Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

ЗАСТОСУНОК ДЛЯ ФІНАНСОВОГО ПЛАНУВАННЯ СІМЕЙНОГО БЮДЖЕТУ

Виконала:
здобувачка IV курсу
групи ІПЗ-41
спеціальності 121 «Інженерія програмного забезпечення»
Сачко Ольга Ігорівна

Науковий керівник:
к.т.н., доцент Шевцова Н. В.

Рівне – 2026

ЗМІСТ

ВСТУП

Актуальність дослідження. Розробка додатків для контролю власних фінансів стала стратегічно важливим кроком для багатьох людей, які бажають забезпечити зручний доступ до інформації про свої доходи та витрати. Особливо актуальною ця тема стає у зв'язку зі створенням власної сім’ї та, відповідно, зміною у структуризації бюджету − розподілу коштів на декількох людей.

Попри те, що вже існує значна кількість програмних рішень для обліку фінансів, багато з них або перевантажені функціоналом, або не враховують специфіку колективного керування бюджетом в межах сім’ї.

Незважаючи на значну кількість досліджень, питання створення сучасного кросплатформного застосунку для управління сімейним бюджетом із підтримкою аналітики, автоматизації накопичень та спільного доступу все ще залишається актуальним і потребує подальшого розвитку.

Метою роботи є розробка та реалізація програмного застосунку для фінансового планування сімейного бюджету, який забезпечує облік доходів і витрат, підтримку заощаджень, автоматизацію регулярних фінансових операцій та зручний інтерфейс для користування декількома користувачами водночас.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Дослідити сучасні підходи та програмні рішення для управління особистими і сімейними фінансами.
2. Провести аналіз існуючих мобільних застосунків та визначити їх переваги й недоліки.
3. Сформувати функціональні та нефункціональні вимоги до програмного продукту.
4. Обґрунтувати вибір технологічного стеку для реалізації мобільного застосунку.
5. Спроєктувати архітектуру програмної системи та структуру бази даних.

6. Розробити інтерфейс користувача з урахуванням принципів адаптивності та зручності використання.
7. Реалізувати функціональні модулі застосунку для управління доходами, витратами та накопиченнями.
8. Імплементувати систему авторизації та синхронізації даних із хмарним середовищем.
9. Реалізувати модуль аналітики та візуалізації фінансової інформації.

Об’єктом дослідження є процес управління сімейним бюджетом із використанням інформаційних технологій.

Предметом дослідження є методи та засоби розробки програмного забезпечення для автоматизації обліку фінансів, планування бюджету та управління заощадженнями.

У роботі використано теоретичні та практичні **методи дослідження**. Теоретичні методи включають аналіз наукової літератури, порівняння існуючих програмних рішень, систематизацію та узагальнення інформації щодо сучасних підходів до розробки мобільних застосунків. Практичні методи передбачають проєктування архітектури системи, моделювання структури даних, програмну реалізацію функціональних модулів, тестування та оптимізацію програмного забезпечення.

Апробація і впровадження результатів дослідження. Основні теоретичні та практичні результати дослідження були апробовані шляхом доповідей на XIX Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих учених «Наука, освіта, суспільство очима молодих» (м. Рівне, 15 травня 2026) [1], а також на VI Міжнародній науково-практичній конференції «Science and Information Technologies in the Modern World» (м. Афіни, 15-17 квітня 2026) [2].

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто теоретичні основи фінансового планування, сучасні підходи до організації сімейного бюджету та аналіз існуючих програмних рішень. У другому розділі описано процес проєктування архітектури системи, структури бази даних

та взаємодії між компонентами застосунку. Третій розділ присвячений практичній реалізації програмного застосунку, розробці користувацького інтерфейсу, тестуванню та аналізу отриманих результатів.

У роботі використано 3 таблиці та 24 рисунки, які ілюструють архітектуру системи, інтерфейс застосунку та результати реалізації окремих функціональних модулів. Загальний обсяг кваліфікаційної роботи становить 46 сторінок. Список використаних джерел налічує 28 найменувань.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Проблема контролю сімейних фінансів

Організації економічного співробітництва та розвитку (ОЕСР) визначає фінансову грамотність як «комплекс знань, вмінь, навичок, ставлення та поведінки людини, необхідних для ухвалення важливих фінансових рішень та для досягнення особистого фінансового добробуту» [3].

В сучасному світі життєво необхідно вміти розпоряджатися власними заощадженнями правильно. Розумне планування бюджету в сучасній родині – це не тільки ретельний контроль за витратами, а й вміння ефективно використовувати свої кошти задля їх примноження.

Сучасні методи планування бюджету базуються на гнучкості, розумінні економічної ситуації та максимальному використанні цифрових можливостей, не потрапляючи при цьому у пастки інтернет маркетологів.

У 2021 році дослідницька агенція Info Sapiens за підтримки USAID – незалежної агенції федерального уряду США, та Національного банку України провела дослідження фінансової грамотності, інклюзії та добробуту українців [4].

Аналіз визначив рівень знань про інфляцію та відсоткові ставки, окрему увагу приділили фінансовій стійкості та здатності громадян долати економічні кризи за допомогою заощаджень.

У 2021 році половина опитуваних користуються мобільним телефоном (мобільним додатком банку) при здійсненні платежів, як показано на рисунку 1.1.

Загалом 38% респондентів використовує і комп’ютер, і мобільний телефон для здійснення фінансових операцій.

Діджитал-активи не є пріоритетними для дорослого населення України, 42% не знає чи є зараз підходящим час для інвестицій в них, ще 10% вважає це питання неактуальним для себе.

Рисунок 1.1. Статистика технологічної досвідченості українців 2021 року

Документ загалом зафіксував позитивну динаміку: індекс фінансової обізнаності населення зріс до 12,3 бала, що наблизило Україну до середніх показників країн ОЕСР.

Дослідження визначає молодь та пенсіонерів як найбільш вразливі групи населення – це виявляє суттєві розбіжності в обізнаності залежно від віку, освіти та рівня доходів громадян України.

Громадян було опитано які дії вони роблять для контролю власного бюджету (рисунок 1.2). На основі отриманих даних було сформовано стратегічні рекомендації щодо державної політики, які в свою чергу спрямовані на посилення захисту прав споживачів та розвиток інклюзивності фінансового ринку.

Одною з таких рекомендацій стало створення навичок моніторингу власних операцій, тобто регулярна перевірка виписок та проведених грошових транзакцій.

У 2024 році Національним Банком України було затверджено Національну стратегію розвитку фінансової грамотності до 2030 року, що представляє собою ідею підвищення фінансової обізнаності населення України [5].

Однією із цілей проєкту є Просунута цифрова фінансова грамотність – особливу увагу буде приділено саме розробці інтерактивних цифрових інструментів, які спрямовані на підвищення фінансової грамотності.

Дії	Україна, 2021 р.	Україна, 2018 р.	Зміна, в.п.
Планувати доходи та витрати	68%	70%	-2 в.п.
Робити записи своїх витрат	23%	19%	+4 в.п.
Тримати гроші для сплати рахунків окремо від грошей на щоденні витрати	31%	27%	+4 в.п.
Робити записи про майбутні рахунки, щоб їх не пропустити	14%	7%	+7 в.п.
Використовувати банківський додаток або інструмент управління грошовими коштами, щоб відстежити витрати	13%	6%	+7 в.п.
Налаштовувати автоматичні платежі для регулярних витрат	6%	2%	+4 в.п.
НІЧОГО	16%	18%	-2 в.п.
(можливо зазначити декілька відповідей)			

Рисунок 1.2. Дії опитуваних українців для контролю за бюджетом

1.2. Огляд існуючих рішень

Існує багато програмних рішень для спрощення ведення власного бюджету – як вебресурси, так і застосунки на персональний комп'ютер або мобільний пристрій. Однак більшість з них не розраховані для ведення бюджету від обличчя декількох людей водночас.

Аналіз найпопулярніших додатків для ведення фінансів був проведений на платформі Google Play (або Play Маркет) – крамниці застосунків від Google, що дозволяє власникам пристроїв з мобільною операційною системою Android та іншими завантажувати і купувати різні застосунки, книги, фільми і музику [6].

Одним з найпопулярніших серед користувачів Google Play виявився додаток «Monefy – Budget & Expences app» – застосунок дає користувачу можливість вводити доходи та витрати, візуалізуючи їх у вигляді кругової діаграми (рис. 1.4).

Із мінусів слід зазначити відсутність статистики витрат та доходів за певний період часу та неможливість додати нову інформацію про транзакції, що відбулися в минулому.

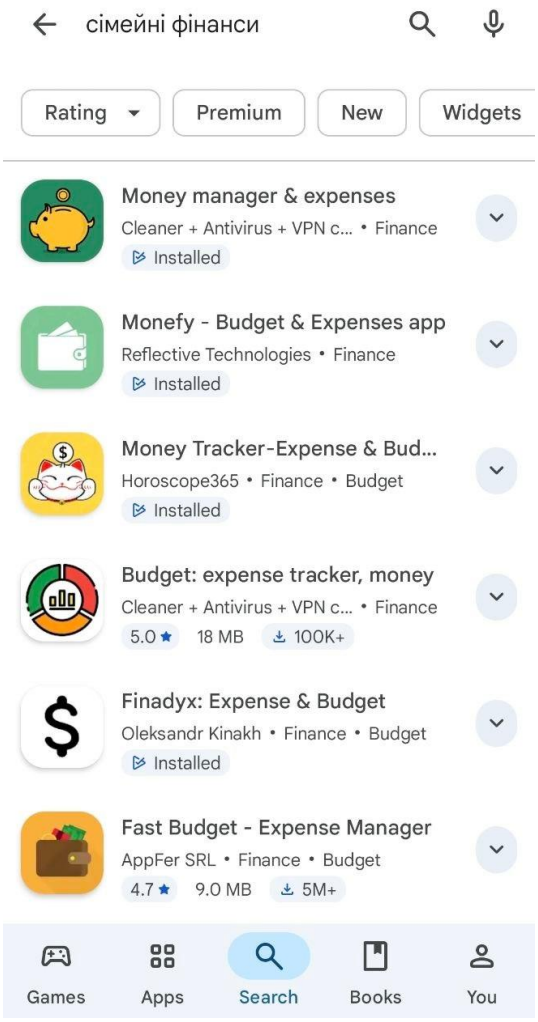


Рисунок 1.3. Найпопулярніші додатки всередині Google Play при пошуку «сімейні фінанси»

Також одним із суттєвих мінусів програми є монетизація більшості функціоналу. Без підписки на сервіс користувач не має можливості додати категорії витрат та прибутків, змінити валюту, додати постійний дохід та синхронізувати декілька девайсів водночас.

Наступним у рейтингу є додаток «Money Tracker-Expense & Budget». Вигляд головного екрану додатку представлено на рисунку 1.5. Програма позиціонує себе як менеджер грошей, касова книга та органайзер рахунків.

Окрім ведення власного бюджету, він надає можливість переглядати статистику або вже готові звіти проведених грошових операцій.

Рисунок 1.4. Початковий екран мобільного додатку «Monefy – Budget & Expences app»

Всередині застосунку існує можливість керування бюджетом від обличчя декількох користувачів водночас, але ця функція доступна тільки після купівлі річної преміум підписки.

Далі розглянемо додаток «Fynadyx: Expense & Budget», створений українським Android розробником Олександром Кінахом. Він має зрозумілий інтерфейс, функції обліку витрат та прибутків із поділом на категорії (рис. 1.6).

Всередині є досить багато функцій, до яких потрібно звикнути, тим не менш споживач може керувати різними обліковими записами, планувати платежі, вписувати різні джерела доходу та розділяти їх на категорії.

Мінусом цього застосунку також є монетизація більшості функціоналу та відсутність поділу акаунту на членів родини.

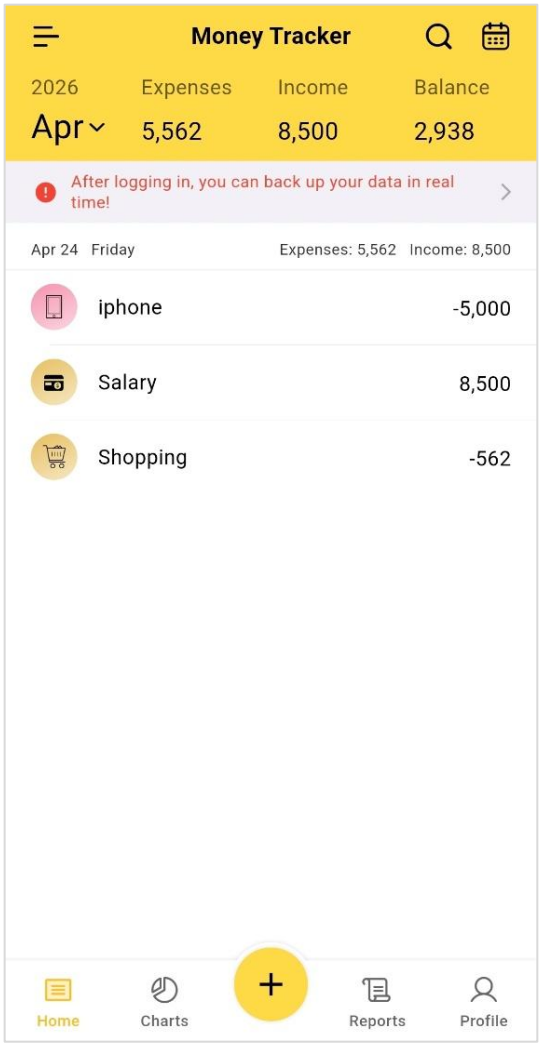


Рисунок 1.5. Початковий екран мобільного застосунку «Money Tracker-Expense & Budget»

Також є доцільним включити у вибірку розділ зі статистикою всередині застосунку monobank, адже більшість громадян України контролюють власні витрати саме там [7].

