**ВСТУП**



В наш час існує безліч додатків для управління фінансами — від загальних персональних фінансів, таких як Mint, до спеціалізованих додатків для бюджету. Але вони також занадто складні для середньостатистичних користувачів і їм відсутні деякі додаткові функції.

Ось чому з’явився BudgetBee. Його мета — стати простим у використанні програмним забезпеченням для платежів і виставлення рахунків, з доступними звітами для допомоги в управлінні коштами.

Я додатково маю інструменти — досвід програмування на C++ і роботи з інтерфейсами Qt. Бізнес-логіка і спосіб подання даних розробляються на основі патерну MVVM.

Ось кілька ключових кроків для створення BudgetBee:

* по-перше, ми повинні дослідити ринок фінансових додатків — це добра бізнес-модель, основні функції та інтереси користувачів;
* потім ми спроектуємо систему, серцем якої є процедури транзакцій, категорії витрат та звіти;
* API для інтеграції з іншими додатками або системами" дозволить клієнту завантажувати контент без затримки;
* ця функція буде піддана ретельному тестуванню (як функціональному, так і продуктивності). Саме тут Google Benchmark перевіряє чутливість нашої системи, а gtest — наявність помилок.

Життєвий цикл програмного забезпечення зараз розглядається, щоб забезпечити процес обробки фінансових даних та їх перетворення на корисну інформацію для прийняття рішень.

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

## **1.1 Опис предметної області та актуальність розробки**

Автоматизована інформаційна система не тільки відображається у балансі активів і капіталу, але також у закритому або відкритому балансі, де мультимедійні технології вирішують проблеми. Фінансові системи можуть ставати більш дорогими у використанні в процесі позики.

Фінансові дані, наприклад, включають надання даних фінансових послуг, транзакції та доходи, кредити та кредитні ліміти тощо, а також записи всіх інших типів атрибутів «номер». Нижче наведено типовий приклад, і автор зазначає, що для того, щоб зрозуміти його, йому завжди потрібно багато аналізувати, використовуючи формули та розпізнавання шаблонів. Швидкість розповсюдження інформації через фінансові ринки перевершує розвиток кібернетики. Сьогодні фінансова інформація (горішки скрізь) випускається по мережах, а потім перетворюється в інформацію. (Корпоративні титани вивчають фінанси під час роботи, а не до неї; вже недостатньо припускати, що всі вже розуміють «щось подібне», тому що «ми займаємося бізнесом», зникає в минулому; чесно кажучи, ми завжди можемо дізнатися щось нове про фінанси. Ви вже будете зворушені самим існуванням точок входу.)

BudgetBee переосмислює фінанси. З цим програмним забезпеченням - BUDGET, ми плануємо вирішити багато питань. Для цього — тобто ви змушені не тільки вести окремий запис книг, а й заборонено планувати наперед (заощадження) через облік на щоденній основі. Ми зазвичай називаємо це «BudgetBee».

* записуйте свої витрати;
* замисліться про фінансову сторону вашого досвіду;
* фінансова грамотність. Без фінансових знань ви не зможете зрозуміти, які вигоди вона може запропонувати;
* підтримуйте гідного кандидата замість того, щоб підтримувати невдалу угоду доларами;
* зніміть тягар свого фінансового здоров'я з роботи. Коли у вас десятки, а то й сотні завдань, простіше керувати всім одночасно. Це здається клопітким, звісно, відстежувати кожну копійку, що заходить та виходить з вашої кишені. Можна також вийти з себе, коли робите це таким чином, щось, що неважко зробити.

Додатки, такі як BudgetBee та інші, які автоматично реєструють вашу фінансову історію кожного разу, коли ви витрачаєте гроші, можуть значно полегшити ваш спосіб мислення. Це часто означатиме знання, на якому етапі процесу ви знаходитесь з точки зору торгового циклу, і який рівень результатів можна розумно очікувати. Хоча ми будемо працювати з API банку, будуть й інші DMS (системи управління даними) на інших платформах.

А що стосується нашого сайту, то тут насправді дуже мало ручного введення. Він поважає конфіденційність і не залежить від банківських API. BudgetBee - дуже простий додаток, майже мінімалістичний, але він працює напрочуд добре. Це не випадковість.

* визнання витрат і доходів;
* витрати можуть автоматично класифікуватися на основі умов, типів витрат;
* маркування всієї інформації, пов'язаної з вартістю, і створення нових міток;
* прогноз ваших витрат у майбутньому;
* відстеження прогнозних моделей і фактичних даних (щомісячні залишкові витрати, щоденні витрати).

У міні-курсі BudgetBee ви навчитеся планувати бюджет, як ви можете врахувати його у своїх витратах на життя і на скільки шлюбна праця буде необхідна. Якщо вони можуть бути новаторами для аудиторії, яка інакше не має прямого доступу через архаїчні системи обліку або банківські системи до того, скільки вони повинні і коли.

## **1.2 Порівняльна характеристика існуючих аналогів розроблюваної системи**

Сьогодні, у цифрову епоху, ця функція існує як фінансові та рейтингові пристрої (будь-якого виду), завдяки яким створюються дані для відповідних даних і дій, які оцінюються.

І все ж, бухгалтери продовжують говорити про "рахунки", де доходи зіставляються з витратами проти взаємних клієнтів: нехай вони це роблять у банках. До тих пір, поки всі дебети (або кредити) та витрати (або доходи) відповідають стандартному обліку для даної транзакції, все добре.

У вас є всі ці десятки пасток у системі фінансового управління, щоб виконувати такі функції; коли всі ці специфічні характеристики повинні співпрацювати (продукт і послуга не мають різниці, і в будь-якому випадку ніхто більше не знає, які продукти і внутрішні можливості, як дані про ціну, адже компанія тут "і" вже "створена" в якійсь роботі або продукті всі аспекти для використання, і що це і те і це виклик тільки один резюме, як "один" рахунок витрат, зарплата тощо), тоді всі ці проблеми виникають.

