію.

НОВИЙ БЮДЖЕТ

Заповніть дані для створення бюджету

Бюджет на пальне

Новий — EUR

EU EUR

Тиждень

Місяць

Півріччя

Рік

10.06.2026

09.07.2026

Категорії

Розподіл бюджету

Пальне

500

€

Загальна сума бюджету

500,00 €

Створити бюджет

+ Додати

Рисунок 3.12 – Модальне вікно для створення бюджету

Новостворений бюджет відображається на сторінці бюджетів у вигляді картки (рис.3.13). Або на головній сторінці у вигляді кружечка.

Бюджети

Вікторія Чебера

Вийти

Створити бюджет

Бюджет на пальне

Новий

eu EUR

Ліміт

500,00 €

Витрачено

0,00 €

Залишок

500,00 €

Пальне

0,00 € / 500,00 €

10.06.2026 — 09.07.2026

Рисунок 3.13 – Результат додавання бюджету

Для демонстрації роботи створеного бюджету потрібно створити транзакцію з типом витрати відповідної категорії бюджету. Після проведення

операції система автоматично перераховує показники: на головному екрані інтерфейсу відображаються оновлений статус та актуальний залишок коштів (рис.3.14).

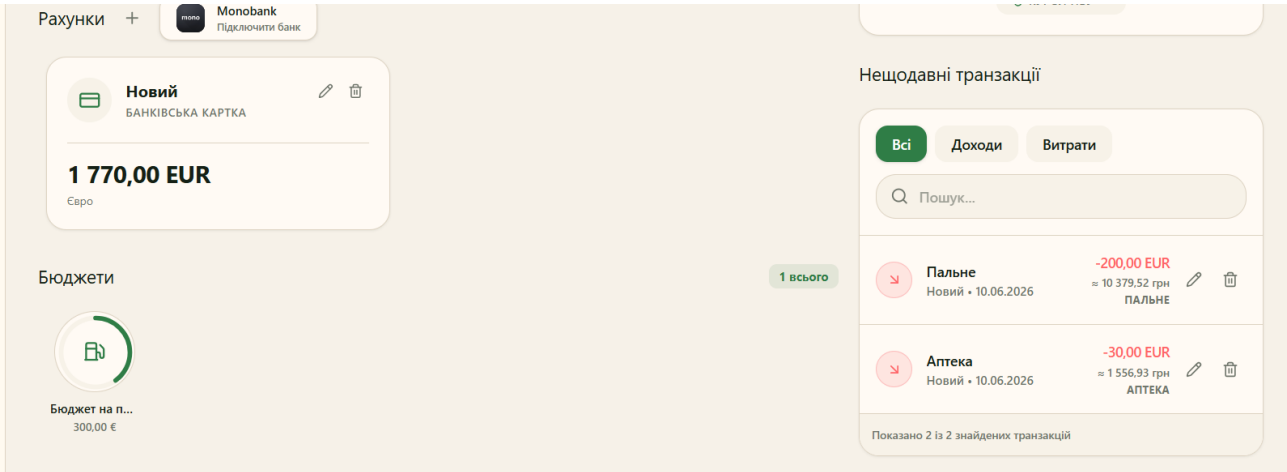


Рисунок 3.14 – Результат актуального залишку коштів

Натиснувши на бюджет, користувач переходить до детального перегляду, де вказується загальна сума ліміту, витрачені кошти та залишок (рис.3.15). Крім того, у вікні детальної інформації передбачена функція видалення бюджету у разі його відпрацювання або скасування.

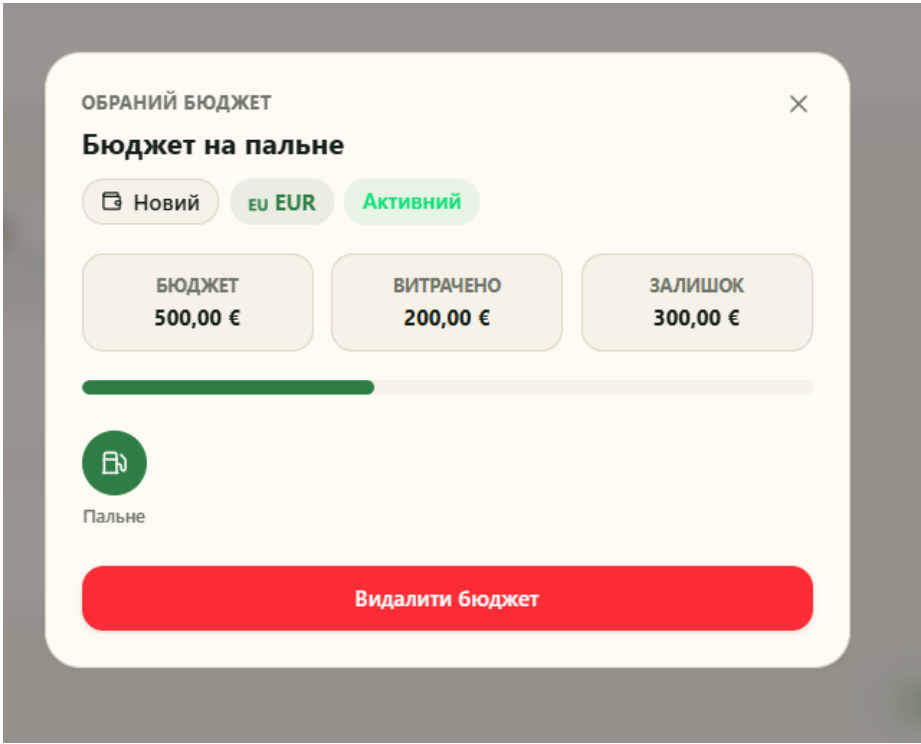


Рисунок 3.15 – Модальне вікно перегляду інформації про бюджет

У разі перевищення ліміту витрат система автоматично змінює статус бюджету та припиняє його дію. Користувач отримує сповіщення у правій верхній частині інтерфейсу (рис. 3.16), що допомагає швидко відреагувати та зупинити подальше витрачання коштів. Повідомлення можна переглянути або видалити.