Як показано на рисунку 1.7, якщо у користувача банку декілька карток, він може вибрати перегляд всіх водночас або подивитись статистику по кожній окремо.

Зверху необхідно вказати період, за який потрібно переглянути витрати – для цього потрібно натиснути на іконку календаря і вибрати початкову і кінцеву дату. Після цього користувач може побачити всі витрати за вказаний період, скільки

зроблено транзакцій і по яким категоріям. Статистика також відображає накопичений кешбек – це програма лояльності, що дозволяє повертати частину коштів (до 20% і більше) за покупки, вибрані на початку місяця, що нараховується на спеціальний рахунок сумою від 100 гривень [8].

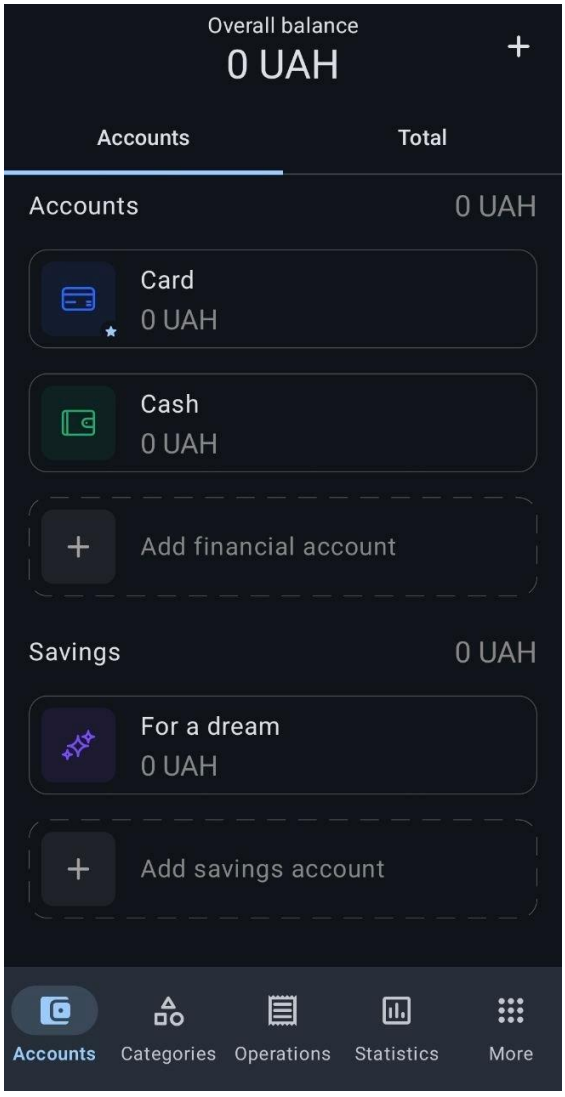


Рисунок 1.6. Розділ гаманців мобільного застосунку «Fynadyx: Expense & Budget»

Недоліком статистики всередині додатку monobank є відсутність більшості функціоналу, який представлений у самостійних мобільних застосунках для ведення фінансів, та можливість опрацювання тільки транзакцій безпосередньо всередині самого банку.

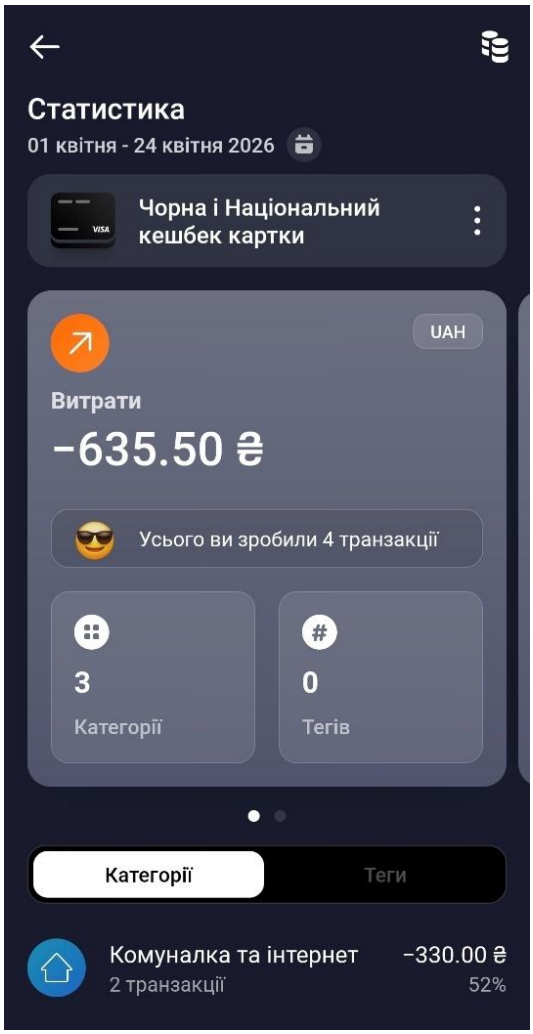


Рисунок 1.7. Розділ статистики застосунку monobank

В результаті було проведено порівняльний аналіз функціональних можливостей вже існуючих застосунків для ведення фінансів та було сформовано таблицю застосунків-аналогів розроблюваного продукту (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця застосунків-аналогів розроблюваного продукту

Застосунок	**Інтерфейс**	**Ведення доходів та витрат**	**Цілі**	**Статистика**	**Декілька акаунтів**
Monefy – Budget & Expences app	+	+	-	-	-

Продовження табл. 1.1

Money Tracker- Expense & Budget	-+	+	+	+	-+
Fynadyx: Expense & Budget	+	-+	-+	+	-+
monobank	+	+	-	+	-

У висновку було виявлено основні проблеми вже існуючих програмних рішень: відсутність безкоштовних комплексних рішень, якісних застосунків українською мовою та автоматичних регулярних платежів у більшості.

1.3. Постановка задачі дослідження

На основі виявлених недоліків було поставлено наступну задачу - розробити безкоштовний вебзастосунок для управління сімейним бюджетом. Було створено таблицю функціональних (табл. 1.2) та нефункціональних (табл. 1.3) вимог до створюваного продукту.

Таблиця 1.2

Функціональні вимоги до застосунку управління сімейним бюджетом

Вимога	**Опис**
Реєстрація користувача	Система повинна забезпечувати можливість створення нового облікового запису користувача.
Авторизація користувача	Застосунок повинен підтримувати вхід користувача за електронною поштою та паролем.
Управління профілем	Система повинна дозволити редагування персональних даних користувача.

Продовження табл. 1.2

Створення сімейного бюджету	Користувач повинен мати можливість створювати спільний сімейний бюджет.
Додавання учасників	Система повинна підтримувати додавання членів сім’ї до спільного бюджету.
Додавання доходів та витрат	Користувач повинен мати можливість додавати записи про доходи та витрати.
Редагування транзакцій	Користувач повинен мати можливість змінювати інформацію про фінансові операції.
Видалення транзакцій	Повинна бути реалізована можливість видалення фінансових операцій.
Категоризація операцій	Система повинна підтримувати поділ доходів і витрат за категоріями.
Перегляд балансу	Застосунок повинен відображати актуальний баланс бюджету.
Формування статистики	Система повинна формувати статистику витрат і доходів.
Накопичення коштів	Користувач повинен мати можливість створювати фінансові цілі та накопичення.
Автоматичне накопичення	Система повинна підтримувати автоматичне перенесення частини коштів до накопичень.
Локалізація	Застосунок повинен підтримувати декілька мов інтерфейсу.
Синхронізація даних	Дані користувачів повинні автоматично синхронізуватися через хмарний сервіс.

Також було створено UML Use Case Diagram, що знаходиться у додатку А. Діаграма випадків використання UML візуально відображає взаємодію між користувачами (акторами) та системою, зосереджуючись на основних

функціональних можливостях без детального опису внутрішніх процесів [9]. Вони описують у текстовій формі поведінку системи, коли один із суб'єктів системи надсилає певний сигнал.

Таблиця 1.3

Нефункціональні вимоги до застосунку управління сімейним бюджетом

Вимога	**Опис**
Кросплатформність	Застосунок повинен працювати на Android, iOS та Web-платформах.
Продуктивність	Час завантаження основних екранів не повинен перевищувати декількох секунд.
Масштабованість	Архітектура системи повинна забезпечувати можливість розширення функціоналу.
Безпека даних	Дані користувачів повинні передаватися та зберігатися у захищеному вигляді.
Зручність використання	Інтерфейс застосунку повинен бути інтуїтивно зрозумілим для користувача.
Адаптивність інтерфейсу	Інтерфейс повинен коректно відображатися на різних розмірах екранів.
Підтримуваність	Код програми повинен бути структурованим та придатним для подальшої модифікації.
Модульність	Архітектура застосунку повинна передбачати незалежність окремих компонентів.
Сумісність	Застосунок повинен підтримувати сучасні версії мобільних операційних систем і браузерів.
Локалізація	Повинна бути забезпечена підтримка багатомовного інтерфейсу.
Розширюваність	Повинна бути можливість інтеграції нових модулів та сервісів у майбутньому.

У підсумку, для реалізації всіх необхідних функцій, з урахуванням усіх вищеописаних функціональних та нефункціональних вимог, необхідно спроектувати і розробити розділи застосунку: систему реєстрації та авторизації користувачів, головний екран із відображенням поточного балансу та фінансової статистики, модуль управління доходами і витратами, розділ аналітики та звітності, систему накопичень і фінансових цілей, модуль спільного керування бюджетом, налаштування профілю та локалізації застосунку.

Таким чином необхідно забезпечити синхронізацію даних із хмарною базою даних, адаптивність інтерфейсу на різних платформах і пристроях. Разом із тим необхідно забезпечити належний рівень безпеки та стабільності роботи розроблюваного програмного продукту.

РОЗДІЛ 2
МЕТОДОЛОГІЯ ТА ТЕХНОЛОГІЇ РОЗРОБКИ

2.1 Огляд та обґрунтування вибору технологій

React – це бібліотека JavaScript для створення користувацьких інтерфейсів, розроблена у Facebook і випущена у 2013 році [10].

Можна сміливо сказати, що React є найвпливовішою бібліотекою для створення інтерфейсів за останні роки. Її використовують для створення компонентів, які представляють логічні частини інтерфейсу, що можуть бути повторно використані.

Суть React полягає в тому, що складність створення компонента зведена до мінімуму – це просто функція JavaScript. Значенням, що повертається цією функцією, є HTML або інтерфейс користувача, написаний у спеціальній синтаксичній форматі JSX, що дозволяє легко поєднувати JavaScript з HTML-розміткою.

Для того, щоб передати дані в компонент, просто передається аргумент props, на який потім можна посилатися всередині тіла функції або в інтерфейсі користувача, використовуючи фігурні дужки.

Хук – це функція, яка повертає значення, а також функція для зміни значення. Якщо потрібно надати компоненту власний внутрішній стан, ми можемо використовувати хук useState.

React сам по собі не займається маршрутизацією, управлінням станом, анімацією, натомість він дозволяє цим аспектам природно розвиватися в рамках спільноти відкритого програмного забезпечення. Незалежно від того, що саме розробник намагається зробити, найімовірніше, така бібліотека вже існує. Потрібен статичний сайт – є Gatsby, серверний рендеринг – Next, для анімації є Spring, для форумів – Formic, для управління станом існують Redux, MobX, Flux, Recoil, X State та багато іншого.

Після опанування React, досить легко перейти до React Native і почати створювати мобільні додатки. Сьогодні знання цієї бібліотеки є одним із найбільш затребуваних навичок для фронтенд-розробників [11].

Якщо зазирнути всередину одного з файлів TypeScript, код там буде точно таким самим, як і у файлі JavaScript.

TypeScript як мова є надмножиною JavaScript, а це означає, що звичайний JavaScript також є на 100% дійсним TypeScript. Відносини між SCSS і CSS такі самі, як між TypeScript і JavaScript. Іншими словами, він просто додає додаткові функції поверх звичайного JavaScript.

Браузери не знають, як виконувати код TypeScript, потрібен компілятор, щоб перетворити код TypeScript на звичайний JavaScript. У проєкті застосунку для ведення сімейних фінансів можна помітити спеціальний файл під назвою tsconfig. Його призначення – налаштувати поведінку компілятора, який сам здатний перетворити код TypeScript на будь-який варіант JavaScript, який запускається в браузері.

Компілятори дозволяють використовувати спеціальні функції сучасного синтаксису JavaScript, не турбуючись про те, чи підтримуватиметься цей код у старих браузерах. Єдиним недоліком є те, що тепер всередині проєкту існує досить великий файл конфігурації.

Всередині JavaScript-версії Create React App використовується компілятор під назвою Babel, який виконує те саме. Таким чином можна писати сучасний код як у проєктах React.js, так і в проєктах TS.

