Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розробка веб-застосунку для планування особистих фінансів з використанням Ruby on Rails»

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

_____ Денисюк О.В.
(підпис) (прізвище та ініціали)

Керівник _____ Михалик Д.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Липак Г. І.
(підпис) (прізвище та ініціали)

Тернопіль 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

студенту Денисюку Олександру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка веб-застосунку для планування особистих фінансів з використанням Ruby on Rails»

Керівник роботи Михалик Дмитро Михайлович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Мова програмування Ruby, фреймворкRuby on Rails, Tailwind CSS, мова розмітки HTML, база даних MySQL, бібліотека JavaScript: Stimulus, інструмент графіків: Chartkick + Chart.js

4. Зміст роботи (перелік питань, які потрібно розробити)
Сформувати завдання, проаналізувати предметну область, визначити функціональні вимоги, спроєктувати, розробити та протестувати веб-застосунок для планування особистих фінансів з використанням «Ruby on Rails»

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми варіантів використання, зв'язків, зображення архітектури системи, графічного інтерфейсу користувача та результатів тестування, таблиці

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми роботи		
2	Аналіз предметної області		
3	Проєктування загальної структури системи		
4	Проєктування структур окремих модулів		
5	Програмна реалізація вебзастосунка		
6	Тестування розробленої системи		
7	Виконання розділу з безпеки життєдіяльності та основ охорони праці		
8	Оформлення пояснювальної записки		
9	Нормоконтроль та рецензування		
10	Перевірка на плагіат		
11	Попередній захист		
12	Захист роботи		

Студент _____ Денисюк О. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Михалик Д. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-застосунку для планування особистих фінансів з використанням Ruby on Rails // Кваліфікаційна робота освітнього рівня «Бакалавр» // Денисюк Олександр Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41 // Тернопіль, 2025 // с. – 64, рис. – 28, таб. – 4, кресл. – 11, додат. – 3, бібліогр. – 28.

Ключові слова: фінанси, планування, web, застосунок, ruby, html, css, rails, база даних.

Кваліфікаційна робота присвячена розробці вебзастосунку для управління особистими фінансами.

В першому розділі кваліфікаційної роботи описано актуальність теми, проведено аналіз існуючих рішень. Висвітлено основні функціональні вимоги до системи. Розглянуто архітектурні підходи та вибір технологій.

В другому розділі кваліфікаційної роботи здійснено проєктування структури застосунку, бази даних, реалізовано серверну логіку та інтерфейс користувача. Досліджено особливості побудови інтерфейсу з використанням фреймворку Tailwind CSS.

В третьому розділі кваліфікаційної роботи описано функціональність реалізованого застосунку, виконано тестування.

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці, а саме: класифікація стихійних явищ, розрахунок блискавкозахисту.

Об'єкт дослідження: процес управління особистими фінансами за допомогою вебзастосунків.

Предмет дослідження: методи та засоби реалізації вебсистеми для обліку, аналізу й планування особистих фінансів.

ABSTRACT

Development of a Web Application for Personal Finance Planning Using Ruby on Rails // Denysiuk Oleksandr Volodymyrovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, Group SP-41 // Ternopil, 2025 // p. – 64, fig. – 28, tab. – 4, chair. – 11, app. – 3, bibl. – 28.

Keywords: finance, planning, web, application, ruby, html, css, rails, database.

This qualification work is dedicated to the development of a web application for personal finance management.

The first chapter presents the relevance of the topic and analyzes existing solutions. The main functional requirements of the system are outlined, and architectural approaches and technology choices are discussed.

The second chapter focuses on the design of the application structure and database, implementation of server-side logic, and development of a user interface. Special attention is given to the specifics of interface development using the Tailwind CSS framework.

The third chapter describes the implemented functionality of the application and presents the results of manual testing.

The fourth chapter addresses occupational safety and health issues, including the classification of natural hazards and lightning protection calculations.

Object of research: the process of managing personal finances using web applications.

Subject of research: methods and tools for implementing a web system for accounting, analysis, and planning of personal finances.

ЗМІСТ

ВСТУП

У сучасних умовах швидкого розвитку цифрових технологій та зростаючого фінансового навантаження на громадян, питання ефективного управління особистими фінансами набуває особливої актуальності. Більшість людей мають постійні доходи й витрати, однак часто не контролюють фінансові потоки, що призводить до нераціонального використання коштів, боргів та нездатності до накопичення. І хоч дослідження НБУ показують що, рівень фінансової грамотності українців покращився з 11,6 до 12,2 бала (шкала від 1 до 21), до рівня європейських країн ми ще не дотягуємо.

У зв’язку з цим виникає потреба в інструментах, які допомагають користувачам вести облік доходів, витрат, планувати бюджет, контролювати фінансові цілі та аналізувати власну платоспроможність. На ринку існує низка подібних застосунків, однак багато з них — комерційні, перевантажені функціями, не мають підтримки української мови або не враховують локальні особливості. Це створює нішу для створення власного, адаптованого до потреб користувачів, рішення.

Метою кваліфікаційної роботи є розробка веб-застосунку для планування особистих фінансів, який дозволить користувачам здійснювати облік доходів та витрат, встановлювати бюджет, переглядати аналітику й контролювати фінансові показники у зручному онлайн-інтерфейсі. Для досягнення поставленої мети необхідно вирішити наступні завдання:

– Провести аналіз існуючих аналогів та визначити їхні сильні і слабкі сторони.

– Сформувати вимоги до функціональності, безпеки та дизайну майбутнього веб-застосунку.

– Спроєктувати архітектуру системи, базу даних та користувацький інтерфейс.

– Реалізувати веб-застосунок на базі Ruby on Rails із використанням CSS, HTML, та MySQL.

– Забезпечити базову авторизацію, CRUD-функціонал та візуалізацію даних.

– Провести тестування системи та оцінити її ефективність і зручність.

– Надати рекомендації щодо подальшого розвитку й удосконалення системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Що таке веб-застосунок, та його структура

Веб-застосунок (англ. web application) — це програмне забезпечення, яке користувачі можуть використовувати через веб-браузер, без необхідності встановлення на пристрій. Він працює на віддалених серверах і надає різноманітні функції або послуги користувачам через Інтернет.

На відміну від традиційних веб-сайтів, які переважно надають статичну інформацію, веб-застосунки дозволяють користувачам взаємодіяти з даними, виконувати дії, зберігати та обробляти інформацію в режимі реального часу [2]. Прикладами таких застосунків є електронна пошта (наприклад, Gmail), системи управління контентом (CMS) та соціальні мережі.

Типова архітектура застосунку базується на клієнт-серверній моделі та складається з трьох основних компонентів [3]:

1. Клієнтська частина (Front-end): Це інтерфейс, з яким взаємодіє користувач через веб-браузер. Вона розробляється за допомогою HTML, CSS та JavaScript і відповідає за відображення даних та обробку взаємодії користувача.
2. Серверна частина (Back-end): Це логіка, яка обробляє запити від клієнта, взаємодіє з базою даних та формує відповіді. Серверна частина може бути реалізована за допомогою різних мов програмування та фреймворків, таких як Ruby on Rails, Python (Django), PHP (Laravel).
3. База даних: Це система зберігання даних, яка використовується для збереження та отримання інформації, необхідної для роботи застосунку. Популярні системи управління базами даних включають MySQL, PostgreSQL, MongoDB тощо.

Ця трирівнева архітектура дозволяє розділити відповідальність між різними компонентами, що сприяє кращій масштабованості, підтримці та безпеці застосунку. Принцип роботи веб-застосунку зображено на рисунку 1.1.

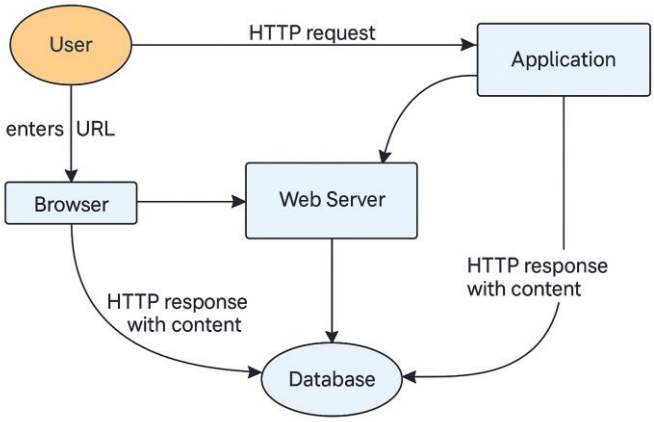

Рисунок 1.1 – Принцип роботи веб-застосунку

Принцип роботи веб-застосунку складається з того, що користувач вводить URL-адресу у веб-браузері. Після цього браузер надсилає HTTP-запит до веб-сервера. Сервер, отримавши цей запит, передає його до серверної частини застосунку, де відбувається обробка. Якщо потрібно — серверна логіка звертається до бази даних для зчитування або зміни інформації. На основі отриманих даних сервер формує відповідь і повертає її клієнту. Браузер користувача, отримавши відповідь, відображає її у вигляді веб-сторінки, інтерфейсу або іншого результату взаємодії.

1.2 Аналіз існуючих рішень

На сучасному етапі розвитку інформаційних технологій існує значна кількість програмних рішень, які спрямовані на ведення особистого фінансового обліку та планування бюджету. Ці рішення суттєво відрізняються між собою як за функціональністю, так і за підходами до організації взаємодії з користувачем. Зокрема, окремі з них орієнтовані на ручне введення транзакцій, що дозволяє користувачу самостійно фіксувати витрати, призначати категорії та контролювати баланс. Інші ж застосунки передбачають автоматизовану роботу з банківськими

рахунками, здійснюючи імпорт даних із фінансових установ і автоматичну класифікацію операцій [4].

Крім того, існують сервіси, які акцентують увагу лише на фіксації витрат, тоді як інші пропонують інструменти для детального планування бюджету, постановки фінансових цілей та аналітики [5]. Така різноманітність функціоналу та підходів зумовлює потребу в аналізі наявних рішень перед розробкою власного веб-застосунку.

У межах кваліфікаційної роботи було проаналізовано три популярні сучасні фінансові рішення, які мають велику користувацьку базу та відповідають сучасним вимогам до програм для обліку особистих фінансів. До розгляду були обрані: Wallet, Spendee та YNAB (You Need A Budget).

Розглянемо перший застосунок Wallet (BudgetBakers). Wallet — це популярний сервіс для управління особистими фінансами, розроблений чеською компанією BudgetBakers. Дане рішення орієнтоване на широке коло користувачів і дозволяє вести облік доходів та витрат, створювати бюджети, фінансові цілі, здійснювати аналіз грошових потоків, а також отримувати звіти у вигляді графіків і таблиць [6]. Однією з ключових переваг Wallet є наявність як мобільної, так і веб-версії, що забезпечує зручність доступу з будь-якого пристрою. Інтерфейс веб-застосунку зображено на рисунку 1.2.

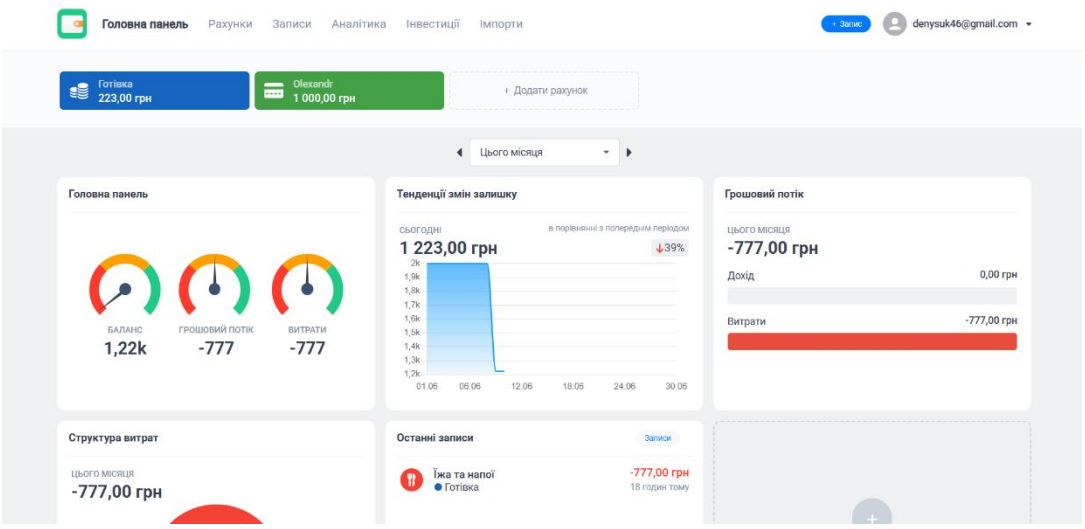

Рисунок 1.2 – Інтерфейс застосунку Wallet

На головному екрані Wallet відображається поточний баланс, останні транзакції, аналітика витрат у вигляді діаграм, а також налаштування категорій витрат. Інтерфейс досить зрозумілий, із сучасною візуальною структурою.

Переваги: підтримка української мови, наявність веб-інтерфейсу, візуальна аналітика, можливість синхронізації з банками (у платній версії). Недоліки: обмежений функціонал у безкоштовній версії, проблеми з повноцінною інтеграцією українських банків.

Розглянемо наступне рішення. Spendee — це багатоплатформовий застосунок для управління особистими фінансами, розроблений компанією Cleevio s.r.o. Програма пропонує користувачеві простий і візуально привабливий інтерфейс для контролю щоденних витрат, а також зручні інструменти для бюджетування й аналітики. Особливістю Spendee є можливість створення окремих «гаманців» для різних сфер життя (наприклад, особисті витрати, подорожі, домашній бюджет), що дозволяє ефективно структурувати фінансову інформацію. Загальний вигляд інтерфейсу Spendee продемонстровано на рисунку 1.3.

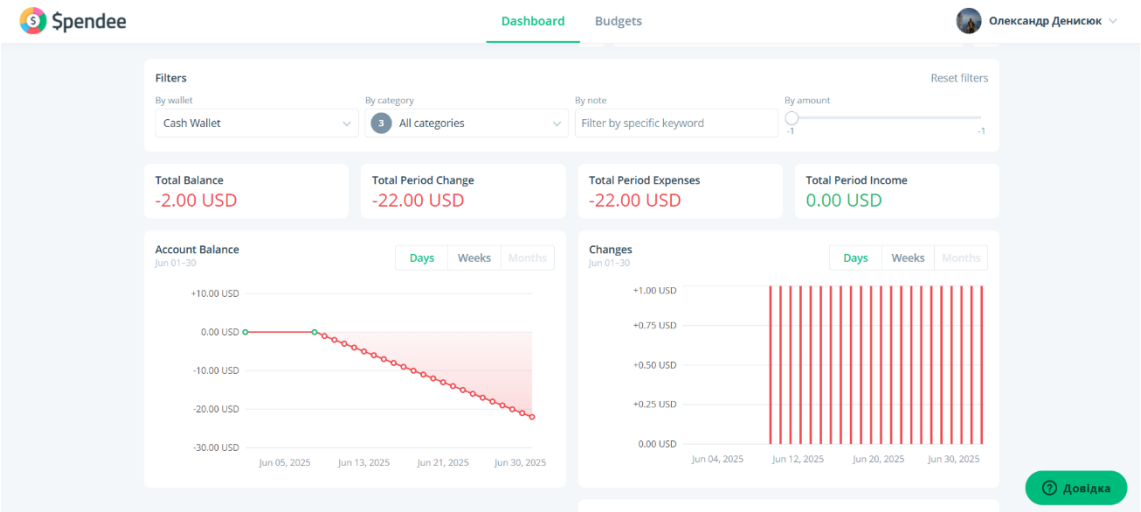

Рисунок 1.3 – Інтерфейс застосунку Spendee

Інтерфейс Spendee побудований за принципом карток: кожна категорія представлена кольором, сума витрат — графіком, а навігація інтуїтивно проста. Присутній вибір валют, базова статистика та імпорт транзакцій.