Цього разу ми зосереджуємося строго на мінімальних числах, щоб з'ясувати, хто виграє. Для всіх попередніх пунктів (таблиця 1.1) тут у BudgetBee, ми використовуємо наш загальний алгоритм розподілу з єдиною моделлю для поведінки і ціни... яка може в різних формах або ні, добре вписуватися!

QuickBooks

Практично будь-який бізнес може отримати вигоду від використання QuickBooks як бек-енд облікового програмного забезпечення. Ви можете розкласти бухгалтерію - круто :) це не так організовано (категорія → М'ясо → Телятина → Фірма) І немає порівняння об'єму на долар, з автоматичним ціноутворенням продукту.

Xero

Так ми любимо думати про це: Xero для них як бухгалтерія малого бізнесу. Ви можете записувати всі типи інформації, від грошей, які отримала компанія до того, наскільки прострочено є рахунки до отримання. Але вони ніколи не зуміли створити підсистему для нарахування у цих установах. Немає засобів порівняння дефіциту через зміни ціни, так само.

Mint

Mint (Mint.com) інструмент для візуалізації коштів, я дивлюся на суму в ваших руках і всі витрати такі як без контролю над можливістю витрат Інтенсивно, коли доходить до стадії перевірки витрат це для перевірки витрат після "хронології місячних витрат кожної транзакції". Він лише надає прості категорії (Їжа, Автомобіль, Розваги) і ніколи не розділяє підкатегорії або каже вам, що ви заплатили більше/менше за одиницю.

BudgetBee

BudgetBee явно одне з програмного забезпечення для особистих фінансів. Це не просто те, що він робить все добре, у нього є деякі дуже, дуже базові істини про бухгалтерію рівня М, аналіз зміни цін і т.п., і, звичайно, чудова графіка. Він малює чітке, повне економічне бачення, яке будь-хто повинен вміти оперувати самостійно; він намалює чітку картину на екрані найскладнішої філософії життя.

Таблиця 1.1 – 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 |
| Продовження таблиці 1.1 | | | | |
| 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**

Сьогодні, як ніколи раніше в історії, люди потребують фінансової грамотності. Зараз вони зосереджуються на утриманні своїх витрат на низькому рівні, знаючи, звідки надходять їхні гроші, і працюючи зі своїм бюджетом. Але для отримання високого рівня деталізації ваших фінансових даних, усе, що я бачив, було або занадто складне, або занадто специфічне, чи дуже спеціалізоване, але не пристосоване до того, що люди хотіли б використовувати цей інструмент.

Зустрічайте BudgetBee. Він розроблений, щоб бути інтуїтивно зрозумілим і зручним для вас, кінцевого користувача, а також для того, щоб ви мали можливість керувати своїми особистими чи сімейними витратами, здійснювати дії самостійно (і позначати кілька речей в історії користувача) і/або зараз, щоб найбільш інтуїтивні (та найкращі) — користувацькі або проаналізовані дії у пропозиції певних моментів. І потім отримувати і/або розміщувати замовлення(я) і/або ініціювати інші замовлення для тієї ж ринкової дії, що позначають дійсність як значну маніфестацію введених (і/або проаналізованих) дій(й), і [діапазон форм коду], також кластеризації (понад 50) або "зміни цін".

Загальні цілі:

* а тепер давайте продемонструємо, що ми вже робимо. Подумайте про фінансові додатки: QuickBooks, Xero, Mint. Визначте, чого не вистачає в таких системах з багаторівневою категоризацією і що зайве, занадто складне. Дізнайтеся, що хочуть ваші користувачі і що є лише збагаченням чи надмірною складністю;
* загальний дизайн та архітектура DA: дозвольте користувачеві визначати власні набори категорій та/або підкатегорій. ВИ ПОВИННІ мати це, щоб можна було класифікувати по транзакціям, щоб бачити витрати більш організованими:
  + я можу мати більше одного облікового запису, я хочу мати можливість мати більше одного облікового запису (ім'я\_рахунка = готівка, картка, депозит), якщо я хотів, наприклад, показати баланс як баланс або баланс = баланс + загальний баланс. Мене цікавить спрощене управління рахунками, і я хочу перевіряти кожен обліковий запис у додатку;
  + автоматичний розрахунок вартості за одиницю (кг/л і т.д.) товару: функція вартості за одиницю. Це також повідомить їх про підвищення чи очікувану зміну цін;
* модуль звітності за розкладом: оскільки нам потрібно оцінити загальні зобов’язання (період (день, місяць...)) витрати для категорій, витрати для статті за минулий (місяць, рік), прогноз витрат (на останній тиждень). Буде надано;
* для користувачів: дизайн інтерфейсу системи не повинен бути "невідомим". Не може бути більше ніж один чи два кліки до чого-небудь;
* тестування / оптимізація: після періоду інтенсивного технічного тестування і BudgetBee повторно тестується на функціональному рівні, де це необхідно, система виправляється і оптимізується для продуктивності.

І коли ви робите все це, користувачі "матимуть чудовий та зручний інструмент для управління безпліддям вдома." Ви не тільки отримаєте силу домашнього/фабричного витратного приводу, але також зможете дозволити собі подумати про життя вашого гаманця, керувати своїми речами та планувати нові покупки.

## **1.4 Обґрунтування вибору технологій та архітектури BudgetBee**

Тим не менш, масштабування нашого бізнесу — це те, що дійсно необхідне для того, щоб зрозуміти, як фінансова програма працює на практиці.

Тож, залежно від того, чого хочуть користувачі і як їх заохочують вводити свої фінансові дані, BudgetBee доведеться вибирати технологію на майбутнє. Тобто, мені потрібно запустити базу даних, і базою даних, яку я використовував у BudgetBee, була SQLite. Чому?