Vite – це інструмент для компіляції JavaScript, який дозволяє миттєво запускати проєкт і відразу бачити внесені зміни, незалежно від розміру розроблюваної програми [12]. Vite має ліцензію MIT і є безкоштовним та з відкритим вихідним кодом.

Він виконує дві функції: по-перше, забезпечує локальне виконання коду під час розробки, а по-друге, об’єднує файли JavaScript, CSS та інші ресурси в єдиний пакет для розгортання у виробничому середовищі.

З використанням Vite не потрібно постійно перекомпілювати код, натомість розробники відразу бачать внесені зміни.

Firebase та Supabase є платформами «бекенд як послуга», тому розробникам, що користуються їх послугами не потрібно керувати власними серверами, вони масштабуються автоматично.

На відміну від багатьох NoSQL рішень, Supabase використовує PostgreSQL – реляційну базу даних з відкритим вихідним кодом і, по суті, є відкритою альтернативою Firebase. Firebase – це набагато старіший продукт і він має набагато більше пропозицій порівняно з Supabase, але Supabase має велику перевагу, оскільки він використовує тільки технологої з відкритим кодом.

Автентифікація та авторизація Supabase схожі, вони дуже прості у використанні. Можна легко реалізувати автентифікацію на обох платформах – вони обидва мають інтуїтивно зрозумілі інтерфейси користувача для обробки автентифікації, досвід розробників щодо впровадження автентифікації на Firebase та Supabase дуже схожий.

Firestore є провідною пропозицією у сфері баз даних. Базу даних Firestore робить унікальною те, що це база даних реального часу, побудована на базі даних NoSQL. Вона є власністю саме Firebase, а Supabase в свою чергу фактично створює свою базу даних на основі PostgreSQL.

Як було зазначено, Supabase побудований на повністю відкритій технології – розробникам не потрібно турбуватися про прив'язку до постачальника. Тоді як Firebase змусить набагато більше прив'язуватися до постачальника послуг, оскільки база даних, яку власники платформи використовують, є їх власністю.

Як Firebase, так і Supabase пропонують правила безпеки, але спосіб, у який вони пропонують ці правила, дуже відрізняється. Правила безпеки на основі Supabase написані повністю на SQL, тому для тих розробників, хто добре володіє цією мовою програмування, буде дуже просто їх написати.

Firebase для написання правил безпеки не використовує SQL і використовує лише власний синтаксис, який є досить інтуїтивно зрозумілим. Але коли розробник

потрапляє в більш складні випадки розробки баз даних, правила безпеки Firebase буде написати складніше.

В цілому вони обидва пропонують користувачеві дуже схожі функції, але спосіб їх реалізації відрізняється. У цьому випадку було надано перевагу Supabase, оскільки все написано на SQL, від правил автентифікації – до налаштування бази даних.

Хмарне сховище дуже просте у використанні та налаштуванні: спочатку створюється сховище, додаються правила безпеки, а потім всередину завантажується все інше.

Як Supabase, так і Firebase пропонують хмарні служби. Обидва дуже легко налаштувати, але єдиний недолік функцій Supabase полягає в тому, що можна запускати лише одну за раз. Звичайно, є обхідне рішення для тестування всіх функцій одночасно, але воно офіційно не підтримується Supabase.

Важливо також згадати, що однією з функцій Supabase є можливість підписки на зміни в базі даних у реальному часі через вебсокети, що критично для чатів або динамічних дашбордів

Є певні речі, які дешевші на Firebase порівняно з Supabase, і є певні речі, які дешевші на Supabase порівняно з Firebase. Наприклад, Firebase Authentication пропонує безкоштовну необмежену автентифікацію для користувачів, тоді як Supabase пропонує лише 50000 активних користувачів щомісяця безкоштовно [13, 14].

Supabase не стягує плату за запити на читання та запис, а просто за передачу даних, тоді як Firebase стягує плату за запити на читання та запис. Тому використання платформи Firebase може бути досить дорогим, якщо у проєкті виконується багато операцій читання та запису.

Ще одна річ, яку слід врахувати, це той факт, що Supabase побудована на повністю відкритій технології, тому насправді не обов’язково платити за їхні послуги, розробник може розмістити все в образі Docker, а потім опублікувати створений проєкт на своєму постачальнику хмарних послуг за вибором.

У випадку використання Firebase користувач їх послуг буде повністю замкнений на платформі Google Cloud та всередині екосистеми Google.

Якщо потрібне універсальне рішення для підтримки розроблюваних мобільних додатків, ігор Unity або вебдодатку, Firebase буде безперечним переможцем, оскільки вони підтримують вебверсії Android, iOS та, Unity. В свою чергу Supabase офіційно підтримує лише JavaScript, тому це зазвичай буде вебдодаток або мобільний додаток React Native.

У розроблюваному застосунку кваліфікаційної роботи використовувється база даних PostgreSQL, на якій відповідно побудовано Supabase. Вона підтримує як традиційні SQL-дані, так і AI-векторні ембедінги завдяки розширенню PGVector. Розробники можуть взаємодіяти з базою через візуальний редактор таблиць (Table Editor) або безпосередньо через SQL Editor.

Платформа автоматично генерує RESTful API завдяки компоненту PostgREST, що дозволяє маніпулювати даними без написання додаткового серверного коду [15].

Важливою темою також є подолання мовного бар'єру для зручності користувачів розроблюваного продукту. Для надання можливості користувачеві змінювати мову застосунку безпосередньо всередині нього було використано фреймворк i18n з його розширенням react-i18next, створений для JavaScript з динамічним сховищем JSON [16]. Під час його використання додатковий парсинг не потрібен – нові рядки додаються на ходу, коли вони вперше використовуються.

Capacitor – це міжплатформове нативне середовище виконання, яке спрощує розробку високопродуктивних мобільних додатків, що працюють у нативному режимі на iOS, Android та інших платформах із використанням сучасних вебінструментів.

Capacitor створює нативні вебдодатки, пропонуючи сучасний підхід на основі нативних контейнерів для команд, які прагнуть розробляти з орієнтацією на веб, не відмовляючись при цьому від повного доступу до нативних SDK [17].

З його допомогою розробники можуть отримувати доступ до файлової системи, камери або функції вібрації пристрою, щоб безпосередньо активувати вібрацію через створений додаток.

Capacitor дає можливість робити все це, використовуючи лише вебтехнології та звичайний JavaScript. Він сумісний з будь-яким фреймворком JavaScript, що особливо актуально при створенні мобільних додатків, де зазвичай доводиться докладати вдвічі більше зусиль, щоб код працював нативно в різних середовищах. З його допомогою можна створити вебдодаток один раз, а потім використовувати ту саму базу коду, щоб відтворити той ж досвід для користувача на мобільних пристроях.

Саме тому його було обрано для реалізації мобільного застосунку для керування сімейними фінансами, адже можна швидко створювати додатки, не витрачаючи при цьому багато грошей на розробку і отримати кінцевий результат у вигляді мобільного застосунку.

2.2 Проєктування моделі бази даних

База даних – це сукупність взаємопов'язаних даних, які зберігаються разом. Ця сукупність має таку мінімальну надмірність, що припускає використання даних оптимальним чином для однієї або декількох задач [18].

Під час проєктування бази даних застосунку для управління сімейним бюджетом було розроблено реляційну модель, яка б забезпечила реалізацію функцій поставленої задачі дослідження.

Під час розробки моделі бази даних було застосовано принципи нормалізації даних для того, щоб запобігти аномаліям оновлення, видалення та вставки. Схема була розроблена відповідно до третьої нормальної форми (3NF), що забезпечує мінімальну надмірність даних та оптимальну вибірку інформації [19].

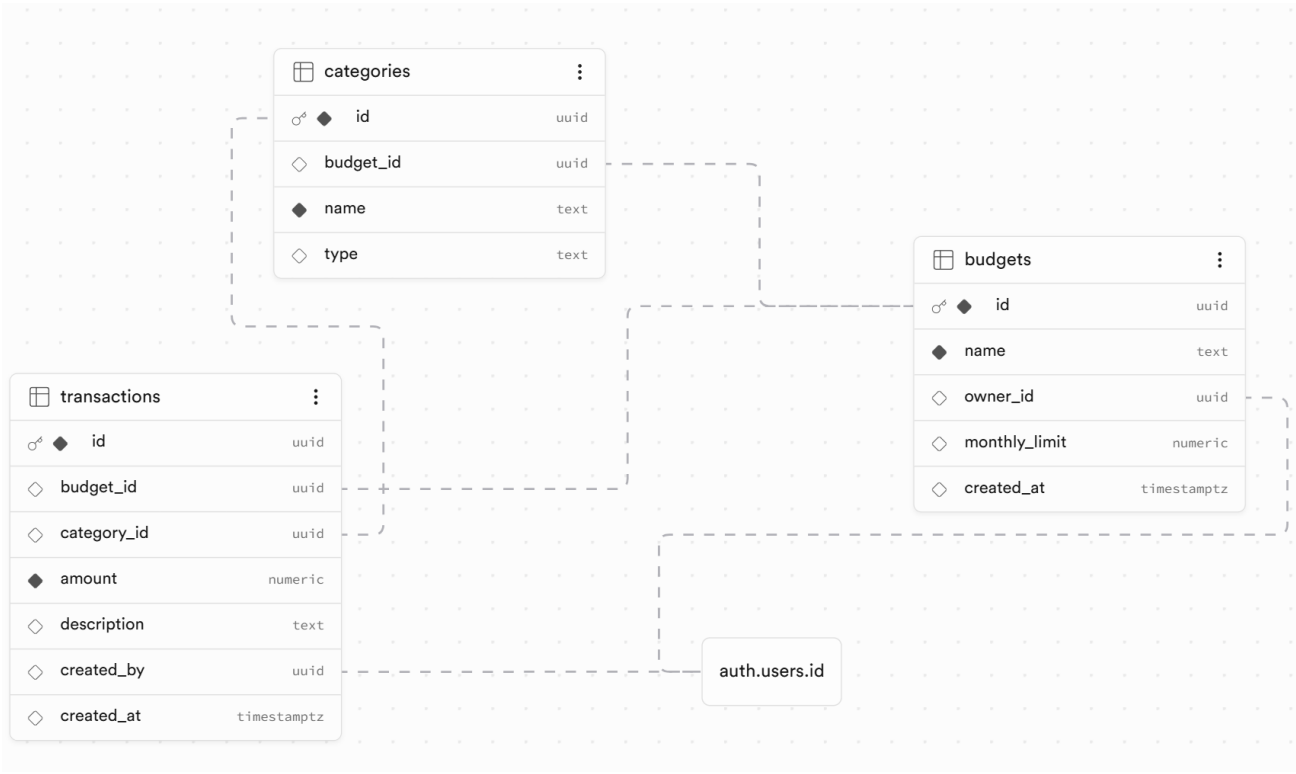


Рисунок 2.1 Структура взаємозв’язаних таблиць моделі бази даних

Реляційна модель передбачає собою наступні основні зв'язки між сутностями: одна сім'я може мати багато профілів (до 5) та безліч бюджетів; одна сім'я може визначити власні категорії, кожна з яких може бути застосована до багатьох транзакцій; один профіль може виконати багато операцій; один бюджет може мати кілька налаштувань регулярних доходів, множину цілей заощаджень та містить багато транзакцій.

На рисунку 2.1 представлено ER-діаграму, що демонструє структуру та взаємозв'язки між основними таблицями бази даних. Діаграма показує первинні ключі (primary keys), зовнішні ключі (foreign keys) і характер зв'язків між таблицями.

Кожна таблиця містить унікальний ідентифікатор, що дозволяє однозначно визначати записи та запобігає виникненню дублювань у системі. Між таблицями було налаштовано зв’язки із використанням механізму каскадного видалення ON DELETE CASCADE для підтримки консистентності даних. Завдяки цьому при видаленні сім’ї автоматично видаляються всі пов’язані записи, зокрема бюджети, транзакції та профілі користувачів.

Окрему увагу було приділено безпеці доступу до інформації. На рівні Supabase було реалізовано політики Row Level Security (RLS), які забезпечують ізоляцію даних між різними сім’ями.

Для кожної таблиці було налаштовано правила доступу, які перевіряють значення family_id поточного користувача перед виконанням операцій читання, додавання, редагування чи видалення. Такий підхід гарантує, що користувачі можуть працювати лише з інформацією, яка належить їхній сімейній групі.

У базі даних було застосовано додаткові обмеження та констрейнти для перевірки правильності введених значень, що дозволяє мінімізувати ризик внесення некоректних даних та підвищує надійність роботи системи. Наприклад, сума фінансової транзакції повинна бути додатною, а тип категорії може приймати лише визначені значення income або expense.

Під час кожного запиту передаються не всі записи, а лише поточна сторінка, це значно прискорює роботу застосунку під час слабкого з’єднання.

Структуру таблиць було спроєктовано так, щоб забезпечити цілісність даних та дозволити легке розширення функціоналу в майбутньому.