Переваги: простий і привабливий інтерфейс, можливість створення окремих гаманців, наявність веб-версії, кросплатформеність. Недоліки: обмеження у безкоштовній версії, англомовний інтерфейс, відсутність прямої підтримки українських банків [7].

You Need A Budget (YNAB) — відомий американський фінансовий застосунок, головним завданням якого є допомогти користувачам набути фінансової дисципліни за допомогою детального бюджетування. Дане рішення відрізняється від інших сервісів тим, що не просто фіксує витрати, а навчає користувача планувати кожну гривню/долар ще до моменту її витрати.

YNAB орієнтований на довготривале фінансове планування, розподіл коштів за категоріями, створення фондів (на кшталт заощаджень, позапланових витрат тощо), а також відстеження змін бюджету з часом. Користувачі мають змогу формувати щомісячні бюджети, керувати боргами та досягати поставлених фінансових цілей. Інтерфейс застосунку YNAB зображено на рисунку 1.4.

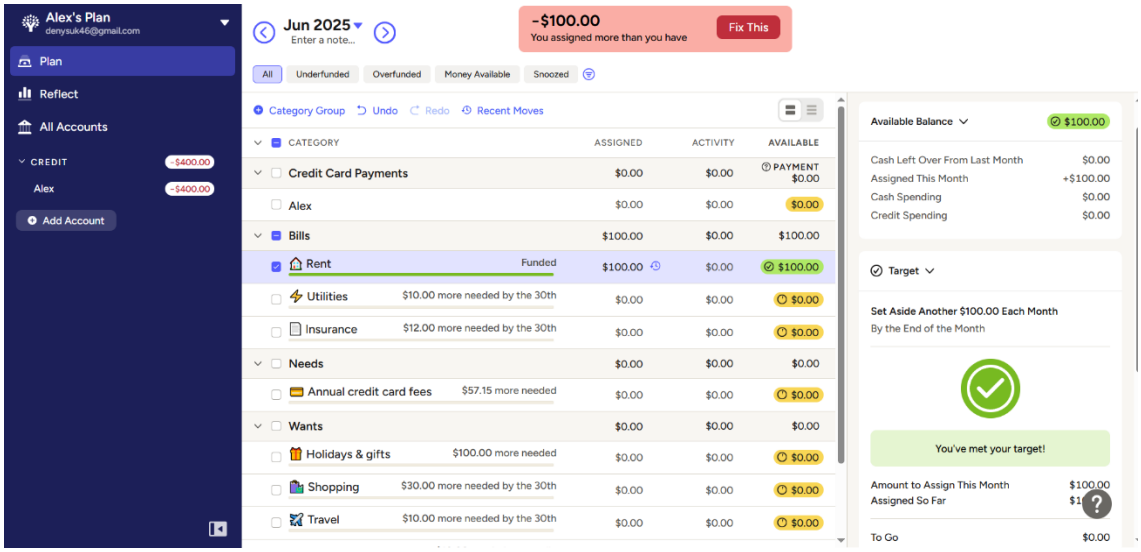

Рисунок 1.4 – Інтерфейс застосунку YNAB

На головному екрані відображено категорії витрат, наявні кошти, бюджет на місяць та прогноз залишків. Інтерфейс побудований у вигляді таблиці з можливістю редагування, фільтрації та перегляду аналітики.

Переваги: веб-версія з повним функціоналом, потужна система бюджетування, навчальні матеріали. Недоліки: висока вартість підписки (від $99/рік), відсутність локалізації, важкий до освоєння функціонал [8].

Короткий аналіз кожного рішення, їх недоліки та переваги наведено в таблиці 1.1.

Таблиця 1.1. – Порівняння переваг та недоліків

Критерій	Wallet	Spendee	YNAB
Мова інтерфейсу	Українська, англійська	Англійська	Англійська
Розуміння функціоналу	Легко	Середній рівень	Важко освоїти
Підтримка валют	Багатовалютність	Багатовалютність	Багатовалютність
Доступність українським користувачам	Висока	Середня	Низька
Фокус функціоналу	Облік витрат, бюджети, аналітика	Візуальний контроль фінансів	Жорстке планування бюджету
Безкоштовний функціонал	Частково доступний	Частково доступний	Лише пробний період

Отже, на основі проведеного аналізу сучасних програмних рішень для управління особистими фінансами можна зробити висновок, що ринок пропонує широкий спектр застосунків із різноманітним функціоналом, орієнтованим на різні потреби користувачів.

Водночас, усі три сервіси мають свої обмеження — від платного функціоналу до відсутності локалізації чи інтуєтивно незрозумілого інтерфейсу. Це свідчить про необхідність створення веб-застосунку, який би поєднував локалізацію та доступний функціонал для ефективного управління фінансами, що є актуальним завданням для розробки в межах кваліфікаційної роботи.

1.3 Постановка завдання та визначення цільової аудиторії

Для успішної реалізації веб-застосунку з планування особистих фінансів необхідно чітко визначити вимоги до системи та цільову аудиторію. Відповідність цим вимогам забезпечить ефективність, масштабованість та зручність у використанні розробленого програмного забезпечення.

Мета веб-застосунку — створення зручного та локалізованого інструменту для управління особистими фінансами, який дозволяє користувачам вести облік доходів і витрат, планувати бюджет, аналізувати фінансові показники та отримувати нагадування про перевищення бюджету, забезпечуючи інтуїтивний інтерфейс, підтримку української мови.

Проаналізувавши предметну область і існуючі рішення, можна сформувати загальні вимоги до розробки застосунку:

– Реєстрація та автентифікація користувача
– Створення та редагування власного бюджету.
– Введення та перегляд транзакцій(доходи та витрати).
– Вибір категорії, дати, опису та суми транзакції.
– Можливість фільтрувати транзакції за датою, категорією тощо.
– Можливість встановлення бюджету на місяць/категорію.
– Отримання сповіщенньь про перевищення бюджету.
– Можливість створення власних фінансових цілей та їх редагування.
– Загальний баланс, сума витрат і доходів, відображення у вигляді графіків.
– Перегляд анатілити за допомогою графіків для розуміння куди було витрачено бюджет.

Нефункціональні вимоги:

– Мови: українська як основна.
– Сумісність: веб-застосунок має коректно відображатись у сучасних браузерах.

– Розширюваність: архітектура повинна дозволяти масштабування та додавання нових функцій.

– Безпека: обов’язкове шифрування паролів, захист від SQL-ін’єкцій, CSRF та інших типових загроз.

– Продуктивність: час відгуку інтерфейсу не має перевищувати 1 секунди при типових діях.

Визначимо цільову аудиторію розроблювального застосунку [9]. Основна аудиторія — особи віком від 18 до 55 років, які мають потребу в ефективному управлінні особистими фінансами. Це включає молодь (18–30 років) — студентів і молодих фахівців, які контролюють витрати та накопичують на короткострокові цілі, наприклад, подорожі чи покупку техніки; осіб середнього віку (30–45 років) — сім’ї та працівників, які планують сімейний бюджет, відстежують регулярні витрати (комунальні послуги, продукти, освіта дітей) і заощадження; а також старше покоління (45–55 років), орієнтоване на довгострокові фінансові цілі, такі як пенсійні накопичення чи великі покупки.

Отже основне завдання роботи — створити зручний і локалізований веб-застосунок для управління фінансами, орієнтований на широку аудиторію користувачів (18-55років) із різним рівнем фінансової грамотності.

1.4 Вибір технологій

У процесі розробки веб-застосунку для планування особистих фінансів було обрано сучасний набір технологій, що забезпечують стабільну архітектуру, високу швидкодію, безпеку та зручність використання. Враховуючи особливості проєкту та власні навички, було зроблено акцент на гнучких open source рішеннях [10].

Основні технології та інструменти:

Ruby on Rails — сучасний веб-фреймворк, що працює за принципом Model-View-Controller (MVC). Його використано для побудови логіки застосунку,

обробки запитів користувача та взаємодії з базою даних. Rails дозволяє значно скоротити час розробки завдяки великій кількості готових рішень та зрозумілій структурі проєкту.

MySQL — реляційна система управління базами даних з відкритим вихідним кодом, яка відзначається високою продуктивністю та масштабованістю. У проєкті MySQL використовується як основна система управління для зберігання інформації про користувачів, транзакції, бюджети й категорії витрат.

HTML5 — мова розмітки, яка відповідає за побудову структури веб-інтерфейсу. Вона використовується для формування сторінок, форм, таблиць та інших елементів UI.

TailwindCSS — утилітарний CSS-фреймворк, який дозволяє швидко створювати адаптивні та сучасні інтерфейси. Його перевагою є висока гнучкість у компонуванні стилів без потреби в написанні кастомного CSS.

JavaScript (Stimulus) — використовується для реалізації динамічної поведінки інтерфейсу: обробки подій, валідації форм, асинхронної взаємодії з сервером. Stimulus дозволяє підключати JS-логіку до HTML-структури у зрозумілий та модульний спосіб.

Chartkick або ApexCharts — це графічні бібліотеки, які застосовуються для побудови діаграм та візуалізації фінансових даних. У межах реалізованого веб-застосунку вони використовуються для формування інтерактивних графіків, що дозволяє користувачеві візуально аналізувати доходи, витрати та залишки бюджету.

Devise — Ruby-гем, який реалізує функціональність автентифікації та авторизації користувачів. Забезпечує підтримку реєстрації, входу в систему, скидання пароля та керування сесіями.

Simple Form — бібліотека для Ruby on Rails, що спрощує створення HTML-форм. Вона дозволяє ефективно будувати адаптивні та зручні для користувача форми з урахуванням стандартів оформлення інтерфейсу.

Ransack — Ruby-гем для реалізації фільтрації, пошуку та сортування даних у таблицях. Його інтеграція у веб-застосунок дозволяє здійснювати швидке знаходження потрібної транзакції чи категорії.

Kaminari — засіб для розбиття великих масивів даних на сторінки (пагінації). Застосовується у таблицях з транзакціями та фінансовими цілями для покращення навігації та зручності користувача.

Hotwire (Turbo) — технологія, яка дозволяє реалізовувати динамічну взаємодію на стороні клієнта без повного використання JavaScript. Завдяки використанню Turbo Stream реалізовано миттєве оновлення частин інтерфейсу без перезавантаження сторінки, зокрема під час додавання транзакцій, оновлення сповіщень або редагування фінансових цілей.

Figma — онлайн-платформа для створення та прототипування користувацьких інтерфейсів. У межах даного проєкту за її допомогою було спроєктовано макети основних сторінок веб-застосунку з урахуванням вимог до юзабіліті та дизайну.

Усі компоненти веб-застосунку тісно пов’язані між собою. Серверна логіка реалізована на Ruby on Rails, де контролери приймають запити від користувача та обробляють їх через моделі, взаємодіючи з базою даних MySQL. Представлення (view) формуються за допомогою HTML-шаблонів і стилізуються з використанням TailwindCSS. JavaScript і Stimulus забезпечують динамічну поведінку сторінок без повного перезавантаження.

Графічна аналітика реалізується через Chartkick або ApexCharts, що інтегруються з Rails та дозволяють будувати діаграми в режимі реального часу. Ruby-геми Devise, Ransack, Simple Form та Kaminari забезпечують авторизацію, пошук, форми і зручну навігацію у списках записів.

Таким чином, обраний стек технологій дозволяє реалізувати ефективний, адаптивний та сучасний веб-застосунок, орієнтований на потреби користувачів і особливості локального фінансового середовища.

1.5 Висновки до першого розділу

У першому розділі було проаналізовано актуальність створення веб-застосунку для управління особистими фінансами, розглянуто основи його архітектури та принципи роботи. Проведено огляд трьох популярних сервісів — Wallet, Spendee та YNAB, визначено їхні переваги та недоліки в контексті потреб українського користувача.

На основі аналізу сформульовано функціональні та нефункціональні вимоги до майбутнього застосунку, а також окреслено його цільову аудиторію. Також обґрунтовано вибір технологій, серед яких Ruby on Rails, MySQL, TailwindCSS та JavaScript (Stimulus), що дозволяють створити гнучкий, безпечний і зручний у використанні веб-продукт.

Отримані результати створюють базу для переходу до наступного етапу — проєктування системи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ЗАСТОСУНКУ

2.1 Архітектура розроблювального застосунку

У процесі проєктування веб-застосунку для управління особистими фінансами було обрано архітектурний підхід Model-View-Controller (MVC), який є стандартом у фреймворку Ruby on Rails. Ця архітектура забезпечує чітке розділення відповідальностей між компонентами системи, що спрощує розробку, тестування та подальший розвиток програмного забезпечення. Загальну схему архітектури зображено на рисунку 2.1.

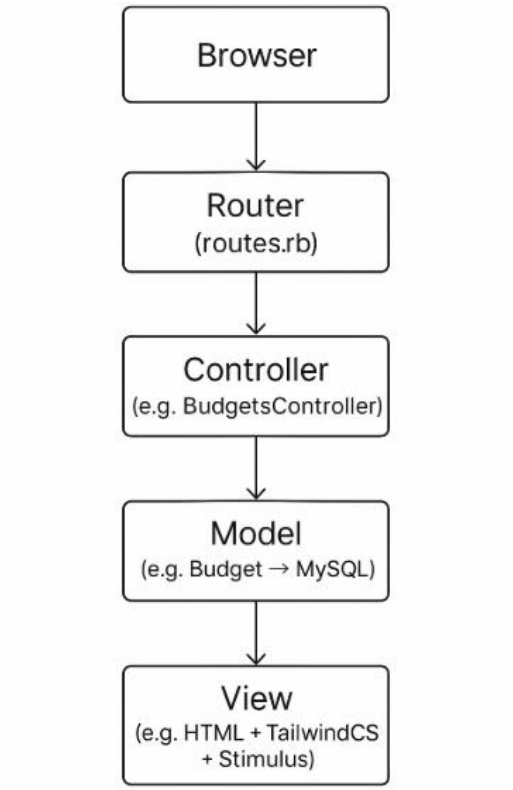

Рисунок 2.1 – Схема архітектури MVC

Компоненти архітектури MVC:

- Model (Модель) — відповідає за взаємодію з базою даних, зберігає бізнес-логіку та оперує даними. У проєкті основними моделями є User, Transaction,

Category та Budget. Вони визначають структуру таблиць, зв’язки між ними (асоціації) та правила валідації даних.