По-перше, SQLite — це легка, клієнтська система бази даних, яка не вимагає налаштування управління. Вона також повинна забезпечувати сильну конфіденційність, залишаючи дані на пристрої. Це важлива річ для фінансових додатків, оскільки вона усуває необхідність відправляти фінансові дані до будь-якого провайдера хмарних послуг. І ще один бонус: SQLite насправді підтримує транзакції. Іншими словами, якщо щось неправильно виконано в транзакції (рідко, але трапляється), ви можете просто скасувати всі зміни.

А тепер перейдемо до архітектури застосунку. Ми використовували паттерн MVVM.

Як результат, бізнес-логіка стає чітко розділеною від шару представлення. Таким чином, у BudgetBee моделі — це просто дані, ViewModels — це бізнес-логіка, а UI передає дані через ViewModels. Це дуже добре для початківців, щоб навчатися, і легше масштабувати, коли ви випробували або змінили додаток.

Ми вибрали інтерфейс на базі Qt/QML. Чому? Ми вибрали інтерфейс Qt/QML з двох причин: швидкість впровадження (нам потрібно було щось динамічне) і естетичність інтерфейсу. Не тільки ви можете створити простий прототип, ви також можете включити прототип анімації у ваш додаток. Це не буде те саме, звісно, але буде близько. Оскільки BudgetBee написаний на Qt (Win, Android), він досить легко працює на інших подібних середовищах.

BudgetBee використовує движок, написаний на C++. Це гарантує відмінну продуктивність обробки історії транзакцій/цін у будь-якій реалізації додатку. C++ також може обробляти множинні процесори, оскільки це необхідно для актуалізації даних у реальному часі. Також планується впровадження мультикористувацького кодування та історії зміни цін у BudgetBee. Якщо у вас є декілька облікових записів, більше простору для всього, що веде до його частого використання в загальному сенсі. Тобто: багато облікових записів для невеликих частин бюджету в іншому обліковому записі, дозволи, збережені гроші та витрачені гроші і так далі! І потім, у застосовуваній мірі, до профілювання всіх залишків на рахунках, підсумовування рахунків, використовуючи кожну згадану рекламну сторінку, яка потім ведеться.

BudgetBee також підтримує аналог для вас. Справа не стільки в тому, щоб відстежувати витрати, скільки в аналізі витрат. Не загальну суму, а скільки коштує одиниця товару зараз і в майбутньому.

Наприклад, якщо клієнт купив телятину, 1 кг, за 250 грн, а через 3 дні — за 300 грн, система сама автоматично обчислить середню вартість цієї події, і буде показана динаміка подій. Діаграма таких суттєвих змін цін на білети і "бінго!" Споживач закінчує з деякими думками, чи варто щось змінювати з їх фінансами.

Ось чому BudgetBee написаний таким чином, що тільки ядро може бути додано або модифіковано без втручання у кожен окремий модуль. Наприклад, модуль звітності є фоновим процесом до основної контрольної логіки програми. Це дозволяє додавати нові типи звітів або оновлювати старі алгоритми, не піддаючи ризику іншу функціональність.

Отже, з підсумку вище, не важко побачити, що ваш вибір — це те, що ми використовуємо, тому вони будуть змінними (наприклад, нові речі можуть бути додані в якийсь спосіб або навіть повністю замінені). BudgetBee написаний з урахуванням дати, щоб спробувати відповідати.

## **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 1.2 – 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 1.2).

Таблиця 1.3 – 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 1.3).

Таблиця 1.4 – Порівняльний аналіз фреймворків та мов програмування

|  |  |  |  |
| --- | --- | --- | --- |
| Критерій | QT/QML | Flutter | React Native |
| Продовження таблиці 1.4 | | | |
| 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 |

Нарешті, такі технології, як SQLite, MVVM, Qt/QML, C++, сприяють плавним операціям з даними та легкому обслуговуванню інтерфейсу користувача, забезпечуючи можливість задовольняти зростаючий попит на BudgetBee (таблиця 1.4).

Це гарантує, що обробка місцевих фінансових даних зосереджується повністю на пристрої без необхідності серверних рішень, з простою реалізацією та високою продуктивністю.

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

## **2.1 Ідея та концепція BudgetBee як сучасного фінансового асистента**

Якщо положення старого м'якого МВА за тими стриманими інструментами були повністю переглянуті в стилі, то сьогоднішні нові фінансові інструменти можуть вважатися формою мистецтва. І ще одне, нові фінансові менеджерські інструменти сьогодення все ще виконуються в когнітивному режимі, найвищий рівень якого візуалізується і, отже, не потребує пояснень від інших людей.

Тому ми часто не помічаємо, як низка невеликих речей протягом місяця може скластися в одне дуже велике витрачання. Це та сфера, яку покликаний охопити BudgetBee — він показує все про ваше фінансове життя в одному місці.

BudgetBee було винайдено з простої причини, що сьогодні люди більше не задовольняються переглядом даних "в-описі" або "поза-пояснення". Отже, вам потрібен додаток як ваш особистий помічник фінансового менеджменту для відстеження витрат, а в деяких випадках вказуючи, куди йдуть ваші гроші — компас для справжньої впевненості.

Поза простим додатком, насправді це фінансовий компас, здатний вести вас до місць, де бажання і обставини співпадають.

Намагаючись, як вони сьогодні, BudgetBee надає користувачам повну автономію в їхніх облікових записах для того, щоб контролювати свої власні гроші та бачити управління коштами з першого погляду. Вони можуть рахувати свій дохід, коли завгодно, виявити, через що з'являються готівкові кошти, планувати наперед бюджет на наступний місяць (вони тільки що натрапили на розпродажі під кінець року), або нагадування про те, що є ще якісь рахунки, що мають бути сплачені на цьому тижні.

Все це разом в одному пристрої чи місці, вміщено в зміст зі стилем.

З цієї причини BudgetBee відрізняється від інших трекерів витрат на вашому телефоні. І ви можете називати себе "фінансовим радником" або "рятівником", врешті-решт, адже "вони" дійсно роблять це для вас, і не лише спрощують, а й позбавляють всіх тих непотрібних зусиль у звільняючому американському стилі середньостатистичної особи, відвертаючи атаки банкрутства: в прямому сенсі вони дійсно мають на увазі бізнес.