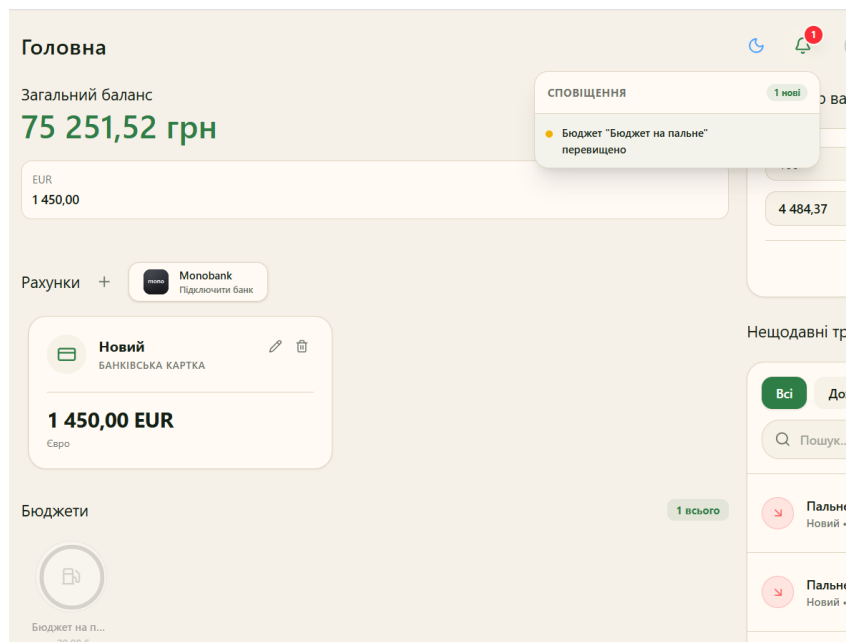


Рисунок 3.16 – Результат сповіщення при перевищенні бюджету

Для автоматизації фінансового обліку реалізовано інтеграцію з Monobank. Щоб підключити картки, користувач натискає кнопку з іконкою сервісу, після чого відкривається форма, де потрібно ввести персональний токен, який попередньо потрібно згенерувати в додатку банку та вставити у відповідне поле. Після натискання кнопки «Підключення та синхронізація» система надсилає запит до банку.

Після успішної авторизації на головній сторінці відображаються підключені карткові рахунки з актуальними залишками (рис. 3.17). Загальний баланс оновлюється автоматично, а операції за картками додаються до списку транзакцій. Імпортовані з Monobank транзакції не можна редагувати або видаляти, щоб зберегти відповідність банківській виписці.

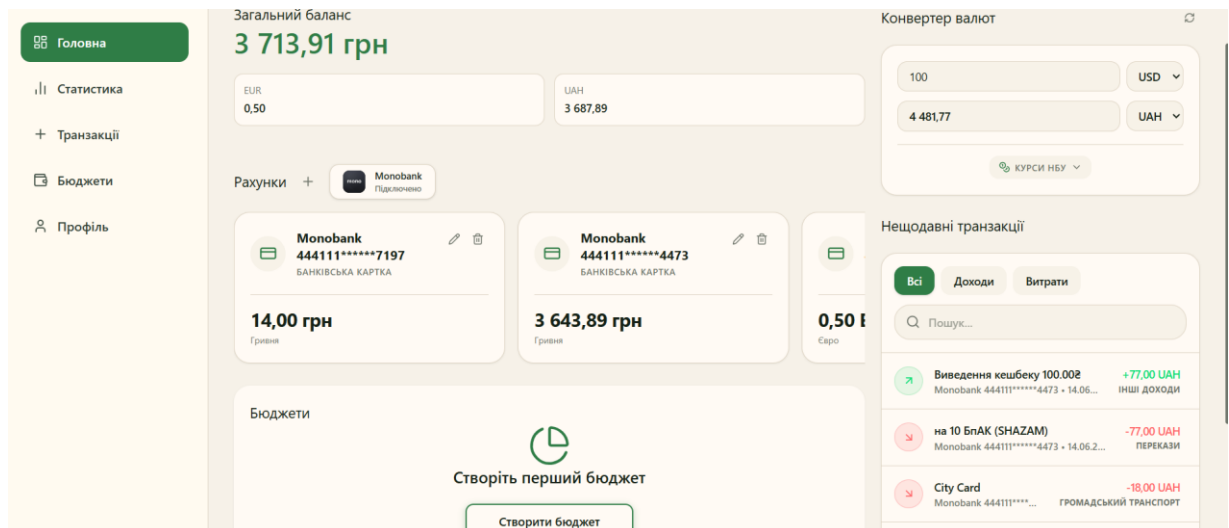


Рисунок 3.17 – Сторінка з підключеним Monobank сервісом

Допоміжним інструментом на головній панелі є вбудований міні-конвертер валют (рис.3.18). Він дозволяє користувачеві швидко розрахувати еквівалент введеної суми. Нижче поля конвертації розміщено довідник актуальних курсів найпопулярніших світових валют відносно гривні, що забезпечує швидкий доступ до фінансової інформації без переходу на сторонні ресурси.

