

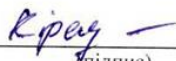
**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ**

КВАЛІФІКАЦІЙНА РОБОТА

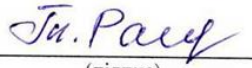
на тему: ЗАСІБ ВІДСТЕЖЕННЯ ТА УПОРЯДКУВАННЯ ДОХОДІВ І ВИТРАТ

Здобувача вищої освіти другого (магістерського) рівня,
II курсу, групи КМІз-24-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»


(підпис)

Кіреу Дмитра Руслановича
(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Ратушняк Тетяна Володимирівна,
канд. фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Омельчук Антон
Анатолійович, канд. техн. наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «09» січня 2026 № 11

Завідувач кафедри


(підпис)

Філоненко Михайло Миколайович,
канд. фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2026

Реєстрація № 37

02.02.2026
(дата)


(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на
доопрацювання)

09.04.2026
(дата)

Г. Раєв
(підпис)

Результат захисту

4 (добре)
(оцінка)

11.02.2026
(дата)


(підпис)

Голова комісії:


(підпис)

О.С. Дюбанов

Члени комісії:


(підпис)

А.Ю. Горбовий


(підпис)

М.М. Філоненко


(підпис)

І.Д. Погореловська

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	24		11.02.2026 Дюбанов О.С.
2. Дотримання вимог щодо оформлення	від 0 до 10	8		Горбовий А.Ю.
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	3		Філоненко М.М. Погореловська І.Д.
4. Рецензент	від 0 до 5	4		Суховерхня Л.П.
РАЗОМ БАЛІВ	від 0 до 50	39		

Оцінювання кваліфікаційної роботи науковим керівником - 41 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
80	B	4 (добре)	11.02.2026	Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погореловська І.Д. Суховерхня Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую

Завідувач кафедри  Філоненко М.М.

«11» грудня 2024

**Завдання на кваліфікаційну роботу
здобувачу вищої освіти**

Кіреу Дмитру Руслановичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Засіб відстеження та упорядкування доходів і витрат

Затверджена наказом ректора від «10» грудня 2024 № 1852

2. Строк здачі закінченої роботи «09» січня 2026

3. Мета роботи Розробка та реалізація десктопної системи управління особистими фінансами FinPal, яка забезпечує зручний облік фінансових операцій, аналітичну обробку даних та застосування інтелектуальних методів для автоматичної категоризації та рекомендацій щодо оптимізації витрат. Досягнення цієї мети передбачає створення продуктивного, масштабованого та інтуїтивного програмного продукту з сучасною архітектурою та високим рівнем автоматизації.

Об'єкт дослідження Процеси та інформаційні системи управління особистими фінансами, включаючи програмні засоби, методи обліку витрат і доходів, алгоритми аналізу фінансових даних та інструменти візуалізації.

Предмет дослідження Методи, архітектурні рішення та програмні засоби, використані для проєктування й реалізації системи FinPal.

Орієнтовна база дослідження Система FinPal.

4. Вихідні дані до роботи Технічні вимоги до інтелектуальної інформаційної системи моніторингу витрат;

5. Перелік графічного матеріалу Інтерфейс інформаційної системи;

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП; РОЗДІЛ 1. АНАЛІЗ ВИДІВ ДОДАТКІВ ТА СПОСОБІВ ЇХ СТВОРЕННЯ; РОЗДІЛ 2 АНАЛІЗ ПІДХОДІВ ДО ТЕСТУВАННЯ ТА ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ; РОЗДІЛ 3. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІНСТРУМЕНТІВ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ФІНАНСОВОГО ДОДАТКУ; РОЗДІЛ 4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ; ВИСНОВКИ; СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.

7. Календар план виконання роботи:

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2024	До 15.12.2024
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.05.2025	26.05.2025
3	II розділ (аналіз експериментальної частини про методики досліджень)	02.09.2025	02.09.2025
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	24.11.2025	24.11.2025
5	Висновки	18.12.2025	18.12.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	02.01.2026	02.01.2026
7	Отримання відгуку керівника	02.02.2026	02.02.2026
8	Отримання зовнішньої рецензії	05.02.2026	05.02.2026

8. Дата видачі завдання «11» грудня 2024

9. Керівник кваліфікаційної роботи


(підпис) Ратушняк Т.В.
(прізвище, ініціали)

10. Керівник освітньо-професійної програми


(підпис) Омельчук А.А.
(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти


(підпис) Кіреу Д.Р.
(прізвище, ініціали)

АНОТАЦІЯ

ЗАСІБ ВІДСТЕЖЕННЯ ТА УПОРЯДКУВАННЯ ДОХОДІВ І ВИТРАТ

У рамках виконаної роботи було проведено комплексне дослідження, проєктування та реалізацію системи управління особистими фінансами FinPal, яка поєднує в собі сучасні технології розробки програмного забезпечення, раціональні архітектурні принципи та інтелектуальні компоненти для автоматизації ключових операцій. У процесі дослідження проаналізовано підходи до створення десктопних додатків, вивчено актуальні інструменти та технології для побудови фінансових систем, після чого реалізовано повнофункціональний застосунок, здатний забезпечувати користувача інтуїтивним інтерфейсом, високою продуктивністю та глибокими аналітичними можливостями.

Кваліфікаційна робота містить 61 сторінок, 8 рисунків, список використаних джерел 47 найменувань.

Ключові слова: інформаційна система, контроль витрат, інтелектуальна система.

Рік захисту роботи 2026

ABSTRACT

TOOL FOR TRACKING AND ORDERING INCOME AND EXPENSES

As part of the work performed, a comprehensive study, design and implementation of the FinPal personal finance management system was carried out, which combines modern software development technologies, rational architectural principles and intelligent components for automating key operations. In the process of research, approaches to creating desktop applications were analyzed, relevant tools and technologies for building financial systems were studied, after which a fully functional application was implemented, capable of providing the user with an intuitive interface, high performance and deep analytical capabilities.

The qualification work contains 61 pages, 8 figures, a list of sources used 47 items.

Keywords: information system, cost control, intelligent system.

The year of defense of the work is 2026

ЗМІСТ

ВСТУП.....	3
РОЗДІЛ 1. АНАЛІЗ ВИДІВ ДОДАТКІВ ТА СПОСОБІВ ЇХ СТВОРЕННЯ	5
1.1 МЕХАНІЗМИ СТВОРЕННЯ ТА ВИДИ ДОДАТКІВ	5
1.2 СТРУКТУРА ДОДАТКУ ТА АРХІТЕКТУРНІ ПАТТЕРНИ	7
РОЗДІЛ 2 АНАЛІЗ ПІДХОДІВ ДО ТЕСТУВАННЯ ТА ЗАБЕЗПЕЧЕННЯ	
ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	10
2.1 ЖИТТЄВИЙ ЦИКЛ ТЕСТУВАННЯ ТА ЗАБЕЗПЕЧЕННЯ ЯКОСТІ.....	10
2.2 ПІРАМІДА ТЕСТУВАННЯ ТА РІВНІ ПОКРИТТЯ	13
РОЗДІЛ 3. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІНСТРУМЕНТІВ ДЛЯ	
ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ФІНАНСОВОГО ДОДАТКУ	18
3.1 ВИБІР МОВИ ПРОГРАМУВАННЯ ДЛЯ РОЗРОБКИ ДЕСКТОПНОГО ДОДАТКУ	18
3.2 ПОРІВНЯННЯ ФРЕЙМВОРКІВ ДЛЯ СТВОРЕННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ	
КОРИСТУВАЧА.....	21
3.3 ВИБІР СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ ДЛЯ ЛОКАЛЬНОГО ЗБЕРІГАННЯ .	23
3.4 АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ ВІЗУАЛІЗАЦІЇ ФІНАНСОВИХ ДАНИХ.....	28
РОЗДІЛ 4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ УПРАВЛІННЯ	
ОСОБИСТИМИ ФІНАНСАМИ.....	33
4.1 АРХІТЕКТУРА СИСТЕМИ ТА ПРОЕКТУВАННЯ КОМПОНЕНТІВ	33
4.2 РЕАЛІЗАЦІЯ МОДУЛІВ СИСТЕМИ	37
4.3 РЕАЛІЗАЦІЯ АІ-КОМПОНЕНТІВ ДЛЯ АНАЛІЗУ ФІНАНСОВИХ ДАНИХ	41
4.4 ТЕСТУВАННЯ ТА ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ СИСТЕМИ	44
ВИСНОВКИ	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	59

ВСТУП

У сучасних умовах діджиталізації та зростання фінансової активності користувачів управління особистими фінансами стає важливою частиною повсякденного життя. Збільшення кількості фінансових операцій, різноманіття джерел витрат та доходів, а також потреба в точному контролі бюджету вимагають використання спеціалізованих інструментів, здатних забезпечити ефективний аналіз фінансової інформації та підтримку прийняття рішень. Традиційні методи обліку — такі як таблиці чи ручні записи — стають недостатньо ефективними через свою обмеженість, складність оновлення та відсутність автоматизації.

Метою дослідження є розробка та реалізація десктопної системи управління особистими фінансами FinPal, яка забезпечує зручний облік фінансових операцій, аналітичну обробку даних та застосування інтелектуальних методів для автоматичної категоризації та рекомендацій щодо оптимізації витрат. Досягнення цієї мети передбачає створення продуктивного, масштабованого та інтуїтивного програмного продукту з сучасною архітектурою та високим рівнем автоматизації.

Об'єктом дослідження є процеси та інформаційні системи управління особистими фінансами, включаючи програмні засоби, методи обліку витрат і доходів, алгоритми аналізу фінансових даних та інструменти візуалізації.