- View (Представлення) — відображає дані, отримані від контролера, у зручному для користувача вигляді. Представлення реалізоване у вигляді HTML-шаблонів з використанням TailwindCSS для стилізації та JavaScript (Stimulus) для динамічної взаємодії.
- Controller (Контролер) — обробляє запити від користувача, взаємодіє з моделлю для отримання або зміни даних та передає їх до відповідного представлення. Наприклад, контролер TransactionsController відповідає за створення, редагування, видалення та фільтрацію транзакцій.

Цей архітектурний підхід дозволяє легко розширювати функціонал, підтримувати логіку на стороні сервера, дотримуватись принципів чистої архітектури та забезпечувати хорошу читабельність коду [11]. Завдяки розділенню логіки, інтерфейсу та зберігання даних, веб-застосунок є гнучким і масштабованим.

2.2 Пошук актантів та варіантів використання

Дуже важливим етап проєктування програмного забезпечення є ідентифікація користувачів (акторів) системи та опис сценаріїв їхньої взаємодії з функціональністю [12]. У межах кваліфікаційної роботи було проведено аналіз майбутніх взаємодій користувачів з веб-застосунком та сформовано перелік актантів і типових варіантів використання (Use Cases), що дало змогу сформувати технічне бачення основних процесів, які система має підтримувати.

Основні актанти розроблювального застосунку:

1. Зареєстрований користувач (User) — основний актант, який має доступ до всіх функцій застосунку: створення транзакцій, перегляд аналітики, встановлення бюджету, налаштування категорій тощо.

2. Гість (Guest) — неавторизований відвідувач, який має обмежений доступ до системи. Може переглядати загальну інформацію, форму реєстрації або входу до облікового запису.

3. Система (System) — внутрішній виконавець дій, який автоматично обробляє запити, валідує введені дані, зберігає їх у базі, будує графіки, обчислює залишки бюджету та сповіщає про перевищення лімітів.

Після визначення актантів було визначено варіанти використання (Use Cases):

– Реєстрація користувача: гість заповнює форму реєстрації, вказуючи email, пароль та ім’я. Система перевіряє дані та створює новий обліковий запис.

– Авторизація користувача: користувач вводить логін і пароль для входу до системи. При успішному вході відкривається доступ до функціоналу.

– Головний екран — після входу користувач переходить на головний екран, на якому відображається дашборд із короткою аналітикою, а також меню з вкладками: корисне, сповіщення, аналітика, керування фінансами та профіль.

– Профіль — користувач може переглянути інформацію про свій обліковий запис, редагувати його, а також вийти з нього.

– Редагування профілю — надається можливість змінити пароль або email.

– Видалення акаунту — користувач може повністю видалити свій обліковий запис із системи.

– Вихід із системи — завершує поточну сесію користувача з та перенаправляє на сторінку входу.

– Транзакції — користувач переглядає список фінансових операцій (доходів та витрат).

– Додавання транзакції — створення нової операції із обов’язковим зазначенням типу (дохід/витрата), категорії, суми, дати та опису.

– Редагування транзакції — можливість внесення змін до вже раніше доданої транзакції.

– Видалення транзакції — видалення окремої транзакції зі списку.

– Пошук транзакції — користувачу доступно здійснення фільтрації або пошук за категорією, сумою, типом або датою.

– Бюджет — перегляд поточного бюджету користувача.

– Створення/редагування бюджету — користувач встановлює або змінює свій бюджет.

– Цілі — розділ для управління фінансовими цілями.

– Створити ціль — користувач задає назву цілі, бажану суму, дату досягнення та опис.

– Активні цілі — перегляд поточних фінансових цілей та прогресу їх виконання.

– Керування фінансами — користувач переходить на сторінку, на якій відразу може, відредагувати бюджет, створити транзакцію та змінити фінансову ціль.

– Перегляд аналітики — система візуалізує статистику витрат, доходів і залишку бюджету у вигляді графіків.

– Сповіщення — користувач отримує повідомлення, якщо витрати перевищують 80 або більше відсотків бюджету.

– Корисне — окремий розділ із порадами та матеріалами для кращого планування бюджету.

На рисунку 2.2 зображено діаграму варіантів використання системи.

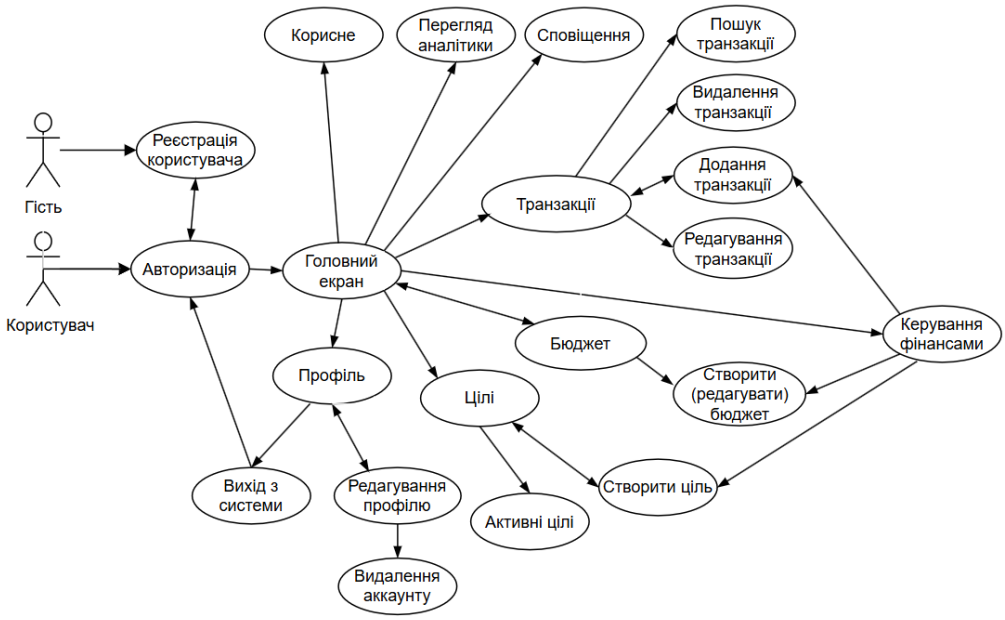

Рисунок 2.2 – Діаграма варіантів використання

Побудова даної діаграми використання дозволила чітко структурувати функціональні вимоги до системи, забезпечити повноту сценаріїв взаємодії з користувачем та слугує важливим етапом у розробці логіки веб-застосунку [20].

2.3 Розробка серверної логіки веб-застосунку

При створенні будь-якого сучасного веб-застосунку дуже важливим етапом є розробка серверної частини. Саме серверна логіка відповідає за збереження, обробку та передачу даних між інтерфейсом користувача та БД, а також реалізує основну логіку застосунку. У даному проєкті для реалізації серверної частини було обрано фреймворк Ruby on Rails, який забезпечує швидку та ефективну розробку.

Фреймворк написаний мовою програмування Ruby та дотримується архітектурного патерну MVC. Rails пропонує велику кількість генераторів коду, інтегровані засоби роботи з базою даних, що дозволяє розробляти повнофункціональні веб-застосунки за короткий час [13]. Більшість налаштувань у Rails встановлюється за замовчуванням, що зменшує кількість ручної конфігурації.

Також у фреймворку існує безліч офіційних і сторонніх бібліотек (гемів), таких як Devise (автентифікація), Chartkick (візуалізація даних), Kaminari (пагінація), Ransack (фільтрація). Rails має вбудовані механізми захисту від XSS, CSRF та SQL-ін’єкцій, що є критично важливими у фінансовому застосунку.

Завдяки цим перевагам Ruby on Rails став оптимальним вибором для розробки особистого фінансового менеджера, де важливі як швидкість реалізації, так і підтримка масштабованості та безпеки.

У таблиці 2.1 продемонстровано контролери, що обслуговують різні модулі:

Таблиця 2.1 – Контролери програмованого застосунку

Контролер	Призначення
TransactionsController	Керування доходами та витратами користувача
BudgetsController	Встановлення та редагування місячного бюджету
GoalsController	Створення та моніторинг фінансових цілей
NotificationsController	Формування й відображення повідомлень
AnalyticsController	Підготовка агрегованих даних для візуалізації
CategoriesController	Створення категорій витрат і доходів
HomeController	Дашборд із графіками та загальною фінансовою інформацією

Лістинг кодів окремих контролерів наведено в Додатку А.

Створення серверної логіки відбувалося поетапно — від базової генерації коду до реалізації повноцінної бізнес-логіки. У рамках фреймворку Ruby on Rails для прискорення розробки активно використовувалися генератори (rails generate), які дозволяють автоматично створювати шаблони контролерів, моделей, представлень, маршрутів тощо.

Для кожного ресурсу спочатку створювався базовий контролер командою, наприклад (див. рис. 2.3).

```
bin/rails generate controller Transactions index new edit
```

Рисунок 2.3 – Команда створення контролера

Ця команда створює файл transactions_controller.rb, відповідні файли представлень (index.html.erb, new.html.erb тощо), а також автоматично додає маршрути в routes.rb, якщо додати resources. Після створення контролера вручну додавалися RESTful-маршрути в config/routes.rb. Це забезпечує підтримку стандартних дій: index, new, create, edit, update, destroy.

Оскільки застосунок призначений для роботи з персональними фінансовими даними, до кожного контролера було додано фільтр before_action :authenticate_user!, який обмежує доступ неавторизованим користувачам. Для зручного створення форм використовувався гем Simple Form, а для виведення помилок — стандартна механіка Rails(див. рис. 2.4).

```
<%= form_with(model: @transaction, local: true) do |form| %>
  <% if @transaction.errors.any? %>
    <div class="text-red-600">
      <%= @transaction.errors.full_messages.to_sentence %>
    </div>
  <% end %>
  ...
<% end %>
```

Рисунок 2.4 – Команда виведення помилок

Це дозволяє відображати користувачу конкретні причини, чому форма не збереглася (наприклад, "Сума не може бути порожньою").

Також деякі контролери мають специфічну поведінку:

1. TransactionsController автоматично оновлює бюджет після додавання витрати.

2. GoalsController відслідковує прогрес і створює сповіщення, якщо ціль досягнута.

3. NotificationsController дозволяє позначати повідомлення як прочитані без перезавантаження сторінки (через remote: true або StimulusJS).

4. AnalyticsController агрегує дані з бази за категоріями або місяцями та передає їх у форматі, зручному для побудови графіків.

У процесі розробки серверної частини реалізовано ключову функціональність веб-застосунку, що охоплює обробку транзакцій, бюджету, фінансових цілей, категорій та сповіщень. Логіка розподілена між окремими контролерами, кожен з яких виконує чітко визначене завдання. Передбачено захист доступу, валідацію введених даних, а також підтримку фільтрації та часткового оновлення вмісту сторінки. Такий підхід дозволив забезпечити стабільну, зручну у використанні та придатну до розширення серверну основу застосунку.

2.4 Проєктування та розробка бази даних

Наступним етапом при створенні веб-застосунку є проєктування структури бази даних , адже саме вона відповідає за зберігання, впорядкування й забезпечення швидкого доступу до фінансових даних користувача — зокрема транзакцій, бюджетів, категорій, цілей та сповіщень [14].

У межах даного проєкту було обрано реляційну систему управління базами даних MySQL [15]. Це стабільне, масштабоване та функціональне рішення, яке підтримує складні запити, транзакції та індексацію. У поєднанні з фреймворком Ruby on Rails, що реалізує архітектуру MVC, MySQL дозволяє організувати ефективну взаємодію між моделями застосунку та відповідними таблицями бази даних.

Структура бази даних розроблена з урахуванням потреб системи управління особистими фінансами. До основних сутностей належать:

- User (Користувач) – основна сутність, що представляє зареєстрованого користувача. Містить облікову інформацію (email, зашифрований пароль) і є пов’язаною з усіма іншими сутностями системи. Саме через користувача реалізовано ізоляцію даних: кожен обліковий запис працює виключно з власною інформацією.
- Transaction (Транзакція) – відображає окрему фінансову операцію. Основні атрибути: сума, дата, тип операції (дохід або витрата), опис та зв’язок з певною категорією. Кожна транзакція належить до конкретного користувача та категорії.
- Category (Категорія) – описує тип доходу або витрати (наприклад, «Їжа», «Зарплата»). Категорії створюються користувачем та використовуються при класифікації транзакцій.
- Budget (Бюджет) – дозволяє встановити загальну фінансову межу на конкретний місяць і рік. Пов’язаний із користувачем і не деталізується по категоріях.
- Goal (Фінансова ціль) – функція для планування накопичення коштів. Містить назву мети, цільову та поточну суму, дедлайн і опис. Прив’язується до користувача, але не напряму до транзакцій.
- Notification (Сповіщення) – система сповіщень для користувача про події, пов’язані з фінансами (наприклад, досягнення мети або перевищення ліміту). Кожне сповіщення має заголовок, текст і статус прочитання.

При створенні моделей у Ruby on Rails використовується механізм міграцій — спеціальні файли, які формують структуру таблиць у базі даних та дозволяють відслідковувати зміни структури схеми з часом. Однією з центральних таблиць проєкту є Transactions, що зберігає дані про фінансові операції користувача. Її структура включає такі основні поля:

- amount — числове значення транзакції, збережене у форматі decimal(10,0), що забезпечує точність обчислень і придатне для фінансових розрахунків;

- transaction_type — тип операції, який може приймати значення "income" або "expense";
- category_id — зовнішній ключ, що встановлює зв’язок з таблицею categories для класифікації транзакцій;
- user_id — зовнішній ключ, який вказує на користувача, якому належить відповідна транзакція;
- date — дата здійснення операції, використовується для хронологічної фільтрації та побудови аналітики;
- description — текстовий опис транзакції, що дозволяє додавати примітки або пояснення;
- created_at та updated_at — службові поля, автоматично створювані Rails, що зберігають дату створення та останнього оновлення запису.

Завдяки такій структурі таблиця transactions підтримує повноцінну обробку фінансових даних: дозволяє ефективно фільтрувати записи за типом, категорією, користувачем або датою, а також виконувати групування й аналіз динаміки доходів і витрат.

У такий самий спосіб, із використанням механізму генерації моделей та міграцій у Ruby on Rails, було створено й інші таблиці бази даних. Кожна з таблиць була сформована окремою міграцією, яка описує перелік атрибутів, типи полів та зовнішні ключі. Для забезпечення цілісності даних у базі реалізовано зв’язки між таблицями (через references і foreign_key), а також створено індекси для пришвидшення пошуку та фільтрації. Загальна структура сформована відповідно до вимог реляційної моделі й відображена у вигляді діаграми на рисунку 2.5.