2.3. Проєктування архітектури застосунку

Як любить нагадувати фахівець із мобільних технологій Джейсон Грігсбі: «Вебпосилання не відкривають додатки, а ведуть на вебсторінки». Незалежно від того, чи це пошук, електронна пошта, соціальні мережі чи вебсторінки, якщо у вас є контент в Інтернеті, люди знайдуть і поділяться посиланнями на нього. Відсутність мобільного вебрішення означає, що будь-хто, хто перейде за цими посиланнями на мобільному пристрої, не отримає гарного досвіду (якщо взагалі зможе отримати доступ до вашого контенту). Наявність нативного мобільного додатка не допоможе [20].

На рисунку 2.2 представлено загальну архітектуру мобільного застосунку управління сімейним бюджетом. Архітектура системи побудована за клієнт-серверним принципом із використанням React для реалізації інтерфейсу користувача та Supabase як серверної платформи.

Для створення розроблюваного застосунку було обрано односторінковий додаток – single page application (SPA).

У SPA додаток працює як єдина вебсторінка, а рівень візуалізації обробляється безпосередньо в браузері, що покращує користувацький досвід завдяки відсутності повного оновлення сторінки [21].

Початкова структура SPA нагадує традиційний вебдодаток, але між ними є суттєві відмінності. У SPA «види» (views) – це фрагменти DOM, а не цілі HTML-сторінки.

Модель об’єктів документа (DOM) — це програмний інтерфейс, який виконує роль сполучної ланки між веб-сторінками та скриптами або мовами програмування. Вона представляє веб-документ (наприклад, HTML або XML) у вигляді дерева об’єктів, як показано на рисунку 2.3, де кожна гілка дерева закінчується вузлом, а кожен вузол вже містить самі об’єкти [22]. Після першого завантаження нові «види» створюються в браузері динамічно.

Починається все з ініціалізації та запуску простого проєкту, що призведе до створення HTML-сторінки «index.html», де односторінковий додаток завантажується в оболонку додатка. Сам додаток – це просто лічильник на основі цього скрипта. DOM візуалізується, а до інтерактивних елементів підключаються слухачі подій.

Об’єднання HTML-коду та даних відбувається на стороні клієнта, що мінімізує участь сервера у процесі відображення даних.

Обмін даними з сервером зазвичай відбувається асинхронно за допомогою API XHR у форматі JSON. Тому сервер залишається ключовим елементом у забезпеченні даними, навіть якщо рендеринг використовується на стороні клієнта.

Додаток був розроблений повністю у вебсередовищі, використовуючи стилі Tailwind, а потім перейшов до стану нативного додатку.

На рисунку 2.5 зображена діаграма послідовностей, що показує взаємодію між користувацьким інтерфейсом, логікою додатка та сервісами бекенду під час виконання.

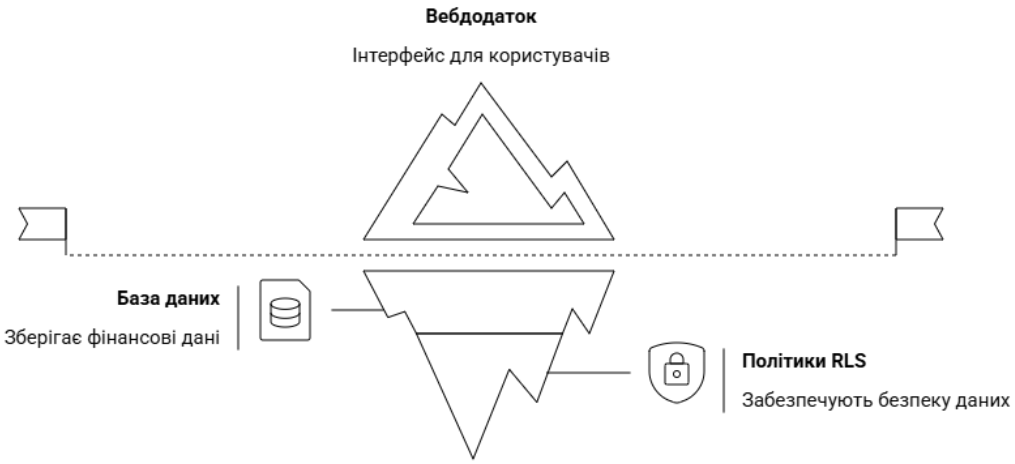


Рисунок 2.2. Архітектура розроблюваної системи

Користувач взаємодіє з інтерфейсом застосунку через сторінки та компоненти React UI, а керування станом здійснюється за допомогою Context API та кастомних hooks.

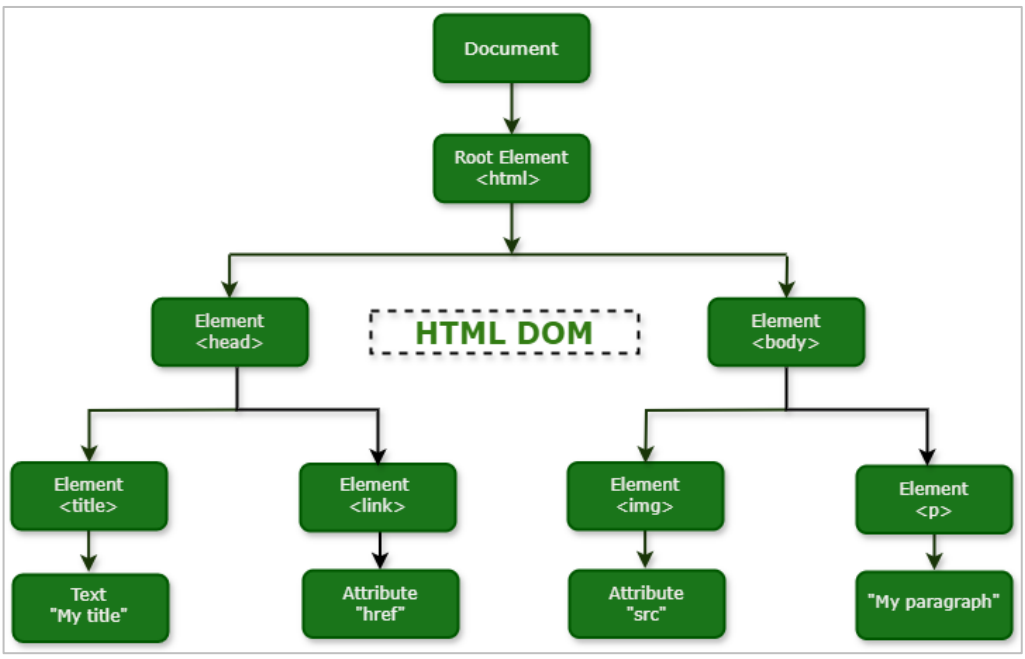


Рисунок 2.3. Деревоподібна структура HTML DOM

Для взаємодії з серверною частиною використовується Supabase Client SDK, який забезпечує доступ до API, авторизації та бази даних PostgreSQL.

Додатково в системі реалізовано виконання автоматизованих фонових процесів і запланованих операцій. Після дії користувача в інтерфейсі React UI відбувається виклик відповідного обробника подій, який передає дані до шару Context та hooks (рис. 2.4). Далі формується HTTP-запит через Supabase Client SDK до серверного API.

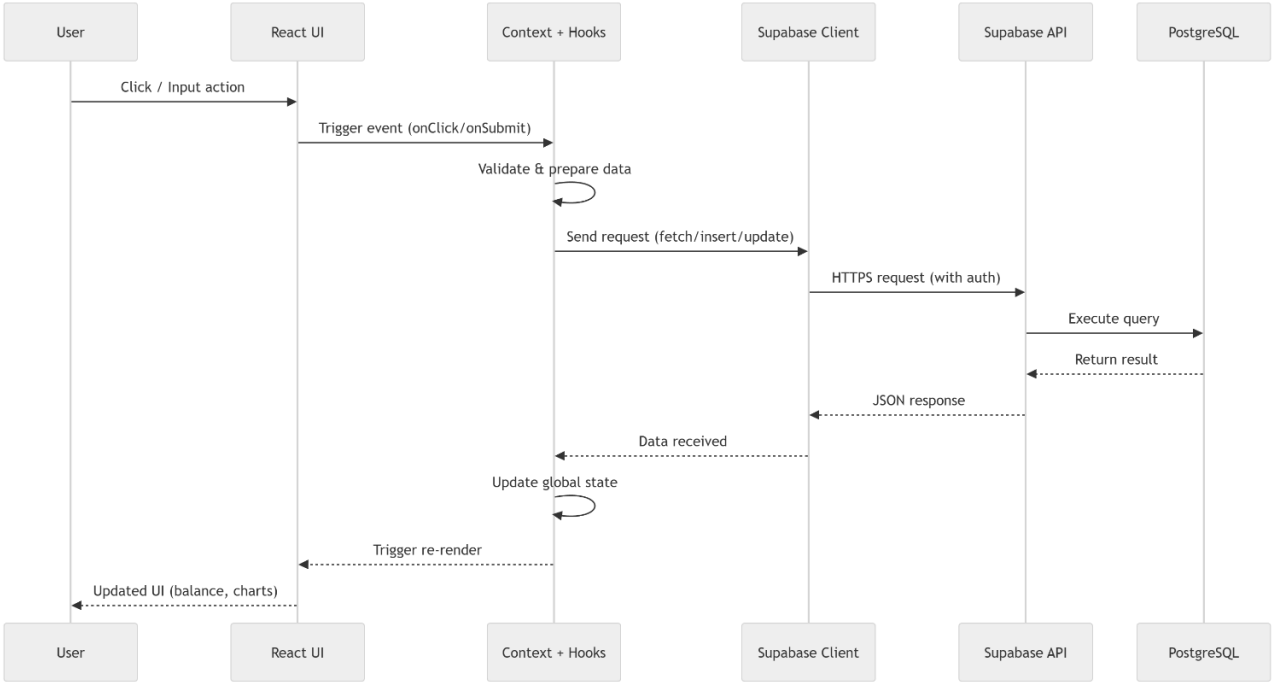


Рисунок 2.5. Діаграма послідовностей UML
(основний алгоритм роботи програми)

Після дії користувача в інтерфейсі React UI відбувається виклик відповідного обробника подій, який передає дані до шару Context та hooks (рис. 2.3). Далі формується HTTP-запит через Supabase Client SDK до серверного API.

Серверна частина виконує обробку запиту та взаємодію з базою даних PostgreSQL, після чого результат повертається у вигляді JSON-відповіді. Отримані дані оновлюють глобальний стан застосунку, що викликає повторний рендер компонентів та оновлення інтерфейсу користувача.

На рисунку 2.6 зображено процес автоматизованого виконання періодичних фінансових операцій у системі. Для реалізації автоматичного додавання

регулярних доходів або транзакцій використовується механізм pg_cron, який запускає заплановані задачі у визначений час.

Через pg_net формується HTTP POST-запит до Edge Function, де здійснюється перевірка наявності відповідних записів у базі даних PostgreSQL. У разі відсутності дублювання створюється нова транзакція та оновлюються фінансові дані користувача.

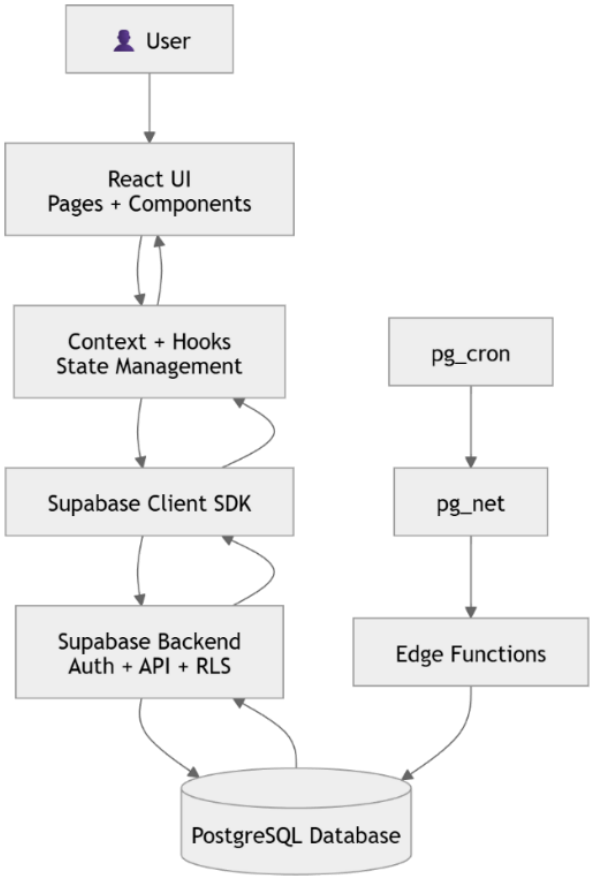


Рисунок 2.4. Діаграма послідовностей UML (архітектура застосунку)

Після синхронізації даних React-застосунок отримує оновлену інформацію через API та відображає актуальний баланс і статистику в інтерфейсі користувача.

Схема автоматизації демонструє обробку на основі подій та за розкладом із використанням cron на рівні бази даних і безсерверних функцій (рис. 2.6).

Запропонована архітектура забезпечує модульність, масштабованість і зручність подальшого супроводу програмного забезпечення. Використання

сучасних вебтехнологій дозволило реалізувати швидкий інтерфейс користувача та ефективну взаємодію із серверною частиною.

Обраний підхід надає можливість подальшого розширення функціональності застосунку без суттєвих змін у загальній структурі системи.