## **2.2 Формулювання вимог до системи**

Якщо ми збираємось перетворити BudgetBee на повноцінного фінансового асистента, нам потрібно вирішити, що саме він буде робити.

З одного боку, додаток має бути дуже простим і зрозумілим у використанні.

З іншого боку, він має містити всі необхідні функції для бухгалтерії, аналізу та фінансового планування.

Так, BudgetBee – це більше, ніж калькулятор витрат. Це – всебічний менеджер особистих фінансів.

* аутентифікація: користувачі реєструються, входять в систему і можуть попросити скинути пароль;
* управління рахунками: ви можете створити безліч рахунків (наприклад, картковий, депозитний, готівковий), щоб усе було в одному місці;
* управління транзакціями: додайте нові транзакції (сума, категорія, дата, опис). Таким чином, ви можете бачити, куди саме йдуть ваші гроші;
* візуалізація потоків: переглядайте, де ваші гроші "витікають" у корисних графіках та візуалізаціях. Це може допомогти вам зрозуміти, куди йдуть ваші гроші;
* бюджетування: створюйте бюджет, додавайте категорії (наприклад, "Покупки" або "Їжа") та відстежуйте витрати в цих категоріях;
* нагадування про платежі: за бажанням, дозвольте додатку нагадувати забудькуватим клієнтам сплачувати;
* звітування: дізнайтеся, як ви фінансово справляєтеся, звідки прийшли гроші і куди вони пішли на ваших рахунках за період використання.

І це ще не все. Єдине, що ми можемо гарантувати про BudgetBee - це те, що вона працює! Виглядає стильно, і ваші дані завжди в безпеці.

Нефункціональні вимоги до системи:

* контроль:тепер, коли дані введені в систему, ми повинні мати моніторинг у реальному часі;
* зручність для користувачів: Інтерфейс повинен бути зручним для користувача, який не знає системи;
* шифрування:дані зашифровані під час збереження (повністю або частково); передача відбувається через безпечне з'єднання;
* переносимість: ноутбуки та подібне. У випадку, якщо BudgetBee вийде з ладу, я хочу мати доступ до інформації з останнього разу, коли я відкривав додаток. Платформа повинна мати змогу легко масштабуватися до більшої системи з часом.

Отже, з описаної тут перспективи, BudgetBee також є інструментом фінансової аналітики, відстеження та прогнозування. Цей концепт може бути реалізований на практиці на більш високому рівні амбіцій для реалізації проекту з певними словами про технічні обмеження розробки та впровадження.

# **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.

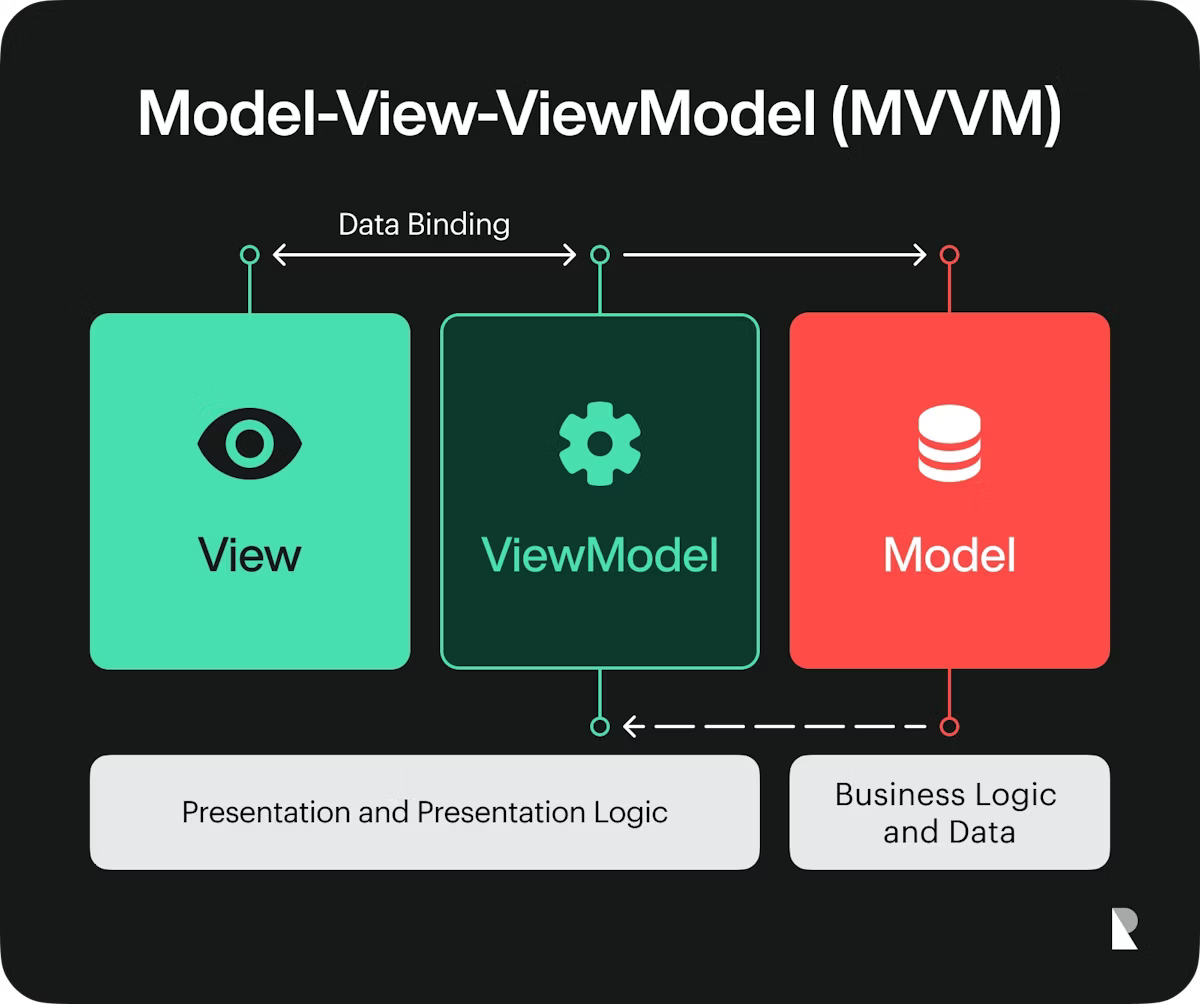


Рисунок 3.1 – MVVM архітектура

Model — це місце, де ви працюєте з логікою даних. Для BudgetBee це означає операції з базою даних SQLite, роботу з даними транзакцій, категорій та рахунків. Всі моделі — це C++ класи, які пропонують методи CRUD (додати, оновити, видалити, прочитати) для обробки даних.