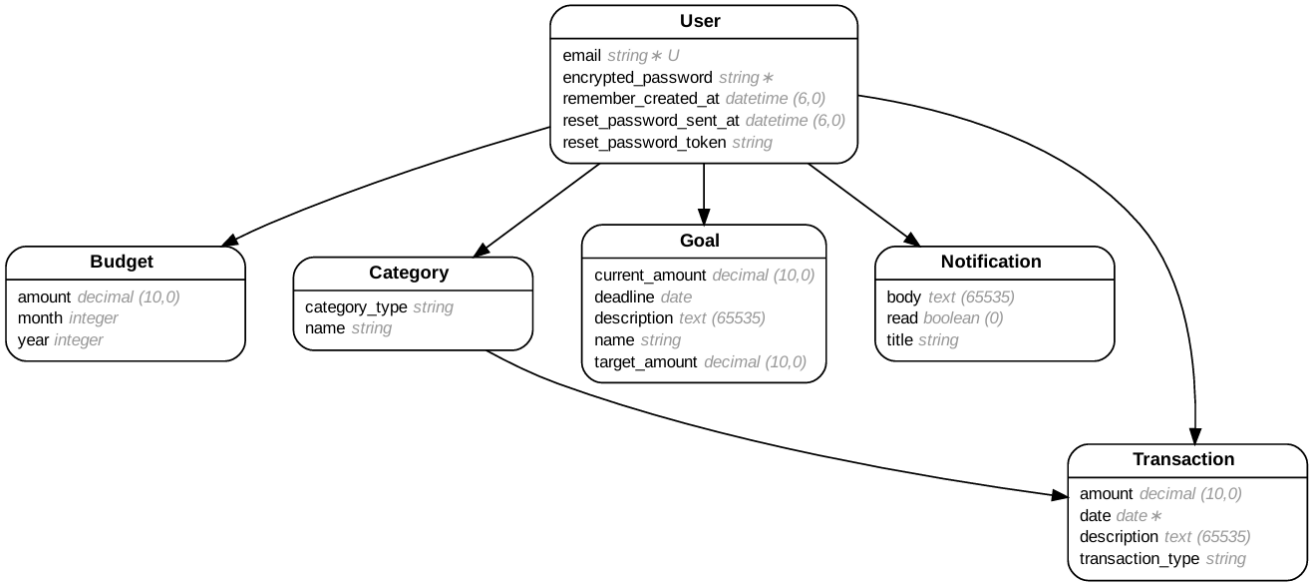

Рисунок 2.5 – ER-діаграма бази даних застосунку для планування фінансів

Основні зв’язки реалізовано наступним чином:

- User — Transaction: один користувач може мати багато транзакцій (1:N).
- User — Category: кожен користувач створює власні категорії (1:N).
- User — Budget: кожен бюджет належить конкретному користувачу (1:N).
- User — Goal: фінансова ціль створюється окремим користувачем (1:N).
- User — Notification: сповіщення надсилаються конкретному користувачу (1:N).
- Category — Transaction: транзакція належить певній категорії, одна категорія може мати багато транзакцій (1:N).

Таким чином, структура бази даних (див. додат. Б) реалізована відповідно до логіки доменної області, є масштабованою та забезпечує надійне зберігання інформації в межах облікового запису. Всі зв’язки реалізовано через зовнішні ключі, що дозволяє підтримувати узгодженість даних на рівні моделі та ефективно реалізовувати запити з урахуванням користувацької ізоляції [16].

2.5 Розробка інтерфейсу користувача

Інтерфейс користувача — компонент веб-застосунку завдяки якому відбувається вся взаємодія користувача з системою. У процесі розробки було поставлено завдання створити простий, інтуїтивно зрозумілий, адаптивний та візуально привабливий інтерфейс, який би відповідав очікуванням користувача особистого фінансового менеджера [17].

Для побудови інтерфейсу використано фреймворк Tailwind CSS, що забезпечує гнучке та ефективне створення сучасної верстки завдяки утилітарному підходу [18]. Стилі можуть задаватися безпосередньо в HTML-шаблонах окремих сторінок або оформлюватися у вигляді окремих CSS-класів, що використовуються повторно. Такий підхід дозволяє з одного боку оперативно адаптувати вигляд елементів під потреби конкретного компонента, а з іншого — підтримувати загальну стилістичну цілісність у межах застосунку. З метою уніфікації зовнішнього вигляду було створено набір власних компонентів, які охоплюють найуживаніші елементи інтерфейсу: картки, заголовки, сповіщення, посилання тощо. Приклад оголошення таких компонентів подано на рисунку 2.6.

```
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer components {
  .card {
    @apply bg-white p-6 rounded-lg shadow-md border border-gray-100;
  }

  .card-title {
    @apply text-2xl font-bold text-gray-800 mb-4;
  }

  .card-subtitle {
    @apply text-sm font-semibold text-gray-400;
  }

  .text-muted {
    @apply text-sm text-gray-500;
  }

  .link {
    @apply text-blue-500 hover:text-blue-700 underline text-sm mt-2 inline-block;
  }

  .header-link {
    @apply hover:text-orange-600 transition;
  }

  .nav-link {
    @apply flex space-x-6 text-sm font-medium text-gray-700;
  }

  .flash-success {
    @apply mb-2 p-3 rounded bg-green-100 text-green-700 text-sm;
  }

  .flash-error {
    @apply mb-2 p-3 rounded bg-red-100 text-red-700 text-sm;
  }
}
```

Рисунок 2.6 – Лістинг коду стилів застосунку

Інтерфейс структурується на кілька основних розділів, кожен з яких відповідає певному типу функціональності. Після входу в систему користувач потрапляє на головну сторінку (дашборд), реалізовану дією index контролера HomeController. На цій сторінці подається зведена інформація про останні транзакції, загальний бюджет, стан фінансових цілей, а також інтерактивні графіки для візуального аналізу фінансової активності.

Окремий розділ призначено для роботи з транзакціями (TransactionsController#index). Тут користувач може переглядати, створювати, редагувати та видаляти записи про фінансові операції. Форма створення розміщена безпосередньо на сторінці перегляду, що сприяє зручності та швидкості роботи.

Функціональність роботи з бюджетом реалізовано через відповідний контролер BudgetsController. Користувач може встановити щомісячний бюджет, який у подальшому буде змінюватись в залежності від транзакцій. Дані бюджету відображаються як у спеціальному розділі, так і безпосередньо на дашборді.

Модуль фінансових цілей (GoalsController) дозволяє створювати цілі накопичення із зазначенням бажаної суми, дедлайну та опису. Прогрес виконання відслідковується системою і відображається у вигляді графічної шкали. Після досягнення цілі генерується відповідне сповіщення, яке подається користувачу у спеціальному розділі.

Розділ аналітики (AnalyticsController#index) містить інтерактивні графіки, що відображають структуру витрат за категоріями та їх зміну в динаміці. Користувач має можливість фільтрувати дані за періодом, що дозволяє здійснювати глибший аналіз фінансової поведінки.

Система сповіщень (NotificationsController#index) інформує користувача про важливі події: досягнення мети, перевищення ліміту бюджету тощо.

Додатково реалізовано сторінку налаштувань профілю (SettingsController#show), де користувач може переглядати й редагувати персональні параметри. Також наявна сторінка довідкової інформації (UsefulController#show), яка містить поради з ефективного планування бюджету.

Інтерфейс повністю адаптивний — всі елементи автоматично підлаштовуються під ширину екрана, що забезпечує комфортну взаємодію як на настільних комп’ютерах, так і на мобільних пристроях. Приклад відображення інтерфейсу на мобільному подано на рисунку 2.7.

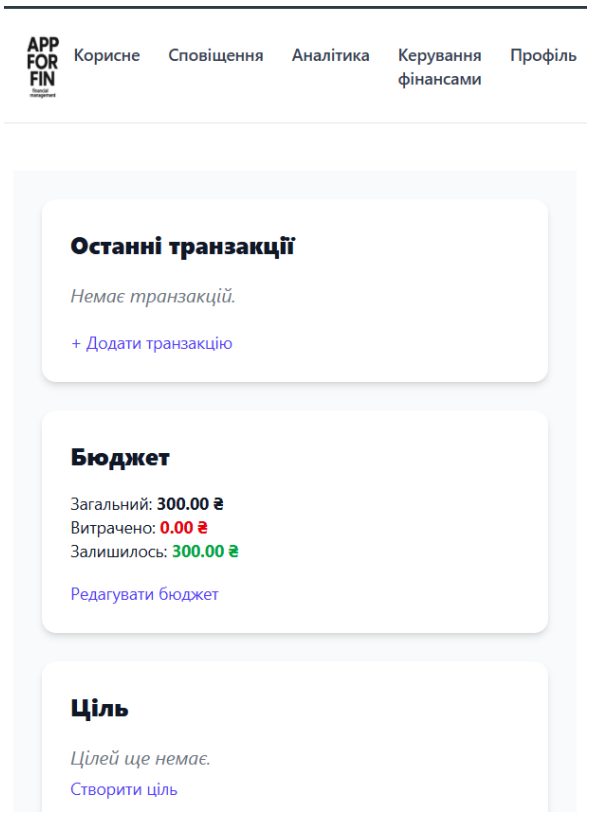

Рисунок 2.7 – Вигляд інтерфейсу на мобільному пристрої

Загалом інтерфейс реалізовано відповідно до принципів мінімалізму, інтуїтивності, консистентності та доступності — використано спокійні кольори, читабельні шрифти та достатні відступи між елементами. Інтерфейс є інтуїтивним — користувач легко орієнтується в діях без додаткових інструкцій. Усі елементи оформлено у єдиному стилі, що забезпечує консистентність, а достатній контраст і зручні поля вводу сприяють доступності та зручності користування на різних типах пристроїв.

2.6 Висновки до другого розділу

У другому розділі було розроблено основні складові веб-застосунку для управління особистими фінансами. Обрано архітектуру Model-View-Controller

(MVC), яка дозволила розділити логіку, інтерфейс і роботу з даними, що значно спростило реалізацію та підтримку системи.

На основі фреймворку Ruby on Rails реалізовано серверну частину, що обробляє запити користувача, взаємодіє з базою даних і забезпечує необхідну функціональність. Створено моделі, контролери та маршрути, які охоплюють ключові процеси — транзакції, бюджет, фінансові цілі, категорії та сповіщення. Усі дії захищено авторизацією, а введені дані проходять перевірку на відповідність.

Окрему увагу приділено розробці бази даних. За допомогою механізму міграцій створено таблиці, які відображають логіку предметної області, а також реалізовано зв’язки між ними.

Для побудови інтерфейсу використано Tailwind CSS, що дозволило створити адаптивний, простий та візуально привабливий дизайн. Користувач має змогу зручно взаємодіяти з функціоналом через дашборд, таблиці, графіки та форми.

У результаті було реалізовано повнофункціональну серверну та клієнтську частину системи, що відповідає поставленим вимогам і готова до подальшого розширення.

РОЗДІЛ 3. ОГЛЯД ФУНКЦІОНАЛЬНОСІ ТА ТЕСТУВАННЯ ПРОЄКТУ

Після завершення етапів проєктування, розробки архітектури, серверної частини та інтерфейсу користувача варто перейти до огляду та тестування застосунку. У цьому розділі розглянуто основні реалізовані можливості системи, а також проведено тестування її роботи з метою перевірки правильності функціонування та відповідності вимогам. Окрема увага приділена опису ключових модулів застосунку, а також методам контролю якості розробленого програмного забезпечення.

3.1 Огляд функціоналу

Після відкриття веб-застосунку користувач потрапляє на сторінку входу (див. рис. 3.1). Якщо користувач уже має обліковий запис, він вводить електронну пошту та пароль для авторизації.

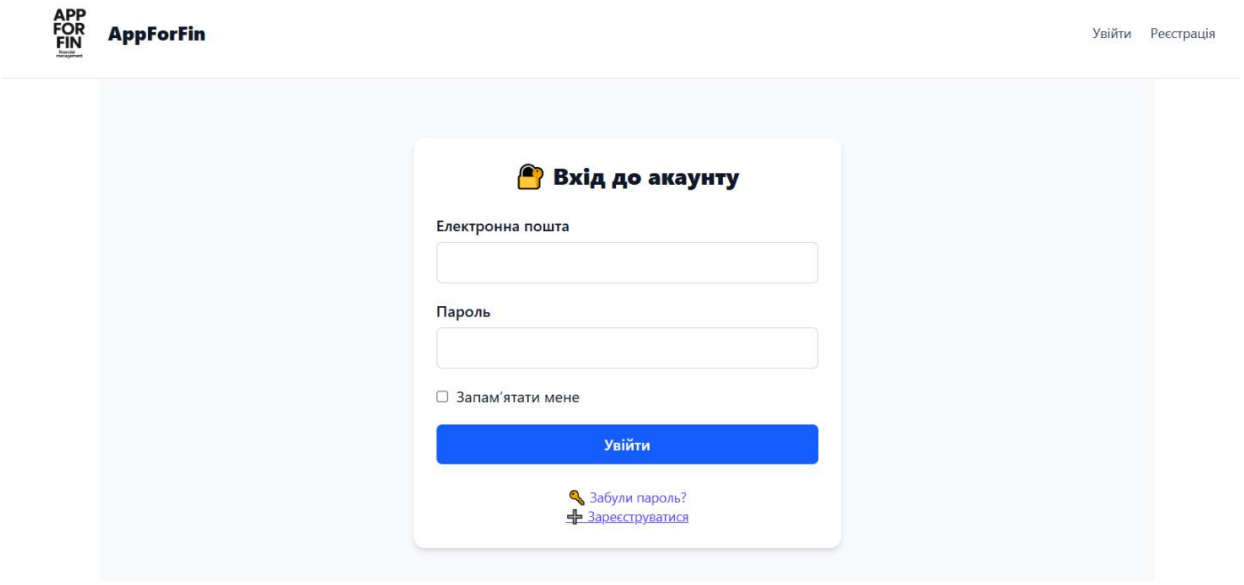

Рисунок 3.1 – Вигляд форми авторизації

У разі відсутності облікового запису необхідно пройти реєстрацію, заповнивши стандартну форму із вказанням імені, адреси електронної пошти та бажаного пароля. Усі дані перевіряються сервером на валідність, зокрема перевіряється унікальність електронної пошти, довжина пароля та правильність заповнення полів. Форма реєстрації(див. рис. 3.2) у веб-застосунку була реалізована за допомогою бібліотеки Devise — популярного Ruby on Rails-гему для реалізації автентифікації користувачів. Основою служить хелпер form_for, який генерує HTML-форму, прив’язану до моделі User (ресурсу Devise).

```
<%= form_for(resource, as: resource_name, url:
registration_path(resource_name)) do |f| %>
  <%= render "devise/shared/error_messages", resource: resource %>
  <div>
    <%= f.label :email, "Електронна пошта" %>
    <%= f.email_field :email, autofocus: true, autocomplete: "email" %>
  </div>
  <div>
    <%= f.label :password, "Пароль" %>
    <%= f.password_field :password, autocomplete: "new-password" %>
    <% if @minimum_password_length %>
      <p>Мінімум <%= @minimum_password_length %> символів</p>
    <% end %>
  </div>
  <div>
    <%= f.label :password_confirmation, "Підтвердження пароля" %>
    <%= f.password_field :password_confirmation, autocomplete: "new-password"
%>
  </div>
  <div>
    <%= f.submit "Зареєструватися" %>
  </div>
<% end %>

<div>
  <%= render "devise/shared/links" %>
</div>
```

Рисунок 3.2 – Лістинг коду форми реєстрації

Звичайно ж реєстрація обов’язкова для продовження роботи в застосунку. Після успішної реєстрації користувач автоматично перенаправляється на головну сторінку — дашборд. Цей екран слугує основним інформаційним центром, де відображаються останні транзакції, поточний стан бюджету, прогрес виконання фінансових цілей, а також інтерактивні графіки, які показують динаміку бюджету та прогрес виконання фінансової цілі(див. рис. 3.3). Враховуючи що користувач тільки що здійснив реєстрацію, поки що йому не відображається інформація з «колонок» та графіків, оскільки він не надав ніякої інформації.