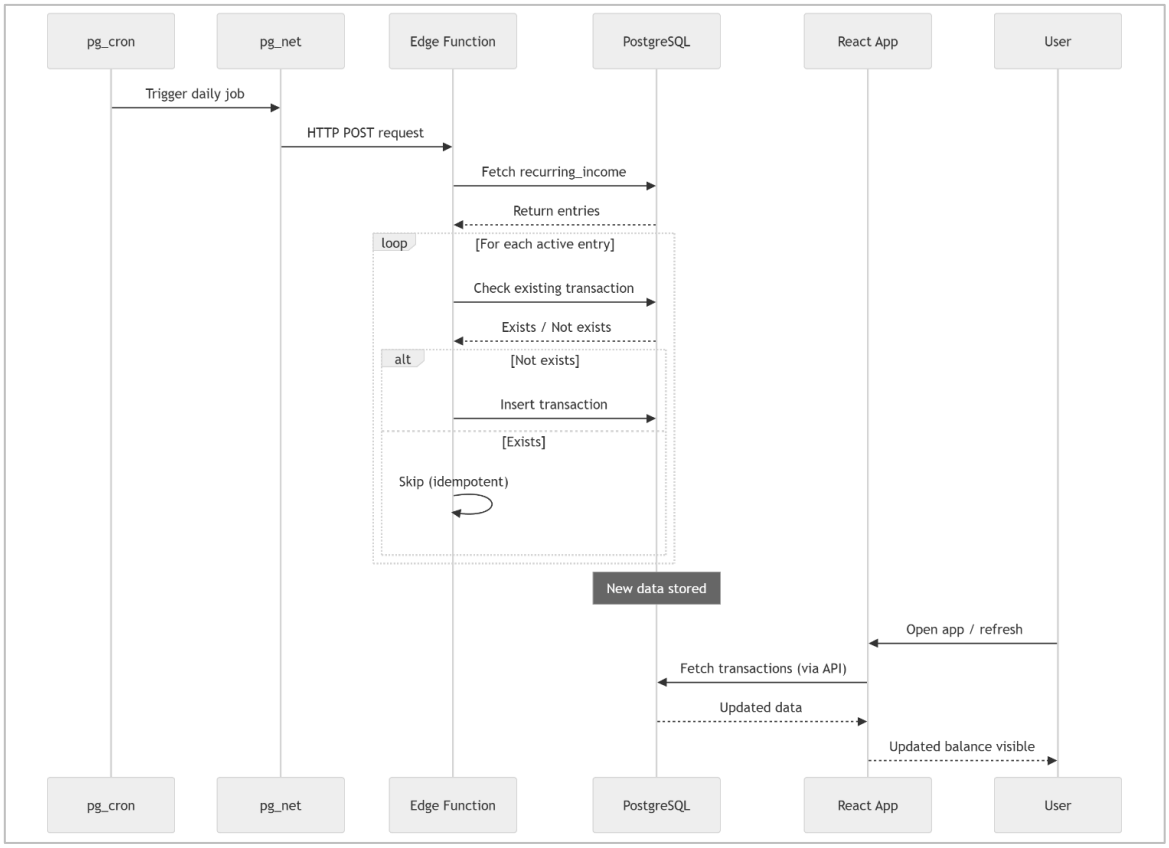


Рисунок 2.6 Діаграма послідовностей UML (потік автоматизації)

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ ПЛАНУВАННЯ СІМЕЙНОГО БЮДЖЕТУ

3.1. Розробка інтерфейсу користувача

Само собою зрозуміло, що інтерфейс користувача – критично важливий елемент будь-якого застосунку для управління сімейним бюджетом. Саме через нього користувач взаємодіє з усіма функціональними можливостями системи.

Якісно розроблений інтерфейс всередині додатку забезпечує інтуїтивність використання, швидкий доступ до необхідної інформації та комфортну роботу з фінансовими даними на пристроях різних розмірів.

При розробці інтерфейсу застосунку використовувався підхід Mobile-First, що передбачає створення дизайну спочатку для найменших екранів з подальшим розширенням функціональності для більших дисплеїв [23]. Оскільки, як було з’ясовано, більшість користувачів взаємодіють з фінансовими додатками саме через мобільні пристрої цей підхід є оптимальним для створюваного PWA-застосунка.

В проєкті було створено інтерфейс для екранів шириною від 320px з поступовим масштабуванням для планшетів та desktop. Також було проведено оптимізацію всіх інтерактивних елементів для touch-управління з мінімальним розміром кнопок 44×44 пікселі згідно з рекомендаціями Apple Human Interface Guidelines [24].

Особлива увага була приділена touch-friendly елементам інтерфейсу. Відстань між сусідніми інтерактивними елементами становить не менше 8 пікселів для запобігання випадкових натискань .

Для оптимальної читабельності тексту на екранах будь-якого розміру без необхідності ручного масштабування типографіка також адаптована для мобільних екранів. Використовується система fluid typography з функцією clamp() для плавного масштабування розмірів шрифтів.

Навігаційна структура застосунку побудована з урахуванням специфіки мобільних інтерфейсів та частоти використання різних функцій. Основою навігації є Bottom Navigation Bar, що розміщується внизу екрану та залишається доступною на всіх сторінках застосунку, частина коду її реалізації показана на рисунку 3.1.

Рис. 3.1. Скріншот коду, що визначає компонент BottomNav (Bottom Navigation Bar)

Додатковим елементом навігації є Budget Switcher – горизонтальна смуга з можливістю перемикання між різними бюджетами. Цей компонент розміщується у верхній частині екрану на сторінках Dashboard, Reports та Savings, дозволяючи користувачу швидко переключатися між сімейними бюджетами в різних валютах:

Dashboard (головний екран) – точка входу в застосунок, що надає швидкий огляд фінансового стану користувача. Екран складається з наступних компонентів:

Balance Card – карточка з відображенням загального балансу, виконана у формі градієнтної картки з білим текстом. Використання градієнта від синього до темно-синього кольору привертає увагу до найважливішої інформації. Розмір шрифту для суми балансу становить 36 пікселів, що забезпечує легку читабельність без необхідності зосередження зору.

Expected Income блок – новий компонент, доданий в рамках реалізації функціональності регулярних доходів. Розміщується безпосередньо під Balance Card і відображає суму очікуваного доходу на поточний місяць на основі налаштованих recurring income entries. Формат відображення: «Очікуваний дохід цього місяця: 25,000 ₴».

Budget Progress Cards – секція з картками прогресу бюджетів. Кожна картка містить назву категорії, іконку, прогрес-бар з відсотком виконання та числову інформацію у форматі «витрачено/ліміт». Прогрес-бар було реалізовано з використанням CSS для плавної анімації:

На головному екрані розташований розділ Recent Transactions – список останніх транзакцій з можливістю переходу до повного списку. Кожна транзакція відображається як окремий рядок з іконкою категорії, описом, сумою та часом її виконання.

Використання «type="text"» замість «type="number"» дозволило повністю контролювати введення та уникнути появи стрілочок збільшення/зменшення значення. Атрибут «inputMode="decimal"» викликає числову клавіатуру на мобільних пристроях, що зручно для введення грошових сум. Варто зазначития, що для встановдення PIN для входу в профіль члена сім'ї натомість використовується атрибут «inputMode="numeric"».

Savings екран представляє собою нову функціональність управління цілями заощаджень. Верхня частина екрану містить карточку Savings з інформацією про загальний дохід поточного місяця, рекомендовану суму заощаджень та залишок після відрахувань на цілі.

Кожна ціль заощаджень відображається у вигляді картки з круговим індикатором прогресу, реалізованим через SVG – формат векторної графіки на основі XML для створення двовимірних зображень, що підтримує інтерактивність та анімацію, приклад використання показано на рисунку 3.2 [25].

Щоб запобігти reflow браузера (процес, під час якого браузер перераховує розмір, положення та геометрію елементів на сторінці) та забезпечити високу частоту кадрів навіть на слабких пристроях для плавності анімацій було використано CSS transform замість зміни position або width/height [26].

```tsx
const SavingsRing = ({ percent, color }: Props) => {
  const r = 28;
  const c = 2 * Math.PI * r;
  const offset = c - (Math.min(percent, 100) / 100) * c;
  return (
    <svg width="72" height="72" viewBox="0 0 72 72">
      <circle cx="36" cy="36" r={r} stroke="hsl(var(--secondary))" strokeWidth="6" fill="none" />
      <circle
        cx="36"
        cy="36"
        r={r}
        stroke={color}
        strokeWidth="6"
        fill="none"
        strokeLinecap="round"
        strokeDasharray={c}
        strokeDashoffset={offset}
        transform="rotate(-90 36 36)"
      />
      <text x="36" y="40" textAnchor="middle" className="fill-foreground" fontSize="13" fontWeight="600">
        {Math.round(percent)}%
      </text>
    </svg>
  );
};
```

Рис. 3.2. Приклад використання SVG у створенні кругових діаграм для функціоналу заощаджень

3.2. Реалізація функціональних можливостей

Функціональність recurring income дозволяє користувачам налаштувати автоматичне створення транзакцій доходу в певний день кожного місяця. Це особливо корисно для зарплат, пенсій, орендних платежів та інших регулярних надходжень.

Політики Row Level Security забезпечують, що користувачі можуть переглядати та змінювати лише регулярні доходи тих бюджетів, до яких вони мають доступ.

Автоматичне створення транзакцій реалізовано через комбінацію Supabase Edge Function та pg_cron. Edge функція виконує бізнес-логіку перевірки та створення транзакцій. Щоб запобігти дублюванню транзакцій при повторному запуску функції вона містить у собі ідемпотентну логіку, що перевіряє наявність вже створеної транзакції для даного recurring entry на поточну дату.

Планування виконання Edge функції здійснюється через pg_cron. Налаштування виконується безпосередньо використовуючи мову програмування SQL.

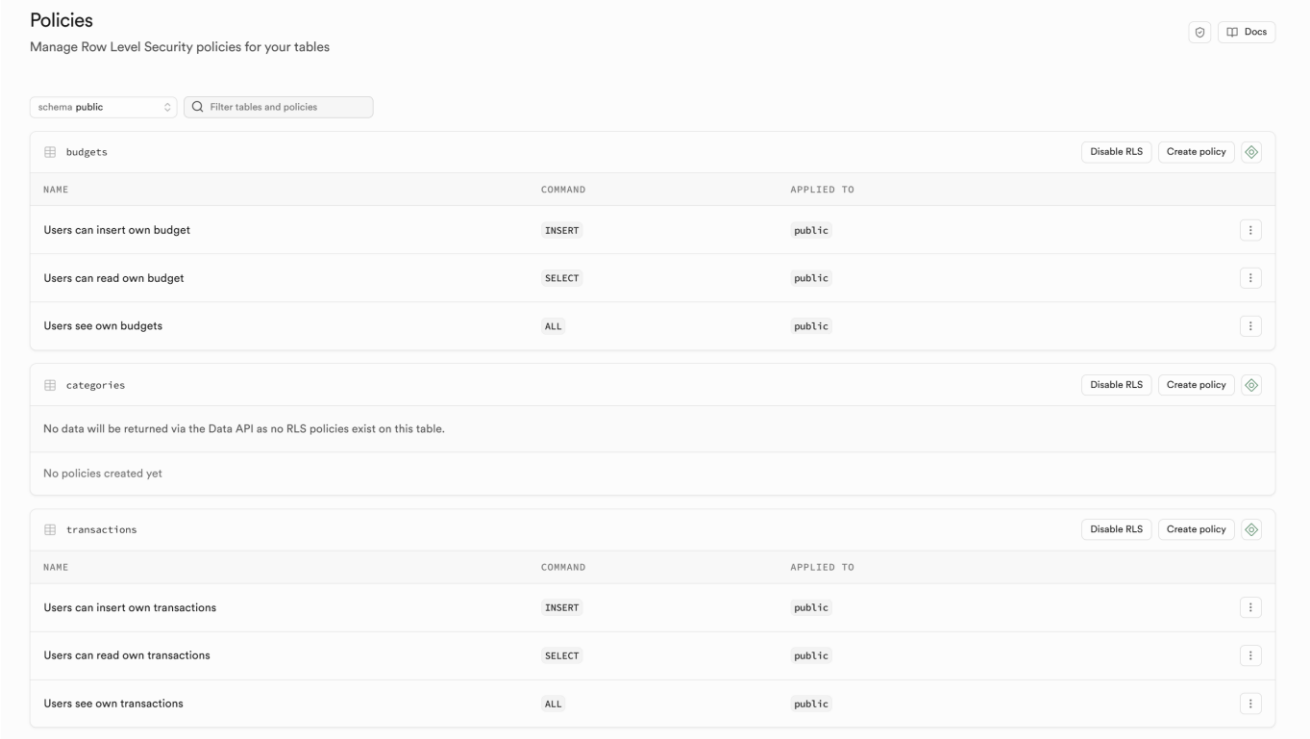


Рис. 3.2. Реалізовані політики Row Level Security (Policies) всередині застосунку

Supabase Cron – це модуль Postgres, який спрощує планування періодичних завдань із використанням синтаксису cron та моніторинг їх виконання безпосередньо в Postgres.

Завдання Cron можна створювати за допомогою SQL або через інтерфейс «Інтеграції → Cron» на панелі управління; їх можна запускати з періодичністю від кожної секунди до одного разу на рік, залежно від потреб користувача [27].