Предметом дослідження є методи, архітектурні рішення та програмні засоби, використані для проєктування й реалізації системи FinPal, зокрема:

- ✓ архітектурний патерн MVP;
- ✓ методи оптимізації бази даних SQLite;
- ✓ засоби побудови інтерфейсу на базі CustomTkinter;
- ✓ алгоритми автоматичної категоризації транзакцій та рекомендацій на основі локальних AI-моделей;
- ✓ технології аналітичної обробки, візуалізації та резервного копіювання фінансових даних.

У відповідь на ці виклики було розроблено систему FinPal — десктопний застосунок для комплексного управління особистими фінансами. FinPal поєднує у собі зручний інтерфейс, сучасну архітектуру та інтелектуальні можливості, що дає змогу користувачам не лише вести облік транзакцій, а й отримувати глибоку аналітику, виявляти фінансові закономірності та оптимізувати власні витрати. Завдяки інтеграції AI-компонентів система допомагає автоматизувати рутинні дії, зокрема категоризацію транзакцій та формування рекомендацій на основі індивідуальних фінансових моделей.

Під час розробки застосунку використано архітектурний патерн MVP, який забезпечив чітке розділення відповідальностей, високу модульність та зручність подальшого розширення. Для інтерфейсу обрано CustomTkinter, що дозволило створити сучасний та інтуїтивний дизайн, орієнтований на комфорт користувача. Зберігання даних реалізовано на основі SQLite, що забезпечує надійність, автономність та високу продуктивність при роботі з великими обсягами інформації.

FinPal вирішує широкий спектр задач: від додавання та структуризації транзакцій до побудови графіків, аналізу витрат і резервного копіювання даних. Усі функції системи розроблені з урахуванням продуктивності, надійності та простоти використання, що робить застосунок ефективним інструментом для персонального фінансового менеджменту.

Створення цього застосунку стало можливістю дослідити сучасні підходи до розробки програмного забезпечення, апробувати інтелектуальні алгоритми для аналізу фінансових даних та продемонструвати практичне застосування архітектурних і технологічних рішень у реальному продукті. Отримані результати підтверджують, що FinPal є масштабованим, продуктивним та інтелектуально гнучким рішенням, здатним стати надійним помічником у сфері управління особистими фінансами.

РОЗДІЛ 1. АНАЛІЗ ВИДІВ ДОДАТКІВ ТА СПОСОБІВ ЇХ СТВОРЕННЯ

1.1 Механізми створення та види додатків

Сучасний ринок програмного забезпечення пропонує величезну різноманітність додатків, кожен з яких має свої особливості, переваги та недоліки. Розуміння різних типів додатків та механізмів їх створення є критично важливим для прийняття обґрунтованих рішень при розробці програмного продукту. Додатки можна класифікувати за різними критеріями: за платформою виконання, за архітектурою, за способом розподілу, за призначенням та іншими характеристиками.

Веб-додатки є одним з найпоширеніших типів програмного забезпечення в сучасному світі. Вони працюють через веб-браузер та не потребують встановлення на комп'ютер користувача, що робить їх доступними з будь-якого пристрою з інтернет-з'єднанням. Веб-додатки можуть бути статичними, де весь контент завантажується одночасно, або динамічними, де контент генерується на сервері залежно від запитів користувача. Сучасні веб-додатки часто використовують архітектуру SPA (Single Page Application), де вся логіка інтерфейсу виконується на клієнті через JavaScript, а сервер надає тільки дані через API. Це забезпечує швидку роботу та зручність використання, але потребує постійного інтернет-з'єднання та може мати обмеження у функціональності через обмеження браузера.

Десктопні додатки встановлюються безпосередньо на комп'ютер користувача та працюють як окремі програми операційної системи. Вони мають повний доступ до ресурсів системи, можуть працювати без інтернет-з'єднання та забезпечують кращу продуктивність порівняно з веб-додатками. Десктопні додатки можуть бути нативними, написаними спеціально для конкретної операційної системи з використанням її API, що забезпечує максимальну продуктивність та інтеграцію з системою, але потребує окремої розробки для

кожної платформи. Крос-платформенні десктопні додатки використовують фреймворки як Electron, Qt або Java, що дозволяє створювати один код для різних платформ, але може мати обмеження у продуктивності або інтеграції з системою.

Мобільні додатки розробляються для смартфонів та планшетів та можуть бути нативними для конкретної платформи iOS або Android, гібридними з використанням веб-технологій, або крос-платформенними через фреймворки як React Native або Flutter. Нативні мобільні додатки забезпечують найкращу продуктивність та інтеграцію з пристроєм, але потребують окремої розробки для кожної платформи. Гібридні додатки використовують веб-технології всередині нативного контейнера, що дозволяє використовувати один код для різних платформ, але може мати обмеження у функціональності та продуктивності.

Для фінансових додатків вибір типу додатку залежить від конкретних вимог та обмежень. Веб-додатки забезпечують доступність з будь-якого пристрою та не потребують встановлення, але потребують інтернет-з'єднання та можуть мати обмеження у безпеці через необхідність передачі даних через мережу. Десктопні додатки забезпечують повну приватність даних через локальне зберігання, можуть працювати без інтернету та мають кращу продуктивність, але потребують встановлення та обмежені конкретною платформою. Мобільні додатки забезпечують зручність використання на ходу та можливість швидкого введення транзакцій, але мають обмеження у складності інтерфейсу та обробці великих об'ємів даних.

Механізми створення додатків включають різні підходи до архітектури та розробки. Монолітна архітектура об'єднує всі компоненти в одному додатку, що спрощує розробку та розгортання, але ускладнює масштабування та підтримку. Мікросервісна архітектура розділяє додаток на окремі сервіси, що забезпечує гнучкість та масштабованість, але ускладнює розробку та потребує додаткової інфраструктури. Клієнт-серверна архітектура розділяє інтерфейс користувача та логіку обробки даних, що забезпечує централізоване управління даними та можливість оновлення без зміни клієнта, але потребує стабільного з'єднання між клієнтом та сервером.

Для локального фінансового додатку оптимальним є десктопний додаток з локальним зберіганням даних, що забезпечує повну приватність, роботу без інтернету та швидку продуктивність. Архітектура MVP забезпечує чітке розділення компонентів для легкої підтримки та тестування, що критично важливо для забезпечення якості та надійності фінансового додатку.

1.2 Структура додатку та архітектурні паттерни

Структура додатку визначає організацію коду, взаємодію між компонентами та загальну архітектуру системи. Правильно спроектована структура забезпечує легкість розробки, підтримки та розширення функціональності, тоді як погана структура може призвести до проблем з підтримкою, тестуванням та розширенням проєкту. Архітектурні паттерни надають перевірені рішення для організації коду та забезпечення якості програмного забезпечення.

MVC (Model-View-Controller) є одним з найпоширеніших архітектурних паттернів для розробки додатків. Model відповідає за дані та бізнес-логіку, зберігаючи стан додатку та забезпечуючи доступ до даних. View відповідає за відображення інтерфейсу користувача, показуючи дані з Model та передаючи дії користувача до Controller. Controller координує взаємодію між Model та View, обробляючи події користувача та оновлюючи Model та View відповідно. Такий підхід забезпечує розділення відповідальностей та легкість тестування компонентів незалежно один від одного.

MVP (Model-View-Presenter) є варіантом MVC з більш чітким розділенням між View та Presenter. У MVP View є пасивним компонентом, який тільки відображає дані та передає події до Presenter, не містивши бізнес-логіки. Presenter містить всю логіку обробки подій та координації між Model та View, що робить View легшим для тестування та заміни. Model залишається незмінним порівняно з MVC, відповідаючи за дані та бізнес-логіку. Такий підхід особливо ефективний

для десктопних додатків, де View компоненти можуть бути складними та потребувати легкої заміни.

MVVM (Model-View-ViewModel) є паттерном, який особливо популярний для веб-та мобільних додатків. ViewModel виступає як проміжний шар між View та Model, надаючи дані у форматі, зручному для View, та обробляючи команди від View для оновлення Model. ViewModel часто використовує data binding для автоматичного оновлення View при зміні даних, що спрощує синхронізацію між даними та інтерфейсом. Такий підхід особливо ефективний для складних інтерфейсів з багатьма взаємопов'язаними компонентами.

Структура проєкту визначає організацію файлів та модулів у проєкті. Модульна структура розділяє код на окремі модулі за функціональністю, що забезпечує легкість навігації та підтримки. Кожен модуль відповідає за конкретну функціональність, має чітко визначені інтерфейси для взаємодії з іншими модулями та може бути тестований незалежно. Така структура особливо важлива для великих проєктів, де кілька розробників працюють над різними частинами системи.

Для фінансового додатку FinPal обрано архітектуру MVP, яка забезпечує оптимальний баланс між простотою та гнучкістю. Model Layer включає Database клас для роботи з даними та AI компоненти для інтелектуальної обробки. View Layer складається з окремих View компонентів для кожної функціональності - DashboardView, TransactionsView, AnalyticsView та інші. Presenter Layer представлений MainController, який координує взаємодію між View та Model. Така структура забезпечує легкість додавання нових функцій, тестування компонентів та підтримку коду в довгостроковій перспективі.

Висновки до розділу 1

Проведений аналіз видів додатків та механізмів їх створення показав різноманітність підходів до розробки програмного забезпечення. Веб-додатки забезпечують доступність та не потребують встановлення, але потребують інтернет-з'єднання та можуть мати обмеження у безпеці. Десктопні додатки забезпечують повну приватність, роботу без інтернету та кращу продуктивність,

що критично важливо для фінансових додатків. Мобільні додатки забезпечують зручність використання на ходу, але мають обмеження у складності інтерфейсу. Архітектурні паттерни MVC, MVP та MVVM надають перевірені рішення для організації коду та забезпечення якості. Для фінансового додатку обрано десктопну архітектуру з MVP паттерном, що забезпечує оптимальний баланс між функціональністю, безпекою та продуктивністю.