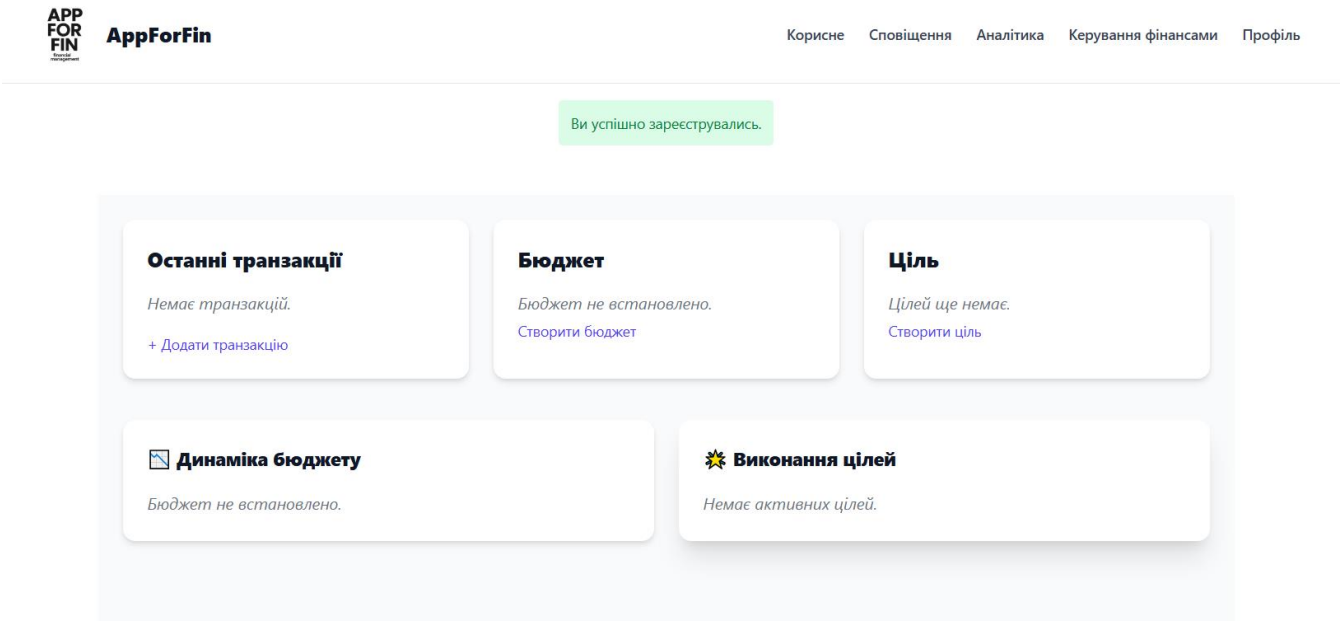

Рисунок 3.3 – Головна сторінка користувача

В верхній частині сторінки користувачу доступне меню застосунку, а саме:

- Корисне.
- Сповіщення.
- Аналітика.
- Керування фінансами.
- Профіль.

Розпочати користування застосунком слід зі створення бюджету, для цього користувачу потрібно натиснути на посилання «Створити бюджет». Після переходу на сторінку потрібно вказати бюджет(місячний дохід) в гривнях на

поточний місяць, який відображається атоматично. Приклад формування бюджету та вигляд сторінки зображено на рисунку 3.4.

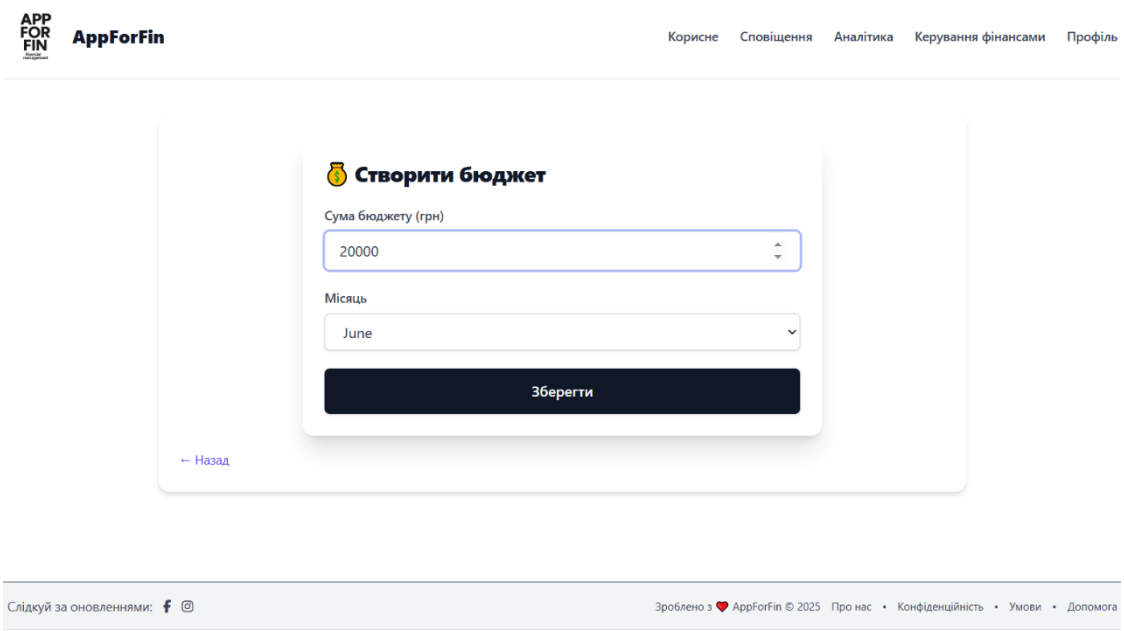

Рисунок 3.4 – Сторінка створення бюджету

Після збереження бюджету клієнта поверне на головну сторінку, де він може продовжити редагувати(створювати) доступні компоненти системи. Перейшовши на вкладку «Створити ціль» користувач може створити фінансову мету, вказавши бажану суму накопичення, строк досягнення та опис(див. рис. 3.5), в правій «колонці» будуть відображатись усі активні цілі з можливістю видалення на редагування.

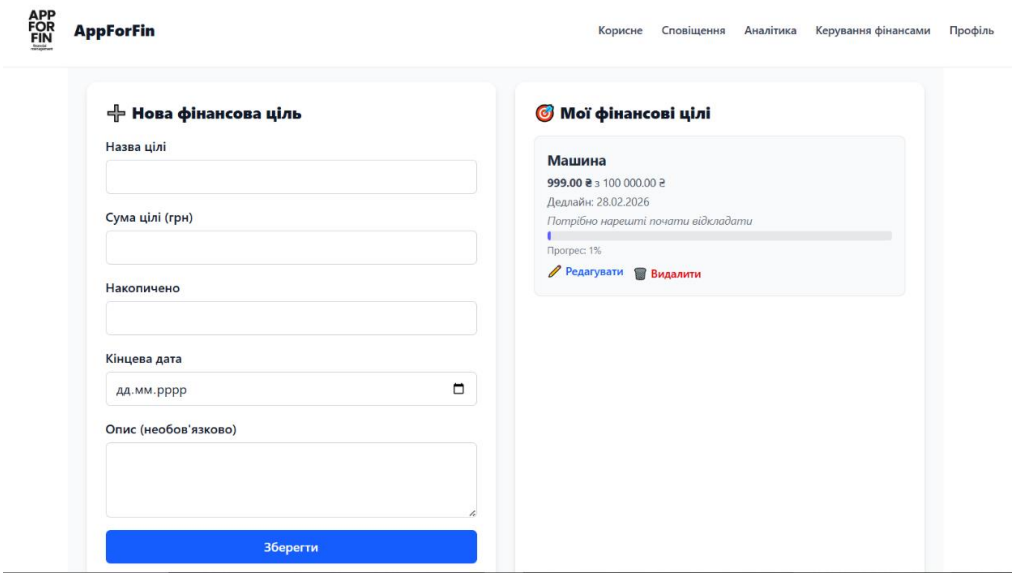

Рисунок 3.5 – Сторінка створення фінансової цілі

Застосунок відслідковує прогрес накопичення й відображає його у вигляді візуальної шкали. При досягненні мети користувач отримує повідомлення про успішне завершення цілі.

З головної сторінки користувач може перейти до вкладки «Додати транзакцію», де відкривається сторінка зі списком усіх фінансових операцій користувача. У лівій частині розміщена форма для створення нової транзакції, де потрібно зазначити тип операції (дохід або витрата), категорію, суму, дату та опис. У веб-застосунку реалізовано динамічну зміну списку категорій залежно від вибраного типу транзакції (дохід або витрата). Це зроблено за допомогою StimulusJS — невеликої JavaScript-бібліотеки, що інтегрується з HTML-розміткою через data-атрибути. На рисунку 3.6 зображено лістинг коду Stimulus-контролеру.

```
import { Controller } from "@hotwired/stimulus"


export default class extends Controller {
  static targets = ["categorySelect", "transactionType"]
  connect() {
    this.loadCategories()
  }
  loadCategories() {
    const type = this.transactionTypeTarget.value
    const options =
JSON.parse(this.element.getAttribute("data-transaction-form-categories-valu
e") || "[]")
    const filtered = options.filter(cat => cat.category_type === type)
    this.categorySelectTarget.innerHTML = ""
    filtered.forEach(cat => {
      const option = document.createElement("option")
      option.value = cat.id
      option.textContent = cat.name
      this.categorySelectTarget.appendChild(option)
    })
  }
}
```

Рисунок 3.6 – Лістинг коду динамічної зміни категорій

В праві частині вкладки «Транзакцій» — таблиця зі всіма записами, які можна фільтрувати за типом, категорією або діапазоном дат. Для кожної транзакції реалізовано можливість редагування або видалення. Приклад додання транзакції та відображення її в таблиці продемонстровано на рисунку 3.7.

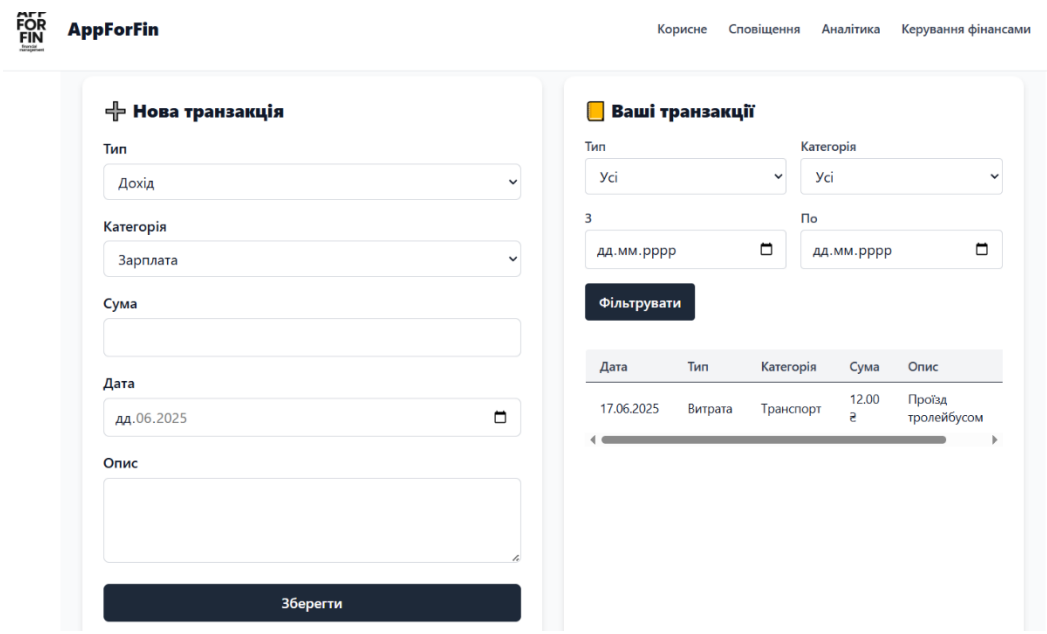

Рисунок 3.7 – Створення та відображення транзакції

Після огляду ключових функцій та їх налаштування, клієнт може скористатись верхнім меню. Вкладка «Корисне» в верхньому меню містить поради щодо ефективного управління фінансами(див. рис. 3.8).

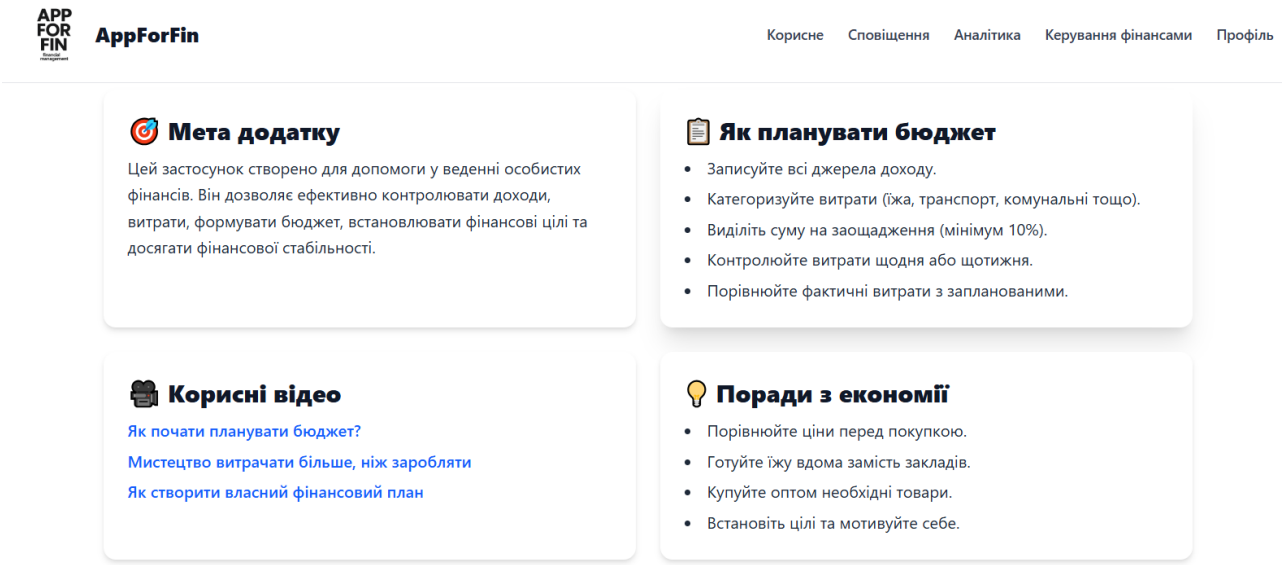

Рисунок 3.8 – Вигляд вкладки «Корисне»

У вкладці «Сповіщення» користувачу будуть відображатись повідомлення про використання певного відсотку бюджету. На сторінці реалізовано функціональність, яка дозволяє користувачеві миттєво видалити або позначити всі

сповіщення як прочитані без необхідності перезавантаження сторінки. Такий інтерактивний механізм досягається за допомогою технології Hotwire, а саме її складової — Turbo Stream. Turbo Stream — це інструмент, який дозволяє динамічно оновлювати частину HTML-сторінки, реагуючи на дії користувача або відповіді сервера. Приклад надходжень сповіщень при використанні бюджету зображено на рисунку 3.9.