UI компонент для управління recurring income реалізовано у вигляді списку на екрані Settings з можливістю додавання нових записів через bottom sheet. Кожен запис recurring income відображається як картка з можливістю перемикання активності, редагування та видалення. На Dashboard екрані додано відображення очікуваного доходу поточного місяця, що розраховується як сума всіх активних recurring income.

Функціональність savings goals дозволяє користувачам встановлювати та відстежувати цілі заощаджень двох типів: відсоткові (percentage) та фіксовані (fixed). Відсоткові цілі автоматично розраховують цільову суму на основі доходу

поточного місяця, тоді як фіксовані цілі мають статичну цільову суму. Для відсоткових цілей поле value містить відсоток від доходу, для фіксованих – цільову суму в валюті бюджету. Поле current_amount використовується для відстеження досягнутого прогресу.

Функціональність мультивалютності дозволяє користувачам створювати окремі бюджети в різних валютах та легко перемикатися між ними. Це особливо корисно для сімей, що мають доходи чи витрати в декількох валютах, або користувачів, що подорожують. Українська гривня отримує пріоритетний статус і відображається першою в селекторі валют.

Всі компоненти, що відображають грошові суми, використовують currentBudget.currency для форматування. Це забезпечує консистентність відображення фінансових даних по всьому застосунку та автоматичне оновлення при перемиканні між бюджетами різних валют.

3.3. Практичне використання та впровадження

Завантажуючи мобільний додаток для ведення особистого бюджету, користувач потрапляє на початкову сторінку входу в акаунт (рис. 3.3). Тут він має можливість увійти у вже створений акаунт або натиснути кнопку реєстрації, яка переміщує його безпосередньо на екран реєстрації.

Інтерфейс початкового екрану виконаний у мінімалістичному стилі для забезпечення швидкої та зрозумілої взаємодії із системою.

Вигляд екрану реєстрації майже ідентичний до екрану входу у вже створений обліковий запис. Такий підхід дозволяє зберегти єдину стилістику інтерфейсу та спростити навігацію для користувача. Під час реєстрації нового користувача системи, окрім паролю та електронної пошти, додатково тільки потрібно вписати ім’я першого члена сім’ї всередині загального акаунту. За основу було взято принцип роботи спільного використання акаунта стримінгового сервісу Netflix, як показано на рисунку 3.4.

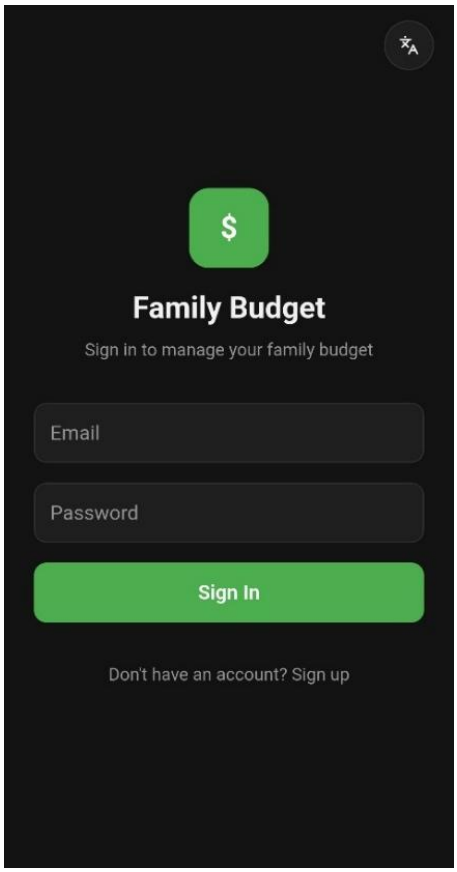


Рисунок 3.3. Початковий екран розроблюваного застосунку

Після авторизації завантажується вікно з можливістю створення та вибору профілю учасника сім'ї – декілька користувачів можуть працювати в межах одного сімейного бюджету, зберігаючи при цьому індивідуальні налаштування (рис. 3.5).

Рисунок 3.4. Обліковий запис родини стримінгового сервісу Netflix

Щоб частково обмежити доступ до окремих профілів у межах одного облікового запису, користувач має можливість створювати до п'яти членів сім'ї та

за бажанням ставити чотирьохзначний пароль на вхід у профіль члена сім’ї. Під час видалення одного з індивідуальних профілів усі грошові операції які були ним проведені теж видаляються – такий механізм забезпечує цілісність та консистентність даних у системі. Вихід з загального облікового запису теж відбувається у вікні керування профілями (рис. 3.6).

Рисунок 3.5. Вибір профілю в загальному акаунті користувача

Після заповнення даних профілю або вибору вже створеного профілю члена сім’ї завантажується головне вікно з загальною інформацією про обраний гаманець користувача. На головному екрані відображається поточний баланс, список останніх транзакцій та коротка фінансова статистика. Також тут доступний перехід у розділи «Add», «Saving», «Reports», «Settings», що відображається на панелі навігації в нижній частині екрану (рис.3.7).

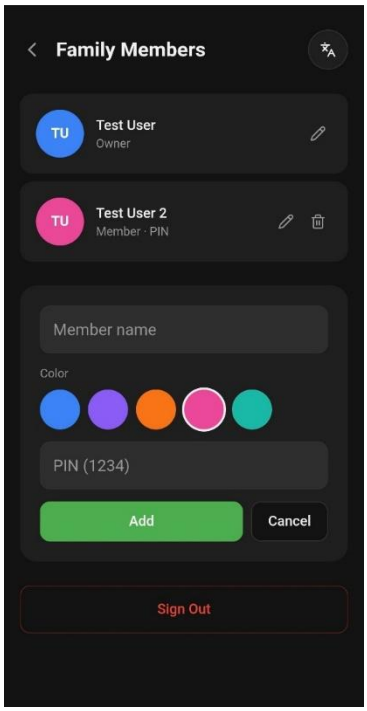


Рисунок 3.6. Вікно керування профілями

На головній сторінці можна побачити залишки або вже витрачені гроші бюджету, перемикання між відображенням коштів відбувається при натисненні на загальну суму.

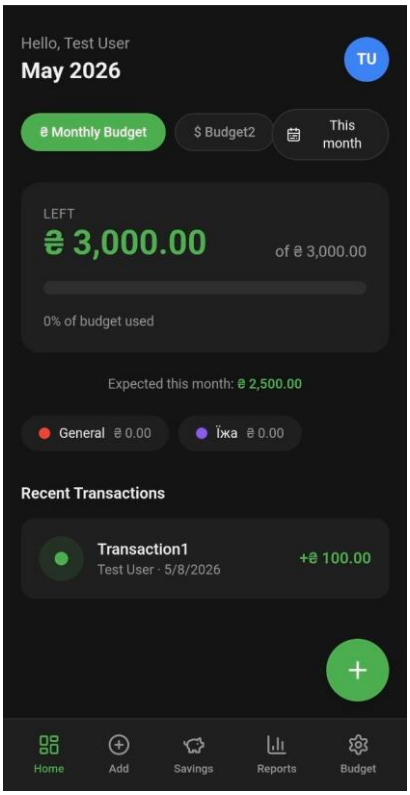


Рисунок 3.7. Головна сторінка застосунку

Під час створення загального акаунту автоматично формується перший бюджет під назвою «Monthly Budget» у розмірі 3000 гривень – користувач може видалити або змінити його згодом у розділі налаштувань.

На сторінці «Головна» у правому нижньому куті екрану також є кнопка швидкого доступу до розділу «Add» у вигляді зеленого знаку плюс. Її використання прискорює створення нових фінансових операцій.

Користувач може редагувати вже проведені транзакції, натиснувши на них в частині «Recent Transactions». Після відкриття транзакції доступна зміна суми, категорії, дати або опису операції у розділі Add, що показаний на рисунку 3.8.

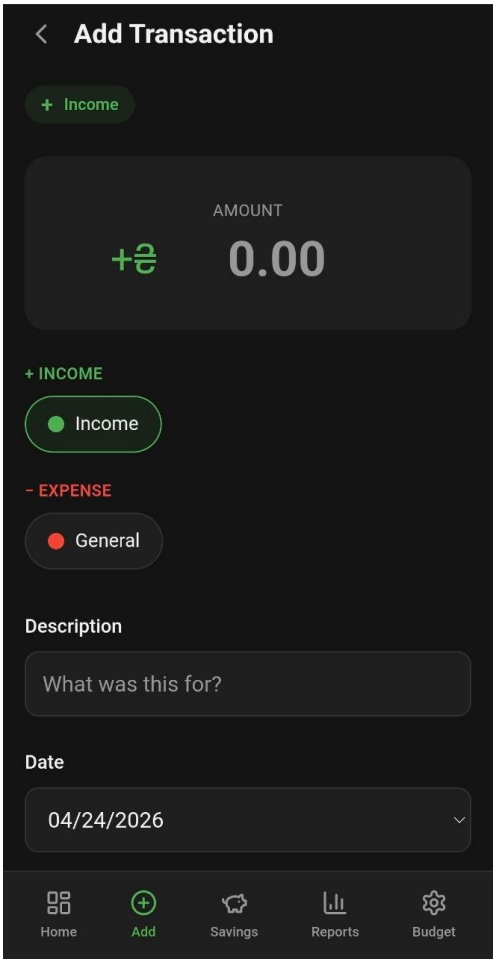


Рисунок 3.8. Розділ «Add» створюваного застосунку

У застосунку також реалізована підтримка декількох мов інтерфейсу. Користувач може змінювати мову в налаштуваннях без необхідності перезапуску додатка, приклад натискання на кнопку показано на рисунку 3.7. Після зміни мови всі елементи інтерфейсу оновлюються автоматично

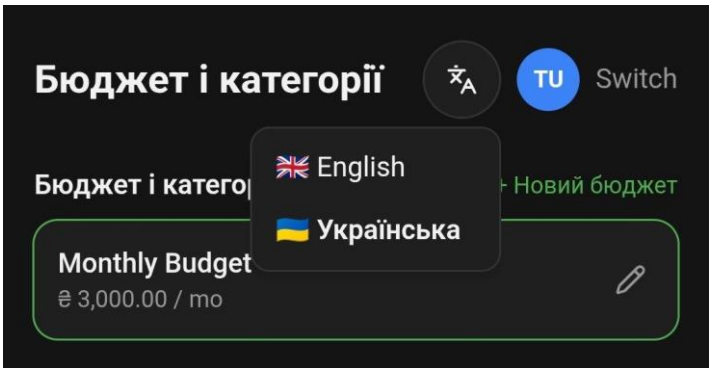


Рисунок 3.9. Скріншот натискання кнопки зміни мови застосунку

Користувач також може обрати період за який буде показано інформацію про бюджет на кожному з розділів, натиснувши на кнопку з вибором часового періоду поруч з списком бюджетів (рис. 3.10). Під час натискання на кнопку з'являється випадаючий список з вже визначеними періодами та вікно «Користувацький період», де користувач власноруч може поставити дати для відображення інформації.

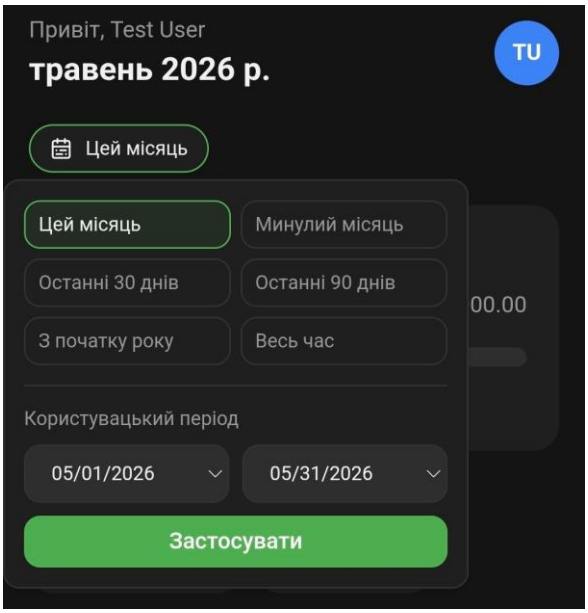


Рисунок 3.10. Скріншот вибору часового періоду для показу інформації

Також користувачі мають можливість користуватися додатком планування сімейного бюджету через браузер власного персонального комп'ютера. Приклад такого використання зображений на рисунку 3.11, у прикладі використовується браузер «Brave» версії 1.90.121 (Chromium: 148.0.7778.96).

Процес використання додатку абсолютно ідентичний в обох варіантах – як у мобільній версії, так і на вебсайті.

