МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Державний університет інтелектуальних технологій і зв’язку

|  |  |
| --- | --- |
| Факультет | Інформаційних технологій та кібербезпеки |
| Кафедра | Інженерії програмного забезпечення |

**БУХТА Микита Миколайович**

**КВАЛІФІКАЦІЙНА (БАКАЛАВРСЬКА) РОБОТА**

з теми

**Розробка додатку для контролю та аналізу фінансових ресурсів та витрат (Back-end та архітектура додатку)**

|  |  |
| --- | --- |
| Спеціальність | F2 Інженерія програмного забезпечення |
|  |  |
| Освітня програма | Інженерія програмного забезпечення |
|  |  |
| Керівник | Калініна Т.О., посада керівника |

ОДЕСА – 2025

**ЗМІСТ**

Міністерство освіти і науки України

Державний університет інтелектуальних технологій і зв’язку

|  |
| --- |
| Факультет інформаційних технологій та кібербезпеки |
| Кафедра інженерії програмного забезпечення |

|  |  |
| --- | --- |
| Рівень вищої освіти | перший (бакалаврський) |
| Спеціальність | F2 Інженерія програмного забезпечення |
| Освітня програма | Інженерія програмного забезпечення |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ЗАТВЕРДЖУЮ | | | | | | | | | | |
|  | Завідувач кафедри | | | | | | | | | | |
|  | |  | | | Матін ГАДЖИЕВ | | | | | | |
|  | | (підпис) | | |  |  | | | | | |
|  | |  |  |  | | |  | 20 |  | р. |  | |

**ЗАВДАННЯ**

**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

|  |
| --- |
| ПІБ (родовий) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. Тема роботи: |  | | | | | | | | | | | | | | | | | |
|  |  | | | | | | | | | | | | | | | | | |
| Керівник роботи: | | |  | | | | | | | | | | | | | | | |
| затверджені наказом ректора від | | | |  | |  |  |  | |  | | р. | | № | |  | |  | | |
|  | | | |  | |  | | |  | |  | |  | |  | |  | | |
|  | | | |  | |  | | |  | |  | |  | |  | |  | | |
| 2. Зміст роботи: | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
|  | |  | | | | | | | | | | | | | | | | |
| 3. Консультанти розділів роботи | | | | |  | | | | | | | | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада  консультанта | Підпис, дата | |
| завдання видав | завдання  прийняв |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4. Дата видачі завдання |  |  |  |  |  | р. |  |  |  |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №№ п/п | Назва етапів кваліфікаційної роботи | Строк виконання | Примітка |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Здобувач вищої освіти** |  |  |  |  |  |
|  |  | (підпис) |  | (прізвище та ініціали) |  | |
| **Керівник роботи** |  |  |  |  |  | |
|  |  | (підпис) |  | (прізвище та ініціали) |  | |

**Петров О.О.** Тема роботи [кваліфікаційна (бакалаврська) робота зі спеціальності F2 Інженерія програмного забезпечення; ОПП «Інженерія програмного забезпечення»] / наук. кер.: посада, звання Петров О.О.; Державний університет інтелектуальних технологій і зв’язку. Одеса: ДУІТЗ, 2025. 52 с.

**Анотація**

Текст

**Ключові слова**: хоча б 4-5

**Petrov O.O.** Тема англійською [qualification (bachelor's) thesis in the speciality F2 Software engineering; EPP «Software engineering»] / research supervisor: посада та звання англ Petrov O.O.; State University of Intellectual Technologies and Communications. Odesa: SUITC, 2025. 52 p.

**Abstract**

Текст англ

**Keywords:** англ 4-5 слів

**ВСТУП**

З кожним роком зростає кількість цифрових фінансових операцій, а разом з цим і потреба в ефективних інструментах для контролю особистих витрат та доходів. Сьогодні на ринку є багато програм для управління фінансами — від великих систем на зразок Mint до простіших застосунків для ведення бюджету. Проте більшість із них або надмірно складні для звичайного користувача, або обмежені за функціоналом.

Проєкт BudgetBee створений для того, щоб заповнити цю прогалину. Його головна мета — стати простим та інтуїтивним інструментом для обліку фінансів, що дозволяє користувачам не лише фіксувати транзакції, але й аналізувати свої витрати через зручну систему звітності.

Для реалізації цього завдання буде використано сучасні технології: мова програмування C++, фреймворк Qt для створення інтерфейсу та SQLite як основну базу даних. Проєкт побудований за архітектурою MVVM (Model-View-ViewModel), що забезпечує чітке розділення логіки та представлення даних.

Розробка BudgetBee включає кілька ключових етапів:

* Спочатку необхідно дослідити ринок фінансових додатків, виділити основні функції та зрозуміти потреби цільової аудиторії.
* Наступним кроком стане створення архітектури системи, де головними компонентами виступатимуть модулі для роботи з транзакціями, категоріями витрат та звітами.
* Реалізація API для взаємодії між бекендом та фронтендом дозволить користувачам отримувати актуальну інформацію у зручному форматі.
* Окрему увагу буде приділено тестуванню — як функціональному, так і продуктивному. За допомогою Google Benchmark буде проаналізовано ефективність роботи системи, а GTest допоможе виявити потенційні помилки.

Таким чином, предметом дослідження виступає саме процес розробки програмного забезпечення, яке надає можливість не лише зберігати фінансові дані, але й перетворювати їх на корисну інформацію для прийняття зважених рішень.

**1. АНАЛІЗ СУЧАСНИХ ІНСТРУМЕНТІВ ДЛЯ АВТОМАТИЗОВАНОЇ ОБРОБКИ ФІНАНСОВИХ ДАНИХ**

**1.1 Опис предметної області та актуальність розробки**

Автоматизована обробка фінансових даних є важливою складовою сучасних інформаційних систем, що спрямовані на забезпечення ефективного управління фінансовими потоками підприємства. Основною метою автоматизації фінансової обробки є зменшення часу на обробку даних, підвищення точності розрахунків, мінімізація людських помилок та оптимізація процесів аналізу та звітності.

Фінансові дані охоплюють широкий спектр інформації, яка може включати транзакції, доходи, витрати, кредитні операції, інвестиційні дані та інші фінансові показники. Обробка таких даних зазвичай передбачає виконання складних розрахунків, аналізу тенденцій, виявлення аномалій та формування звітів.

З розвитком цифрових технологій та зростанням обсягів даних, що обробляються, виникла необхідність у створенні програмних продуктів, які здатні ефективно обробляти великі масиви фінансової інформації в реальному часі. Це дозволяє керівникам підприємств приймати своєчасні та обґрунтовані рішення на основі актуальних фінансових показників.

Одним із сучасних інструментів, спрямованих на автоматизацію обробки фінансових даних, є BudgetBee. Основною метою BudgetBee є розробка інтегрованої системи для управління особистими фінансами, що дозволяє користувачам контролювати свої витрати, аналізувати фінансові потоки та прогнозувати майбутні витрати.

Таким чином, актуальність розробки BudgetBee обумовлена необхідністю забезпечення користувачів зручним інструментом для управління особистими фінансами, що сприяє підвищенню фінансової грамотності, зменшенню фінансових втрат та підвищенню ефективності управління особистими коштами.

У сучасних умовах зростаюча кількість фінансових транзакцій, які здійснює середньостатистичний користувач, ускладнює процес контролю за особистими фінансами. Ведення витрат та доходів вручну, використання таблиць та нотаток — це трудомісткий процес, який часто призводить до помилок, пропусків важливої інформації та втрати часу.