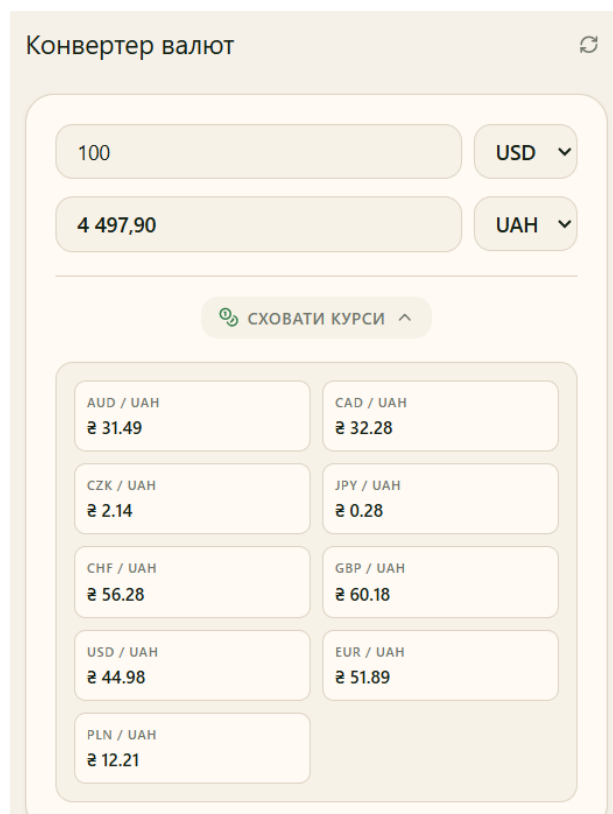


Рисунок 3.18 – Конвертер валют

Для аналізу фінансового стану в системі передбачено розділ «Статистика» (рис. 3.19). У верхній частині сторінки показано основні показники за обраний період: баланс, доходи, витрати та відсоток економії. Кругова діаграма відображає витрати за категоріями, а лінійний графік показує зміну фінансових операцій у часі.[16]

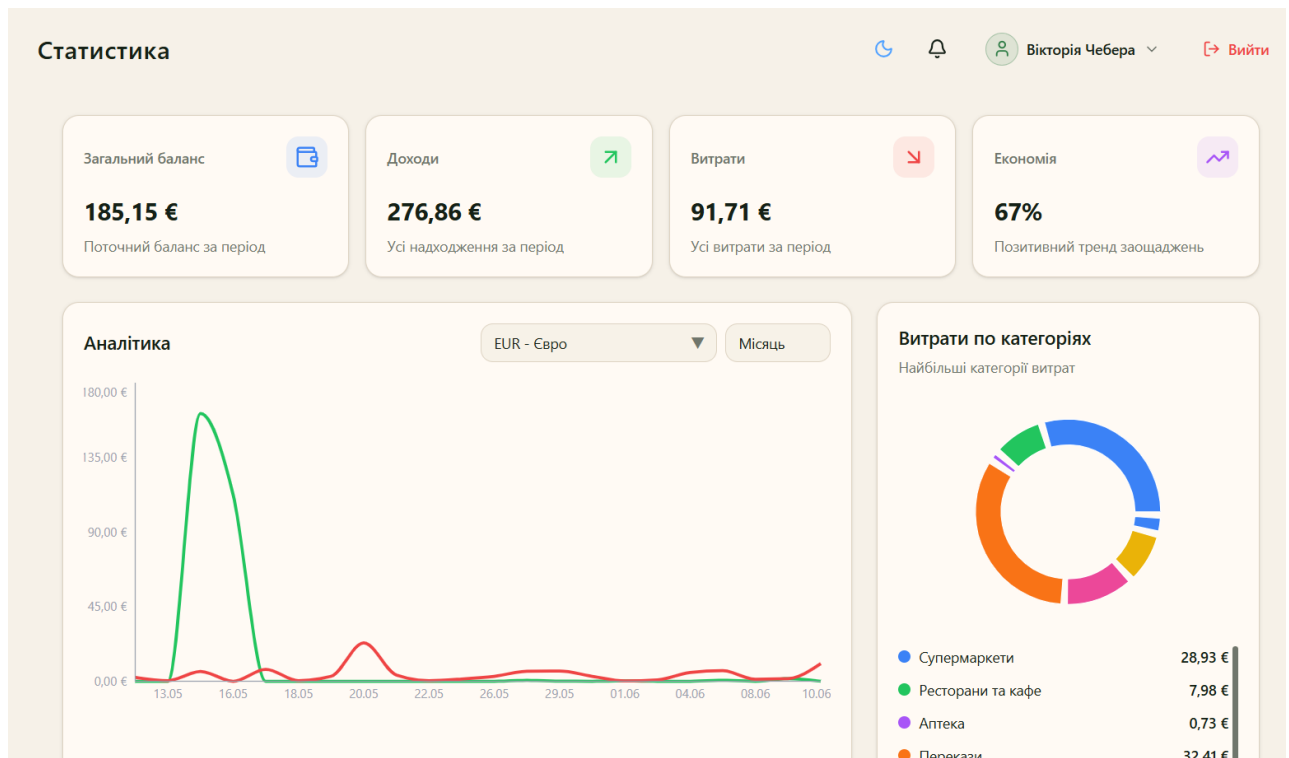


Рисунок 3.19 – Сторінка статистики

Керування даними виконується на сторінці «Профіль» (рис. 3.20). Тут користувач може змінити ім'я, переглянути статус підтвердження email і вибрати базову валюту для балансу, транзакцій та статистики.

Блок зміни пароля відображається лише для акаунтів, створених через email і пароль (рис. 3.20). Для Google-акаунтів цей блок прихований. Також на сторінці є «Небезпечна зона» з кнопкою «Видалити акаунт», яка дозволяє повністю видалити профіль і пов'язані фінансові дані.