РОЗДІЛ 2 АНАЛІЗ ПІДХОДІВ ДО ТЕСТУВАННЯ ТА ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Життєвий цикл тестування та забезпечення якості

Тестування становить фундаментальний етап життєвого циклу розробки програмного забезпечення (SDLC), оскільки воно безпосередньо корелює з показниками надійності та функціональної відповідності продукту встановленим специфікаціям. Рациональна організація верифікаційних процесів сприяє детекції дефектів на початкових стадіях проектування, що суттєво мінімізує витрати на їх усунення. Структурований життєвий цикл тестування (STLC) охоплює комплекс послідовних фаз — від стратегічного планування до пост-релізного супроводу, кожна з яких базується на специфічних методологіях та інструментарії.

Планування тестування є першим етапом життєвого циклу, де визначаються цілі, обсяг, стратегія та ресурси для тестування. На цьому етапі аналізуються вимоги до системи, визначаються критичні функції, які потребують особливої уваги, та планується покриття тестами різних компонентів системи. Стратегія тестування визначає підходи до різних типів тестів - функціональних, інтеграційних, системних та приймальних, а також пріоритети тестування залежно від важливості функцій. Планування також включає визначення критеріїв готовності до релізу, які визначають мінімальний рівень якості, необхідний для випуску продукту.

Етап розробки тестів (Test Design) полягає у формуванні структурованих тестових сценаріїв та кейсів на базі аналізу технічних вимог і специфікацій. Тестові сценарії (Test Scenarios) визначають логічну послідовність дій для верифікації окремих функціональних модулів, охоплюючи преумови (preconditions), ітераційні кроки та постумови. У свою чергу, тестові випадки (Test Cases) забезпечують максимальну деталізацію перевірок, фіксуючи

конкретні вхідні параметри та еталонні очікувані результати. Ця фаза також передбачає генерацію релевантних наборів тестових даних та конфігурацію тестового середовища, архітектура якого має максимально дублювати умови реальної експлуатації системи.

Виконання тестів є одним із ключових етапів процесу забезпечення якості програмного забезпечення. На цьому етапі здійснюється прогін підготовлених тестових сценаріїв та фіксація отриманих результатів. Тестувальники виконують кожен сценарій відповідно до описаних кроків, після чого порівнюють фактичну поведінку системи з очікуваною, визначеною в документації або специфікації вимог. Це дозволяє виявити можливі розбіжності або аномалії в роботі продукту.

У ході тестування всі знайдені відхилення від очікуваного результату реєструються у вигляді звітів про дефекти. Такий звіт містить детальний опис проблеми, чіткі кроки для її відтворення, інформацію про середовище, у якому виникає помилка, а також очікувану та фактичну поведінку системи. Окрім цього, кожному дефекту присвоюються пріоритет та рівень серйозності, що допомагає команді розробників правильно планувати виправлення та визначати критичність проблеми для кінцевого користувача. Систематична реєстрація результатів тестування робить можливим відстеження прогресу, аналіз тенденцій у виявленні дефектів та прийняття рішень щодо подальших кроків команди.

Аналіз результатів та звітність є наступним важливим етапом, який зосереджений на глибокому вивченні результатів тестування та формуванні підсумкової інформації про якість програмного продукту. На цьому етапі оцінюється рівень покриття функціоналу тестами, визначаються модулі, у яких спостерігається найбільша концентрація помилок, та аналізуються ризики, пов'язані з випуском продукту. Команда тестування формує рекомендації щодо готовності системи до релізу та пропонує напрями для удосконалення якості.

Формування звітності включає підрахунок кількості виявлених дефектів, аналіз їхнього розподілу за рівнями серйозності та пріоритетами, а також оцінку швидкості та ефективності процесу виправлення помилок. Окрім цього,

надається загальна характеристика якості продукту, яка допомагає менеджерам приймати рішення щодо релізу або необхідності проведення додаткових ітерацій тестування й доопрацювання функціоналу. Така аналітична інформація є цінною для планування майбутніх етапів розробки та оптимізації процесів.

Підтримка тестів — це завершальний, але не менш важливий етап життєвого циклу тестування. Оскільки вимоги й функціональність програмного забезпечення постійно змінюються, виникає необхідність регулярно оновлювати тестову базу. Коли до системи додаються нові функції, створюються відповідні нові тестові сценарії. Якщо ж існуючі функції модифікуються, необхідно актуалізувати й ті тести, які їх покривають, щоб вони коректно відображали нову поведінку системи.

Підтримка тестів також передбачає їх рефакторинг — удосконалення структури, підвищення зрозумілості та зручності супроводу. Це може включати оптимізацію кроків, скорочення дублювання, упорядкування тестових даних та видалення застарілих або неактуальних тестових випадків. Додатково проводиться оптимізація часу виконання тестів, що дозволяє прискорити зворотний зв'язок для команди розробки та підвищити загальну ефективність процесу тестування.

Таким чином, виконання тестів, аналіз результатів та підтримка тестів є взаємопов'язаними етапами, які забезпечують стабільну якість програмного продукту. Вони дозволяють своєчасно виявляти та усувати помилки, підтримувати актуальність тестової документації та формувати обґрунтовані висновки щодо готовності системи до релізу.

Для фінансового додатку тестування є особливо критичним через важливість точності обчислень та безпеки даних. Функціональне тестування перевіряє коректність обчислення балансу, статистики та прогнозів, що критично для фінансового додатку, де помилки в обчисленнях можуть призвести до неправильних рішень користувача. Тестування безпеки перевіряє захист даних від несанкціонованого доступу, коректність резервного копіювання та відновлення даних, що критично для забезпечення конфіденційності фінансової

інформації. Тестування продуктивності перевіряє швидкість роботи при великих об'ємах даних, що забезпечує зручність використання навіть при накопиченні великої кількості транзакцій.

2.2 Піраміда тестування та рівні покриття

Піраміда тестування — це концептуальна модель, що визначає оптимальне співвідношення різних рівнів тестування в межах програмного проєкту. Вона допомагає забезпечити високу якість продукту при раціональному використанні часу та ресурсів команди. Модель передбачає, що більшість тестів має знаходитись на нижньому рівні, а найменша їх частина — на верхньому. Класична піраміда включає три основні рівні: unit-тести, інтеграційні тести та end-to-end (E2E) тести. Кожен рівень виконує свою роль у виявленні дефектів, має власні особливості, складність написання та цінність для проєкту. Правильний баланс між цими рівнями дає змогу сформувати стійкий, надійний та ефективний процес тестування.

Unit-тести розташовуються на нижньому, найбільш широкому рівні піраміди, оскільки саме їх має бути найбільше. Вони тестують невеликі частини системи — окремі функції, методи або класи — у повній ізоляції від інших компонентів. Такий підхід дозволяє швидко виявляти помилки в логіці на ранніх стадіях розробки.

Unit-тести є найоперативнішими, оскільки виконуються буквально за частки секунди, і саме це робить їх надзвичайно цінними для розробників. Вони забезпечують миттєвий зворотний зв'язок про те, чи коректно працює код після внесення змін, дозволяють безпечно проводити рефакторинг і підтримувати високу стабільність системи.

Цей тип тестів також ефективний для перевірки:

- ✓ бізнес-логіки;
- ✓ математичних обчислень;

- ✓ обробки помилок;
- ✓ граничних сценаріїв;
- ✓ валідації вхідних даних.

У фінансових застосунках unit-тести набувають особливої важливості. Наприклад, вони критично необхідні для перевірки точності розрахунків балансу, статистичних показників, курсових змін чи прогнозних моделей. Будь-яка помилка на цьому рівні може призвести до неправильних аналітичних висновків, що в реальному фінансовому середовищі може мати значні негативні наслідки.

Інтеграційні тести займають середній рівень піраміди, оскільки їх кількість повинна бути меншою, ніж unit-тестів, але більшою, ніж E2E. Основна їхня мета полягає у перевірці взаємодії між компонентами системи. Якщо unit-тести гарантують правильність роботи окремих частин, то інтеграційні тести перевіряють, чи правильно вони працюють разом.

Такі тести дозволяють знаходити помилки, які виникають не всередині окремого модуля, а саме в процесі взаємодії — наприклад, коли передаються неправильні дані між класами або API, або коли одна частина системи очікує інший формат інформації, ніж фактично отримує.

Інтеграційні тести складніші в реалізації, оскільки вимагають налаштування оточення: роботи з базою даних, моків зовнішніх сервісів, або реальних компонентів, що взаємодіють між собою. Вони також виконуються повільніше, але дають більш цінну інформацію про цілісність системи.

У контексті фінансового застосунку інтеграційні тести особливо важливі для перевірки таких процесів, як:

- ✓ взаємодія між Database та інтерфейсними компонентами View;
- ✓ передача та обробка даних між модулями, що відповідають за штучний інтелект та базою даних;
- ✓ коректність оновлення UI після зміни даних у системі;
- ✓ правильність синхронізації та оновлення статистичних показників.

Інтеграційні тести допомагають гарантувати, що система працює узгоджено, а всі модулі передають і приймають дані у відповідності до вимог проєкту.

End-to-end тести є верхнім рівнем піраміди та тестують повний сценарій використання системи від початку до кінця. Вони імітують реальну поведінку користувача та перевіряють коректність роботи всієї системи разом. End-to-end тести є найповільнішими у виконанні та найскладнішими у написанні та підтримці, але надають найбільшу впевненість у коректності роботи системи з точки зору користувача. Для фінансового додатку end-to-end тести перевіряють повні сценарії використання, такі як додавання транзакції, перегляд статистики, генерація звітів та резервне копіювання, забезпечуючи що всі компоненти працюють разом коректно.