ViewModel — це проміжний простір між Моделлю і Виглядом.

У BudgetBee ми маємо декілька класів ViewModel, таких як TransactionViewModel і CategoryListViewModel. Вони дозволяють отримати дані моделі і подати ці дані на Вигляд у зручний спосіб. Наприклад, TransactionViewModel має список транзакцій, які компоненти QML можуть використати для відображення, і може оновити свої дані при зміні бази даних.

View — інтерфейс користувача, розроблений з використанням Qt Quick Controls 2 і QML.

Це частина, де дані вводяться, а користувач взаємодіє. Надані компоненти використовуються BudgetBee для створення інтерфейсу, а саме ListView, Button і TextField. Жодного зв'язку ViewModel з Виглядом не встановлено, а QML взаємодіє з ViewModel та об'єктами моделі Вигляд, що покриває волатильність змін даних.

І у вас дуже проста комунікація між вашими шарами MVVM завдяки сигналу і слоту, і ви можете створити реактивний інтерфейс.

Наприклад, коли я додаю нову транзакцію і TransactionViewModel надсилає сигнал, тоді Вигляд оновлює список транзакцій самостійно.

Протокол обміну повідомленнями серед компонентів показаний на Рисунку 3.1. Вигляд викликає методи та сигнали ViewModel. Обробляючи ці команди у ViewModel, відбувається реальне оновлення Моделі, і надаються скориговані дані. Модель, заснована на SQLite, записує ці дані і передає зміни ViewModel через сигнали. ViewModel — це те, що живить Вигляд, і Вигляд оновлюється ViewModel, коли ми надаємо дані для відображення.

Результатом є суттєве розосередження відповідальностей, що полегшує тестування логіки вашого додатку (мета MVVM).

Система дуже гнучка і масштабована, тому що різний інтерфейс користувача може бути замінений, а його бізнес-логіка залишиться незмінною.

## **3.2 Структура бази даних**

База даних на BudgetBee організована за подіями, такими як фінансові операції, що відбуваються, розподіл витрат та доходів, а також операції, пов'язані з переказами. Її можна використовувати для обліку в будь-якій валюті.

Нижче наведено таблицю взаємозв’язків між інформацією в нашій базі даних. На рисунку 3.2 зображено планування бази даних, та на рисунку 3.3 вже діаграма самої бази даних.

Таблиця 3.1 надає повні деталі типів даних та таблиць бази даних.

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);