НАЛАШТУВАННЯ АКАУНТУ

ПОВНЕ ІМ'Я

ОСНОВНИЙ EMAIL
 Підтверджено

ОСНОВНА ВАЛЮТА
\$

Зміна пароля (залиште порожнім, якщо не змінюєте)

ПОТОЧНИЙ ПАРОЛЬ

НОВИЙ ПАРОЛЬ

Скасувати Зберегти зміни

Рисунок 3.20 – Сторінка налаштування профілю для звичайного акаунту

Також у системі передбачено захищену адміністративну панель, доступну лише користувачу з роллю ADMIN. Вона дає змогу контролювати стан платформи, облікові записи та категорії.

Головна сторінка адміністратора (рис. 3.21) містить дашборд із ключовою статистикою: кількістю користувачів, транзакцій, успішних синхронізацій з банком і подій зі статусом "FAILED". Нижче розміщено журнал аудиту, де фіксуються важливі події безпеки, зокрема невдалі входи та спроби несанкціонованого доступу. Для кожної події вказано час і IP-адресу.

Finance

Головна

Користувачі

Логи безпеки

Додати категорію

Головна

Користувачі

5

+5 цього місяця

Транзакції

107

Всього оброблено

Синхронізації

1

Операцій з банком

Невдалі події

13

Подій зі статусом FAILED

Останні невдалі події аудиту

CONNECT_MONOBANK_FAILED FAILED vt221_chva@student.ztu.edu.ua • 193.194.126.37 16.06, 12:39

CONNECT_MONOBANK_FAILED FAILED vt221_chva@student.ztu.edu.ua • localhost 16.06, 12:25

CONNECT_MONOBANK_FAILED FAILED cheberavika@gmail.com • localhost 16.06, 12:00

CONNECT_MONOBANK_FAILED FAILED cheberavika@gmail.com • localhost 16.06, 11:56

Переглянути всі

Рисунок 3.21 – Головна сторінка адміністратора

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	70
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Перейшовши за посиланням «Переглянути всі», адміністратор потрапляє на повну сторінку логів (рис.3.22). У таблиці відображаються дата, дані користувача, IP-адреса та опис помилки для подій зі статусом «FAILED», що допомагає швидко виявляти проблеми й підозрілу активність.

Логи безпеки

Журнал подій

Failed

Дата	Користувач	Email	Подія	Статус	IP	Опис
10.06.2026, 09:48:39	Адміністратор Платфор...	admin@gmail.com	LOGIN_FAILED	FAILED	—	Невдала спроба входу для email: admin@gmail.com
09.06.2026, 19:44:27	Вікторія Чебера	sdfgh99@student.ztu.edu.ua	CREATE_TRANSACTION_FAILED...	FAILED	localhost	Недостатньо коштів. Сума транзакції: 4000, баланс рахунку: 399
09.06.2026, 19:42:12	Вікторія Чебера	vt221_chva@student.ztu.edu.ua	CREATE_TRANSACTION_FAILED...	FAILED	193.194.126...	Недостатньо коштів. Сума транзакції: 200, баланс рахунку: 14
09.06.2026, 19:33:15	Вікторія Чебера	vt221_chva@student.ztu.edu.ua	CONNECT_MONOBANK_FAILED	FAILED	193.194.126...	Невалідний Monobank token
09.06.2026, 19:33:06	Вікторія Чебера	vt221_chva@student.ztu.edu.ua	CONNECT_MONOBANK_FAILED	FAILED	193.194.126...	Невалідний Monobank token
09.06.2026, 19:26:40	Вікторія Чебера	sdfgh99@student.ztu.edu.ua	CREATE_TRANSACTION_FAILED...	FAILED	localhost	Недостатньо коштів. Сума транзакції: 200, баланс рахунку: 14

Показано 6 із 6 записів

Назад Сторінка 1 з 1 Вперед

Рисунок 3.22 – Сторінка журналу аудиту

У розділі «Користувачі» адміністратор переглядає базу клієнтів платформи (рис. 3.23). У таблиці показано ім'я, email, роль користувача (USER або ADMIN), статус профілю та дату реєстрації. Роль можна змінити безпосередньо в таблиці, а для швидкого керування акаунтом передбачено відповідні іконки. Також доступний пошук за ініціалами, статусом або роллю.

Користувачі

Пошук користувачів...

КОРИСТУВАЧ	РОЛЬ	СТАТУС	РЕЄСТРАЦІЯ	ДІЇ
Вікторія Чебера vt2214@student.ztu.edu.ua	USER	Активний	10 черв. 2026 р.	🗑️
Вікторія Чебера vt221@student.ztu.edu.ua	USER	Заблокований	10 черв. 2026 р.	✅ 🗑️
Віка Чебера cheberavika@gmail.com	ADMIN	Активний	8 черв. 2026 р.	
Вікторія Чебера vt221_chva99@student.ztu.edu.ua	USER	Активний	7 черв. 2026 р.	🗑️
Вікторія Чебера vt221_chva@student.ztu.edu.ua	USER	Активний	6 черв. 2026 р.	🗑️
Адміністратор Платформи admin@gmail.com	ADMIN	Активний	6 черв. 2026 р.	

Рисунок 3.23 – Сторінка зі списком користувачів

Функції блокування та видалення потрібні для безпеки системи. Якщо в логах помітна підозріла активність, адміністратор може заблокувати профіль або видалити його (рис. 3.24). Видалення реалізовано як «м'яке»: користувач отримує статус «Видалений», вхід для нього блокується, але дані залишаються в базі для збереження статистики.