Оптимальний розподіл тестів у піраміді забезпечує баланс між швидкістю виконання, покриттям тестами та впевненістю у якості. Велика кількість unit тестів забезпечує швидкий feedback та детальне тестування логіки, середня кількість інтеграційних тестів забезпечує перевірку взаємодії компонентів, а невелика кількість end-to-end тестів забезпечує перевірку критичних сценаріїв використання. Такий розподіл забезпечує ефективне використання ресурсів та швидке виявлення проблем на різних рівнях системи.

Покриття тестами вимірює відсоток коду, який виконується під час тестування, та допомагає визначити зони, які потребують додаткового тестування. Високе покриття тестами забезпечує більшу впевненість у якості коду, але не гарантує відсутність помилок, оскільки тести можуть не перевіряти всі можливі сценарії використання. Для фінансового додатку критично важливі компоненти, такі як обчислення балансу, статистики та прогнозів, повинні мати високе покриття тестами для забезпечення точності обчислень.

Для фінансового додатку FinPal тестування організовано з урахуванням піраміди тестування. Unit тести перевіряють окремі методи Database класу, алгоритми AI компонентів та утиліти для обробки даних. Інтеграційні тести перевіряють взаємодію між компонентами, таку як додавання транзакції через

TransactionsView та оновлення статистики в DashboardView. End-to-end тести перевіряють повні сценарії використання, такі як створення резервної копії та відновлення даних, генерація звітів та експорт графіків. Така організація тестування забезпечує комплексну перевірку якості системи на всіх рівнях.

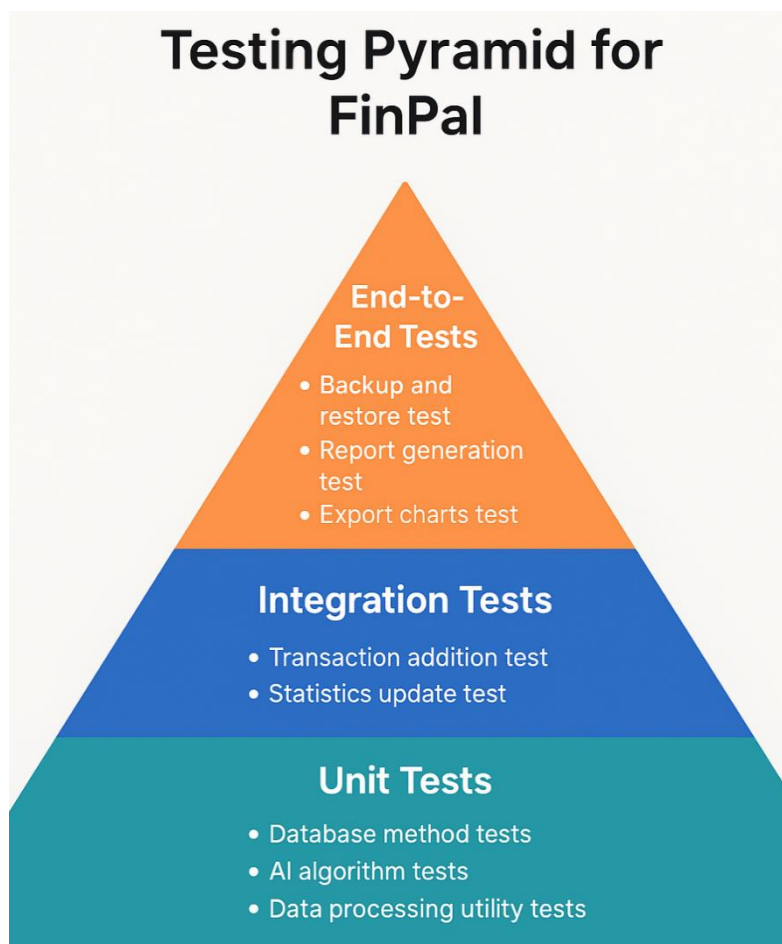


Рис. 1.1. Тестування

Висновки до розділу 2

Проведений аналіз підходів до тестування показав важливість правильно організованого процесу тестування для забезпечення якості програмного забезпечення. Життєвий цикл тестування включає планування, розробку тестів, виконання, аналіз результатів та підтримку, кожен етап якого має критичне значення для забезпечення якості. Піраміда тестування надає оптимальний розподіл різних типів тестів для ефективного тестування при мінімальних витратах. Unit тести забезпечують швидкий feedback та детальне тестування логіки, інтеграційні тести перевіряють взаємодію компонентів, а end-to-end тести

забезпечують перевірку критичних сценаріїв використання. Для фінансового додатку тестування є особливо критичним через важливість точності обчислень та безпеки даних, тому необхідно забезпечити високе покриття тестами критичних компонентів та комплексне тестування на всіх рівнях системи.

РОЗДІЛ 3. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІНСТРУМЕНТІВ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ФІНАНСОВОГО ДОДАТКУ

3.1 Вибір мови програмування для розробки десктопного додатку

Вибір мови програмування є одним із ключових стратегічних рішень під час розробки будь-якого програмного продукту, оскільки саме він визначає технічні можливості системи, швидкість її створення, простоту підтримки та потенціал для масштабування у майбутньому. Для десктопного застосунку з управління особистими фінансами це рішення є особливо важливим, адже такі системи мають специфічні вимоги: надійне локальне зберігання даних, високу швидкість обробки фінансових операцій, інтеграцію з інструментами для аналізу даних та машинного навчання, а також можливість швидко вносити зміни у структуру бізнес-логіки. Усе це потребує вибору мови, яка поєднує продуктивність, гнучкість та простоту розробки.

Однією з причин, чому Python став оптимальним вибором для розробки фінансового застосунку, є його поєднання простоти та функціональної потужності. Мова має надзвичайно лаконічний та зрозумілий синтаксис, що дозволяє розробникам фокусуватися на вирішенні логічних задач, а не на технічних деталях реалізації. Це є важливою перевагою для систем, які постійно розширюються, де бізнес-логіка регулярно уточнюється та доповнюється. Зменшення обсягу коду позитивно впливає на швидкість розробки, а також спрощує підтримку та внесення змін навіть у великих командах чи в проєктах, де розробники можуть часто змінюватися.

Важливою перевагою Python є одна з найбагатших екосистем бібліотек серед сучасних мов програмування. Для фінансового застосунку це означає, що практично будь-яке завдання — від роботи з даними до побудови графіків або навчання моделей — можна реалізувати, використовуючи готові та перевірені інструменти.

Зокрема:

Pandas забезпечує зручні структури даних та методи для операцій із таблицями, що є незамінним при роботі з транзакціями, категоріями витрат та фінансовими звітами.

NumPy надає доступ до швидких математичних і статистичних функцій, що прискорює розрахунки та аналітику.

Scikit-learn дозволяє реалізувати машинне навчання для прогнозування витрат, категоризації транзакцій або побудови рекомендацій, не створюючи алгоритми вручну.

Matplotlib та Seaborn забезпечують інструменти для створення графіків та різноманітних видів візуалізацій, що є невід'ємною частиною сучасних фінансових застосунків.

Таким чином, Python дає можливість швидко створити функціональний та гнучкий фінансовий інструмент без значних витрат на розробку власних аналітичних чи візуальних модулів.

Порівняння Python із Java показує суттєві відмінності, які впливають на вибір мови залежно від типу проєкту. Java дійсно демонструє високу продуктивність та масштабованість завдяки роботі на віртуальній машині (JVM) та розвинутим механізмам оптимізації. Це робить її ефективним вибором для великих корпоративних рішень, серверних систем або високонавантажених платформ. Однак разом із цими перевагами Java характеризується складнішим синтаксисом та більшою кількістю необхідного «шаблонного» коду. Реалізація навіть базових функцій часто вимагає значно більше часу та зусиль, що може уповільнювати розробку.

Крім того, екосистема Java, хоча й потужна для корпоративних задач, менш орієнтована на швидку аналітику, машинне навчання та обробку даних. Бібліотеки існують, але вони менш зручні та не такі численні, як у Python. Для локального десктопного застосунку, який повинен бути гнучким, легко розширюваним і швидко оновлюваним, переваги Python у швидкості розробки, зручності та універсальності переважають над продуктивністю Java. Краща

продуктивність Java не виправдовує збільшення часу створення, складність підтримки та трудомісткість проєктної конфігурації.

У результаті Python є оптимальним вибором для фінансового додатку, де важливо поєднати швидку розробку, зручність підтримки, широкі аналітичні можливості та доступ до сучасних інструментів обробки даних. C# та .NET Framework також пропонують хорошу продуктивність та інтеграцію з Windows, але обмежені платформою Microsoft та потребують більше коду порівняно з Python. Розробка GUI на C# через WPF або WinForms є потужною, але потребує значно більше часу на створення складних інтерфейсів. Екосистема бібліотек для аналітики та машинного навчання в C# менш розвинена порівняно з Python.

JavaScript з Electron дозволяє створювати крос-платформенні додатки з використанням веб-технологій, що дає можливість використовувати React або Vue для інтерфейсу. Однак Electron додатки потребують значно більше ресурсів пам'яті, працюють повільніше через додатковий шар абстракції та мають більший розмір виконуваних файлів. Для локального додатку, де важлива ефективність використання ресурсів, Electron може бути надлишковим.

C++ забезпечує максимальну продуктивність та контроль над ресурсами, але розробка займає значно більше часу через складність мови, необхідність ручного управління пам'яттю та відсутність готових бібліотек для швидкої розробки GUI та аналітики. Для фінансового додатку, де продуктивність Python є достатньою, використання C++ не виправдане через збільшення часу розробки.