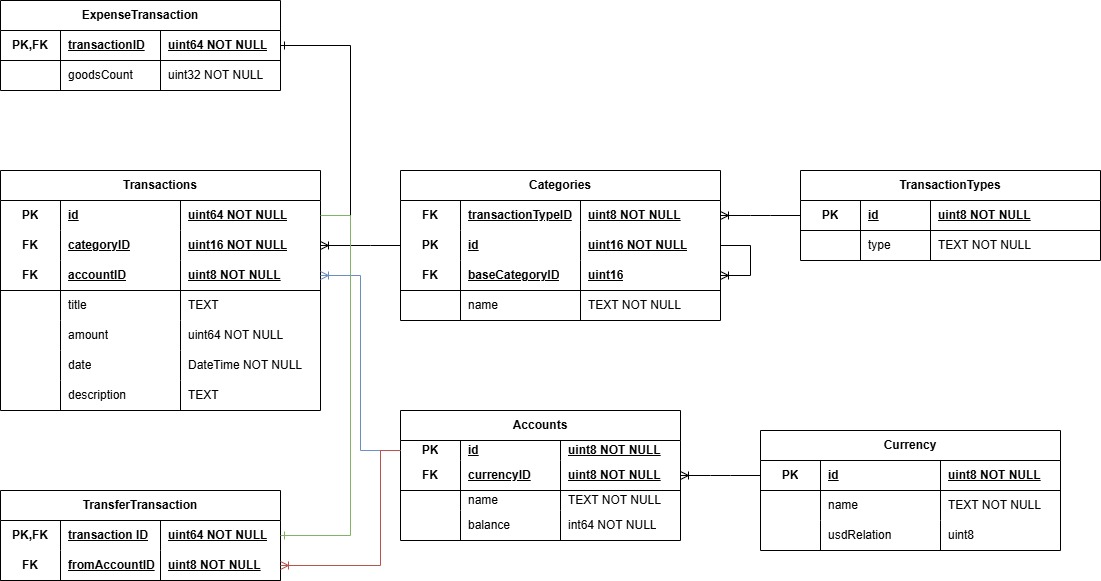


Рисунок 3.2 – ER діаграма на стадії планування

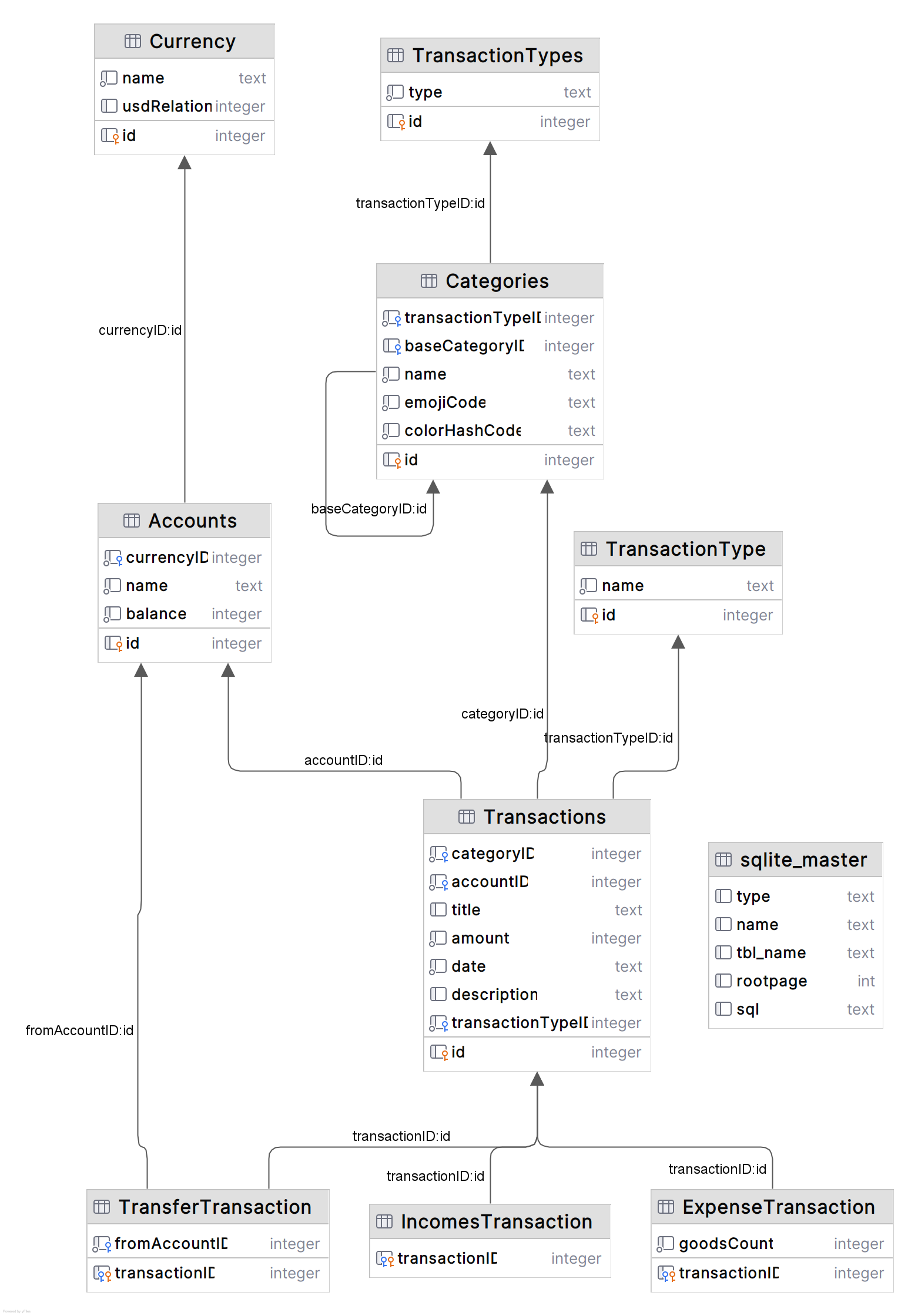


Рисунок 3.3 – Діаграма бази даних

Таблиця 3.1 – 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) |
| Продовження таблиці 3.1 | | |
| 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) | | |
| Продовження таблиці 3.1 | | |
| transactionID | INTEGER | Ідентифікатор транзакції (Foreign Key) |

## **3.3 Проектування моделі ViewModel**

У BudgetBee ViewModel є посередником між моделями даних (Model) та користувацькими інтерфейсами (View) — забезпечуючи як реактивні оновлення, так і абстракцію. Її нові функції називаються інтерфейсами.

Водночас це також вилучає бізнес-логіку з функцій процесу DecorationMethods (які будуть описані пізніше в іншому розділі). Діаграма класу (додаток A) дає детальні описи асоціацій між ViewModel, Model та інтерфейсами.

На основі реалізації TransactionViewModel.cpp можна побачити, якою є основна структура та принципи взаємодії даних у TransactionViewModel. TransactionViewModel відповідає за управління транзакціями та перетворює дані, необхідні для відображення в QML, у відповідний формат.

Клас реалізує логіку обробки транзакцій доходів та витрат, операції, такі як додавання, оновлення та видалення транзакцій, а також обчислення загальних сум транзакцій.

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 програмного забезпечення: тестування BudgetBee є частиною процесу розробки програмного забезпечення. На цьому етапі ринок, фізична реальність або обидва підтверджують функціональність і якість програмного забезпечення. Тестування є фазою, коли ми можемо виявити можливі помилки, баги та проблеми, перш ніж вихід на виробництво і рання відповідь стають доконаним фактом.

Фаза підготовки позначає початок тестування. Вона визначає кроки, які тести повинні виконати з критеріями успіху і межами, і розробляє тести у відповідності до плану. Ці тестові випадки призначені для перевірки на відсутність модулів (тести для всіх модулів — модуль обробки транзакцій, модуль розрахунку балансу, модуль генерації звітів і модуль підтримки даних оцінюються разом). Кожен тест матиме тестові дані, очікуваний результат і умову проходження/провалу.

Ви пишете тестові випадки і потім запускаєте їх після того, як щось розробили. Вам потрібно спочатку розробити спеціальні тестові дані, провести їх через умови введення тесту, подати дані, згенеровані в результатах процесу до кінцевого результату, і потім порівняти результати з очікуваними. Помилки реєструються в адаптованій системі управління багами та аналізуються для визначення причини помилки та подальших дій. Нарешті, крок валідації перевіряє результати тесту, щоб визначити задоволення операцією або іншою дією.