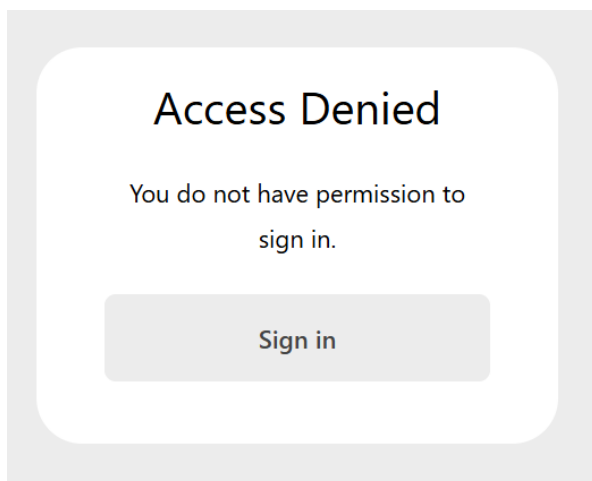


Рисунок 3.24 – Результат входу з заблокованого акаунту

Окремий інструмент адміністратора — розділ управління категоріями (рис. 3.25). Категорії відображаються у вигляді карток з іконкою, назвою, підкатегоріями та МСС-кодами. Ці коди потрібні для автоматичної класифікації банківських операцій, зокрема з Monobank. Для зручності зверху розміщено рядок пошуку.

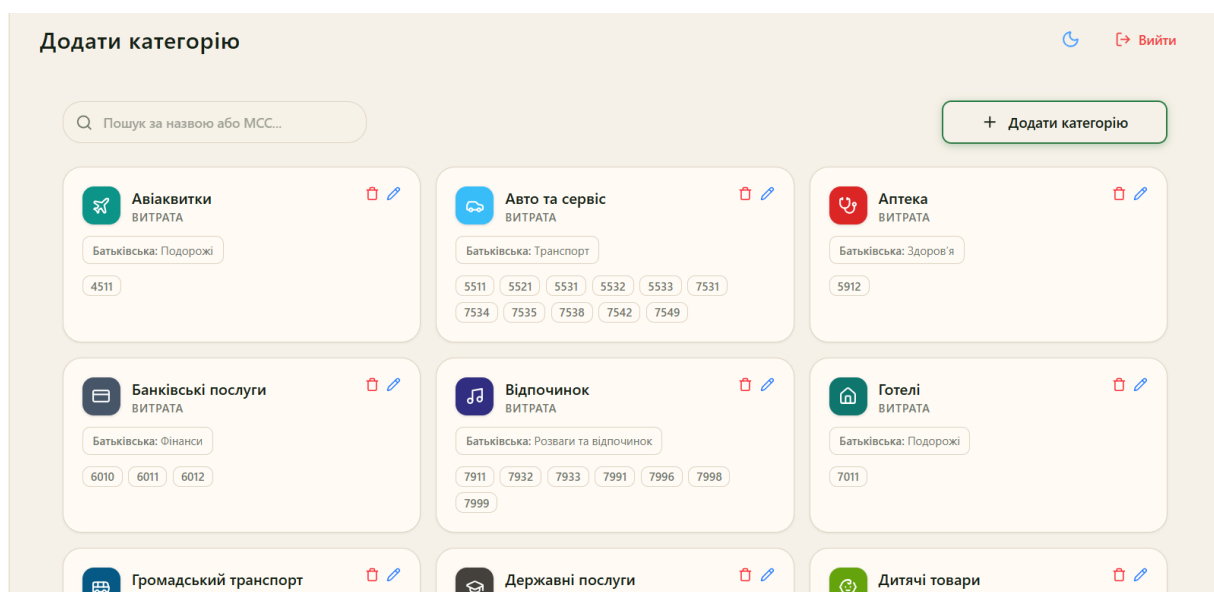


Рисунок 3.25 – Сторінка управління категоріями

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	72
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Щоб створити нову категорію, адміністратор натискає «Додати категорію» та заповнює форму (рис. 3.26). У ній зазначаються назва, тип, батьківська категорія, колір, іконка та МСС. Саме коди МСС визначають, до якої категорії автоматично потраплять транзакції з Monobank, оскільки такі операції не редагуються користувачем.

Рисунок 3.26 – Форма для додавання категорії

Категорії також можна редагувати або видаляти за допомогою іконок на картках. Під час редагування відкривається модальне вікно з уже заповненими даними, де можна змінити іконку, МСС-коди або батьківську категорію.

3.3 Тестування роботи програмного комплексу

Для перевірки точності й коректності роботи програмного комплексу було проведено функціональне тестування. Оскільки система працює з фінансовими даними, окремо перевіряли розрахункові алгоритми, логіку інтерфейсу та адміністративні механізми безпеки [30].

Спочатку тестувалося ядро системи: математичні операції та робота з даними. Під час перевірки конвертації валют виявлено похибки через представлення чисел з рухомою комою у JavaScript, саме результати містили зайві знаки після коми. Проблема було усунуто завдяки використанню типу Decimal на рівні бази даних через Prisma та примусовому округленню в серверній логіці. Також під час перевірки бюджетних лімітів з'ясувалося, що система

фіксувала перевищення вже після запису транзакції в базу даних. Логіку змінили: тепер API-запит перевіряється до збереження, тому заборонені операції не створюються. Після виявлення помилок окремі фрагменти логіки було уточнено та спрощено без зміни зовнішньої поведінки системи, що відповідає підходам рефакторингу програмного коду [23].

Після усунення виявлених дефектів фінальні результати перевірки розрахункових алгоритмів зафіксовано (таблиці 3.1).