Автоматизовані інструменти для обробки фінансових даних, такі як BudgetBee, спрямовані на систематизацію та впорядкування фінансових записів користувача. На відміну від складних бухгалтерських систем, що інтегруються з банківськими рахунками та зовнішніми системами, BudgetBee працює виключно з даними, які вводить сам користувач. Це дозволяє уникнути проблем з конфіденційністю та залежністю від банківських API.

Актуальність розробки BudgetBee обумовлена необхідністю створення простого, але ефективного інструменту для:

Систематизації особистих витрат та доходів у зрозумілому для користувача форматі;

Автоматичного підрахунку загальної суми витрат за період, категоріями або окремими статтями;

Виявлення надмірних витрат та формування аналітичних звітів на основі наявних даних;

Формування прогнозів майбутніх витрат та доходів на основі історичних записів;

Відстеження залишків за кожною категорією витрат для уникнення перевитрат.

Таким чином, BudgetBee не тільки спрощує процес обліку фінансових операцій, але й підвищує фінансову дисципліну користувача, дозволяючи швидко аналізувати власні витрати та доходи. Це особливо актуально для користувачів, які прагнуть краще розуміти власний фінансовий стан без складних систем бухгалтерського обліку та інтеграції з банківськими сервісами.

**1.2 Порівняльна характеристика існуючих аналогів розроблюваної системи**

На сучасному ринку фінансових додатків існує велика кількість інструментів, спрямованих на обробку та аналіз фінансових даних. Однак, більшість з них орієнтовані на загальний облік витрат та доходів без детальної категоризації та без можливості відстеження змін цін на конкретні товари. У цьому розділі розглянемо основні інструменти та їх відмінності від BudgetBee, який реалізує мульти-категорійну систему, аналіз цінових змін та мульти-аккаунтинг. Порівняння з конкурентами відображено на таблиці 1.2.1.

**QuickBooks**

QuickBooks — це багатофункціональна платформа для обліку фінансів, яка переважно орієнтована на бізнес-користувачів. Вона забезпечує загальний облік витрат та доходів, проте не має системи детальної категоризації товарів (наприклад, продукти → м'ясо → телятина → стегно). Також відсутня можливість автоматичного аналізу змін цін на одиницю товару.

**Xero**

Xero пропонує можливість ведення обліку витрат та доходів для малого бізнесу. Вона дозволяє створювати рахунки та контролювати залишки, але не підтримує складну багаторівневу категоризацію витрат та аналіз цінових змін.

**Mint**

Mint орієнтована на управління особистими фінансами та дозволяє контролювати витрати за основними категоріями. Проте її функціональність обмежується лише базовою категоризацією (наприклад, Продукти, Транспорт, Розваги), без можливості створення підкатегорій чи відстеження змін цін за одиницю товару.

**BudgetBee**

BudgetBee поєднує можливості детальної категоризації, аналізу змін цін та мульти-аккаунтингу.

Таблиця 1.2.1 – порівняння з конкурентами.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Функціональність** | **QuickBooks** | **Xero** | **Mint** | **BudgetBee** |
| Мульти-категорійність | ні | ні | ні | так |
| Деталізація витрат | так | так | так | так |
| Аналіз змін цін | ні | ні | ні | так |
| Мульти-аккаунтинг | так | так | ні | так |
| Автоматичних підрахунок залишків | так | так | так | так |
| Формування звітів | так | так | так | так |
| Прогнозування витрат | так | так | так | так |
| Інтеграція з банківськими системами | так | так | так | ні |
| Зберігання даних локально | так | так | так | так |
| Цільова аудиторія | Бізнес | Бізнес | Приватні | Приватні / Малі бізнеси |
| Вартість | Висока | Висока | Безкоштовно | Безкоштовно (частково тим часово) |

**1.3 Постановка мети та задачі**

У сучасному світі фінансова грамотність стає все більш важливою для кожного. Люди прагнуть контролювати свої витрати, відстежувати доходи та краще розуміти, куди йдуть їхні гроші. Однак, коли справа доходить до детального аналізу фінансових даних, більшість інструментів на ринку виявляються або надто складними, або недостатньо гнучкими для індивідуальних потреб користувачів.

Проєкт BudgetBee спрямований на вирішення цієї проблеми. Його головна мета — створити зручний та зрозумілий інструмент для управління особистими фінансами, який дозволяє користувачам не лише відстежувати витрати, але й глибше аналізувати дані за допомогою мульти-категорійності та аналізу цінових змін.

**Основні задачі проєкту:**

* **Зрозуміти, що вже існує на ринку.** Спершу потрібно вивчити інші фінансові додатки, такі як QuickBooks, Xero та Mint. Це дозволить виявити, чого їм бракує у плані мульти-категорійності та аналізу цінових змін. Важливо зрозуміти, які функції є корисними для користувачів, а які — зайві або надто складні.
* **Розробити структуру даних:** Необхідно запланувати систему категорій та підкатегорій, які користувач може налаштовувати під свої потреби. Це дозволить організувати дані таким чином, щоб було зручно аналізувати витрати як за категоріями, так і за конкретними товарами.
* **Створити систему мульти-аккаунтингу:** Потрібно передбачити можливість створення кількох рахунків (готівка, картка, депозит), щоб користувач міг окремо відстежувати залишки за кожним із них та бачити загальний баланс у програмі.
* **Реалізувати алгоритм розрахунку цін за одиницю товару:** Потрібно створити функцію для автоматичного розрахунку середньої ціни за одиницю товару (наприклад, за кілограм або літр). Це дозволить користувачам відслідковувати цінові коливання та прогнозувати можливі зміни витрат.
* **Розробити модуль формування звітів:** Важливо реалізувати систему звітів, яка дозволить користувачам бачити загальні витрати за період, витрати за категоріями, аналіз змін цін та прогнозування витрат на основі історичних даних.
* **Забезпечити зручність використання:** Необхідно розробити інтуїтивний інтерфейс, який буде простим у навігації та не вимагатиме додаткових знань для використання. Кожна функція має бути доступною в кілька кліків.
* **Випробувати систему та оптимізувати її:** Потрібно провести комплексне тестування BudgetBee, щоб переконатися, що всі основні функції працюють коректно та без збоїв. Виявлені проблеми необхідно усунути, а продуктивність — оптимізувати.

Таким чином, реалізація цих задач дозволить створити зручний, гнучкий та зрозумілий інструмент для управління особистими фінансами. BudgetBee допоможе користувачам не лише вести облік витрат, але й бачити реальну картину своїх фінансів, аналізувати зміни цін на товари та планувати свої витрати на майбутнє.

**1.4 Обґрунтування вибору технологій та архітектури BudgetBee**

Коли справа доходить до розробки фінансового додатку, важливо обрати такі технології, які не лише забезпечать стабільну роботу програми, але й дозволять масштабувати її у майбутньому. Вибір технологій для BudgetBee базується на потребах користувачів та специфіці обробки фінансових даних.

Перше, з чого варто почати, — це база даних. Для BudgetBee було вирішено використовувати **SQLite**. Чому саме вона? По-перше, SQLite — це легковажна СУБД, яка не вимагає окремого серверу та складної конфігурації. Вона дозволяє зберігати дані локально, що підвищує рівень конфіденційності. Для фінансових додатків це особливо важливо, адже користувачі не завжди готові віддавати свої фінансові дані у хмару. Крім того, SQLite підтримує транзакції, що дозволяє коректно обробляти зміни у базі даних навіть у випадку збоїв.

Наступний крок — архітектура додатку. Тут обрано **MVVM (Model-View-ViewModel)**. Ця архітектура дозволяє чітко відокремити бізнес-логіку від користувацького інтерфейсу. У випадку BudgetBee це виглядає так: дані зберігаються у моделях, бізнес-логіка працює у ViewModel, а інтерфейс взаємодіє з даними через ViewModel. Така структура не лише спрощує тестування та оновлення додатку, але й дозволяє легко масштабувати його у майбутньому.