Таким чином, тестування BudgetBee буде більше, ніж зворотним циклом виявлення недоліків і багів — це також буде спосіб показати людям, що вони можуть почати відчувати себе безпечно в тому, що вони отримують. На рисунку 4.1 зображено функціональне тестування продукту.

Таблиця 4.1 – 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**

Різні перевірки застосовуються до різних частин. Модульне тестування системи BudgetBee виконується на рівні окремих модулів. У вас буде безліч можливостей протестувати ваші розділи Google Test різними методами, просто використовуйте те, що зручно в даний момент.

Інформація в додатку до тестового протоколу передається отримувачу тестового звіту, як описано в прикладі рисунку 4.1. Це дозволяє ідеальну ситуацію перевірки програмних компонентів один проти одного для перевірки на сталість, і все інше. Одиниці можуть бути протестовані індивідуально.

Більш того, результати кожного модуля перевіряються в нашій модульній тестовій системі.

Оскільки тестова система Google Test містить пакетні тести для кожного з модулів, можливо інтегрувати всі набори тестів у Google Test і запускати їх окремо.

Зараз їх можна розглядати переважно як ізольовані сутності, і кожен модуль коду можна перевіряти окремо.

Основні цілі нашого дослідження такі:

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.

Тести були ретельно складені, щоб охопити все, що система могла б сприймати в термінах успіху/валідності та виключень/помилок.

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);

}

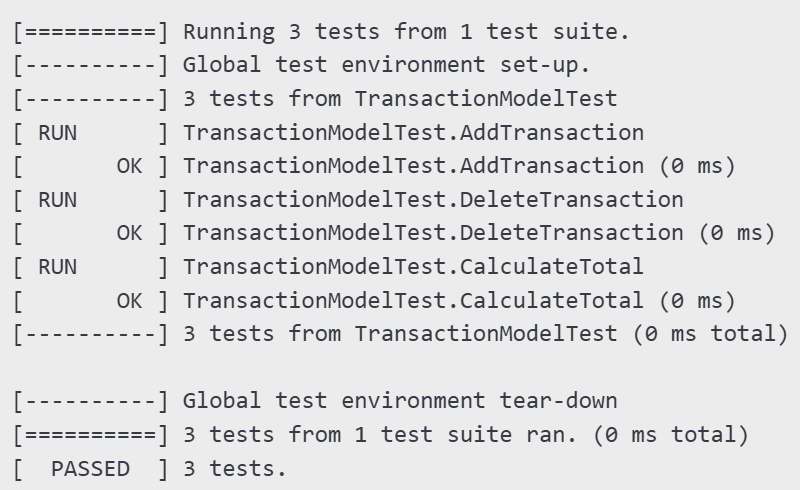


Рисунок 4.1 – Приклад результату модульного тестування

## **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 виходу результатів тестування представлений на рисунку 4.2.

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();

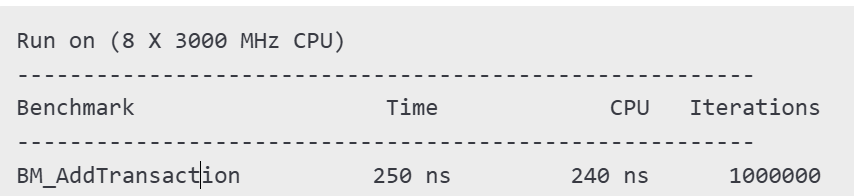


Рисунок 4.2 – приклад результату бенчмарка

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 4.3). Ми можемо змінити поточний рахунок та побачити список усіх транзакцій (рисунок 4.4). 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 4.5). Якщо натиснути на категорію, відкриється його підкатегориї (рисунок 4.6) та так рекурсивно. Ми можемо маніпулювати категоріями. Сторінка для цього зображена на рисунку 4.7. На нижньому полі зображена історія переходу категорій для більш зручного користування. При натисканні на іконку з історії, ми перекнемося на інший список з історії. На рисунку 4.8 зображена сторінка маніпуляції над транзакціями.