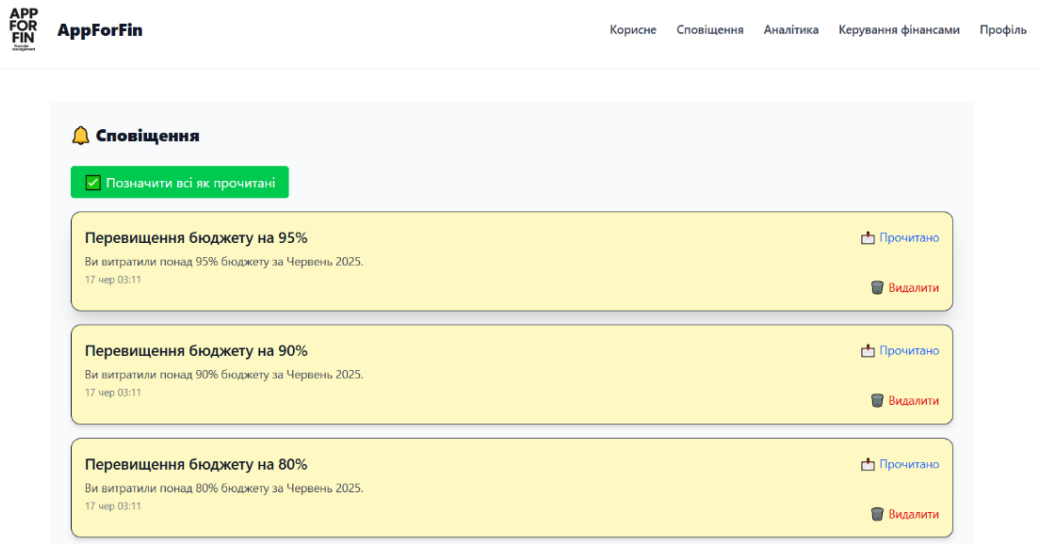

Рисунок 3.9 – Приклад надходження сповіщень

Розділ «Аналітика» забезпечує наочне представлення фінансових даних користувача у вигляді графіків. Для побудови візуалізацій використано гем Chartkick, який у поєднанні з бібліотекою Chart.js дозволяє легко створювати діаграми без потреби в написанні JavaScript-коду.

На рисунку 3.10 зображено лістинг коду форми, яка дає змогу фільтрувати дані за місяцем:

```
<%= form_with url: analytics_path, method: :get, local: true do %>
  <%= month_field_tag :month, @selected_month %>
  <%= submit_tag "Показати" %>
<% end %>
```

Рисунок 3.10 – Лістинг коду форми фільтрування даних

На основі обраного періоду на сервері обчислюються агреговані показники (доходи, витрати, бюджет, цілі), які передаються до відповідних графіків. Наприклад, розподіл витрат по категоріях зображено на рисунку 3.11.

```
<%= pie_chart @expenses_by_category, suffix: " ₴", height: "250px" %>
```

Рисунок 3.11 – Рядок коду подачі витрат по категоріях

Вигляд сторінки «Аналітика» та графіків зображено на рисунку 3.12

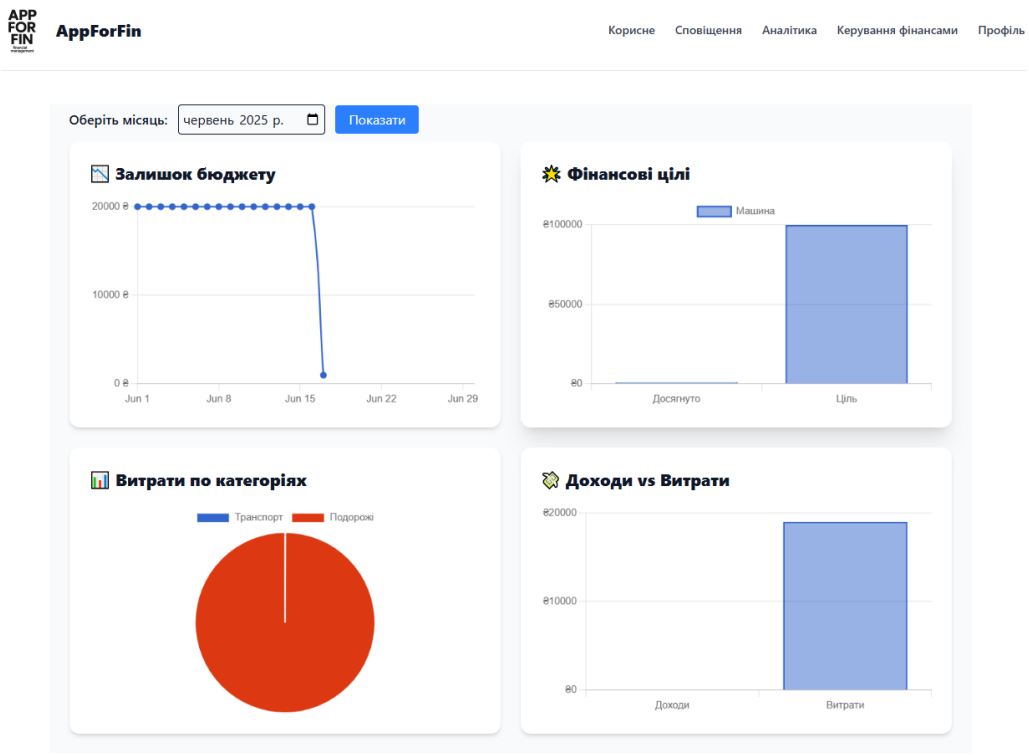

Рисунок 3.12 – Вигляд вкладки «Аналітика»

Перейшовши на сторінку «Керування фінансами» користувачу будуть доступні: створення транзакцій, налаштування бюджету, створення фінансових цілей(все те що на головному дашборді) на одній сторінці(див. рис. 3.13).

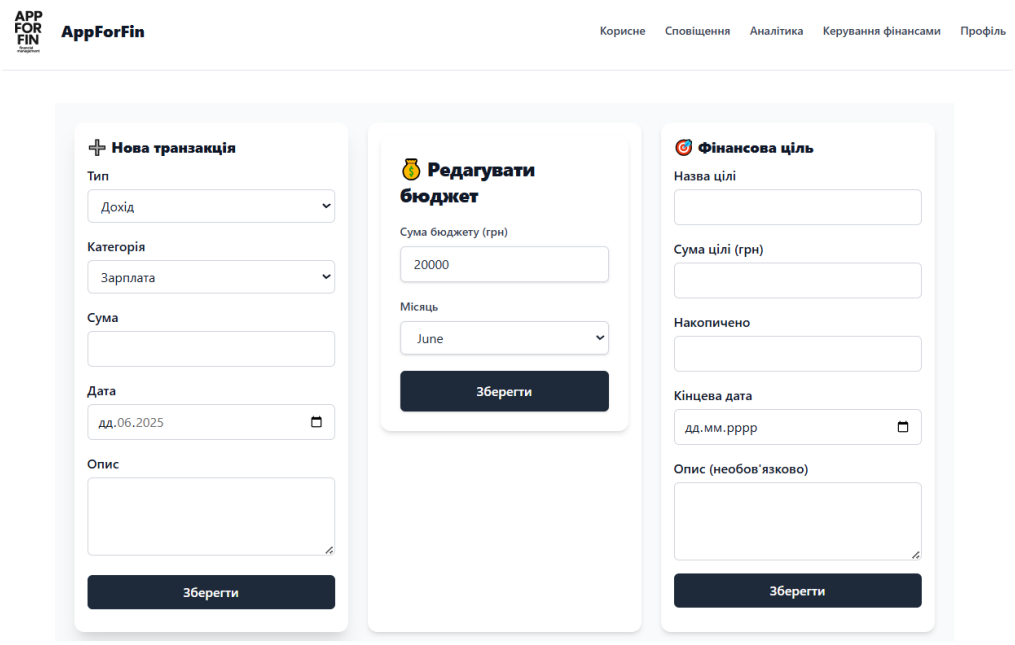

Рисунок 3.13 – Вигляд вкладки «Керування фінансами»

На сторінці «Профіль» клієнт може змінити свої особисті дані(електронну пошту або пароль), повністю видалити обліковий запис, а також завершити поточну сесію натиснувши «Вийти з аккаунту». Вигляд сторінки продемонстровано на рисунку 3.14.

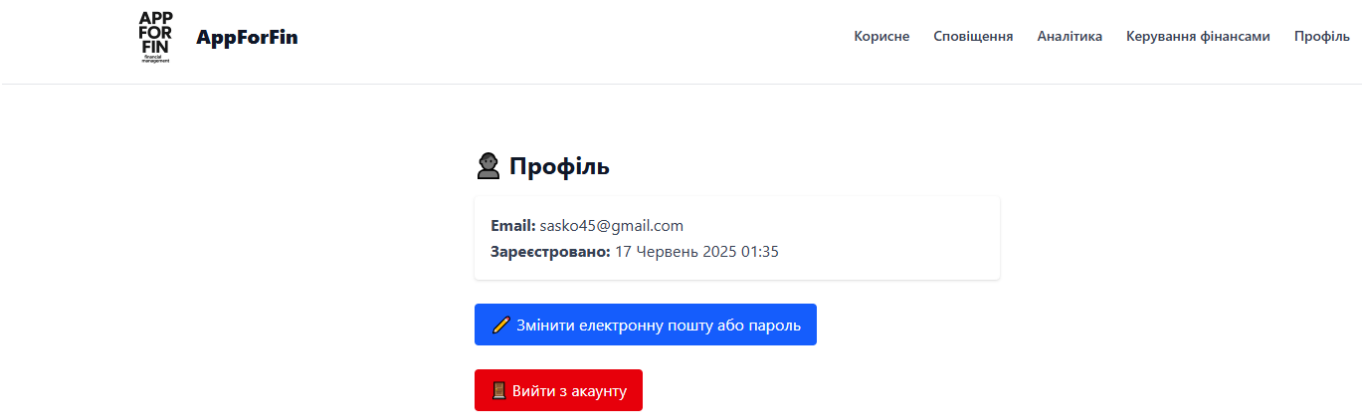

Рисунок 3.14 – Сторінка редагування даних

Розглянувши усі основні функціональні можливості вебзастосунку, та використавши їх, головна сторінка буде мати вигляд зображений на рисунку 3.15.

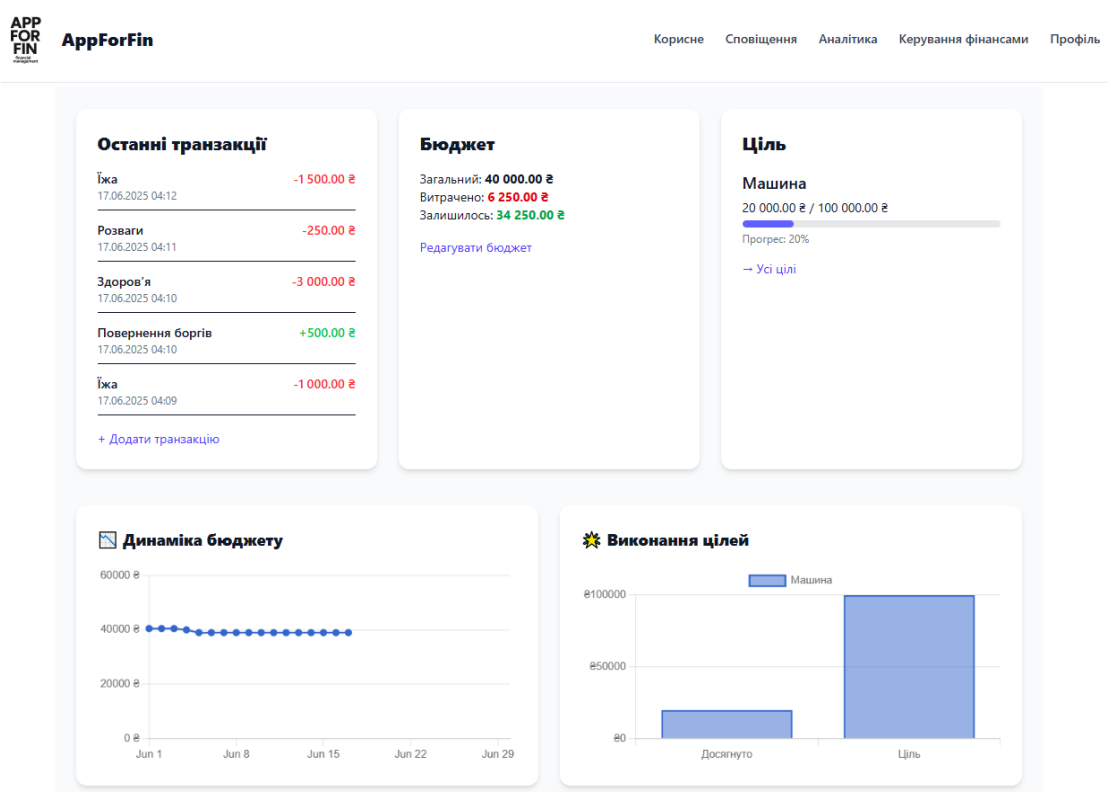

Рисунок 3.15 – Вигляд застосунку при активному користуванні

Таким чином, створені функціональні компоненти веб-застосунку підтверджують реалізацію заявлених можливостей.

3.2 Тестування застосунку

Після завершення етапів розробки веб-застосунку та огляду функціональності було проведено ручне тестування з метою перевірки правильності реалізації основного функціоналу, виявлення можливих помилок, а також оцінки зручності користувацького інтерфейсу [19].

Тестування здійснювалося без використання спеціалізованих інструментів автоматизації, шляхом проходження типових сценаріїв взаємодії з веб-застосунком через браузер [21]. Кожен із функціональних модулів був перевірений на відповідність вимогам, зазначеним у технічному завданні. Результати тестування наведено в таблиці 3.1.