Що стосується інтерфейсу, тут вибір зупинився на **Qt/QML**. Чому саме Qt/QML? Цей фреймворк поєднує у собі два важливі моменти: швидкість розробки та можливість створення сучасних інтерфейсів. З його допомогою можна не лише швидко створити прототипи, але й реалізувати динамічні анімації, що робить додаток привабливим для користувачів. Крім того, Qt дозволяє легко адаптувати BudgetBee для різних платформ — Windows, Android тощо.

Але інтерфейс — це лише видима частина програми. Основна логіка BudgetBee реалізується на **C++**. Це рішення обумовлене потребою в продуктивності, адже додаток повинен швидко обробляти транзакції, аналізувати історію цін та формувати звіти. C++ також дозволяє реалізувати багатопотоковість, що важливо для швидкого оновлення даних у реальному часі.

У контексті функціональності BudgetBee передбачає реалізацію **мульти-аккаунтингу** та **аналізу цінових змін**. Для мульти-аккаунтингу передбачено можливість створення кількох рахунків, таких як готівка, картка або депозит. Це дозволить користувачам бачити загальний баланс або баланс за кожним рахунком окремо.

**Аналіз цінових змін** — це ще одна важлива особливість BudgetBee. Ідея полягає у тому, щоб не просто фіксувати витрати, а й аналізувати динаміку цін за одиницю товару. Наприклад, якщо користувач купує 1 кг телятини за 250 грн, а потім за 300 грн, система автоматично розрахує середню ціну та покаже динаміку. Це дозволить виявити зростання або зниження цін і, можливо, коригувати витрати.

Щоб реалізувати всі ці завдання, структура BudgetBee розробляється так, щоб кожен модуль можна було оновити або розширити без глобальних змін у коді. Наприклад, модуль формування звітів працює окремо від основної логіки програми. Це дозволяє легко додати нові типи звітів або змінити існуючі алгоритми без ризику порушити іншу функціональність.

Отже, обрані технології не лише відповідають поточним вимогам проєкту, але й дозволяють масштабувати BudgetBee у майбутньому, додаючи нові функції або адаптуючи програму під нові платформи.

**1.5 Порівняльний аналіз обраних технологій**

У процесі розробки BudgetBee було обрано ряд технологій, які забезпечують ефективну реалізацію заявленої функціональності. Проте, для підтвердження доцільності цих рішень, варто порівняти їх з іншими альтернативами.

Таблиця 1.5.1 – порівняння СУБД

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерій** | **SQLite** | **PostgreSQL** | **MySQL** |
| Легковажність | Легка, без сервера | Вимагає сервер | Вимагає сервер |
| Локальне зберігання | Так | Ні | Ні |
| Транзації | Підтримує | Підтримує | Підтримує |
| Складні запити | Обмежені | Потужні | Потужні |
| Масштабованість | Обмежена | Висока | Висока |
| Простота налаштування | Дуже проста | Складна | Складна |
| Продуктивність | Висока | Висока | Висока |

SQLite обрана через простоту інтеграції, легковажність та локальне зберігання даних, що відповідає вимогам BudgetBee.

Таблиця 1.5.2 – порівняння архітектури

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерій** | **MVVM** | **MVC** | **MVP** |
| Розділення логіки | Чітке | Може бути розмитим | Чітке |
| Інтеграція з QML | Легко | Вимагає адаптації | Вимагає адаптації |
| Тестування | Просте | Складне | Просте |
| Анімації | Легко | Ускладненее | Ускладнене |
| Залежності | Мінімальні | Мінімальні | Високі |
| Масштабованість | Висока | Середня | Висока |

MVVM обрана через зручну інтеграцію з QML, простоту реалізації анімацій та можливість масштабування.

Таблиця 1.5.3 – Порівняльний аналіз фреймворків та мов програмування

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерій** | **QT/QML** | **Flutter** | **React Native** |
| Цільова аудиторія | Розробка десктопних та мобільних додатків з акцентом на продуктивність та інтеграцію з C++ | Кросплатформенні мобільні додатки з анімаціями та гарним UI | Мобільні та веб-додатки з перевагою швидкої розробки |
| Основна мова | C++ / QML | Dart | JavaScript |
| Продуктивність | Висока завдяки C++ та нативній інтеграції | Середня, оскільки Dart виконується через Flutter Engine | Середня, оскільки JS працює через Bridge |
| Анімації | Потужні, гнучкі та легкі у реалізації | Потужні, завдяки Flutter Widgets | Вимагають оптимізації, особливо для складних анімацій |
| Кросплатформеність | Висока (Windows, Android, Linux, macOS) | Висока (Android, iOS, Web, Desktop) | Висока (Android, iOS, Web) |
| Розмір додатку | Компактний, залежить від обраних компонентів | Більший через Flutter Engine та Widgets | Середній, але може збільшуватись через залежності |
| Інтеграція з нативними модулями | Вбудована, особливо для C++ | Можлива через Platform Channels, але потребує Dart | Можлива через Native Modules, але потребує JS Bridge |
| Крива навчання | Висока через поєднання C++ та QML | Середня, але потребує знання Dart | Низька для розробників JS та React |

Висновок:

* Qt/QML підходить для проєктів з акцентом на продуктивність, нативну інтеграцію та кросплатформенність.
* Flutter орієнтований на швидку розробку анімованих інтерфейсів, але має більший розмір додатку та середню продуктивність.
* React Native зручний для швидкої розробки мобільних та веб-додатків, але вимагає оптимізації для складних анімацій та інтеграції з нативними компонентами.

Таким чином, обрані технології — SQLite, MVVM, Qt/QML та C++ — забезпечують ефективну обробку даних, зручний інтерфейс та можливість масштабування BudgetBee. Це дозволяє зосередитися на локальній обробці фінансових даних без залучення серверних рішень, зберігаючи простоту реалізації та високу продуктивність.

**2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУЛЮВАННЯ ВИМОГ**

**2.1 Опис предметної області**

У сучасному світі фінансове управління — це справжнє мистецтво, яке потребує не лише дисципліни, а й ефективних інструментів. Часто ми навіть не помічаємо, як витрачаємо гроші на дрібниці, які в кінці місяця складаються у чималу суму. Саме тут на допомогу приходить BudgetBee — система, покликана взяти під контроль кожну копійку.

Ідея BudgetBee виникла з розуміння того, що сьогодні людям потрібен не просто калькулятор доходів і витрат. Їм потрібен особистий фінансовий асистент, який не лише підрахує витрати, а й покаже, де гроші «витікають крізь пальці». Це не просто додаток, а справжній фінансовий компас, який допомагає знайти баланс між бажаннями та можливостями.

Користувачі BudgetBee зможуть не лише фіксувати витрати, а й бачити фінансові тренди, відстежувати динаміку доходів, планувати бюджет на наступний місяць та отримувати нагадування про важливі платежі. Усе це — в одній системі, що поєднує простоту використання та потужну аналітику.

Отже, BudgetBee — це не просто черговий додаток для обліку грошей. Це — особистий фінансовий радник, який допомагає взяти ситуацію під контроль, ставити фінансові цілі та досягати їх без зайвих витрат нервів та часу.

**2.2 Формулювання вимог до системи**

Щоб BudgetBee став справжнім помічником у фінансових справах, необхідно чітко визначити, що саме він має робити. З одного боку, додаток має бути простим і зрозумілим. З іншого — він має забезпечувати повний спектр можливостей для обліку, аналізу та планування фінансів.

BudgetBee — це більше, ніж просто калькулятор витрат. Це інструмент для повноцінного контролю над фінансами. Основні функції системи:

**Реєстрація та авторизація:** Користувач може створити обліковий запис, увійти до системи, а також відновити пароль у разі його втрати.