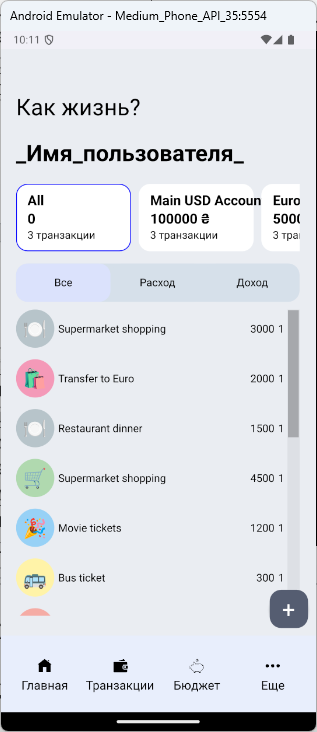


Рисунок 4.3 – Головне меню

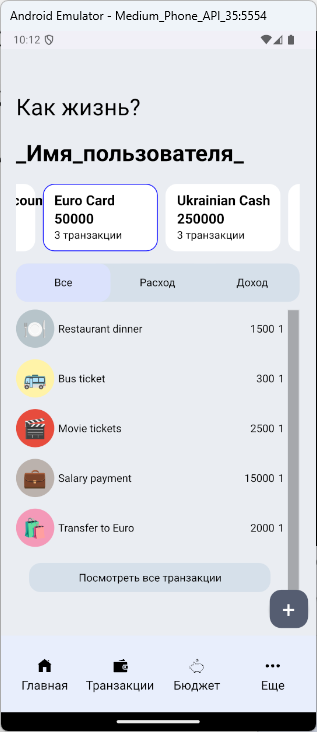


Рисунок 4.4 – Зміна поточного рахунку

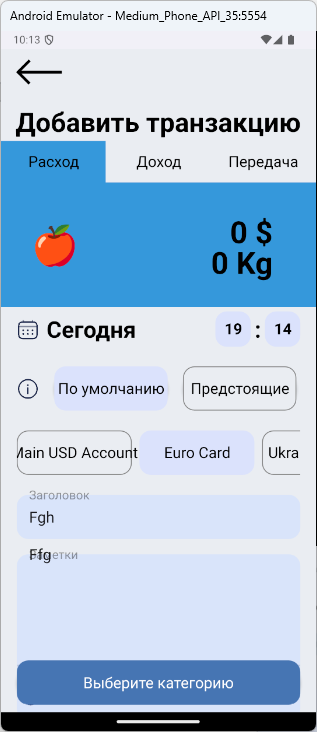


Рисунок 4.5 – Маніпуляція з транкзаціями

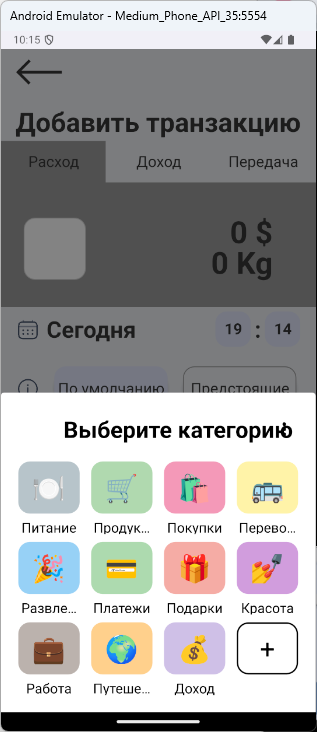


Рисунок 4.6 – Попап категорій

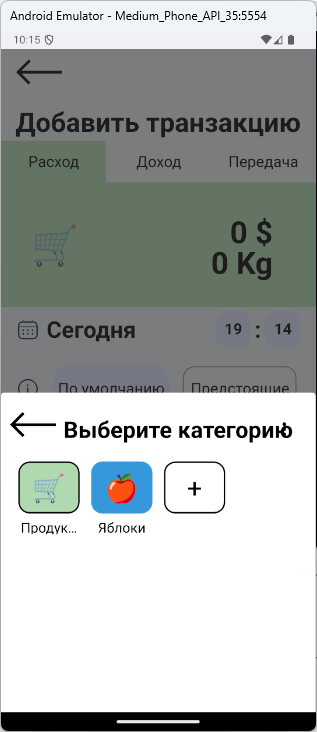


Рисунок 4.7 – Попап під категорій

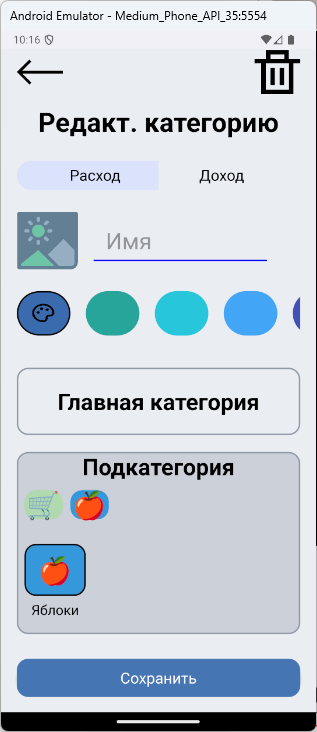


Рисунок 4.8 – Додати / видалити / редагувати категорію

**ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ**



У рамках бакалаврської кваліфікаційної роботи було успішно реалізовано програмний продукт BudgetBee, призначений для зручного та наочного управління особистими фінансами. Застосунок дозволяє користувачам фіксувати доходи й витрати, класифікувати транзакції за типами та здійснювати швидкий аналіз фінансової активності за допомогою візуалізацій і фільтрів.

У процесі розробки було проаналізовано існуючі рішення на ринку та виявлено їхні недоліки, які були враховані при проєктуванні архітектури. Технологічний стек на основі Qt/QML та C++ забезпечив кросплатформенність, високу продуктивність і гнучкість у створенні інтерфейсів користувача.

Архітектура побудована на принципах MVVM, що дозволило розділити відповідальність між компонентами системи, спростити тестування та забезпечити масштабованість. Для зберігання даних використано локальну базу SQLite, що дозволяє застосунку працювати повністю автономно.

Було впроваджено низку сучасних інженерних практик, зокрема використання шаблонів проєктування, асинхронну обробку даних та власну модель взаємодії між логікою та інтерфейсом. Особливу увагу приділено продуктивності, зручності користування та можливості подальшого розширення.

Результати виконаної роботи демонструють практичну цінність BudgetBee і підтверджують, що розроблене рішення може стати основою для наступних ітерацій — зокрема, впровадження хмарної синхронізації, підтримки кількох валют, багатокористувацького режиму та мобільної версії.

Поставлені цілі були досягнуті, а всі завдання успішно виконані.

# **ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**



1. ISO/IEC/IEEE 12207:2017 — Systems and software engineering — Software life cycle processes.
2. Sommerville I. Software Engineering. 10th ed. — Pearson Education, 2015. — 816 p.
3. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — Addison-Wesley, 1994. — 395 p.
4. Шлеєр С., Меллор С. Об'єктно-орієнтований аналіз. — Київ: Діалектика, 2002. — 432 с.
5. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. — Prentice Hall, 2017. — 432 p.
6. Войтович А.І. Проєктування програмного забезпечення: навчальний посібник. — Львів: Видавництво Львівської політехніки, 2018. — 204 с.
7. Qt Documentation — Qt for Application Development
8. Qt QML — Qt Quick Controls 2
9. SQLite Documentation
10. C++ Reference
11. Flutter Documentation
12. Pigeon: Flutter plugin for platform channels
13. Dart Language Tour
14. Козлов Д.Ю. Архітектура програмного забезпечення: навчальний посібник. — К.: НАУ, 2020. — 240 с.
15. Stroustrup B. The C++ Programming Language. 4th ed. — Addison-Wesley, 2013. — 1366 p.
16. Пантелеймоненко А.М. Моделі та засоби автоматизації проєктування інтерфейсів користувача. — К.: КПІ ім. Ігоря Сікорського, 2021. — 180 с.