Таблиця 3.1 — Результати ручного тестування веб-застосунку

№	Функція	Очікувана поведінка	Фактичний результат	Статус	Коментар
1	Реєстрація	Користувач створює обліковий запис	Обліковий запис створено успішно	Успішно	
2	Авторизація	Вхід із коректними даними	Вхід виконано	Успішно	
3	Створення бюджету	Зберігається бюджет для поточного місяця	Бюджет збережено	Успішно	
4	Додавання транзакції	Зберігається нова транзакція	Транзакція додана	Успішно	
5	Додавання фінансової цілі	Зберігається ціль з параметрами	Ціль збережено	Успішно	
6	Аналітика	Графіки оновлюються після фільтрації	Графіки оновлено	Успішно	
7	Видалення профілю	Акаунт користувача видаляється разом із даними	Помилка `Unknown column ‘categories.user_id’`	Виправлено	Було оновлено залежності в моделях
8	Вибір місяця для аналітики	Графіки відображають дані за обраний місяць	При виборі місяця без даних або майбутнього — помилка `Date::Error`	Виправлено	Додано перевірку валідності дати

Під час тестування було виявлено кілька помилок:

– Помилка при видаленні профілю: виникало виключення Mysql2::Error: Unknown column ‘categories.user_id’. Проблема полягала в неправильних залежностях між моделями. Було оновлено асоціації та очищено зайві залежності.

– Помилка при виборі місяця в аналітиці: при виборі майбутнього місяця або місяця без даних система виводила помилку Date::Error. Додано перевірку, яка обмежує вибір майбутніх дат і виводить повідомлення, якщо немає статистики.

– Невірна змінна у формі: у формі створення цілі використовувалась змінна goal, яка не була визначена. Замість неї використано інстанс-змінну @goal.

Ці ситуації підтверджують важливість ручного тестування навіть для проєктів з відносно невеликим обсягом функціоналу. Усі ці виявлені проблеми були проаналізовані та усунуті.

3.3 Висновки до третього розділу

У третьому розділі було здійснено огляд реалізованої функціональності веб-застосунку та проведено ручне тестування його основних модулів. Детально описано послідовність взаємодії користувача із системою — від авторизації до аналітичного перегляду фінансових даних. Результати тестування підтвердили працездатність заявленого функціоналу та виявили окремі недоліки, які були усунуті в процесі вдосконалення системи. Проведене тестування дозволило переконатись у відповідності застосунку функціональним вимогам, а також у стабільності та коректності його роботи в типовому сценарії використання.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Стихійні лиха та їх класифікація

Згідно пункту 42, частини першої, статті 2 Кодексу цивільного захисту України, в редакції Закону № 3441-IX від 08.11.2023, стихійне лихо – природне явище, що діє з великою руйнівною силою, заподіює значну шкоду території, на якій відбувається, порушує нормальну життєдіяльність населення, завдає матеріальних збитків [22].

Залежно від причин виникнення, їх поділяють на:

- Тектонічні (літосферні) – зумовлені процесами в надрах Землі, наприклад, землетруси та вулканічна діяльність;
- Топологічні (гідросферні) – пов’язані з процесами на земній поверхні, як-от повені, зсуви ґрунту, селі;
- Метеорологічні (атмосферні) - спричинені процесами в атмосфері, зокрема аномальна спека, ураганні вітри, посухи тощо.

Тектонічні лиха, такі як землетруси та виверження вулканів, є одними з найруйнівніших через величезну енергію, що вивільняється.

Виверження вулканів належать до найнебезпечніших стихійних лих за масштабами руйнування та енергією. Лавові потоки можуть поширюватися до 100 км, а зона впливу кислотних дощів — до 500 км. На планеті близько 600 активних вулканів, і зменшити шкоду від їх виверженьповністю поки не вдається. Використовуються захисні заходи, як-от охолодження лави водою чи відведення потоків штучними каналами [23].

Землетруси – це коливання земної кори, зумовлені тектонічними процесами. Щорічно фіксується близько мільйона поштовхів, з яких близько 1000 є руйнівними. Їх сила вимірюється за шкалою Ріхтера, і найнебезпечніші землетруси супроводжуються серією ударів різної тривалості. В Україні сейсмічно активними залишаються Карпати й Крим.

Гідросферні (топологічні) стихійні лиха охоплюють повені, цунамі, снігові лавини та селеві потоки.

Повені виникають через надмірні опади, танення снігу чи прориви дамб. Вони завдають значної шкоди інфраструктурі, сільському господарству та можуть спричинити інфекційні спалахи. В Україні часто трапляються на великих річках (Дніпро, Дністер, Тиса).

Цунамі — це потужні хвилі, викликані підводними землетрусами, що несуть масштабні руйнування.

Снігові лавини трапляються в Карпатах і Криму. Вони мають величезну руйнівну силу, особливо через повітряний удар.

Селі — бурхливі потоки води з камінням і уламками — несуть загрозу життю і будівлям. Виникають у горах (Карпати, Крим) через зливи, танення снігу чи обвали. Дуже руйнівні, передбачити їх важко [24].

Метеорологічні стихійні явища. Метеорологічні лиха пов’язані з атмосферними процесами та включають урагани, смерчі, посухи та пожежі.

Урагани (швидкість вітру понад 32 м/с) і смерчі завдають значних руйнувань. В Україні смерчі частіше спостерігаються на Чорному та Азовському морях. Прогнозування ураганів здійснюється за допомогою супутників і метеорологічних літаків.

Лісові та степові пожежі знищують екосистеми та майно. Згідно з дослідженнями, в Україні в середньому на рік виникає близько 3,5 тисячі лісових пожеж, які знищують понад 5 тисяч гектарів лісу. Щодо причин виникнення лісових пожеж, то за статистичними даними, 96–98% з них спричинені діяльністю людини. Гасіння пожеж передбачає використання протипожежних смуг, води та зустрічного вогню [25].

Також, слід зазначити, що війна в Україні має значний вплив на виникнення та загострення стихійних лих, спричиняючи як безпосередні екологічні наслідки, так і опосередковані ефекти через руйнування інфраструктури та порушення природного балансу.

Військові дії спричиняють значні руйнування природних екосистем. Наприклад, підрив Каховської ГЕС у червні 2023 року призвів до масштабної повені, затоплення населених пунктів, загибелі людей та знищення сільськогосподарських угідь. Це не лише безпосередньо вплинуло на екосистему, але й створило передумови для подальших стихійних лих, таких як ерозія ґрунтів та зсуви.

Згідно з Методичними рекомендаціями ДСНС України, громадяни повинні бути обізнані з алгоритмом дій під час стихійних лих [26]. Насамперед, слід уважно стежити за офіційною інформацією (повідомленнями ДСНС, місцевих органів влади), підготувати «тривожну валізу» з найнеобхіднішими речами, заздалегідь визначити безпечні маршрути евакуації. У разі загрози необхідно зберігати спокій, не панікувати, допомагати іншим, особливо дітям, літнім людям і людям з інвалідністю. Дотримання офіційних інструкцій та своєчасне реагування є запорукою збереження життя і здоров’я у надзвичайних ситуаціях.

4.2 Розрахунок блискавкозахисту

Блискавкозахист — це система технічних засобів, яка призначена для захисту будівель, споруд, обладнання та людей від небезпечних впливів блискавки. Ця система забезпечує перехоплення розряду та безпечне відведення струму блискавки в землю. Згідно з Наказом МВС № 1417 від 30.12.2014 (Правила пожежної безпеки в Україні), проєктування блискавкозахисту є обов’язковим етапом проєктування будівель [27].

Таким чином блискавкозахист необхідний при проектуванні: промислових об’єктів, енергетичних підстанцій, житлових і громадських будівель, об’єктів критичної інфраструктури.

Відповідно до стандарту ДСТУ EN 62305-3:2021 (Державний стандарт України. Частина 3. Фізичні пошкодження будівель та небезпека для життя), існує

чотири рівні блискавкозахисту (LPL – *Lightning Protection Level*) [28], які визначають ефективність та надійність системи захисту (див. таб. 4.1):

Таблиця 4.1 – Рівні захисту

Рівень	Ймовірність перехоплення блискавки	Типові сфери застосування
LPL I	99% (найвища ефективність)	Вибухонебезпечні об’єкти, спецпромисловість
LPL II	97%	Промислові підприємства, великі адміністративні будівлі
LPL III	91%	Багатоповерхові житлові будинки, школи, ТЦ
LPL IV	84% (мінімальний захист)	Будівлі невеликого об’єму, низький ризик

Блискавковідводи поділяються на окремо стоячі, які встановлюються окремо від будівлі для захисту великих об’єктів, таких як промислові споруди чи склади, і включають щогли чи тросові системи, та інтегровані в будівлю, що використовують її елементи, наприклад, металеві конструкції даху, для житлових, адміністративних чи промислових об’єктів.

Конструкція блискавковідводу складається з блискавкоприймача, струмовідводу та заземлювальної системи.

Зона захисту блискавковідводу – це простір навколо блискавковідводу, в межах якого забезпечується захист будівель, споруд або об’єктів від прямого влучання блискавки з певною ймовірністю, визначеною нормативними документами. Ця зона формується блискавкоприймачем (стрижневим, тросовим або сітчастим) і залежить від його конструкції, висоти, типу та рівня захисту (LPL I–IV) [28].

Зона захисту визначається за допомогою кількох методів, найпоширеніші з яких – метод «захисного кута», метод «котячої кулі» та метод «сітки». Зона захисту розраховується з урахуванням висоти блискавкоприймача, типу ґрунту, кліматичних умов і категорії об’єкта. Проектування цієї зони виконується

фахівцями для забезпечення безпеки об’єктів від прямого влучання блискавки та її вторинних ефектів.

Для прикладу розглянемо розрахунок блискавкозахисту ділянку цеху з наступними характеристиками:

- Габарити будівлі: довжина — 40 м, ширина — 20 м, висота — 8 м (див. рис. 4.1);

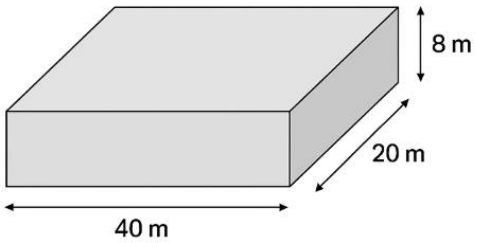

Рисунок 4.1 – Параметри ділянки цеху

- Тип даху: плоский;
- Рівень захисту: LPL II (згідно з ДСТУ EN 62305-3:2021).

Для визначення зони захисту оберемо метод захисного кута (див. Рис. 4.2):

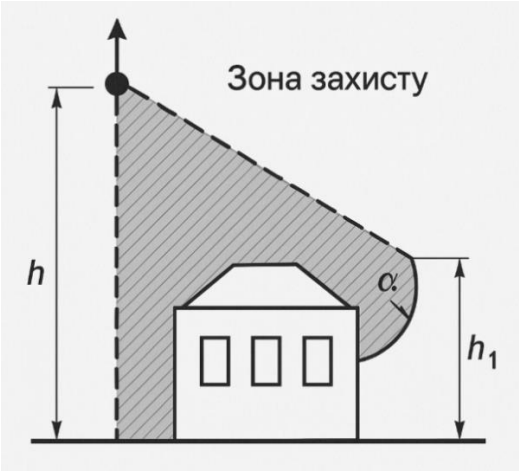

Рисунок 4.2 – Метод захисного кута

При використанні методу захисного кута зона захисту має форму конуса, вершина якого розташована на блискавкоприймачі, а кут при вершині (захисний кут α) залежить від рівня захисту та висоти блискавковідводу. Об’єкти всередині цього конуса вважаються захищеними.

Відповідно до стандарту, для рівня LPL II та висоти блискавкоприймача до 20 м, кут захисту становить 45°.

Для розрахунку радіуса зони захисту одного блискавкоприймача використаємо формулу для визначення горизонтального радіуса захищеної зони [28]:

$$R = H \times \tan(\alpha) \quad (1)$$

де R — радіус зони захисту, м; H — висота блискавкоприймача над захищеною поверхнею, м; α— кут захисту, °.

Припустимо, що висота блискавкоприймача над дахом становить 6 м, тоді:

$$R = 6 \times \tan(45°) = 6 \times 1 = 6 \text{ м} \quad (2)$$

Отже, зона захисту одного блискавкоприймача охоплює площу радіусом 6 м.

Для забезпечення повного покриття даху площею 40 × 20 м, необхідно:

- Встановити по периметру будівлі щонайменше 8 стрижневих блискавкоприймачів висотою 6 м;
- Розмістити блискавкоприймачі з кроком не більше 12 м для надійного перекриття зон;
- За потреби — використовувати тросову систему між приймачами для посилення захисту.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено повноцінний цикл розробки веб-застосунку для планування особистих фінансів. В межах першого розділу проведено аналіз наявних систем фінансового обліку, окреслено їх переваги та недоліки, сформульовано вимоги до майбутньої системи та обґрунтовано вибір архітектурного підходу і технологічного стеку.

У другому розділі описано процес проєктування та реалізації веб-застосунку. Побудовано модель предметної області, розроблено структуру бази даних, реалізовано серверну логіку із застосуванням фреймворку Ruby on Rails, а також створено адаптивний інтерфейс користувача із використанням HTML5, TailwindCSS та JavaScript. Особливу увагу приділено модулю авторизації, управлінню транзакціями, бюджетами та фінансовими цілями, а також побудові візуальної аналітики на основі введених користувачем даних.

У третьому розділі проведено ручне тестування системи, в ході якого перевірено коректність роботи основних функцій. За результатами тестування було виявлено та усунуто низку помилок, що свідчить про ефективність обраного підходу до перевірки працездатності програмного забезпечення.

У четвертому розділі проаналізовано основи безпеки життєдіяльності та охорони праці, зокрема класифіковано стихійні лиха, та здійснено розрахунок блискавкозахисту умовного об’єкта, що є актуальним при проєктуванні безпечних ІТ-систем і офісних приміщень.

У підсумку, розроблений веб-застосунок задовольняє поставлені функціональні вимоги, є зручним у користуванні та має перспективу подальшого розвитку шляхом додавання нових функцій, покращення інтерфейсу та розширення аналітичних можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Національний Банк України. Рівень фінансової грамотності українців - результати дослідження. Національний Банк України. Новини. 26.11.2021 [Електронний ресурс]. Режим доступу до ресурсу: https://bank.gov.ua/ua/news/all/za-ostanni-tri-roki-riven-finansovoyi-gramotnist-ukrayintsiv--polipshivsya--rezultati-doslidjennya?

2. Веб-додаток: що це таке, типи, переваги, принцип роботи. MegaSite. Блог/Web. 16.12.2023. [Електронний ресурс]. Режим доступу до ресурсу: https://megasite.ua/ua/veb-prilogenie-chto-eto-takoe-tipi-preimushchestva-printsip-raboti

3. Розуміння сучасних архітектур веб-додатків. Ranktracker. Веб-розробка та архітектура. 14.06.2024. [Електронний ресурс]. Режим доступу до ресурсу: https://www.ranktracker.com/uk/blog/understanding-modern-web-app-architectures/?.