* **Створення та управління рахунками:** Кожен користувач може додавати кілька рахунків — банківські картки, депозити, готівкові кошти. Це дозволить контролювати усі фінанси в одному місці.
* **Облік транзакцій:** Додавання нових операцій, що включають суму, категорію, дату та опис. Це дозволяє користувачам бачити, на що йдуть гроші.
* **Аналіз фінансових потоків:** Система будує графіки та діаграми, які наочно показують, куди «витікають» гроші. Це допомагає виявити непотрібні витрати.
* **Планування бюджету:** Встановлення лімітів на певні категорії витрат із повідомленнями про досягнення межі.
* **Нагадування про платежі:** Додаток нагадує про регулярні платежі, щоб уникнути прострочень.
* **Генерація звітів:** Система формує детальні звіти за обраний період, щоб користувачі могли оцінити свої фінансові звички.

Функціональність — це ще не все. Важливо, щоб BudgetBee працював стабільно, виглядав сучасно і надійно захищав дані користувачів. Основні нефункціональні вимоги:

* **Продуктивність:** Додаток має швидко реагувати на дії користувача незалежно від обсягу даних.
* **Зручність використання:** Інтерфейс повинен бути інтуїтивно зрозумілим, щоб навіть люди без досвіду могли з легкістю працювати із системою.
* **Безпека даних:** Усі дані мають зберігатися у зашифрованому вигляді. Передача інформації здійснюється за захищеним протоколом.
* **Мобільність:** Система повинна працювати як на комп’ютерах, так і на мобільних пристроях.
* **Відмовостійкість:** У разі збоїв BudgetBee повинен забезпечити збереження даних і їх відновлення.
* **Масштабованість:** Додаток має підтримувати розширення функціоналу без втрати продуктивності.

Отже, BudgetBee визначено як інструмент для обліку, аналізу та планування фінансів. Сформульовані вимоги забезпечують реалізацію ключових завдань проєкту та створюють основу для подальшої технічної реалізації.

**3. Проектування системи**

**3.1 Визначення архітектури системи**

Архітектура системи BudgetBee будується на основі патерну **MVVM (Model-View-ViewModel)**, що забезпечує чітке розділення логіки бізнес-процесів, інтерфейсу користувача та управління даними. Такий підхід дозволяє спростити підтримку коду, сприяє модульності та забезпечує можливість розширення функціональності без суттєвих змін у структурі додатку.

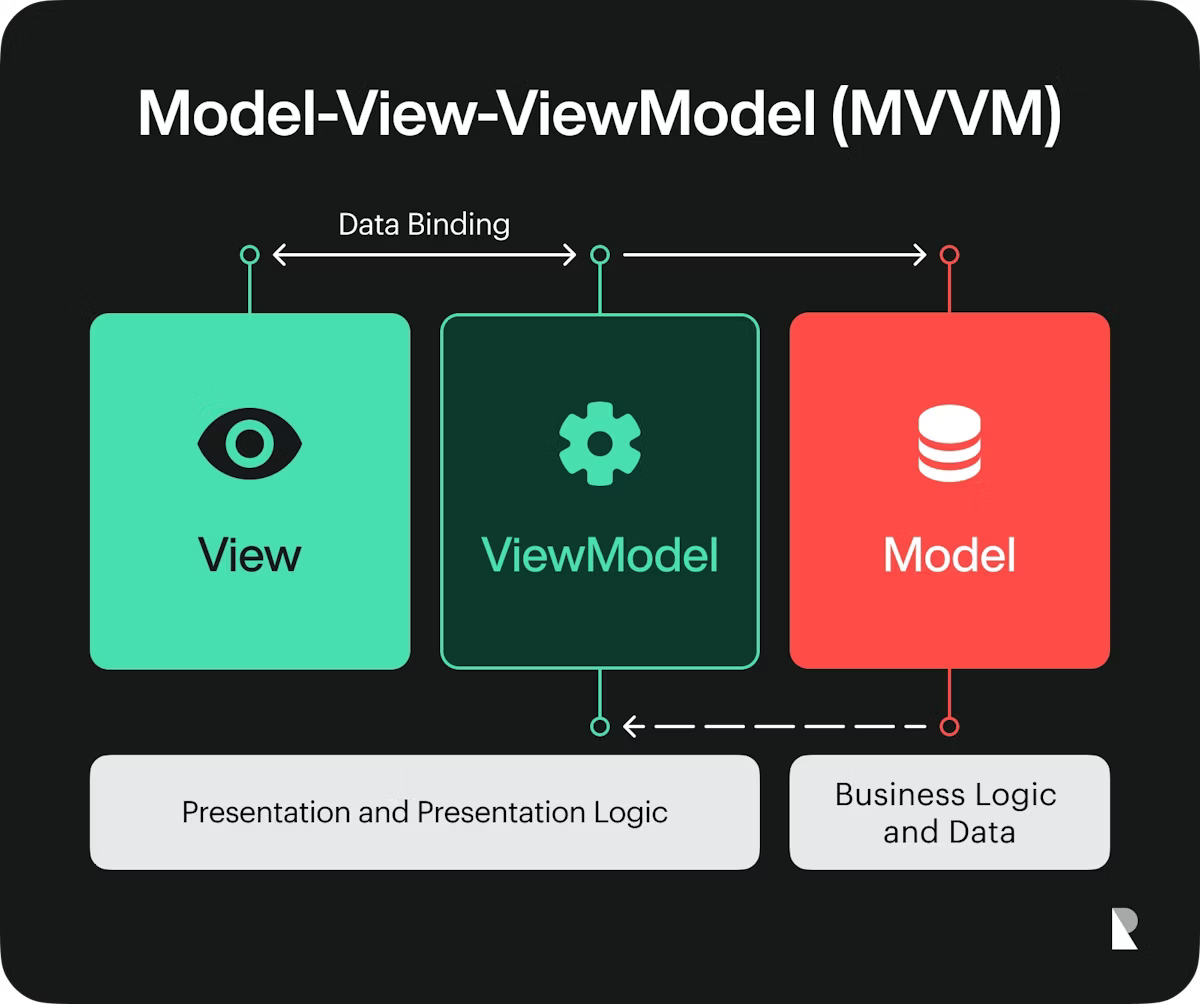


Рисунок 3.1.1 – MVVM архітектура

**Model** — це шар, що відповідає за роботу з даними. У BudgetBee він включає взаємодію з базою даних SQLite, зберігання та обробку фінансових даних, таких як транзакції, категорії та рахунки. Всі моделі реалізовані за допомогою C++ класів та містять методи для маніпуляції даними: додавання, оновлення, видалення та фільтрація.

**ViewModel** — шар, який виступає як посередник між Model та View. У BudgetBee реалізовано кілька ViewModel-класів, таких як TransactionViewModel та CategoryListViewModel. Вони забезпечують доступ до даних моделей та надають їх View у вигляді зручних для відображення форматів. Наприклад, TransactionViewModel надає список транзакцій для відображення у QML-компонентах та оновлює ці дані у реальному часі при зміні бази даних.

**View** — це користувацький інтерфейс, побудований за допомогою Qt Quick Controls 2 та QML. Він відповідає за відображення даних та взаємодію з користувачем. У BudgetBee використовується набір готових компонентів, таких як ListView, Button, TextField для реалізації інтерфейсу. Взаємодія з ViewModel здійснюється через QML-посилання на ViewModel об’єкти, що забезпечує динамічне оновлення даних.