Версія Python 3.10 або новіша забезпечує сучасні можливості мови, які критично важливі для створення надійного коду. Pattern matching дозволяє елегантно обробляти різні типи даних, покращена обробка помилок через структурні винятки спрощує налагодження, типізація через type hints дозволяє виявляти помилки на етапі розробки, а покращення продуктивності у порівнянні з попередніми версіями забезпечує швидку роботу додатку. Для фінансового додатку це означає можливість створювати код, який легко підтримувати та розширювати, при цьому забезпечуючи достатню швидкість роботи.

Інтеграція AI-компонентів через бібліотеки машинного навчання є критичною перевагою Python. Scikit-learn надає готові алгоритми класифікації, які можна використовувати для автоматичної категоризації транзакцій. TensorFlow та PyTorch дозволяють створювати складні нейронні мережі для більш точного прогнозування, хоча для початкового проєкту це може бути надлишковим. Бібліотеки для обробки природної мови, такі як NLTK або spaCy, можуть використовуватися для аналізу описів транзакцій та визначення категорій на основі семантики тексту.

3.2 Порівняння фреймворків для створення графічного інтерфейсу користувача

Графічний інтерфейс користувача є критично важливим компонентом фінансового додатку, оскільки він безпосередньо впливає на досвід користувача та ефективність роботи з системою. Погано спроектований інтерфейс може зробити навіть найпотужніший додаток незручним у використанні, тоді як добре спроектований інтерфейс може значно підвищити продуктивність користувача та задоволення від використання продукту. Вибір фреймворку для створення GUI визначає не тільки зовнішній вигляд додатку, але й швидкість розробки, можливості кастомізації, продуктивність та складність підтримки коду.

CustomTkinter є сучасним розширенням стандартної бібліотеки Tkinter для Python, яке зберігає простоту використання оригінального Tkinter, але додає сучасний дизайн у стилі Material Design. Tkinter є частиною стандартної бібліотеки Python, що означає відсутність необхідності встановлення додаткових залежностей, що спрощує розгортання додатку. CustomTkinter розширює можливості Tkinter, додаючи сучасні компоненти з кращим виглядом та більшою гнучкістю налаштування.

Переваги CustomTkinter включають простоту інтеграції з існуючим Python кодом - розробник може використовувати знайомий синтаксис Python без

необхідності вивчення нових мов розмітки або складних архітектурних паттернів. Легка кастомізація компонентів дозволяє швидко адаптувати інтерфейс під конкретні потреби проєкту. Підтримка темної та світлої теми забезпечує зручність використання в різних умовах освітлення та відповідає сучасним стандартам дизайну. Хороша продуктивність для десктопних додатків забезпечує плавну роботу інтерфейсу навіть при обробці великих об'ємів даних.

Для фінансового додатку важливою є можливість швидко створювати складні форми введення даних з валідацією, таблиці транзакцій з пагінацією та сортуванням, інтерактивні графіки для візуалізації даних та панелі навігації для швидкого перемикавання між розділами. CustomTkinter надає всі необхідні компоненти для реалізації цих функцій з мінімальними зусиллями.

Порівняння з PyQt5 та PySide6 показує, що останні пропонують більше можливостей та професійніший вигляд завдяки використанню нативного рендерингу через Qt framework. PyQt має більш багатий набір компонентів, включаючи складні віджети для роботи з датами, таблицями та графіками. Однак PyQt потребує значно більше часу на розробку через складнішу архітектуру та необхідність вивчення Qt-specific концепцій. PyQt також має ліцензійні обмеження для комерційного використання - PyQt5 потребує придбання ліцензії для комерційних проєктів, тоді як PySide6 є безкоштовним, але має меншу підтримку спільноти.

Kivy пропонує крос-платформенність та сучасний дизайн з підтримкою мультитач-жестів, що робить його ідеальним для мобільних додатків. Однак Kivy орієнтований більше на мобільні додатки та має інший підхід до розробки інтерфейсів через власну мову розмітки KV. Для десктопного Windows-додатку Kivy може бути надлишковим та складнішим у використанні порівняно з CustomTkinter. Продуктивність Kivy на десктопі також може бути нижчою через додаткові шари абстракції.

GTK+ через PyGObject також є опцією для створення десктопних додатків, особливо для Linux систем. GTK+ надає потужні компоненти та добре інтегрований з операційними системами на базі Linux. Однак на Windows GTK+

має складнішу інсталяцію, потребує додаткових залежностей та має меншу підтримку спільноти порівняно з Tkinter-екосистемою. Для Windows-додатку це робить GTK+ менш привабливим вибором.

Для фінансового додатку FinPal CustomTkinter обрано через оптимальний баланс між простотою розробки, сучасним виглядом та продуктивністю. Особливо важливою є можливість швидко створювати складні інтерфейси з багатьма компонентами без необхідності написання великої кількості коду. STkScrollableFrame дозволяє ефективно відображати великі списки транзакцій з прокруткою, STkEntry та STkTextbox забезпечують зручне введення даних з валідацією, STkButton та STkLabel надають сучасний вигляд інтерфейсу, а STkTabview дозволяє організувати складну навігацію між розділами.

3.3 Вибір системи управління базами даних для локального зберігання

Вибір системи керування базами даних (СКБД) є одним із найважливіших технічних рішень під час розробки фінансового застосунку, оскільки саме база даних відповідає за довгострокове та надійне зберігання всієї чутливої фінансової інформації користувача. Будь-які помилки, пошкодження або втрата даних можуть мати серйозні наслідки, тому вимоги до стабільності, цілісності та безпеки СКБД є критично високими. Для десктопного програмного забезпечення, яке працює автономно та не залежить від мережевих ресурсів, особливо важливо обрати файлову базу даних, що не потребує зовнішнього сервера та не вимагає від користувача технічних знань для встановлення або налаштування.

Одним із найоптимальніших варіантів у такому контексті є SQLite — легка, вбудована реляційна база даних, яка зберігає всю інформацію в одному файлі. Такий підхід повністю відповідає потребам локального фінансового застосунку. SQLite працює без виділеного серверного процесу, функціонуючи як

бібліотека в межах самого додатку. Це робить процес розгортання максимально простим: користувачу достатньо запустити програму, і всі функції бази даних працюватимуть автоматично, без додаткових залежностей або налаштувань. Окрім цього, вбудована модель забезпечує мінімальне використання ресурсів системи, що є важливою перевагою для настільних програм.

SQLite пропонує низку переваг, які роблять її ідеальним вибором саме для фінансових застосунків.

По-перше, створення резервної копії є надзвичайно простим — достатньо скопіювати головний файл бази даних, щоб отримати повний бекап. Для користувача це означає можливість легко відновити дані в разі помилок, пошкодження файлу чи перенесення програми на інший пристрій.

По-друге, SQLite забезпечує повну транзакційну підтримку ACID, що гарантує цілісність та коректність даних навіть у випадку раптового завершення роботи програми, збою живлення або помилок під час запису.

По-третє, її продуктивність достатня для роботи з тисячами або десятками тисяч записів, що цілком покриває типові обсяги даних у персональних фінансових системах.

По-четверте, підтримка стандартизованої мови SQL дозволяє виконувати складні запити, сортування, групування та аналітичні вибірки, які є важливими для побудови статистики, графіків та динамічних звітів у фінансовому додатку.

Порівняння SQLite із серверними системами, такими як PostgreSQL чи MySQL, демонструє суттєві відмінності у призначенні та застосуванні. Хоча ці СКБД значно потужніші, підтримують паралельні підключення, масштабовані транзакції, складні схеми оптимізації та розподілені конфігурації, їх використання в локальному десктопному застосунку є недоцільним. Вони потребують встановлення сервера, налаштування користувачів і прав доступу, конфігурації служб безпеки, а також постійного моніторингу. Такі вимоги суперечать основній ідеї настільного застосунку — забезпечити простоту, автономність і мінімальну залежність від зовнішніх сервісів. Тому переваги

серверних СКБД у цьому випадку не компенсують додаткової складності розгортання та підтримки.

Ще один можливий варіант — Microsoft Access, який також використовує файлову модель зберігання даних. Проте у контексті Python застосунків Access має суттєві недоліки: відсутність нативної підтримки, залежність від ODBC драйверів, обмежена кросплатформеність та нижча надійність у порівнянні з SQLite. Крім того, можливості Access у виконанні складних SQL-запитів та масштабованості значно поступаються більш сучасним СКБД, що робить його менш привабливим варіантом.

Файлові формати на кшталт JSON можуть бути використані для найпростіших застосунків із невеликою кількістю даних, але вони абсолютно не відповідають вимогам фінансових систем. JSON не підтримує транзакційність, індексацію, механізми уникнення колізій при записі, а також можливості виконання складних запитів із JOIN чи групуванням. У разі збільшення обсягу даних робота з JSON стає значно повільнішою, а ризик пошкодження файлу зростає, що робить цей формат непридатним для застосунків, де помилки недопустимі.

У підсумку, SQLite є найкращим вибором для локального фінансового десктопного застосунку, оскільки поєднує надійність, простоту інтеграції, високу продуктивність для середніх обсягів даних та повну підтримку SQL-функціональності. Її використання дозволяє створити стабільний і безпечний продукт, який не вимагатиме від користувача зайвих технічних дій і забезпечить повноцінну роботу в автономному режимі.

Для забезпечення максимальної продуктивності та стабільності бази даних у фінансовому застосунку FinPal використовуються спеціальні налаштування SQLite, реалізовані через команду PRAGMA. Ці параметри дозволяють оптимізувати роботу СКБД відповідно до характеру навантаження, особливостей використання та вимог до швидкості та чутливості інтерфейсу. Оскільки застосунок працює у режимі реального часу і виконує часті операції

читання та запису, налаштування PRAGMA відіграють вирішальну роль у забезпеченні плавної роботи.