Таблиця 3.1

Тестування розрахункових алгоритмів та логіки даних

ID	Назва	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
TA-01	Конвертація валюти	Сума: 100 USD. Курс: 44.8437	4484.37 UAH	4484.37 UAH	Пройдено
TA-02	Обчислення загального балансу (мультивалютність)	Рахунок 1: 1000 UAH. Рахунок 2: 50 USD (курс 45.00). Базова валюта: UAH	$1000 + (50 * 45) = 3250.00$ UAH	3250.00 UAH	Пройдено
TA-03	Розрахунок відсотка економії	Дохід: 500 UAH. Витрати: 200 UAH	$((500 - 200) / 500) * 100 = 60\%$	60%	Пройдено
TA-04	Алгоритм перевірки ліміту бюджету	Ліміт категорії: 1000 UAH. Поточні витрати: 900 UAH. Нова транзакція: 150 UAH	$900 + 150 = 1050 > 1000$. Система фіксує перевищення.	Виведено попередження про перевищення ліміту.	Пройдено

Другий етап охоплював перевірку логіки інтерфейсу та користувацьких сценаріїв. Під час тестування виявлено, що система не враховувала спосіб реєстрації: користувачі, авторизовані через Google OAuth, бачили форму зміни пароля, хоча її використання спричиняло серверну помилку. Проблема усунули додаванням перевірки наявності пароля в об'єкті сесії користувача. Тепер блок зміни пароля автоматично приховується для OAuth-акаунтів.

Окремо перевіряли адаптивність інтерфейсу. На мобільних екранах ліве бічне меню перекривало основний контент. Це виправили за допомогою CSS-медіазапитів: на мобільних пристроях бічне меню приховується, а замість нього відображається нижня панель навігації. Також додано заборону на редагування транзакцій, імпортованих з Monobank, щоб дані не розходилися з банківською випискою. Результати тестування умовної логіки та інтерфейсу наведені у (таблиці 3.2).[30]

Таблиця 3.2

Тестування логіки інтерфейсу та користувацьких сценаріїв

ID	Назва	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
ТС-01	Умова відображення зміни пароля	Обліковий запис створено через Google	1. Увійти в систему. 2. Відкрити "Налаштування акаунту".	Блок зміни пароля відсутній, відображається інформаційне повідомлення про зміну пароля через Google.	Пройдено
ТС-02	Зміна пароля (стандартний акаунт)	Обліковий запис створений через Email	1. Відкрити "Налаштування акаунту". 2. Ввести поточний та новий паролі. 3. Зберегти.	Пароль успішно оновлено, при наступному вході працює новий пароль.	Пройдено
ТС-03	Захист транзакцій Monobank від редагування	Підключено Monobank, є імпортована транзакція	1. Перейти до "Недавніх транзакцій". 2. Знайти транзакцію з банку. 3. Перевірити наявність кнопок редагування та видалення.	Кнопки редагування та видалення відсутні.	Пройдено
ТС-04	Адаптивність інтерфейсу (Mobile)	Відкрито головну сторінку на ПК	1. Зменшити ширину вікна браузера до мобільного розміру (напр., 375px).	Ліве меню зникає, з'являється нижня панель навігації. Блоки витрат вишиковуються в стовпчик.	Пройдено

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

На завершальному етапі перевіряли адміністративну панель, механізми безпеки та цілісність даних. Найсерйознішу помилку виявлено під час тестування «м'якого видалення» користувача. Інтерфейс змінював статус облікового запису, але система й далі дозволяла такому користувачу надсилати API-запити, бо middleware авторизації перевіряв лише активну сесію та не враховував прапорець isDeleted. Проблему усунули доповненням логіки перевірки токена: якщо власник сесії позначений як видалений, система одразу завершує сесію й блокує подальші запити. Також перевірено запис подій в AuditLog під час спроб підключення банківського сервісу з невалідним токеном і роботу механізму блокування підозрілих акаунтів. Підсумкове тестування адміністративного функціоналу підтвердило коректну роботу всіх механізмів захисту (табл. 3.3).

Таблиця 3.3

Результати тестування адміністративної панелі

ID	Назва	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
TD-01	Логування помилки підключення Monobank	Відкрито вікно інтеграцій адміністратора	1. Ввести невалідний токен (напр., "test123"). 2. Натиснути "Підключення та синхронізація". 3. Перейти в "Лог безпеки".	У журналі з'являється запис з типом "CONNECT_MONOBANK_FAILED" та описом помилки.	Пройдено
TD-02	"М'яке видалення" користувача	Адміністратор на сторінці "Користувачі"	1. Натиснути іконку видалення біля активного користувача. 2. Перевірити статус користувача в таблиці. 3. Спробувати увійти під видаленим акаунтом.	Статус змінюється на "Видалений". Спроба входу завершується помилкою (LOGIN_FAILED у логах).	Пройдено

TD-03	Блокування підозрілого акаунта	В логах є користувач з частими LOGIN_FAILED	1. Знайти користувача у розділі "Користувачі" (через пошук). 2. Змінити статус на "Заблокований".	Статус оновлено. Користувач втрачає можливість авторизації, система видає повідомлення про блокування.	Пройдено
TD-04	Додавання категорії з MCC	Сторінка "Додати категорію"	1. Заповнити назву "Таксі". 2. Додати MCC 4121. 3. Зберегти.	Категорія створена. При імпорті транзакції з кодом 4121 система автоматично застосовує категорію "Таксі".	Пройдено

Висновки до третього розділу

У цьому розділі наведена детальна схема розгортання вебплатформи. Описано кожен елемент системи та продемонстровано його взаємодію з іншими частинами платформи. Це дозволило зрозуміти архітектуру системи та взаємозв'язки між її компонентами. Далі надано опис інтерфейсу розробленого програмного продукту.

Розглянуті всі функціональні можливості додатка, залежно від ролі користувача. Відповідний користувач отримав свої унікальні функції та можливості в системі, що полегшує кожному користувачу роботу з додатком.