Для забезпечення чіткої взаємодії між компонентами MVVM використовується система сигналів і слотів, що дозволяє реалізувати реактивний інтерфейс. Наприклад, при додаванні нової транзакції TransactionViewModel відправляє сигнал, який автоматично оновлює список транзакцій у View.

Схема взаємодії між компонентами зображено на рисунку 3.1.1. **View** надсилає команди до ViewModel через методи та сигнали. **ViewModel** обробляє ці команди, виконує необхідні операції з **Model** та повертає оновлені дані. Model зберігає ці дані у SQLite та інформує ViewModel про зміни через сигнали. ViewModel оновлює View, передаючи актуальні дані для відображення.

Таким чином, архітектура MVVM забезпечує ефективне розділення відповідальностей та спрощує тестування логіки додатку. Розробник може легко змінювати інтерфейс користувача без необхідності переписувати бізнес-логіку, що робить систему гнучкою та масштабованою.

**3.2 Структура бази даних**

База даних у BudgetBee побудована з урахуванням специфіки фінансових транзакцій, категоризації витрат, доходів та переказів, а також підтримки мультивалютних рахунків. Нижче наведено детальний опис структури бази даних та взаємозв'язків між таблицями (рисунок 3.2.1) та база на стадії планування (рисунок 3.2.2). На таблиці 3.2.1більш детальний опис таблиць бази даних.