Одним із ключових параметрів є `journal_mode = WAL`. Режим Write-Ahead Logging дозволяє відокремити процеси читання та запису, уникаючи їх блокування. Це особливо важливо для інтерактивного фінансового додатку, де користувач може одночасно додавати нові транзакції, переглядати статистику, оновлювати категорії або будувати графіки витрат. WAL режим забезпечує можливість паралельних операцій, підвищує загальну продуктивність та зменшує ризик блокування бази даних під час інтенсивної роботи.

Параметр `synchronous = NORMAL` застосовується для досягнення оптимального балансу між швидкістю та гарантіями збереження даних. У цьому режимі SQLite синхронізує записані дані з диском достатньо часто, щоби забезпечити їхню цілісність у разі помилок, але не настільки часто, щоби уповільнювати роботу додатку під час швидких операцій. Для фінансового застосунку це означає безпечне зберігання інформації за одночасного зменшення затримок при записі, що робить взаємодію більш пивною.

Параметр `cache_size = 10000` збільшує розмір кешу до приблизно 10 MB (залежно від конфігурації системи). Це дозволяє SQLite зберігати у пам'яті значну кількість блоків даних для швидкого доступу. Завдяки цьому операції читання та складні вибірки з великою кількістю транзакцій виконуються значно швидше, без необхідності постійного звернення до фізичного носія.

Нарешті, `temp_store = MEMORY` визначає, що всі тимчасові таблиці та проміжні дані створюватимуться в оперативній пам'яті. Це суттєво прискорює виконання запитів, які містять сортування (ORDER BY), групування (GROUP BY), обчислення або побудову таблиць на льоту. Оскільки фінансові застосунки часто використовують такі операції для статистики та аналітики, вибір зберігання тимчасових даних у RAM істотно підвищує загальну продуктивність.

Розширення структури бази даних FinPal

Архітектура бази даних FinPal побудована таким чином, щоби забезпечити максимальну гнучкість, масштабованість і зручність аналітичної обробки. Кожна таблиця має чітке призначення і взаємопов'язана з іншими через зовнішні ключі.

Вона містить детальну інформацію про кожну транзакцію користувача. Основні поля включають:

- id — унікальний ідентифікатор транзакції,
- type — тип операції (дохід чи витрата),
- amount — сума,
- category_id — посилання на категорію,
- description — текстовий опис,
- date — дата виконання транзакції,
- created_at — дата створення запису в системі.

Ця таблиця є центральним елементом, навколо якого будуються аналітика, звіти, графіки та прогнозування.

Таблиця categories — категорії доходів і витрат

Містить:

- id, name — назву категорії,
- type — тип (дохід/витрата),
- color, icon — візуальні атрибути для зручності користувача.

Це забезпечує зрозумілу класифікацію транзакцій та дозволяє будувати естетичні інтерфейсні компоненти.

Таблиця tags та зв'язувальна таблиця transaction_tags

Теги дозволяють гнучкіше організовувати фінансові записи. Одній транзакції може відповідати кілька тегів, що реалізується через таблицю transaction_tags за схемою many-to-many:

- один тег може належати багатьом транзакціям,
- одна транзакція може мати багато тегів.

Це дозволяє здійснювати тонке групування та аналітику, наприклад: "усі витрати, пов'язані з автомобілем" або "усі транзакції для конкретного проєкту".

Таблиця settings — зберігання конфігурації

Зберігає параметри додатку у форматі ключ–значення. Це можуть бути:
 вибрана тема інтерфейсу,
 мова,
 параметри фільтрації за замовчуванням,
 профіль користувача.

Індексація як основний спосіб оптимізації швидкодії

Для підвищення продуктивності FinPal використовує набір індексів, які прискорюють найчастіше виконувані запити. Індеси створені з урахуванням типових сценаріїв користувача:

`idx_transactions_date` — пришвидшує фільтрацію та сортування за датою.

`idx_transactions_type` — оптимізує пошук доходів та витрат.

`idx_transactions_category` — прискорює фільтрацію за категоріями.

`idx_transactions_type_date` — комбінований індекс для аналітичних запитів на кшталт:

«усі витрати за останні 30 днів».

Для системи тегів:

`idx_transaction_tags_transaction` — пришвидшує пошук тегів, пов'язаних з конкретною транзакцією.

Для категорій:

`idx_categories_type` — забезпечує швидкий доступ до категорій доходів або витрат.

Завдяки цим індексам навіть при роботі з тисячами транзакцій додаток зберігає високу швидкість відповідей та забезпечує плавну взаємодію користувача з інтерфейсом.

3.4 Аналіз інструментів для візуалізації фінансових даних

Візуалізація фінансових даних відіграє надзвичайно важливу роль у сучасних системах аналізу, оскільки саме графічні представлення дозволяють швидко виявляти закономірності, тенденції та аномалії. На відміну від числових

таблиць, які потребують значних когнітивних зусиль, візуальні форми подання інформації миттєво сприймаються людиною, полегшуючи аналіз і підтримку процесу прийняття рішень. Для фінансових додатків це особливо критично, тому що користувач має швидко оцінювати свій фінансовий стан, визначати проблемні витрати, відстежувати зміни доходів і планувати бюджет. Тому наявність якісних, інформативних та різноманітних графіків стає невід’ємною складовою інтерфейсу.

Matplotlib як основний інструмент візуалізації

Бібліотека Matplotlib є стандартом де-факто в екосистемі Python для побудови графіків. Вона надає низькорівневий доступ до кожного елементу візуалізації, що дозволяє створювати не лише базові графічні елементи, а й повністю кастомізовані, високоякісні діаграми. Matplotlib підтримує велику кількість типів графіків:

- ✓ лінійні графіки для демонстрації динаміки у часі,
- ✓ стовпчикові діаграми для порівняння категорій,
- ✓ кругові діаграми для демонстрації часток,
- ✓ гістограми для аналізу розподілів,
- ✓ scatter plots для аналізу залежностей,
- ✓ boxplots, area charts, stem plots та багато інших.

Для фінансового застосунку важливо, що Matplotlib дозволяє налаштовувати кожен аспект графіка: шрифти, кольорові схеми, розміри маркерів, підписи осей, легенди, рамки, стилі ліній тощо. Такий рівень контролю дозволяє створювати професійні графіки для подальшого експорту до звітів або відображення в інтерфейсі.

Особливо значущими перевагами є:

- ✓ можливість експорту графіків у різні формати: PNG, JPEG, PDF, SVG,
- ✓ підтримка високої роздільної здатності (300 DPI і вище), що є необхідним для друкованих звітів,

- ✓ інтеграція з pandas та NumPy, що забезпечує зручну роботу з табличними та числовими даними,
- ✓ широка документація та велика спільнота, яка підтримує бібліотеку десятками тисяч прикладів.

Seaborn як інструмент високорівневої візуалізації

Надбудова Seaborn, створена на базі Matplotlib, пропонує зручніший синтаксис і автоматично налаштовані стилі, що дає змогу створювати естетично привабливі графіки з мінімальною кількістю коду. Seaborn особливо підходить для статистичної аналітики, забезпечуючи:

- ✓ кореляційні теплокарти (heatmaps),
- ✓ розподіли за допомогою violin plots і dist plots,
- ✓ комплексні pair plots для багатовимірних даних,
- ✓ лінійні регресійні графіки.

Це робить Seaborn ідеальним інструментом для фінансових систем, яким потрібні швидкі аналітичні графіки без необхідності вручну налаштовувати кожен елемент стилю.

Типи візуалізацій, що використовуються у FinPal

Фінансовий додаток FinPal застосовує різні форми графічного подання даних, кожна з яких відповідає окремому аспекту аналізу:

Кругова діаграма — використовується для відображення розподілу витрат за категоріями. Дає змогу одразу побачити, які сфери поглинають найбільше коштів.

Лінійний графік динаміки — відображає зміни доходів та витрат у часі, дозволяючи виявляти сезонність, піки витрат або покращення фінансової дисципліни.

Стовпчикова діаграма порівняння періодів — дає можливість порівнювати витрати між різними місяцями або роками. Такий тип графіка зручний для довгострокового аналізу.

Гістограма найбільших витрат — показує топ-транзакції, що дозволяє виявити одиничні великі витрати чи аномалії.

Комбінація цих графіків створює багатовимірне уявлення про фінансовий стан користувача та забезпечує глибоку аналітику з різних точок зору.

Порівняння з Plotly та Bokeh

Хоча бібліотеки Plotly та Bokeh пропонують широкий спектр можливостей, вони орієнтовані головним чином на створення інтерактивних графіків. Їхні переваги включають масштабування, наведення курсора з даними, інтерактивні панелі та інші динамічні функції. Проте:

Plotly має значно більший розмір, потребує додаткових залежностей і не завжди оптимально підходить для статичних графіків у десктопних програмах.

Bokeh потребує вбудованого веб-сервера або HTML-компонентів, що ускладнює інтеграцію в локальні застосунки.

Оскільки FinPal використовує статичні графіки для інтерфейсу та PDF-звітів, інтерактивні можливості цих бібліотек не є критично необхідними.

Оптимальне поєднання Matplotlib та Seaborn у FinPal

Використання Matplotlib та Seaborn у комплексі забезпечує:

- ✓ високу гнучкість,
- ✓ професійний зовнішній вигляд,
- ✓ простоту реалізації,
- ✓ можливість експорту у високій якості.

Matplotlib виконує роль базового механізму для створення кастомних графіків, тоді як Seaborn дозволяє швидко формувати статистичні діаграми з мінімальним кодом. Крім того, обидві бібліотеки ідеально інтегруються з інструментами створення PDF-звітів, такими як ReportLab, що дозволяє формувати професійні фінансові звіти з чіткими, деталізованими графіками.