Також описано всі сторінки системи та їх принцип роботи, зокрема інтерфейс головної панелі, управління транзакціями та адміністративний розділ. При цьому чітко розмежовано статистику: користувач аналізує лише особисті фінанси та конвертацію валют, тоді як адміністратор контролює загальну активність платформи та події аудиту.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	77
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та реалізовано веб-платформу системи управління особистими фінансами, проведено аналіз предметної області та порівняно різні існуючі сервіси обліку коштів. На основі цього аналізу були визначені основні вимоги до функціоналу та безпеки додатку. В процесі дослідження була розроблена архітектура, підходи та основні технології реалізації клієнтської та серверної частин.

Проектування платформи включало аналіз варіантів використання програмного продукту та проектування структури, включаючи моделювання взаємодії між модулями. Також було спроектовано базу даних для надійного зберігання інформації про користувачів, їхні фінансові рахунки, транзакції, бюджети, налаштування категорій та журнали аудиту.

Описано алгоритм роботи з додатком та послідовні кроки його використання. Розглянуто основний функціонал, включаючи ручне створення транзакцій, налаштування бюджетів, а також ключову особливість — автоматичний імпорт операцій з банківської системи Monobank за допомогою персонального токена з подальшою їх автоматичною категоризацією.

Також була надана детальна схема розгортання платформи, описано взаємодію структурних елементів системи та наведено опис інтерфейсу розробленого програмного продукту для різних типів користувачів. Показано роботу статистики: якщо звичайний користувач бачить графіки власних витрат, економії та конвертації валют, то адміністратор отримує загальні метрики платформи, моніторить успішність синхронізацій та аналізує події з логів безпеки.

Архітектура платформи побудована з урахуванням суворих правил цілісності фінансових даних. Навмисна заборона на редагування транзакцій, імпортованих з банку, гарантує точність та неспотвореність статистики. Використання системи кодів MCC для маршрутизації операцій забезпечує автоматизований та достовірний облік, формуючи довіру користувача до

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	78
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

аналітичних даних системи.

Додаток має інтуїтивний інтерфейс з підтримкою адаптивного дизайну та зміни теми, який дозволяє користувачам легко створювати рахунки, відстежувати залишки та контролювати перевищення лімітів бюджетів. Користувачі можуть зручно взаємодіяти зі системою з будь-якого пристрою, фіксувати свої доходи та витрати, а адміністратор має змогу ефективно керувати платформою та блокувати підозрілу активність.

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	79
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MySQL Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.mysql.com/doc/> (дата звернення: 10.01.2026).
2. Next.js Documentation [Електронний ресурс]. – Режим доступу до ресурсу: <https://nextjs.org/docs> (дата звернення: 15.01.2026).
3. TypeScript Documentation [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/> (дата звернення: 20.01.2026).
4. React Documentation [Електронний ресурс]. – Режим доступу до ресурсу: <https://react.dev/>. (дата звернення: 17.02.2026).
5. Prisma Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.prisma.io/docs/> (дата звернення: 13.01.2026).
6. UML Specification [Електронний ресурс] – Режим доступу до ресурсу: <https://www.omg.org/spec/UML> (дата звернення: 23.03.2026).
7. NextAuth Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://next-auth.js.org/> (дата звернення: 08.02.2026).
8. Google Identity. Google Account Authorization. URL Documentation: <https://developers.google.com/identity/protocols/oauth2> (дата звернення: 10.02.2026).
9. Haverbeke M. Eloquent JavaScript, 4th Edition: a modern introduction to programming. No Starch Press, 2024. 459 с. ISBN 9781718504103. URL: <https://eloquentjavascript.net/> (дата звернення: 14.03.2026).
10. JSON Web Tokens Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.jwt.io/introduction#what-is-json-web-token> (дата звернення: 15.02.2026).
11. Next.js Documentation. Rendering: Server-Side Rendering (SSR) and Static Site Generation (SSG) [Електронний ресурс]. – Режим доступу до ресурсу: <https://nextjs.org/docs/app/building-your-application/rendering> (дата звернення: 24.02.2026).

12. Railway Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.railway.com/> (дата звернення: 11.04.2026).
13. Vercel Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://vercel.com/docs> (дата звернення: 22.05.2026).
14. Monobank Api Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://monobank.ua/api-docs> (дата звернення: 14.05.2026).
15. Booch G. The Unified Modeling Language User Guide / G. Booch, J. Rumbaugh, I. Jacobson. – 2nd ed. – Addison-Wesley Professional, 2005. – 496 p. – ISBN 978-0321267979 (дата звернення: 10.03.2026).
16. Recharts Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://recharts.org/en-US/> (дата звернення: 09.04.2026).
17. Tailwind CSS Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com/docs> (дата звернення: 30.01.2026).
18. Connolly T. Database Systems: A Practical Approach to Design, Implementation, and Management / T. Connolly, C. Begg. – 6th ed. – Pearson, 2014. – 1424 p. ISBN 978-1292061188 (дата звернення: 10.03.2026).
19. Sommerville I. Software Engineering / I. Sommerville. – 10th ed. – Pearson, 2015. – 790 p. ISBN 978-1292096131 (дата звернення: 24.02.2026).
20. Norman D. The Design of Everyday Things / D. Norman. – Revised and Expanded Edition. – Basic Books, 2013. – 368 p. ISBN 978-0465050659 (дата звернення: 30.03.2026).
21. Когут П. М. Основи розробки веб-орієнтованих інформаційних систем : навч. посібник / П. М. Когут, Б. А. Лісевич. – Львів : Вид-во Львівської політехніки, 2020. – 228 с (дата звернення: 10.03.2026).
22. Git [Електронний ресурс] – Режим доступу до ресурсу: <https://git-scm.com/docs> (дата звернення: 05.01.2026).
23. Fowler M. Refactoring: Improving the Design of Existing Code / M. Fowler. – 2nd ed. – Boston: Addison-Wesley Professional, 2018. – 448 p. ISBN 978-0134757599 (дата звернення: 05.02.2026).