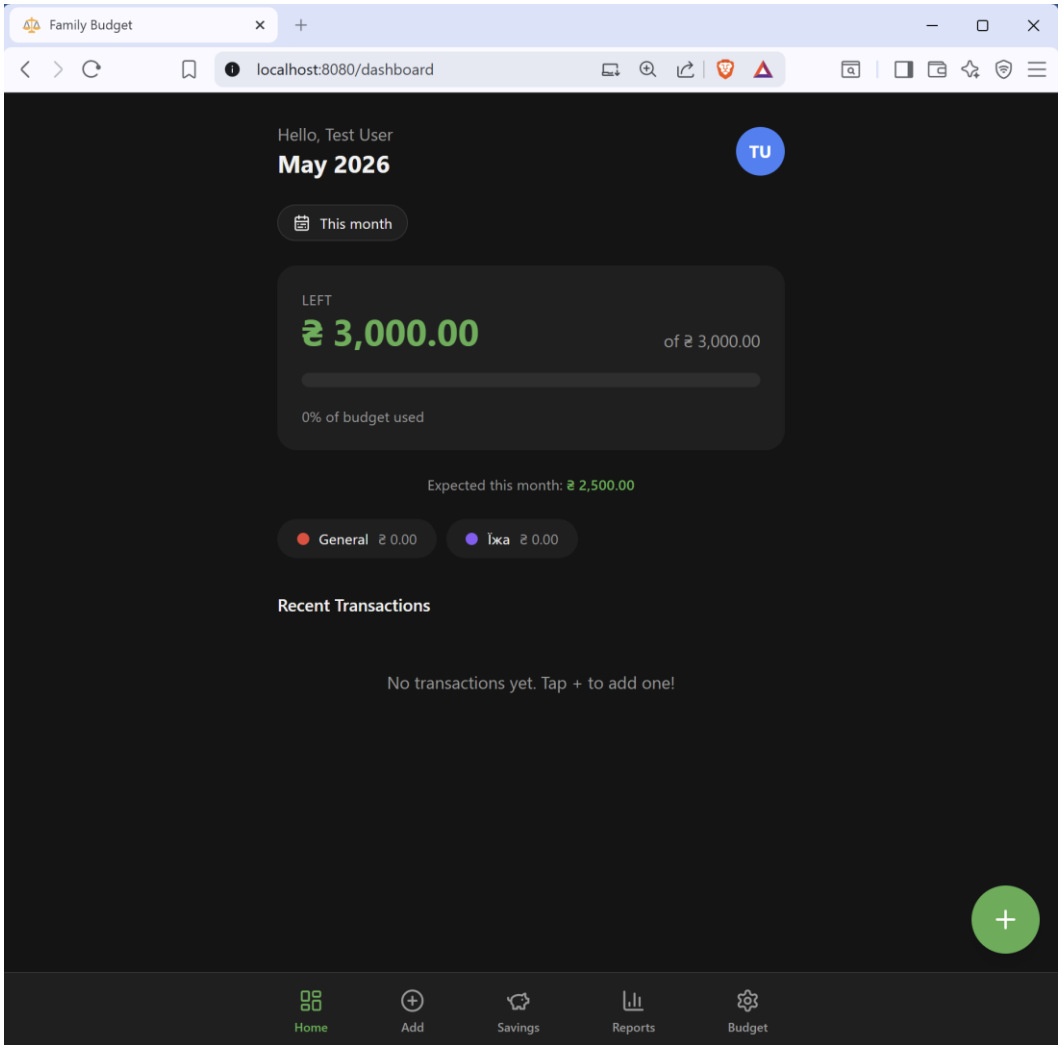


Рисунок 3.11. Використання застосунку у веббраузері комп’ютера

Для контролю роботи системи було використано журналів подій (логів) платформи Supabase. Використання логування дозволило відстежувати виконання запитів до бази даних, процеси авторизації користувачів, виклики Edge Functions, помилки API та результати виконання автоматизованих операцій. Аналіз логів допомагає розробнику оперативно виявляти помилки, перевіряти коректність роботи серверної частини та контролювати стабільність функціонування програмного забезпечення.

Завдяки вбудованим інструментам моніторингу всередині Supabase було забезпечено підвищення надійності системи та спрощено процес подальшого супроводу й оптимізації застосунку.

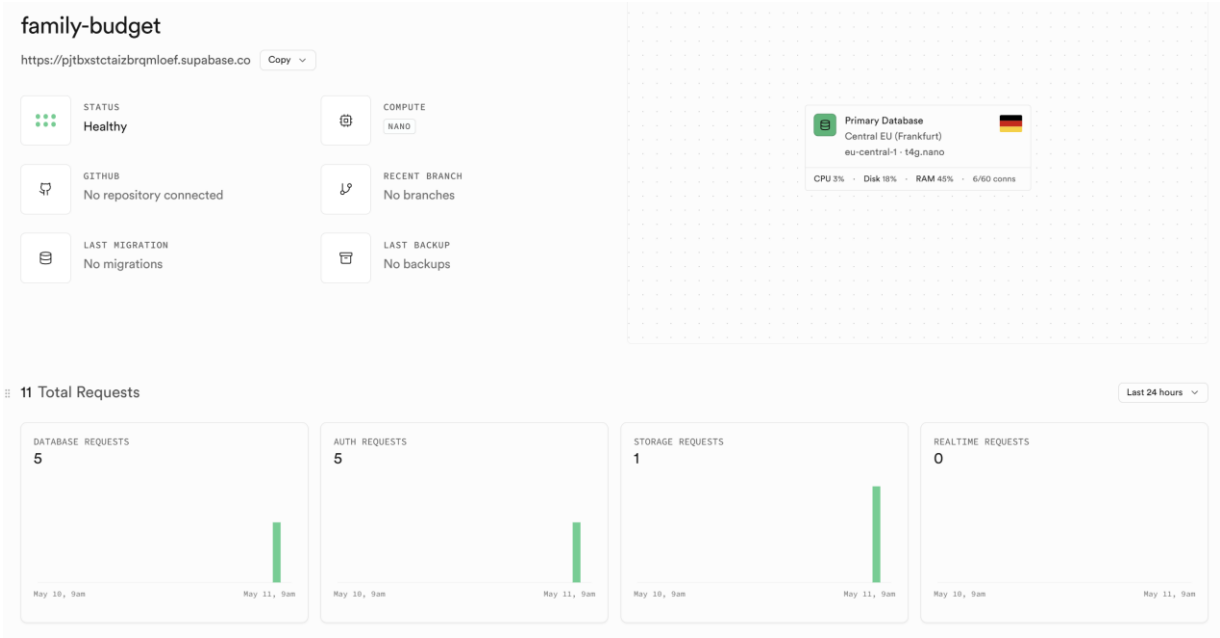


Рисунок 3.12 Project Overview проєкту розроблюваного застосунку для ведення сімейних фінансів на платформі Supabase

Після того, як застосунок готовий, в Android Studio компілюється APK файл, який можна розповсюджувати та встановлювати на Android пристрої. Процес збірки включає оптимізацію ресурсів, перевірку залежностей та генерацію фінального інсталяційного пакета.

Зараз система Android на ринку електроніки стає дедалі популярнішою, особливо на ринку мобільних телефонів [28]. Завдяки відкритому коду частина інструментів розробки є безкоштовною, тому створюється чимало додатків. Процедура розробки та підготовки файлу APK до випуску однакова для всіх додатків і не залежить від моделі поширення, що дозволяє заощадити час і при необхідності автоматизувати ті чи інші процеси.

Як правило, публікація в магазині додатків (наприклад, в Google Play) дозволяє охопити найбільш широку аудиторію. Сервіс ліцензування допомагає розробникам запобігти незаконному встановленню і використанню додатків.

Проте варто зауважити, що за акаунт розробника доведеться заплатити. Але якщо розробник не хоче розміщувати свій продукт в магазині додатків, він може опублікувати його на власному сайті або сервері.

Для цього додаток спочатку готується до публікації, а потім розміщується у вигляді готового файлу APK на сайті з посиланням для його завантаження.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто поставленої мети – розроблено та реалізовано кросплатформний застосунок для планування сімейного бюджету, який дозволяє вести облік доходів і витрат, контролювати фінансовий стан сім’ї, створювати заощадження та автоматизувати регулярні фінансові операції від обличчя декількох користувачів водночас.

В ході написання кваліфікаційної роботи було розглянуто вже існуючі програмні рішення. Проведений аналіз показав, що значна частина наявних застосунків орієнтована переважно на індивідуальне використання або перевантажена функціональністю, яка ускладнює взаємодію користувача із системою. Тому при проєктуванні власного програмного продукту основна увага приділялася простоті інтерфейсу та можливості роботи декількох учасників у межах одного спільного акаунту.

В результаті розробки програмного продукту було створено кросплатформений мобільний та вебзастосунок управління сімейними фінансами. Користувач має можливість ведення обліку власних доходів та витрат, планування бюджету, автоматизації регулярних надходжень, формування заощаджень, працювати з декількома бюджетами та валютами, отримувати аналітичну звітність щодо фінансового стану своєї сім’ї.

У процесі роботи було сформовано функціональні та нефункціональні вимоги до застосунку, які стали основою для подальшої реалізації системи. На основі проведеного аналізу обґрунтовано вибір технологічного стеку, до якого увійшли React, Expo, TypeScript та Supabase.

Використання зазначених технологій дозволило створити гнучку, масштабовану та кросплатформну систему з єдиною кодовою базою для Android, iOS та Web-платформ.

У додатку користувач має можливість обрати мову інтерфейсу: українську або англійську.

Важливим результатом роботи стало створення сучасного інтерфейсу користувача, адаптованого до мобільних пристроїв та орієнтованого на зручність використання.

Для тестування додатку було обрано ручне тестування з використанням як негативного, так і позитивного підходу.

Для введення в експлуатацію, застосунок може бути розміщений на спеціальному віддаленому сервері у вигляді сайту або поширюватися у виді інсталяційного APK файлу та бути розміщеним в магазині додатків.

Перспективи подальших досліджень можуть бути пов’язані з розширенням функціональних можливостей застосунку, зокрема впровадженням інтеграції з банківськими сервісами, підтримкою push-сповіщень, використанням технологій штучного інтелекту для прогнозування витрат, а також реалізацією розширених аналітичних модулів і систем рекомендацій.

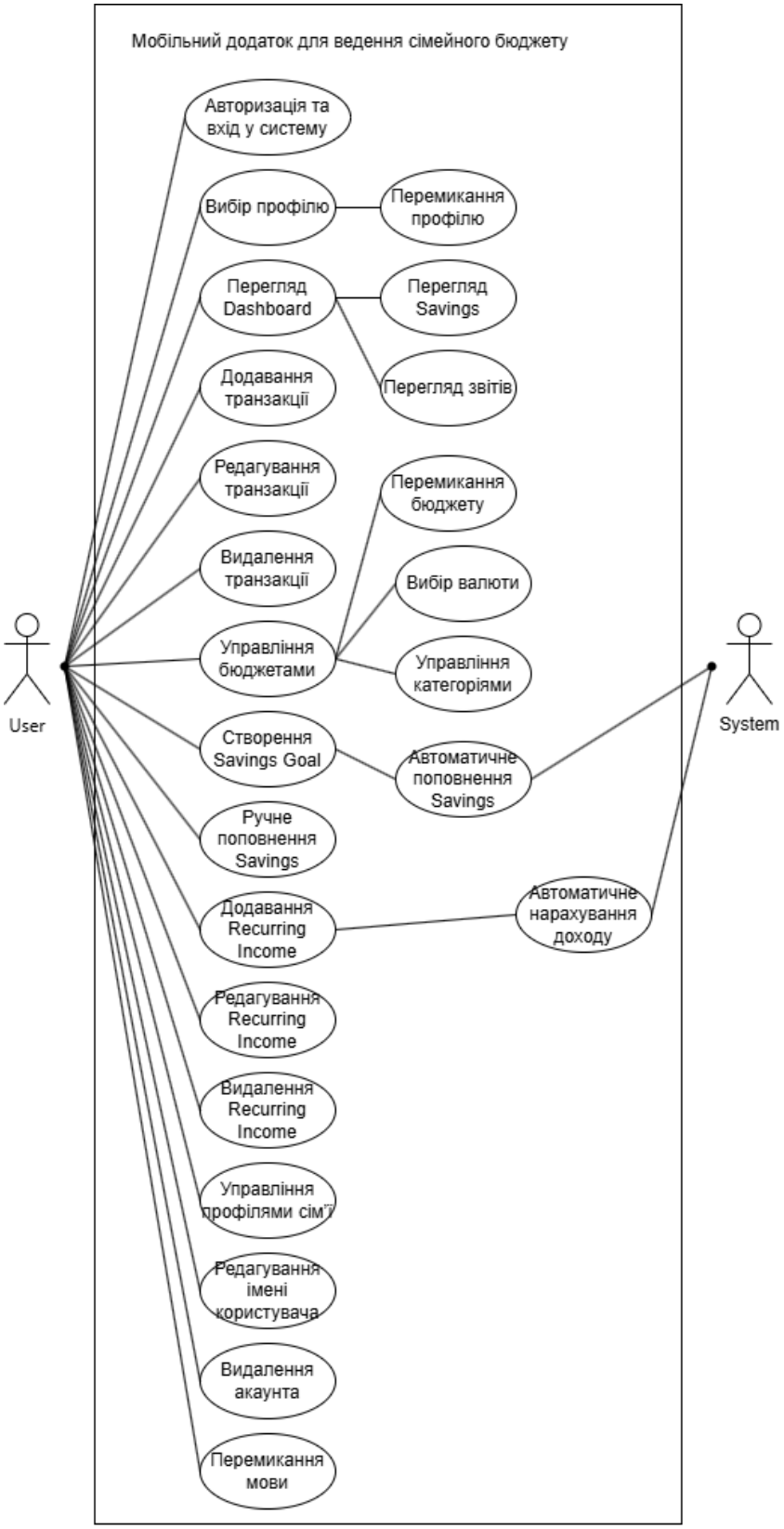
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сачко О. І., Шевцова Н. В. Застосування хмарних технологій у системах управління сімейним бюджетом. Наука, освіта, суспільство очима молодих : матеріали XIX Всеукраїнської науково-практичної конференції здобувачів вищої освіти та молодих учених. м. Рівне, 14 травня 2026 р. Рівне, 2026.
2. Sachko O., Shevtsova N. Banking api integration for the automation of financial accounting. Science and Information Technologies in the Modern World: Collection of Scientific Papers with Proceedings of the 6th International Scientific and Practical Conference. International Scientific Unity. Athens, Greece. April 15-17, 2026. 169-171 p. URL: https://isu-conference.com/en/archive/science-and-information-technologies-in-the-modern-world-15-04-26/
3. OECD/INFE Toolkit for Measuring Financial Literacy and Financial Inclusion / OECD. Paris, 2018. 65 c.
4. Фінансова грамотність, фінансова інклюзія та фінансовий добробут в Україні у 2021 : звіт за результатами дослідження / Info Sapiens ; для DAI Global LLC, Проєкту USAID «Трансформація фінансового сектору». Київ, 2021. URL: https://bank.gov.ua/admin_uploads/article/Research_Financial_Literacy_Inclusion_Welfare_2021.pdf?v=17 (дата звернення: 01.05.2026).
5. Затверджено Національну стратегію розвитку фінансової грамотності до 2030 року. Національний банк України. Новини. 05.07.2024. URL: https://bank.gov.ua/ua/news/all/zatverdjeno-natsionalnu-strategiyu-rozvitku-finansovoyi-gramotnosti-do-2030-roku (дата звернення: 01.05.2026).
6. Google Play. Вікіпедія. 25.11.2025. URL: https://uk.wikipedia.org/wiki/Google_Play (дата звернення: 01.05.2026).
7. Monobank. Статистика. URL: https://monobank.ua/dashboard (дата звернення: 01.05.2026).
8. Monokarta. Монобанк Кешбек – Як Отримати, Використати і Вивести. URL: https://www.monokarta.pp.ua/p/cashback.html (дата звернення: 01.05.2026).