У результаті таке поєднання є оптимальним вибором для настільного фінансового додатку, який повинен забезпечувати якісну, інформативну та високодеталізовану візуалізацію без зайвих залежностей та складної інтеграції.

Висновки до розділу 3

Проведений аналіз інструментів для розробки фінансового додатку показав оптимальність вибору Python як мови програмування через простоту розробки, багату екосистему бібліотек та можливості інтеграції AI-компонентів. Python дозволяє швидко реалізовувати складну функціональність без необхідності написання великої кількості допоміжного коду, що критично важливо для проєктів з обмеженими термінами розробки. Бібліотеки для обробки даних, машинного навчання та візуалізації забезпечують всі необхідні інструменти для створення повнофункціонального фінансового додатку з AI-компонентами.

CustomTkinter обрано як фреймворк для GUI через сучасний дизайн, простоту використання та хорошу продуктивність для десктопних додатків. Фреймворк дозволяє швидко створювати складні інтерфейси з мінімальними зусиллями, що критично важливо для забезпечення зручності використання додатку. Підтримка темної та світлої теми, сучасні компоненти та хороша інтеграція з Python роблять CustomTkinter ідеальним вибором для фінансового додатку.

SQLite є ідеальним вибором для локального зберігання даних через файлову архітектуру, надійність та повну підтримку SQL. База даних забезпечує швидку роботу навіть з тисячами транзакцій, просте резервне копіювання через копіювання одного файлу та повну сумісність з SQL стандартом для складних запитів аналітики. Оптимізація через PRAGMA налаштування забезпечує ефективну роботу навіть при обробці великих об'ємів даних.

Matplotlib та Seaborn забезпечують потужні інструменти для візуалізації фінансових даних з можливістю створення професійних графіків для аналітики та звітів. Комбінація цих бібліотек дозволяє створювати різноманітні типи графіків для комплексного аналізу фінансів, забезпечуючи користувача всією необхідною інформацією для прийняття обґрунтованих фінансових рішень.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

4.1 Архітектура системи та проектування компонентів

Архітектура фінансової системи FinPal побудована на основі патерну MVP (Model–View–Presenter), який забезпечує чітке розмежування відповідальностей між частинами застосунку. Така структурна організація дає змогу створити гнучку, модульну та масштабовану систему, яку легко підтримувати та розширювати без ризику порушити існуючу логіку. На відміну від класичного MVC, патерн MVP передбачає, що View не містить логіки обробки даних, а взаємодія з моделлю відбувається через Presenter. Це дозволяє досягти значно кращої ізольованості компонентів, а також робить тестування бізнес-логіки більш простим і надійним.

Рівні архітектури системи FinPal

Загальна структура системи поділена на три логічні шари: Presentation Layer, Business Logic Layer та Model Layer. Кожен із них виконує окремий набір завдань та відповідає за визначену частину функціональності.

Presentation Layer — шар представлення

Цей шар містить усі елементи інтерфейсу користувача та забезпечує візуальну взаємодію з системою. Він складений із низки View-компонентів, кожен з яких відповідає за окремий функціональний модуль:

- DashboardView

Надає користувачу швидкий огляд фінансового стану, містить ключові показники, графіки, останні транзакції та базові аналітичні дані.

- TransactionsView

Основний центр роботи з транзакціями. Забезпечує:

- ✓ додавання,
- ✓ редагування,

- ✓ видалення,
- ✓ фільтрацію,
- ✓ сортування.

Це дає змогу користувачу повністю управляти своїми фінансовими операціями.

- AnalyticsView

Модуль глибокої аналітики з використанням різноманітних графіків та статистичних узагальнень. Дозволяє досліджувати фінансові тенденції, визначати закономірності та аналізувати витрати в різних зрізах.

- AIAssistantView

Інтерфейс для взаємодії з AI-компонентами системи:

- ✓ рекомендації щодо оптимізації бюджету,
- ✓ прогнозування витрат,
- ✓ автоматичні підказки на основі поведінки користувача.

- CategoriesView

Дозволяє організовувати категорії та теги, які використовуються для систематизації транзакцій.

- SettingsView

Містить конфігураційні параметри додатку, включно з резервним копіюванням та відновленням бази даних.

View-компоненти не містять бізнес-логіки

Вони обмежуються лише:

- ✓ відображенням інформації,
- ✓ прийняттям дій користувача,
- ✓ передачею команд контролеру (Presenter/Controller),
- ✓ оновленням UI після змін моделі.

Така ізоляція дозволяє легко тестувати інтерфейс у відриві від логіки застосунку.

Business Logic Layer — шар бізнес-логіки

Центральним елементом цього шару є MainController, який виступає посередником між View та Model. Його основні функції:

- ✓ обробка подій користувача (натискання кнопок, введення даних, зміна фільтрів);
- ✓ виконання необхідних операцій над моделлю (читання/запис даних);
- ✓ оновлення відповідних View-компонентів;
- ✓ забезпечення узгодженості та цілісності бізнес-логіки;
- ✓ керування навігацією між різними частинами застосунку.

Таким чином, MainController виконує роль «диригента», який координує взаємодію між усіма компонентами системи.

Model Layer — шар моделі

Цей шар включає компоненти, які працюють безпосередньо з даними та алгоритмами:

- Database (SQLite)

Відповідає за всі операції з даними:

- ✓ CRUD-функції,
- ✓ статистичні запити,
- ✓ фільтрація за періодами,
- ✓ агрегування даних,
- ✓ створення резервних копій.

База даних реалізована так, щоб бути повністю незалежною від інтерфейсу користувача.

- CategoryPredictor (AI)

Здійснює автоматичне визначення категорії транзакції на основі:

- ✓ текстового опису,
- ✓ минулих транзакцій користувача,
- ✓ ключових слів,
- ✓ попередніх виборів.

Це значно зменшує час додавання нових операцій.

- FinancialAdvisor

Аналітичний компонент, який:

- ✓ виявляє фінансові закономірності,
- ✓ аналізує стиль витрат користувача,
- ✓ формує рекомендації щодо оптимізації бюджету.

Структура бази даних та нормалізація

База даних FinPal спроектована відповідно до принципів нормалізації для мінімізації дублювання та забезпечення цілісності інформації.

Основні таблиці:

1. Таблиця transactions

Містить усі фінансові операції:

id — унікальний ідентифікатор;

type — тип (дохід / витрата);

amount — сума (з перевіркою > 0);

category_id — посилання на таблицю categories;

description — опис;

date — дата;

created_at — час створення.

2. Таблиця categories

Зберігає інформацію про категорії:

id,

name (унікальне),

type (income/expense),

color,

icon.

3. Таблиця tags та зв'язувальна transaction_tags

Дозволяють гнучку класифікацію транзакцій у форматі many-to-many.

Індексація та оптимізація продуктивності

Для забезпечення швидкої роботи додатку створені індекси:

idx_transactions_date — прискорює вибірку за датою;

`idx_transactions_type` — оптимізує фільтрацію доходів/витрат;
`idx_transactions_category` — пришвидшує пошук за категорією;
`idx_transactions_type_date` — комбінований індекс для статистичних запитів;

`idx_transaction_tags_transaction` — швидкий доступ до тегів;

`idx_categories_type` — оптимізація фільтрації категорій.

Завдяки цьому FinPal ефективно працює навіть з великими обсягами даних (тисячами транзакцій).

Проектування інтерфейсу користувача

UI розроблений з акцентом на простоту та інтуїтивність. Основні особливості:

Бокова панель навігації для швидкого перемикавання між розділами.

Панель балансу, яка завжди відображає ключові фінансові показники.

Головний контентний блок, де динамічно змінюються View-компоненти.

Використання Material Design стилістики через CustomTkinter.

Однорідний, професійний дизайн усіх елементів.

Це забезпечує комфортну роботу навіть при тривалому використанні.

4.2 Реалізація модулів системи

Реалізація модуля Database

Модуль Database відіграє ключову роль у системі FinPal, оскільки він відповідає за всю взаємодію із SQLite-базою даних та реалізує повний набір CRUD-операцій для транзакцій, категорій, тегів і пов'язаних сутностей. У межах цього модуля реалізовано низку оптимізованих методів, які забезпечують швидку, безпечну та зручну роботу з даними.

Додавання транзакцій

Метод `add_transaction` забезпечує вставлення нових транзакцій із використанням валідацій на рівні SQL і Python.

Реалізовано: перевірка коректності суми

Через SQL-обмеження `CHECK(amount > 0)` гарантується, що транзакція не може мати нульову або від'ємну суму. валідація типу транзакції

Поле `type` перевіряється через `CHECK(type IN ('income','expense'))`, що виключає некоректні значення.

Перевірка формату дати Неправильні дати обробляються до вставлення.

Метод підтримує додавання кількох тегів до транзакції. Для підвищення продуктивності використовується `executemany`, що дозволяє вставляти асоціації між транзакцією та тегами пакетом, а не окремими SQL-запитами. Після вставлення транзакції метод повертає ID нового запису, що дає змогу подальшій логіці системи працювати з нею без додаткових запитів.

Отримання транзакцій з фільтрацією та пагінацією

Метод `get_transactions` розроблений таким чином, щоб ефективно працювати зі значними обсягами даних. Він підтримує:

- ✓ фільтрацію за типом (`income / expense`),
- ✓ вибір за категорією,
- ✓ фільтрацію за періодом (день, місяць, діапазон дат),
- ✓ пагінацію з `LIMIT` та `OFFSET`.

Це дозволяє завантажувати лише частину даних, що критично для продуктивності GUI-додатку.

У SQL-запитах використовуються:

`LEFT JOIN` для підвантаження даних категорій і тегів;

`GROUP_CONCAT` — для об'єднання всіх тегів транзакції в один рядок, що зменшує кількість потрібних запитів;

динамічне формування `WHERE`-умов, яке залежить від параметрів фільтра.