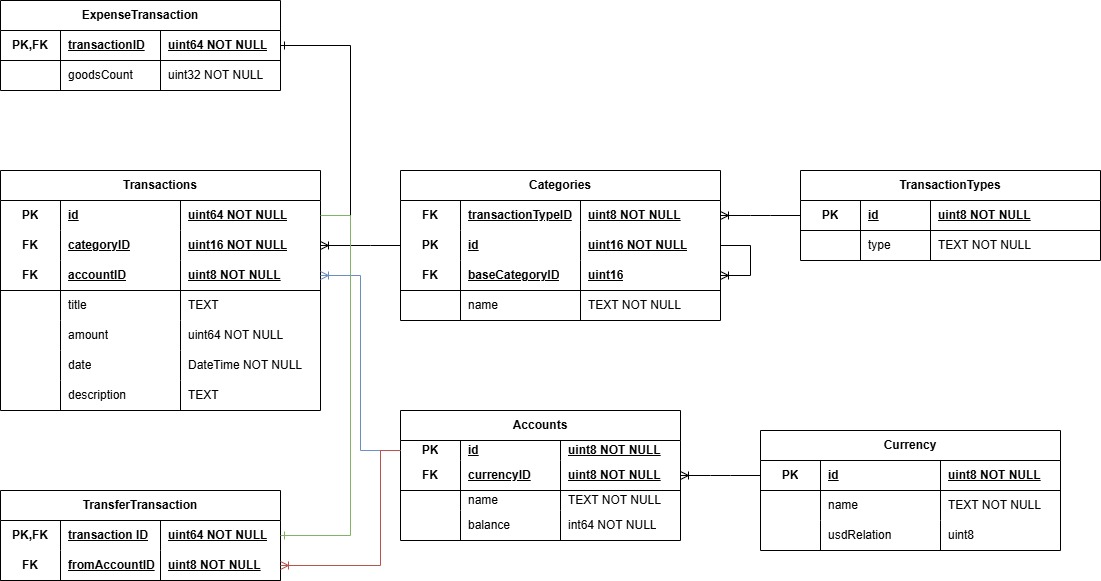


Рисунок 3.2.1 – ER діаграма на стадії планування

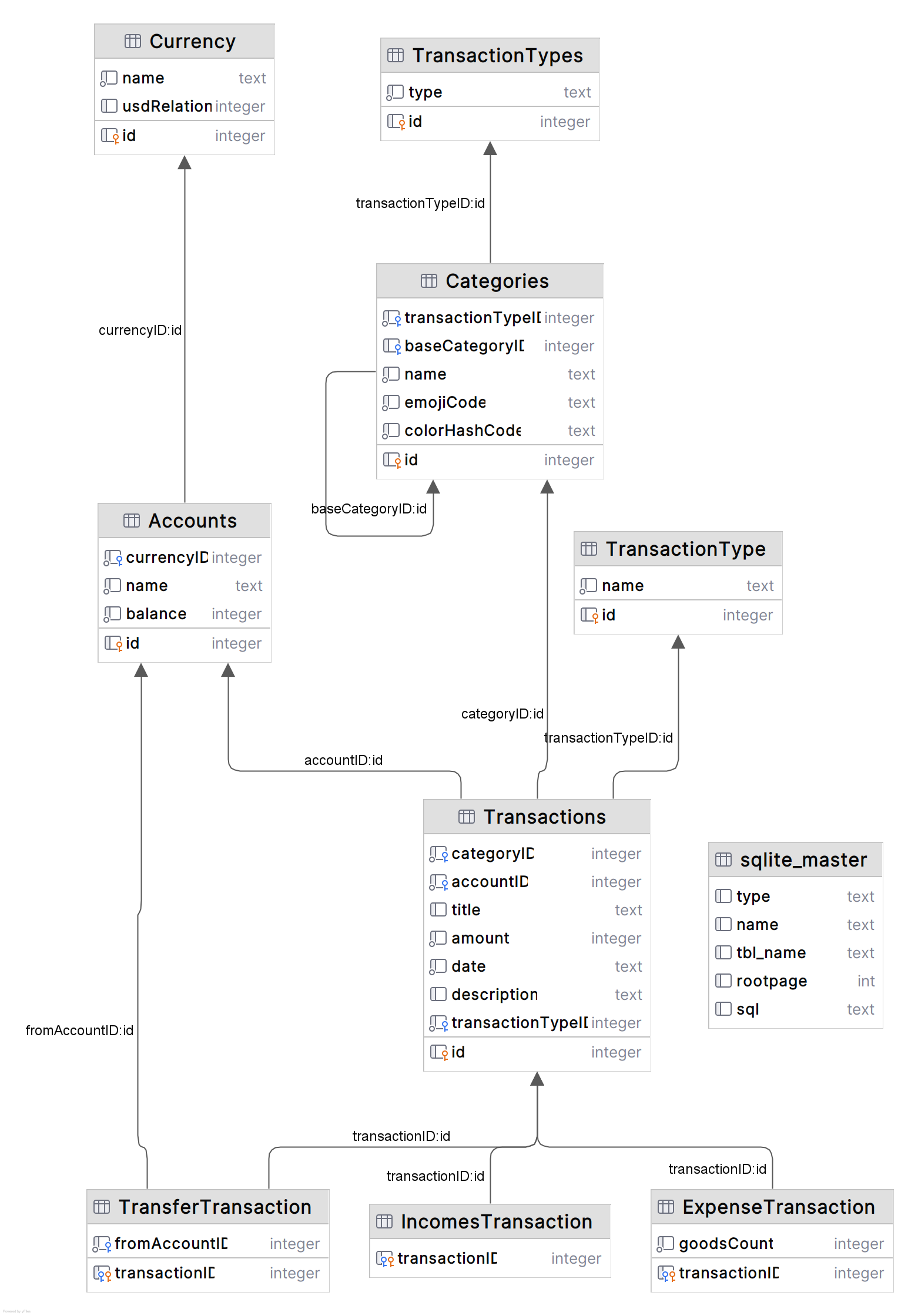


Рисунок 3.2.2 – діаграма бази даних

Таблиця 3.2.1 – опис сутностей та їх властивостей

|  |  |  |
| --- | --- | --- |
| Властивість | Тип даних | Опис |
| Об’єкт Currency (Валюта) | | |
| id | INTEGER | Унікальний ідентифікатор валюти (Primary Key) |
| name | TEXT | Назва валюти (USD, UAH, тощо) |
| usdRelation | INTEGER | Курс валюти до USD, помножений на 100 (наприклад 1.10 = 110) |
| Об’єкт TransactionType (Типи транзакцій) | | |
| id | INTEGER | Унікальний ідентифікатор типу (Primary Key) |
| type | TEXT | Тип транзакції (Income, Expence, Transfer) |
| Об’єкт Accounting (Рахунки) | | |
| id | INTEGER | Унікальний ідентифікатор рахунку (Primary Key) |
| currencyID | INTEGER | Ідентифікатор валюти (Foreign Key) |
| name | TEXT | Назва рахунку |
| balance | INTEGER | Баланс рахунку у найменших одиницях валюти (копійки, центи) |
| Об’єкт Categories (Категорії) | | |
| id | INTEGER | Унікальний ідентифікатор категорії (Primary Key) |
| transactionTypeID | INTEGER | Ідентифікатор типу транзакції (Foreign Key) |
| baseCategoryID | INTEGER | Ідентифікатор батьківської категорії (для підкатегорій) |
| name | TEXT | Назва категорії |
| emojiCode | TEXT | Іконка категорії |
| colorHashCode | TEXT | Колір категорії у HEX |
| Об’єкт Transactions (Транзакції) | | |
| id | INTEGER | Унікальний ідентифікатор транзакції (Primary Key) |
| categoryID | INTEGER | Ідентифікатор категорії (Foreign Key) |
| accountID | INTEGER | Ідентифікатор рахунку (Foreign Key) |
| title | TEXT | Назва транзакції |
| Властивість | Тип даних | Опис |
| amount | INTEGER | Сума транзакції у найменших одиницях |
| date | TEXT | Дата транзакції |
| description | TEXT | Опис транзакції |
| transactionTypeID | INTEGER | Ідентифікатор типу транзакції (Foreign Key) |
| Об’єкт ExpenseTransaction (Витрати) | | |
| transactionID | INTEGER | Ідентифікатор транзакції (Foreign Key) |
| goodsCount | INTEGER | Кількість товару було куплено |
| Об’єкт TransferTransaction (Перекази) | | |
| transactionID | INTEGER | Ідентифікатор транзакції (Foreign Key) |
| fromAccountID | INTEGER | Ідентифікатор транзакції з якої був переклад грошей (Foreign Key) |
| Об’єкт IncomesTransaction (Доходи) | | |
| transactionID | INTEGER | Ідентифікатор транзакції (Foreign Key) |

Між сутностями наявні 2 типу зв’язків:

* 1:N – “один-до-багатьох” . Наявні в таблицях:
  + Accounts та Transactions (accountID)
  + Currency та Account (currencyID)
  + TransactionTypes та Transactions (transactionTypeID)
  + TransferTransaction та Accounts (fromAccountID)
* 1:1 – “один-до-одного” . Наявні в таблицях:
  + Transactions та ExpenseTransaction (transactionTypeID)
  + Transactions та IncomesTransaction (transactionTypeID)
  + Transactions та TransferTransaction(transactionTypeID)

**3.3 Проектування моделі ViewModel**

У BudgetBee модель **ViewModel** виконує роль посередника між даними (Model) та інтерфейсом користувача (View), забезпечуючи реактивне оновлення даних та абстрагування логіки бізнес-процесів. Діаграма класів (додаток А), наведена детально відображає взаємозв’язки між ViewModel, Model та інтерфейсами.

На основі реалізації **TransactionViewModel.cpp** можна визначити основну структуру та принципи роботи з даними у **TransactionViewModel**.

**TransactionViewModel** відповідає за управління транзакціями та забезпечує передачу даних до інтерфейсу користувача у форматі, зручному для відображення у QML.

Клас реалізує логіку обробки транзакцій (**Income**, **Expense**, **Transfer**), забезпечуючи операції додавання, оновлення та видалення транзакцій, а також обчислення загальних сум по транзакціях.

**TransactionViewModel** реалізує такі основні компоненти:

* **Сигнали та слоти** для динамічного оновлення даних у View.
* **Методи для роботи з транзакціями:** **addUpdateExpenseTransaction()**, **addUpdateIncomeTransaction()**, **removeExpenseTransactionCategory()** та **removeIncomeTransactionCategory()**.
* **Методи для обчислення сум транзакцій:** **transactionIncomesSum()**, **transactionExpenseSum()**, **transactionTransferSum()**.
* **Методи для доступу до поточної транзакції:** **currentTransactionID()**, **currentTransactionAmount()**, **currentTransactionDate()** тощо.

При ініціалізації **TransactionViewModel** відбувається встановлення з'єднань з **ITransactionsModel**. Це дозволяє ViewModel динамічно реагувати на зміни у моделі транзакцій.

void TransactionViewModel::init()

{

connect(\_transactionModel.get(),

&models::ITransactionsModel::currentTransactionChanged,

this,

&TransactionViewModel::currentTransactionIDChanged,

Qt::QueuedConnection);

connect(\_transactionModel.get(),

&models::ITransactionsModel::transactionListChanged,

this,

&TransactionViewModel::transactionIncomesSumChanged,

Qt::QueuedConnection);

}

* **currentTransactionChanged** — оновлює властивості поточної транзакції (**ID**, **Amount**, **Date**).
* **transactionListChanged** — оновлює суми доходів, витрат та переказів.

Для роботи з транзакціями у **TransactionViewModel** реалізовано два методи:

* **addUpdateExpenseTransaction()** — створення або оновлення витратної транзакції.
* **addUpdateIncomeTransaction()** — створення або оновлення дохідної транзакції.

Приклад реалізації **addUpdateExpenseTransaction()**:

void TransactionViewModel::addUpdateExpenseTransaction(quint64 id,

quint64 accountID,

quint64 categoryID,

QString title,

qint64 amount,

QString description,

quint32 count,

QDateTime date)

{

types::UniversalTransaction transaction;

transaction.id = id;

transaction.accountID = accountID;

transaction.category.id = categoryID;

transaction.title = std::move(title);

transaction.amount = amount;

transaction.description = std::move(description);

transaction.count = count;

transaction.date = date;

transaction.transactionTypeID = types::TransactionTypeEnum::EXPENSE;

if (transaction.id) {

\_transactionModel->editExpenseTransaction(transaction);

} else {

\_transactionModel->addExpenseTransaction(transaction);

}

\_transactionModel->updateTransactionList();

}

Якщо **id** не дорівнює нулю, виконується оновлення існуючої транзакції через **editExpenseTransaction()**.

Якщо **id** дорівнює нулю, створюється нова транзакція через **addExpenseTransaction()**.

Для видалення транзакцій реалізовано два методи:

* **removeExpenseTransactionCategory()** — видалення витратної транзакції.
* **removeIncomeTransactionCategory()** — видалення дохідної транзакції.

Приклад реалізації **removeExpenseTransactionCategory()**:

void TransactionViewModel::removeExpenseTransactionCategory(qint64 id)

{

\_transactionModel->removeExpenseTransaction(id);

\_transactionModel->updateTransactionList();

}

**Обчислення сум транзакцій**

Для обчислення загальних сум доходів, витрат та переказів реалізовано три окремих методи:

* **transactionIncomesSum()** — сума доходів.
* **transactionExpenseSum()** — сума витрат.
* **transactionTransferSum()** — сума переказів.