9. Lucidchart. UML Use Case Diagram Tutorial URL: https://www.lucidchart.com/pages/tutorial/uml-use-case-diagram (дата звернення: 01.05.2026).
10. Wieruch R. The Road to React. Leanpub, 2024. 410 с.
11. DOU. Робота. Вакансії. URL: https://jobs.dou.ua/vacancies/?category=Front+End&search=react (дата звернення: 01.05.2026).
12. Vite. URL: https://vite.dev/ (дата звернення: 01.05.2026).
13. Firebase. Documentation. Authentication. URL: https://firebase.google.com/docs/auth (дата звернення: 01.05.2026).
14. Supabase. Supabasedocs. Auth. URL: https://supabase.com/docs/guides/auth (дата звернення: 01.05.2026).
15. PostgREST Documentation – PostgREST 14 documentation. URL: https://docs.postgrest.org/en/v14/ (дата звернення: 01.05.2026).
16. I18next. I18next documentation. URL: https://www.i18next.com/ (дата звернення: 01.05.2026).
17. Capacitorjs. Capacitor Docs. https://capacitorjs.com/docs (дата звернення: 01.05.2026).
18. Жученко А. І., Ярощук Л. Д. Проєктування інформаційних систем бази даних. 2-ге вид. Київ : КПІ ім. Ігоря Сікорського, 2022. 166 с.
19. Kirshteyn M. Mastering 3NF. A Comprehensive Guide to Efficient Relational Database Design.. Independently published, 2024. 272 с.
20. Wroblewski L. Mobile First. New York : A book apart, 2011. 130 с.
21. Emmit A. Scott Jr. SPA Design and Architecture. Manning, 2015. 275 с.
22. HTML - Document Object Model. Tutorialspoint. Html. URL: https://www.tutorialspoint.com/html/html_dom.htm (дата звернення: 01.05.2026).
23. Huang Z., Benyoucef M. A systematic literature review of mobile application usability: addressing the design perspective. Univ Access Inf Soc 22. 2023. Т. 22. С. 715–735. URL: https://doi.org/10.1007/s10209-022-00903-w.

24. Human Interface Guidelines. Apple. Developer. URL: https://developer.apple.com/design/human-interface-guidelines/ (дата звернення: 01.05.2026).
25. Svgrepo. Svgrepo. URL: https://www.svgrepo.com/ (дата звернення: 11.05.2026).
26. Premium Frontend. What Triggers Browser Reflow and How To Reduce It ?. Medium. Interview Series. 01.12.2025. URL: https://medium.com/interview-series/what-triggers-browser-reflow-and-how-to-reduce-it-3c94bd3b4d53 (дата звернення: 01.05.2026).
27. Cron. Supabase DOCS. URL: https://supabase.com/docs/guides/cron (дата звернення: 01.05.2026).
28. Qadir S., Mahmood R. Android vs iOS Statistics 2026: Users, Revenue, and Global Trends. Tekrevol. App Development. 31.03.2026. URL: https://www.tekrevol.com/blogs/android-vs-ios-statistics/?utm_source=chatgpt.com (дата звернення: 01.05.2026).

ДОДАТКИ
ДОДАТОК А
UML Use Case Diagram



ДОДАТОК Б
Фрагмент головної конфігурації React-застосунку та системи маршрутизації

```
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { BrowserRouter, Route, Routes, Navigate } from "react-router-dom";
import { Toaster } from "@/components/ui/toaster";
import { TooltipProvider } from "@/components/ui/tooltip";
import { AuthProvider, useAuth } from "@/contexts/AuthContext";
import { ProfileProvider, useProfile } from "@/contexts/ProfileContext";
import Auth from "./pages/Auth";
import ProfilePicker from "./pages/ProfilePicker";
import PinEntry from "./pages/PinEntry";
import Dashboard from "./pages/Dashboard";
import AddTransaction from "./pages/AddTransaction";
import BudgetCategories from "./pages/BudgetCategories";
import Members from "./pages/Members";
import Reports from "./pages/Reports";
import Savings from "./pages/Savings";
import AppLayout from "./components/AppLayout";
import NotFound from "./pages/NotFound";

const queryClient = new QueryClient();

const ProtectedRoute = ({ children }: { children: React.ReactNode }) => {
  const { user, loading } = useAuth();
  if (loading) return (
    <div className="min-h-screen bg-background flex items-center justify-center">
      <div className="w-8 h-8 border-2 border-primary border-t-transparent rounded-
full animate-spin" />
    </div>
  );
  if (!user) return <Navigate to="/auth" replace />;
  return <>{children}</>;
};

const ProfileRequired = ({ children }: { children: React.ReactNode }) => {
  const { activeProfile } = useProfile();
  if (!activeProfile) return <Navigate to="/profiles" replace />;
  return <>{children}</>;
};

const App = () => (
  <QueryClientProvider client={queryClient}>
    <TooltipProvider>
```

```
      <Toaster />
      <AuthProvider>
        <ProfileProvider>
          <BrowserRouter>
            <Routes>
              <Route path="/auth" element={<Auth />} />
              <Route path="/profiles" element={<ProtectedRoute><ProfilePicker
/></ProtectedRoute>} />
              <Route path="/pin/:memberId" element={<ProtectedRoute><PinEntry
/></ProtectedRoute>} />
              <Route path="/" element={<ProtectedRoute><ProfileRequired><AppLayout
/></ProfileRequired></ProtectedRoute>}>
                <Route index element={<Navigate to="/dashboard" replace />} />
                <Route path="dashboard" element={<Dashboard />} />
                <Route path="add" element={<AddTransaction />} />
                <Route path="edit/:id" element={<AddTransaction />} />
                <Route path="budget" element={<BudgetCategories />} />
                <Route path="savings" element={<Savings />} />
                <Route path="reports" element={<Reports />} />
              </Route>
              <Route path="/members" element={<ProtectedRoute><Members
/></ProtectedRoute>} />
              <Route path="*" element={<NotFound />} />
            </Routes>
          </BrowserRouter>
        </ProfileProvider>
      </AuthProvider>
    </TooltipProvider>
  </QueryClientProvider>
);

export default App;
```

ДОДАТОК В
Реалізація системи авторизації користувачів із використанням Supabase Authentication

```
import React, { createContext, useContext, useEffect, useState } from "react";
import { Session, User } from "@supabase/supabase-js";
import { supabase } from "@/integrations/supabase/client";

interface AuthContextType {
  session: Session | null;
  user: User | null;
  loading: boolean;
  signUp: (email: string, password: string, displayName: string) => Promise<void>;
  signIn: (email: string, password: string) => Promise<void>;
  signOut: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children })
=> {
  const [session, setSession] = useState<Session | null>(null);
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const { data: { subscription } } = supabase.auth.onAuthStateChange((_event, session)
=> {
      setSession(session);
      setUser(session?.user ?? null);
      setLoading(false);
    });

    supabase.auth.getSession().then(({ data: { session } }) => {
      setSession(session);
      setUser(session?.user ?? null);
      setLoading(false);
    });

    return () => subscription.unsubscribe();
  }, []);

  const signUp = async (email: string, password: string, displayName: string) => {
    const redirectUrl = `${window.location.origin}/profiles`;
    const { error } = await supabase.auth.signUp({
```

```
      email,
      password,
      options: {
        data: { display_name: displayName },
        emailRedirectTo: redirectUrl,
      },
    });
    if (error) throw error;
  };

  const signIn = async (email: string, password: string) => {
    const { error } = await supabase.auth.signInWithPassword({ email, password });
    if (error) throw error;
  };

  const signOut = async () => {
    const { error } = await supabase.auth.signOut();
    if (error) throw error;
  };

  return (
    <AuthContext.Provider value={{ session, user, loading, signUp, signIn, signOut }}>
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context) throw new Error("useAuth must be used within AuthProvider");
  return context;
};
```

ДОДАТОК Г

Реалізація отримання фінансових транзакцій із бази даних за визначений часовий період

```
import { useEffect, useState, useCallback } from "react";
import { supabase } from "@/integrations/supabase/client";
import { Tables } from "@/integrations/supabase/types";

export type TransactionWithRelations = Tables<"transactions"> & {
  category: Tables<"categories">;
  member: Tables<"family_members">;
};

interface Opts {
  limit?: number;
  withRelations?: boolean;
}

export const useRangeTransactions = (
  budgetId: string | null,
  start: string,
  end: string,
  opts: Opts = {},
) => {
  const [transactions, setTransactions] = useState<any[]>([]);
  const [loading, setLoading] = useState(false);

  const refresh = useCallback(async () => {
    if (!budgetId) {
      setTransactions([]);
      return;
    }
    setLoading(true);
    let query = supabase
      .from("transactions")
      .select(opts.withRelations ? "*, category:categories(*),
member:family_members(*)" : "*")
      .eq("budget_id", budgetId)
      .gte("date", start)
      .lte("date", end)
      .order("date", { ascending: false })
      .order("created_at", { ascending: false });
    if (opts.limit) query = query.limit(opts.limit);
    const { data } = await query;
```

```
    setTransactions(data || []);
    setLoading(false);
  }, [budgetId, start, end, opts.limit, opts.withRelations]);

  useEffect(() => {
    refresh();
  }, [refresh]);

  return { transactions, loading, refresh };
};
```

ДОДАТОК Д
Код налаштування i18n локалізації

```
import i18n from "i18next";
import { initReactI18next } from "react-i18next";
import LanguageDetector from "i18next-browser-languagedetector";
import en from "./locales/en.json";
import uk from "./locales/uk.json";

i18n
  .use(LanguageDetector)
  .use(initReactI18next)
  .init({
    resources: {
      en: { translation: en },
      uk: { translation: uk },
    },
    fallbackLng: "en",
    supportedLngs: ["en", "uk"],
    interpolation: { escapeValue: false },
    detection: {
      order: ["localStorage", "navigator"],
      caches: ["localStorage"],
      lookupLocalStorage: "lang",
    },
  });

export default i18n;
```

ДОДАТОК Е
Механізм автоматичної побудови початкової фінансової структури користувача через trigger-функцію

```
-- Trigger function: auto-create family, budget, categories on signup
CREATE OR REPLACE FUNCTION public.handle_new_user()
RETURNS TRIGGER
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = public
AS $$
DECLARE
  new_family_id UUID;
  new_budget_id UUID;
BEGIN
  -- Create family
  INSERT INTO public.families (name, owner_id)
  VALUES ('My Family', NEW.id)
  RETURNING id INTO new_family_id;

  -- Create owner as first member
  INSERT INTO public.family_members (family_id, user_id, display_name,
avatar_color, role)
  VALUES (new_family_id, NEW.id, COALESCE(NEW.raw_user_meta_data-
>>'display_name', 'Owner'), '#3B82F6', 'owner');

  -- Create default budget
  INSERT INTO public.budgets (family_id, name, monthly_limit, currency)
  VALUES (new_family_id, 'Monthly Budget', 3000, 'USD')
  RETURNING id INTO new_budget_id;

  -- Create starter categories
  INSERT INTO public.categories (budget_id, name, type, color, icon) VALUES
    (new_budget_id, 'Income', 'income', '#4CAF50', 'trending-up'),
    (new_budget_id, 'General', 'expense', '#f44336', 'shopping-bag');

  RETURN NEW;
END;
$$;
```