4. Топ-9 найкращих додатків, які допоможуть контролювати ваші гроші. РБК-Україна. Бізнес. 22.04.2024. [Електронний ресурс]. Режим доступу до ресурсу: https://www.rbc.ua/rus/styler/top-9-naykrashchih-dodatkiv-ki-dopomozhut-1713794027.html

5. Топ найкращих застосунків для особистих фінансів та обліку грошей. Speka. 29.02.2024. [Електронний ресурс]. Режим доступу до ресурсу:https://speka.media/top-naikrashhix-zastosunkiv-dlya-osobistix-finansiv-ta-obliku-grosei-982k86

6. Budgetbakers. Wallet. [Електронний ресурс]. Режим доступу до ресурсу: https://web.budgetbakers.com/dashboard

7. Spendee. Spendee Money Manager & Budget Planner. [Електронний ресурс]. Режим доступу до ресурсу: https://www.spendee.com/

8. Ynab. YNAB. [Електронний ресурс]. Режим доступу до ресурсу: https://www.ynab.com/

9. Як визначити цільову аудиторію за 3 кроки. Remonline. Реклама та маркетинг, клієнти. 30.05.2024. [Електронний ресурс]. Режим доступу до ресурсу: https://remonline.ua/blog/target-audience/

10. 10 порад щодо вибору найкращого технологічного стеку для розробки веб-додатків. SECL Group. Блог. 09.04.2025. [Електронний ресурс]. Режим доступу до ресурсу: https://secl.com.ua/tips-to-choose-tech-stack-for-web-app-development/

11. Побудова динамічних вебзастосунків за допомогою MVC. Hillel. Статті. 14.11.2023. [Електронний ресурс]. Режим доступу до ресурсу: https://blog.ithillel.ua/articles/building-dynamic-web-applications-using-mvc

12. Петрик М. Р. Моделювання програмного забезпечення : науково-методичний посібник / ред. О. Ю. Петрик. Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. 200 с.

13. Ruby on Rails для початківців із проектом і прикладом. Guru99. 26.09.2024. [Електронний ресурс]. Режим доступу до ресурсу: https://www.guru99.com/uk/ruby-on-rails-tutorial.html

14. Гордєєв А. С. Проєктування баз даних та баз знань. Харків : ХНЕУ ім. С. Кузнеця, 2022. 180 с. [Електронний ресурс]. Режим доступу до ресурсу: https://files.znu.edu.ua/files/Bibliobooks/Inshi81/0061625.pdf

15. Система управління базами даних MySQL. Lemon school. 29.04.2024. [Електронний ресурс]. Режим доступу до ресурсу: https://lemon.school/blog/systema-upravlinnya-bazamy-danyh-mysql

16. Права та привілеї користувачів СУБД MySQL. Wiki,Вікіпедія серверів та хостингу. [Електронний ресурс]. Режим доступу до ресурсу: https://vps.ua/wiki/ukr/mysql-privileges/

17. Липова О. М., Люта М. В. Особливості розробки інтерфейсу веб-додатків. Сучасні електромеханічні та інформаційні системи. 2021. С. 112–117. [Електронний ресурс]. Режим доступу до ресурсу: https://er.knutd.edu.ua/handle/123456789/19938

18. Що таке Tailwind CSS?. T3. [Електронний ресурс]. Режим доступу до ресурсу: https://create.t3.gg/uk/usage/tailwind/

19. В чому різниця між автоматизованим і ручним тестуванням?. Sigma Software University. 19.02.2024. [Електронний ресурс]. Режим доступу до ресурсу: https://university.sigma.software/manual-testing-vs-automation-testing/

20. Побудова діаграм. Draw.io. [Електронний ресурс]. Режим доступу до ресурсу: https://www.drawio.com/

21. Авраменко А., Авраменко В., Косенюк Г. Тестування програмного забезпечення : Навчальний посібник. Черкаси : ЧНУ імені Богдана Хмельницького, 2017. 284 с. [Електронний ресурс]. Режим доступу до ресурсу: https://eprints.cdu.edu.ua/1482/1/testyvan.pdf

22. Кодекс цивільного захисту України : Відомості Верховної Ради від 02.12.2012, № № 5403-VI : станом на 01.01.2025. [Електронний ресурс]. Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/5403-17#Text.

23. Грибан В. Г., Фоменко А. Є., Казначеєв Д. Г. Безпека життєдіяльності та охорони праці : підручник. Дніпро : Дніпроп. держ. ун-т внутр. справ, 2022. 388 с. [Електронний ресурс]. Режим доступу до ресурсу: https://fpk.in.ua/images/biblioteka/2FMB_Pravo/Hryban.-2022-Pidruchnyk-BZHD-ta-OP.pdf.

24. Баличева Н. В. Основи охорони праці та безпеки життєдіяльності : навчальний посібник для здобувачів вищої освіти. Умань : Візаві, 2023. 273 с. [Електронний ресурс]. Режим доступу до ресурсу: https://dspace.udpu.edu.ua/bitstream/123456789/16452/1/%D0%9E%D0%9E%D0%9F%20%D1%82%D0%B0%20%D0%91%D0%96%D0%94%202023.pdf?

25. .Гурик О. Я., Окіпний І. Б., Курс Безпеки життєдіяльності та основи охорони праці : електронний курс [Електронний ресурс] – Тернопіль: Тернопільський національний технічний університет ім. І. Пулюя, Серт. №0273, 2020 - Режим доступу: https://dl.tntu.edu.ua/content.php?cid=299396

26. Правила поведінки під час стихійного лиха. ДСНС. Навчально-методичний центр цивільного захисту та безпеки життєдіяльності. 25.05.2021. [Електронний ресурс]. Режим доступу до ресурсу: https://nmc.dsns.gov.ua/zp/news/ostanni-novini/

27. Про затвердження Правил пожежної безпеки в Україні : Наказ МВС України від 30.12.2014, № № 1417 : станом на 14.08.2024. [Електронний ресурс]. Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0252-15#Text

28. ДСТУ EN 62305-3:2021 Блискавкозахист. Частина 3. Фізичні пошкодження будівель (споруд) та небезпека для життя. На заміну ДСТУ EN 62305-3:2012; чинний від 2022–08–01. Вид. офіц. Київ : УкрНДНЦ, 2022. 126 с. [Електронний ресурс]. Режим доступу: https://zakon.isu.net.ua/sites/default/files/normdocs/dstu_en_62305-3_2021_bliskavkozakhist._chastina_3._fizichni_p.pdf

ДОДАТКИ

Додаток А

Код контролерів

Лістинг коду контролера аналітики:

```
class AnalyticsController < ApplicationController
  before_action :authenticate_user!

  def index
    @max_month = Date.today.strftime('%Y-%m') # Обмеження для вибору місяця
у формі

    begin
                    selected_date      =      params[:month].present?      ?
Date.strptime(params[:month], '%Y-%m') : Date.today
    rescue ArgumentError
      flash[:alert] = "Невірний формат дати. Будь ласка, оберіть місяць ще
раз."
      redirect_to analytics_path and return
    end

    if selected_date > Date.today
      flash[:alert] = "Майбутні місяці недоступні для перегляду."
      redirect_to analytics_path and return
    end

    start_date = selected_date.beginning_of_month
    end_date = selected_date.end_of_month

    # Графік залишку бюджету
              @current_budget      =      current_user.budgets.find_by(month:
selected_date.month, year: selected_date.year)
    if @current_budget.present?
      expenses_by_day = current_user.transactions
         .where(transaction_type: 'expense', date: start_date..[Date.today,
end_date].min)
        .group(:date)
        .sum(:amount)

      total = 0
       @budget_chart_data = (start_date..end_date).each_with_object({}) do
|day, hash|
        if day <= Date.today
          total += expenses_by_day[day] || 0
          hash[day] = (@current_budget.amount - total).round(2)
        else
          hash[day] = nil
        end
      end
    else
      @budget_chart_data = {}
    end

    # Графік фінансових цілей
    @goals_data = current_user.goals.map do |goal|
      {
        name: goal.name,
```

```
        data: {
          "Досягнуто" => goal.current_amount,
          "Ціль" => goal.target_amount
        }
      }
    end

    # Витрати по категоріях
    @expenses_by_category = current_user.transactions
      .where(transaction_type: 'expense', date: start_date..end_date)
      .joins(:category)
      .group('categories.name')
      .sum(:amount)

    # Доходи vs Витрати
    income = current_user.transactions
      .where(transaction_type: 'income', date: start_date..end_date)
      .sum(:amount)
    expense = current_user.transactions
      .where(transaction_type: 'expense', date: start_date..end_date)
      .sum(:amount)

    @comparison_chart_data = {
      "Доходи" => income,
      "Витрати" => expense
    }

    # Повідомлення, якщо немає даних у вибраному місяці
    if income.zero? && expense.zero? && @current_budget.blank?
        flash.now[:alert] = "За обраний місяць немає фінансових даних.
Спробуйте інший період."
    end

    @selected_month = selected_date.strftime('%Y-%m')
  end
end
```

Лістинг коду контролера головної сторінки:

```
class HomeController < ApplicationController
  before_action :authenticate_user!

  def index
    # === ОСТАННІ ТРАНЗАКЦІЇ ===
    @recent_transactions = current_user.transactions
                                       .includes(:category)
                                       .order(created_at: :desc)
                                       .limit(5)

    # === БЮДЖЕТ ===
    current_month = Date.today.month
    current_year = Date.today.year
     @current_budget = current_user.budgets.find_by(month: current_month,
year: current_year)

    if @current_budget.present?
      expenses = current_user.transactions
```

```
                                    .where(transaction_type: 'expense', date:
@current_budget.date_range)
                                .sum(:amount)

      incomes = current_user.transactions
                                    .where(transaction_type: 'income', date:
@current_budget.date_range)
                                .sum(:amount)

      @spent = expenses
      @earned = incomes
      @remaining = @current_budget.amount + incomes - expenses

      expenses_by_day = current_user.transactions
                                        .where(transaction_type: 'expense',
date: @current_budget.date_range)
                                    .group(:date)
                                    .sum(:amount)

      cumulative_expenses = {}
      total = 0
      (@current_budget.date_range).each do |day|
        if day <= Date.today
          total += expenses_by_day[day] || 0
           cumulative_expenses[day] = (@current_budget.amount + incomes -
total).round(2)
        else
          cumulative_expenses[day] = nil
        end
      end

      @budget_chart_data = cumulative_expenses
    else
      @spent = 0
      @earned = 0
      @remaining = 0
      @budget_chart_data = {}
    end

    # === ЦІЛІ ===
    @goals_data = current_user.goals.map do |goal|
      {
        name: goal.name,
        data: {
          "Досягнуто" => goal.current_amount,
          "Ціль" => goal.target_amount
        }
      }
    end

    # === СПОВІЩЕННЯ ===
        @unread_notifications   =   current_user.notifications.where(read:
false).order(created_at: :desc)
  end
end
```

Додаток Б

Код моделей та бази даних

Лістинг коду моделі користувача:

```
class User < ApplicationRecord
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable
  has_many :transactions, dependent: :destroy
  has_many :goals, dependent: :destroy
  has_many :categories, dependent: :destroy
  has_many :budgets, dependent: :destroy
  has_many :notifications, dependent: :destroy
end
```

Лістинг коду моделі транзакцій:

```
class Transaction < ApplicationRecord
  belongs_to :user
  belongs_to :category, optional: true

  validates :date, presence: true

  after_create :check_budget_thresholds, if: -> { transaction_type ==
"expense" }

  def self.ransackable_attributes(auth_object = nil)
    %w[id user_id category_id amount transaction_type description date
created_at updated_at]
  end

  private

  def check_budget_thresholds
    return unless date.present?

    month = date.month
    year = date.year

    budget = user.budgets.find_by(month: month, year: year)
```

```
    return unless budget

    expenses = user.transactions
                   .where(transaction_type: "expense", date:
date.beginning_of_month..date.end_of_month)
                   .sum(:amount)

    percent_spent = (expenses / budget.amount.to_f * 100).round

    thresholds = [80, 90, 95, 100]

    thresholds.each do |threshold|
      next unless percent_spent >= threshold

      notification_exists = user.notifications.exists?(
        title: "Перевищення бюджету на #{threshold}%",
        created_at: date.beginning_of_month..date.end_of_month
      )

      unless notification_exists
        user.notifications.create!(
          title: "Перевищення бюджету на #{threshold}%",
          body: "Ви витратили понад #{threshold}% бюджету за #{I18n.l(date,
format: "%B %Y")}.",
          read: false
        )
      end
    end
  end
end
```

Лістинг коду структури бази даних:

```
ActiveRecord::Schema[8.0].define(version: 2025_06_15_201220) do
  create_table "budgets", charset: "utf8mb4", collation:
"utf8mb4_0900_ai_ci", force: :cascade do |t|
    t.decimal "amount", precision: 10
    t.integer "month"
    t.integer "year"
    t.bigint "user_id", null: false
```

```
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["user_id"], name: "index_budgets_on_user_id"
  end

  create_table "categories", charset: "utf8mb4", collation:
"utf8mb4_0900_ai_ci", force: :cascade do |t|
    t.string "name"
    t.string "category_type"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

  create_table "goals", charset: "utf8mb4", collation:
"utf8mb4_0900_ai_ci", force: :cascade do |t|
    t.bigint "user_id", null: false
    t.string "name"
    t.decimal "target_amount", precision: 10
    t.decimal "current_amount", precision: 10
    t.date "deadline"
    t.text "description"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["user_id"], name: "index_goals_on_user_id"
  end

  create_table "notifications", charset: "utf8mb4", collation:
"utf8mb4_0900_ai_ci", force: :cascade do |t|
    t.string "title"
    t.text "body"
    t.boolean "read"
    t.bigint "user_id", null: false
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["user_id"], name: "index_notifications_on_user_id"
  end

  create_table "transactions", charset: "utf8mb4", collation:
"utf8mb4_0900_ai_ci", force: :cascade do |t|
```

```
    t.decimal "amount", precision: 10
    t.string "transaction_type"
    t.bigint "user_id", null: false
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.date "date"
    t.bigint "category_id"
    t.text "description"
    t.index ["category_id"], name: "index_transactions_on_category_id"
    t.index ["user_id"], name: "index_transactions_on_user_id"
  end

  create_table "users", charset: "utf8mb4", collation:
"utf8mb4_0900_ai_ci", force: :cascade do |t|
    t.string "email", default: "", null: false
    t.string "encrypted_password", default: "", null: false
    t.string "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["email"], name: "index_users_on_email", unique: true
    t.index ["reset_password_token"], name:
"index_users_on_reset_password_token", unique: true
  end

  add_foreign_key "budgets", "users"
  add_foreign_key "goals", "users"
  add_foreign_key "notifications", "users"
  add_foreign_key "transactions", "categories"
  add_foreign_key "transactions", "users"
end
```

Додаток В

Диск з роботою