Приклад реалізації **transactionExpenseSum()**:

qint64 TransactionViewModel::transactionExpenseSum() const

{

return accumulateTransactionsSum(types::TransactionTypeEnum::EXPENSE);

}

Метод використовує загальний метод `accumulateTransactionsSum()`, який реалізує обчислення суми для вказаного типу транзакцій.

qint64 TransactionViewModel::accumulateTransactionsSum(types::TransactionTypeEnum transactionType) const

{

auto transactionList = \_transactionModel->transactionList();

if (!transactionList) {

return 0;

}

return std::accumulate(transactionList->begin(), transactionList->end(), static\_cast<qint64>(0), [transactionType](qint64 sum, const types::Transaction &item) {return item.transactionTypeID == transactionType ? sum + item.amount : sum;});

**TransactionViewModel** взаємодіє з **ITransactionsModel** через **std::shared\_ptr**. Це дозволяє зберегти слабку залежність між ViewModel та Model, що полегшує тестування та розширення логіки.

**Приклад передачі моделі:**

TransactionViewModel::TransactionViewModel(

std::weak\_ptr<models::ITransactionsModel> transactionModel, QObject \*parent)

: QObject{parent}

, \_transactionModel{transactionModel.lock()}

{}

transactionModel передається у конструктор як **std::weak\_ptr**, що дозволяє уникнути циклічних залежностей.

**TransactionViewModel** реалізує ключову логіку обробки транзакцій у BudgetBee, забезпечуючи централізоване управління транзакціями (**Income**, **Expense**, **Transfer**). Клас використовує сигнали та слоти для динамічного оновлення даних у View, а також забезпечує розрахунок сум транзакцій. Взаємодія з **ITransactionsModel** реалізована через **std::weak\_ptr**, що дозволяє мінімізувати залежності та забезпечити модульність системи.

**4. Тестування та відлагодження системи**

**4.1 Тестування програмної системи**

Тестування програмної системи BudgetBee є одним із ключових етапів у процесі розробки програмного забезпечення. Цей етап спрямований на перевірку функціональності, надійності та якості програми до її випуску на ринок або введення в експлуатацію. Завдяки тестуванню виявляються можливі помилки, дефекти та недоліки в роботі системи, що дозволяє їх усунути до моменту використання програми кінцевими користувачами.

Процес тестування починається з етапу планування, на якому визначаються основні цілі тестування, його обсяг та критерії успішного завершення. Відповідно до розробленого плану, формуються тестові сценарії, що охоплюють усі основні функції системи, включаючи обробку транзакцій, розрахунок балансу, генерацію звітів та роботу з даними. Кожен тестовий сценарій включає набір вхідних даних, очікувані результати та критерії успішності.

Після розробки тестових випадків проводиться їх виконання, яке передбачає введення тестових даних, аналіз отриманих результатів та їх порівняння з очікуваними. У разі виявлення дефектів, вони фіксуються у спеціальній системі управління дефектами, після чого здійснюється їх аналіз з метою виявлення причини та визначення способів усунення. Завершальним етапом є валідація результатів тестування, що дозволяє переконатися у тому, що система працює відповідно до вимог специфікації.

Таким чином, тестування програмної системи BudgetBee не лише допомагає виявити та усунути помилки, але й забезпечує загальну стабільність та якість програмного продукту, підвищуючи довіру користувачів до системи.

Таблиця 4.1.1 – Функціональне тестування

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Крок сценарію | Очікуваний результат | Отриманий результат | Відмітка про проходження |
| 1 | Відкрити головну сторінку | Відображається головна сторінка | Сторінка відкрилася | Так |
| 2 | Виконати авторизацію користувача | Користувач успішно авторизований | Авторизація пройшла | Так |
| 3 | Додати нову транзакцію | Транзакція успішно додана до бази даних | Транзакція додана | Так |
| 4 | Видалити транзакцію | Транзакція успішно видалена | Транзакція видалена | Так |
| 5 | Змінити суму транзакції | Сума транзакції оновлена у базі даних | Сума успішно змінена | Так |
| 6 | Фільтрувати транзакції за датою | Відображається список транзакцій за обраний період | Список транзакцій коректний | Так |
| 7 | Генерація звіту за категоріями | Звіт створено у PDF форматі з відповідними даними | Звіт згенеровано | Так |
| 8 | Вивантаження звіту у CSV | Звіт збережено у CSV форматі | Звіт успішно завантажено | Так |
| 9 | Пошук транзакції за сумою | Відображається транзакція із заданою сумою | Транзакція знайдена | Так |
| 10 | Вихід з облікового запису | Користувач виходить зі свого акаунту |  | Так |

**4.2 Модульне тестування**

Для перевірки коректності роботи окремих компонентів системи BudgetBee було створено набір юніт-тестів з використанням бібліотеки Google Test. Тести зосереджені на основних модулях системи: обробці транзакцій, обчисленні балансу та генерації звітів.Приклад виведення результатів тестування наведено на рисунку 4.2.1. Таким чином, модульне тестування дозволяє швидко перевірити коректність реалізації окремих компонентів системи, забезпечуючи високу якість та надійність програмного продукту.

Модульне тестування є одним із ключових етапів перевірки коректності функціонування окремих компонентів системи BudgetBee. Для цього етапу використовувалася бібліотека Google Test, яка дозволяє створювати тести для окремих модулів та класів, ізольовано від інших компонентів системи.

Основними цілями модульного тестування було перевірити:

* правильність обробки даних транзакцій;
* коректність виконання CRUD-операцій з базою даних;
* функціональність алгоритмів обчислення балансу та статистичних показників;
* відповідність результатів логіці програми та очікуваним значенням.

Модульні тести розроблялися з урахуванням усіх можливих сценаріїв використання системи, включаючи як коректні дані, так і можливі виключення та помилки.

Приклад модульного тесту для модуля транзакцій:

#include <gtest/gtest.h>

#include "TransactionModel.hpp"

TEST(TransactionModelTest, AddTransaction) {

TransactionModel model;

model.addTransaction("Food", 150);

EXPECT\_EQ(model.getBalance(), 150);

}

TEST(TransactionModelTest, DeleteTransaction) {

TransactionModel model;

model.addTransaction("Transport", 200);

model.deleteTransaction(1);

EXPECT\_EQ(model.getBalance(), 0);

}

TEST(TransactionModelTest, CalculateTotal) {

TransactionModel model;

model.addTransaction("Food", 100);

model.addTransaction("Entertainment", 50);

EXPECT\_EQ(model.calculateTotal(), 150);

}

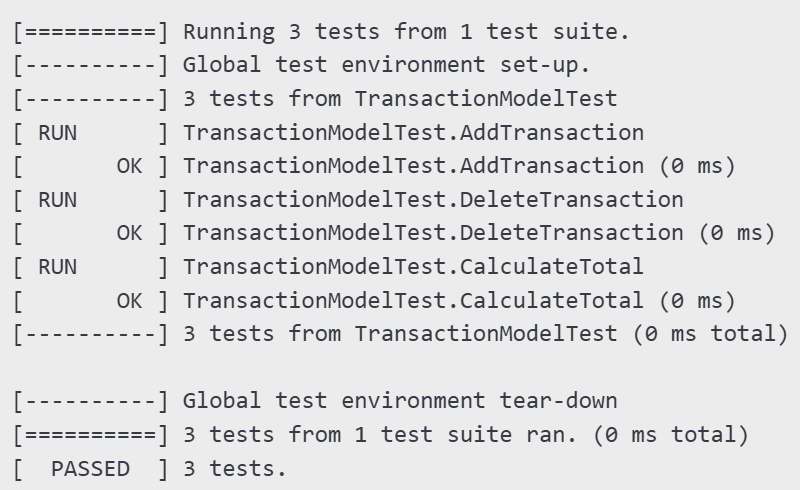


Рисунок 4.2.1 – приклад результату модульного тестування

**4.3 Тестування оптимізації**

Процес тестування оптимізації в системі BudgetBee спрямований на виявлення вузьких місць у коді та забезпечення максимально ефективного використання ресурсів. Для аналізу продуктивності використовувалася бібліотека Google Benchmark, яка дозволяє оцінити час виконання окремих функцій та алгоритмів при різних обсягах даних. Приклад виведення результатів бенчмарку показано на рисунку 4.3.1

Основні цілі тестування оптимізації:

* визначити час обробки транзакцій при різних кількостях даних (1000, 5000, 10 000 записів);
* оцінити швидкість генерації звітів з використанням різних фільтрів та умов;
* проаналізувати роботу алгоритмів обчислення середніх значень та інших статистичних показників.