- 24.SWR Documentation. Data-Fetching Library for React [Електронний ресурс] / Vercel. – Режим доступу до ресурсу: <https://swr.vercel.app/> (дата звернення: 16.05.2026).
- 25.Winand M. SQL Performance Explained: Everything Developers Need to Know about SQL Performance / M. Winand. – 2nd ed. – [S.l.]: Markus Winand, 2020. – 210 p. ISBN 978-3950307878 (дата звернення: 10.02.2026).
- 26.Banks A. Learning React: Modern Patterns for Developing React Apps / A. Banks, E. Porcello. – 2nd ed. – Sebastopol: O'Reilly Media, 2020. – 350 p. ISBN 978-1492051725. (дата звернення: 11.02.2026).
- 27.Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann. – Sebastopol : O'Reilly Media, 2017. – 616 p. ISBN 978-1449373320 (дата звернення: 04.03.2026).
- 28.Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale / B. Cherny. – Sebastopol : O'Reilly Media, 2019. – 346 p. – ISBN 978-1492037651 (дата звернення: 17.03.2026).
- 29.Richardson L. RESTful Web APIs / L. Richardson, M. Ruby. – Sebastopol : O'Reilly Media, 2013. – 404 p. – ISBN 978-1449358068 (дата звернення: 16.04.2026).
- 30.Beck K. Test-Driven Development: By Example / K. Beck. – Boston : Addison-Wesley Professional, 2002. – 240 p. – ISBN 978-0321146533 (дата звернення: 05.05.2026).

ДОДАТКИ

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	83
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

Фрагмент логіки маппінгу категорій

```

const dbMapping = await prisma.mccMapping.findMany({include: {category: true}});
const mccMap = new Map(dbMapping.map(m => [m.mcc, m.category]));

const otherExpenseCategory = await prisma.category.findFirst({
  where: {
    category_name: "Інші витрати",
    category_type: "EXPENSE"
  }
});
const defaultIncomeCategory = await prisma.category.findFirst({
  where: {
    category_name: "Інші доходи",
    category_type: "INCOME"
  }
});

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

const existingTransactions = await prisma.transaction.findMany({
  where: {user_id: currentUserId, source: "MONOBANK"},
  select: {external_id: true}
});
const existingIds = new Set(existingTransactions.map(t => t.external_id).filter(Boolean));

const transactionsToCreate: Prisma.TransactionCreateManyInput[] = [];

for (const monoAccount of monoAccounts) {
  const accountName = monoAccount.maskedPan?.[0] ? `Monobank ${monoAccount.maskedPan[0]}` : "Monobank Картка";
  const currencyCode = MONOBANK_CURRENCIES[monoAccount.currencyCode] ?? "UAH";

  const currentBalance = new Prisma.Decimal(monoAccount.balance / 100);

  const bankAccount = await prisma.bankAccount.upsert({
    where: {account_id: monoAccount.id},
    update: {balance: currentBalance, account_name: accountName},
    create: {
      account_id: monoAccount.id,
      user_id: currentUserId,
      account_name: accountName,
      account_type: AccountTypeEnum.CARD,
      balance: currentBalance,
      currency: currencyCode,
    },
  });
}

```

```

let statementResponse: Response | null = null;
let retryCount = 0;
const maxRetries = 3;
let backoffTime = 1000;

while (retryCount <= maxRetries) {
  const tempResponse = await fetch(
    `https://api.monobank.ua/personal/statement/${monoAccount.id}/${from}/${to}`,
    {headers: {"X-Token": monoToken}}
  );

  if (tempResponse.ok || tempResponse.status !== 429) {
    statementResponse = tempResponse;
    break;
  }

  retryCount++;
  if (retryCount <= maxRetries) {
    await new Promise((resolve) => setTimeout(resolve, backoffTime));
    backoffTime *= 2;
  }
}

if (!statementResponse || !statementResponse.ok) {

  await createAuditLog({
    userId: currentUser.id,
    action: "SYNC_MONOBANK_RATE_LIMIT",
    status: "FAILED",
    details: `Перевищено ліміт запитів (Rate Limit) для картки ${monoAccount.id} після
    ${maxRetries} спроб.`,
    ipAddress: ipAddress,
  });
  continue;
}

const statement = await statementResponse.json();

if (Array.isArray(statement)) {
  for (const item of statement) {
    if (existingIds.has(String(item.id))) continue;

    const transactionType: TransactionType = item.amount > 0 ? "INCOME" : "EXPENSE";

    let categoryId = transactionType === "EXPENSE"
      ? (mccMap.get(item.mcc)?.category_id ?? otherExpenseCategory?.category_id)
      : defaultIncomeCategory?.category_id;

    if (!categoryId) continue;

    transactionsToCreate.push({

```

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	85
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        transaction_id: crypto.randomUUID(),
        user_id: currentUserId,
        account_id: bankAccount.account_id,
        category_id: categoryId,
        amount: new Prisma.Decimal(Math.abs(item.amount / 100)),
        type: transactionType,
        description: item.description || "Транзакція Monobank",
        transaction_date: new Date(item.time * 1000),
        external_id: String(item.id),
        source: "MONOBANK",
    });
}
}

```

Додаток Б

Обробка помилок вебхука

```

    return NextResponse.json({ status: "created" });

  } catch (error) {
    console.error("[Webhook] Критична помилка обробки:", error);
    return new NextResponse("OK", { status: 200 });
  }
}

```

		Чебера В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	86
		Власенко О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		