Приклад коду бенчмарку для функції додавання транзакцій:

#include <benchmark/benchmark.h>

#include "TransactionModel.hpp"

static void BM\_AddTransaction(benchmark::State& state) {

TransactionModel model;

for (auto \_ : state) {

model.addTransaction("Test", 100);

}

}

BENCHMARK(BM\_AddTransaction);

BENCHMARK\_MAIN();

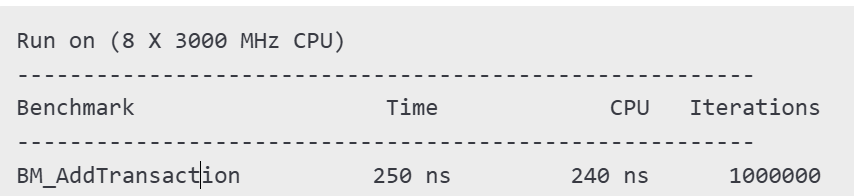


Рисунок 4.2.1 – приклад результату бенчмарка

Аналіз результатів тестування оптимізації дозволяє виявити ділянки коду, що потребують доопрацювання, а також обґрунтовано приймати рішення щодо рефакторингу для досягнення більш високої продуктивності системи.

**4.4 Опис інструкції користувача**

При відкриті ми бачимо головне меню з усіма транзакціями (рисунок 4.4.1)

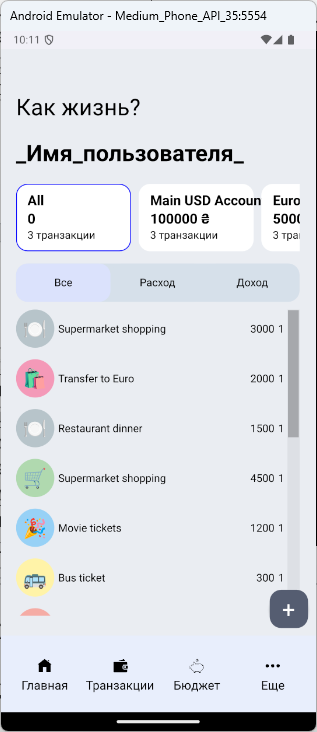
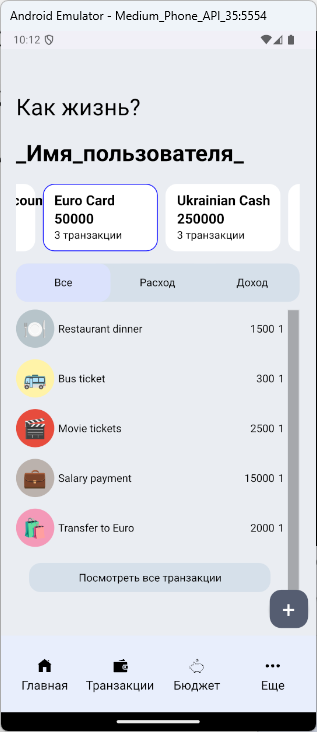
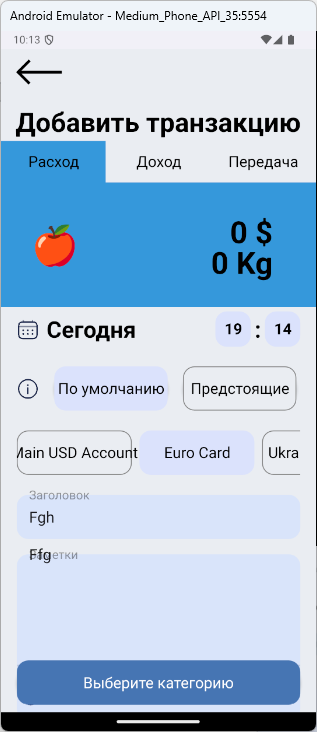


Рисунок 4.4.1 – головне меню

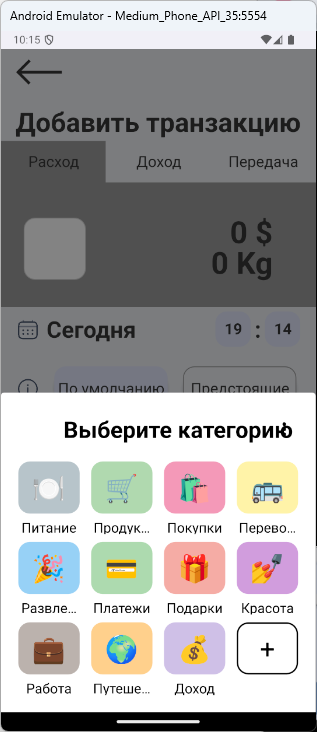
Ми можемо змінити поточний рахунок та побачити список усіх транзакцій (рисунок 4.4.2). Також ми можемо редагувати, додавати або видаляти транзакцію. Реалізація сторінки для цього зображено на рисунку 4.4.3. Для транзації ми можемо вибрати категорію. Для цього потрібно нажати на квадрат та відкриється попап (рисунок 4.4.4). Якщо натиснути на категорію, відкриється його підкатегориї (рисунок 4.4.5) та так рекурсивно. Ми можемо маніпулювати категоріями. Сторінка для цього зображена на рисунку 4.4.6. На нижньому полі зображена історія переходу категорій для більш зручного користування. При натисканні на іконку з історії, ми перекнемося на інший список з історії.



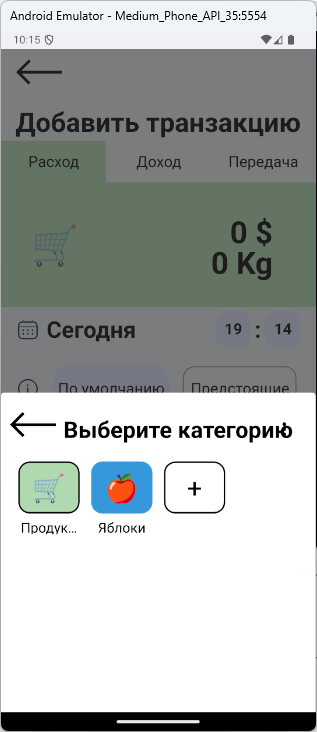
4.4.2 – зміна поточного рахунку



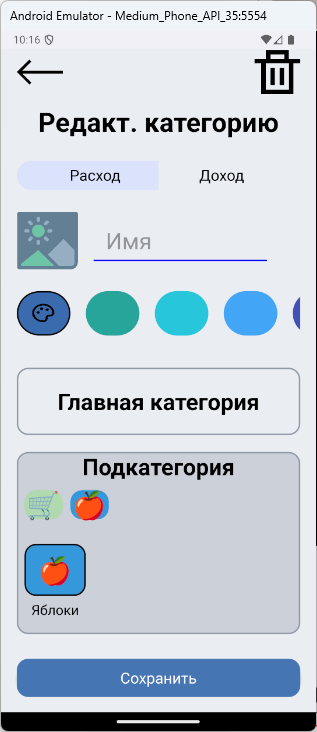
4.4.3 – маніпуляція з транкзаціями



4.4.4 – попап категорій



4.4.5 – попап під категорій



4.4.5 – додати / видалити / редагувати категорію

ДОДАТОК А