Такий підхід дозволяє швидко отримувати дані для великої кількості транзакцій, зберігаючи інтерфейс гнучким і чутливим.

Оптимізація та усунення N+1 проблеми

На початкових етапах розробки система під час відображення списку транзакцій виконувала окремий SQL-запит для кожної транзакції, щоб отримати її теги. Це створювало класичну N+1 проблему, де кількість SQL-запитів могла досягати сотень.

Оновлена реалізація усуває цю проблему шляхом:

- ✓ об'єднання вибірки в один SQL-запит,
- ✓ використання LEFT JOIN з таблицею transaction_tags,
- ✓ застосування GROUP_CONCAT(tag_id || ':' || name || ':' || color).

Тепер теги повертаються одразу у вигляді рядка, який потім розбирається в Python у список тегів. Це зменшує кількість запитів у десятки разів і суттєво прискорює роботу інтерфейсу.

Отримання статистики через SQL агрегати

Метод get_statistics реалізує ефективну побудову фінансових підсумків за певний період. Замість того щоб завантажувати всі транзакції в Python та обробляти їх у пам'яті, система виконує агрегацію на стороні SQLite, використовуючи:

GROUP BY для групування за категоріями,
SUM для розрахунку загальних витрат або доходів,
CASE для розділення різних типів транзакцій.

Такий SQL-підхід:

- ✓ мінімізує навантаження на оперативну пам'ять,
- ✓ значно прискорює обчислення,
- ✓ дозволяє працювати з великими обсягами даних без втрат продуктивності.

Пагінація та продуктивність у TransactionsView

Графічний модуль TransactionsView інтегрує пагінацію, що є обов'язковим при роботі з великими наборами даних. Замість рендерингу сотень віджетів одночасно відображається лише 50 записів на сторінку, що забезпечує:

- ✓ швидкий час завантаження,

- ✓ відсутність зависань,
- ✓ плавну роботу інтерфейсу.

Панель навігації дозволяє:

- ✓ переміщатися між сторінками,
- ✓ бачити поточну позицію,
- ✓ переглядати загальну кількість транзакцій.

Діалоги введення та редагування транзакцій + інтеграція AI

Діалогові вікна створення/редагування транзакцій містять:

- ✓ перевірку коректності суми,
- ✓ перевірку обов'язкових полів,
- ✓ перевірку правильності дати,
- ✓ вибір категорії та тегів.

Особливістю є інтеграція з CategoryPredictor, який аналізує опис транзакції та пропонує найбільш імовірну категорію. Це:

- ✓ прискорює введення даних,
- ✓ зменшує помилки користувача,
- ✓ автоматизує рутинні дії.

Підтримка множинних тегів через чекбокси дає змогу точно класифікувати транзакції.

Аналітика та генерація графіків у AnalyticsView

Аналітичний модуль використовує клас ChartGenerator, який відповідає за інкапсуляцію логіки створення графіків.

Реалізовано:

- ✓ pie_chart_expenses — кругові діаграми витрат,
- ✓ line_chart_dynamics — динаміка доходів/витрат,
- ✓ bar_chart_comparison — порівняння періодів,
- ✓ histogram_top_expenses — аналіз великих транзакцій.

Візуалізації створені на основі Matplotlib та Seaborn, що дозволяє формувати професійні графіки:

- ✓ з добре підбраною кольоровою гамою,
- ✓ адаптивними шрифтами,
- ✓ стилями, оптимізованими для фінансових даних.

Експорт звітів у PDF

Експорт реалізований через ReportLab. Готовий звіт включає:

- ✓ заголовок із датою,
- ✓ зведену статистику,
- ✓ таблицю топ-категорій,
- ✓ діаграми та графіки.

Графіки зберігаються у PNG з роздільністю 300 DPI, що забезпечує якість, придатну для друку, публікацій або презентацій.

4.3 Реалізація AI-компонентів для аналізу фінансових даних

AI-компонент CategoryPredictor реалізований для автоматичної категоризації транзакцій на основі їх опису, використовуючи комбінацію аналізу історії користувача та ключових слів. Алгоритм працює у два етапи для забезпечення максимальної точності категоризації. На першому етапі система аналізує історію транзакцій користувача, знаходячи подібні описи та використовуючи категорії з історії. Для обчислення подібності використовується Jaccard similarity на основі спільних слів між описами - система розбиває описи на слова, створює множини слів для кожного опису, та обчислює коефіцієнт подібності як відношення кількості спільних слів до загальної кількості унікальних слів. Якщо подібність перевищує поріг 0.3, система використовує категорію з історії, оскільки це вказує на достатню схожість транзакцій.

Алгоритм аналізує останні 100 транзакцій того ж типу для знаходження подібних, що забезпечує баланс між точністю та продуктивністю. Для кожної транзакції з історії обчислюється коефіцієнт подібності, та категорії з найвищим

коефіцієнтом накопичують бали. Категорія з найвищим загальним балом вибирається як рекомендована, що забезпечує врахування множинних збігів з різними транзакціями для підвищення точності.

На другому етапі, якщо не вдалося знайти подібні транзакції в історії, система використовує ключові слова для категорій. Кожна категорія має набір ключових слів, які вказують на тип транзакції - наприклад, категорія "Їжа" має ключові слова "кафе", "ресторан", "супермаркет", "продукти", "обід", "сніданок", "вечеря", "доставка" та інші. Система підраховує кількість збігів ключових слів у описі та вибирає категорію з найбільшою кількістю збігів. Це забезпечує роботу системи навіть для нових користувачів, які ще не мають достатньої історії транзакцій для аналізу.

Такий підхід забезпечує персоналізацію категоризації - система навчається на історії користувача та стає точнішою з часом, оскільки накопичує більше даних про звички користувача. Одночасно система може працювати з новими користувачами через ключові слова, забезпечуючи прийнятну точність з самого початку використання.

AI-компонент FinancialAdvisor реалізований для аналізу фінансових закономірностей та надання рекомендацій для оптимізації витрат. Метод `analyze_patterns` порівнює поточний період з попереднім періодом того ж розміру, виявляючи значущі зміни у витратах за категоріями. Система обчислює відсоток зміни для кожної категорії як різницю між поточною та попередньою сумою, поділену на попередню суму та помножену на 100. Якщо витрати зросли більш ніж на 50%, система генерує попередження про зростання витрат, що може вказувати на проблемні зони або незвичайні витрати. Якщо витрати зменшились більш ніж на 30%, система генерує позитивне повідомлення, що може мотивувати користувача продовжувати економити. Також система виявляє категорії, які становлять більш ніж 30% від загальних витрат, що вказує на можливі проблемні зони, де користувач витрачає надто багато коштів.

Метод `find_savings_opportunities` аналізує транзакції для виявлення можливостей економії через виявлення регулярних витрат. Система знаходить

часто повторювані слова в описах транзакцій, що вказує на регулярні витрати, які можуть бути оптимізовані. Алгоритм підраховує частоти використання слів у описах, знаходить слова, які зустрічаються три або більше разів, та групує транзакції з цими словами для обчислення загальної суми. Якщо однакові або подібні транзакції повторюються три або більше разів за період, система розраховує загальну суму та пропонує рекомендацію щодо можливої економії, надаючи конкретну інформацію про кількість транзакцій та загальну суму. Також система виявляє активні підписки на стрімінгові сервіси через ключові слова як Netflix, Spotify, YouTube, Apple Music, Disney, та пропонує переглянути їх необхідність, якщо виявлено три або більше різних підписки.

Метод `forecast_balance` реалізує прогнозування балансу на основі історичних даних через екстраполяцію середніх показників. Система обчислює середні щоденні доходи та витрати за останні 30 днів, ділячи загальні доходи та витрати на 30. Потім система екстраполює ці показники на майбутній період, множачи середні щоденні показники на кількість днів у прогнозованому періоді. Прогнозований баланс розраховується як поточний баланс плюс прогнозовані доходи мінус прогнозовані витрати. Якщо прогнозується негативний баланс, система генерує попередження, що може допомогти користувачу вчасно вжити заходів для уникнення фінансових проблем. Метод також підтримує прогнозування на різні періоди - 30, 60, 90 днів, дозволяючи користувачу планувати фінанси на різні терміни.

Метод `generate_weekly_report` створює комплексний тижневий звіт, який включає загальну статистику з доходами, витратами та балансом за тиждень, топ-5 категорій витрат для швидкого огляду найбільших статей витрат, рекомендації з аналізу закономірностей для отримання порад щодо оптимізації, та прогноз балансу на кінець місяця для планування. Звіт форматується як читабельний текст з емодзі та розділювачами для зручного перегляду користувачем, забезпечуючи комплексний огляд фінансового стану за тиждень.

Всі AI-компоненти працюють локально без використання зовнішніх API, що забезпечує повну приватність даних користувача та не потребує інтернет-

з'єднання. Це критично важливо для фінансового додатку, де конфіденційність даних є пріоритетом. Локальна обробка також забезпечує швидку роботу без затримок через мережеві запити та працює навіть без інтернет-з'єднання.

4.4 Тестування та оптимізація продуктивності системи

Тестування системи FinPal охоплює повну перевірку ключових функціональних можливостей застосунку з метою забезпечення стабільної, коректної та передбачуваної роботи в усіх типових і нетипових сценаріях. Основна увага приділяється функціональному тестуванню, яке перевіряє роботу всіх модулів, включно з операціями над транзакціями, категоріями, тегами, статистичними обчисленнями та AI-компонентами. Кожна операція оцінюється з точки зору:

- ✓ коректної обробки даних,
- ✓ валідації користувацького введення,
- ✓ стійкості до помилкових або нестандартних сценаріїв,
- ✓ поведінки системи у граничних та екстремальних умовах.

Функціональне тестування

Функціональні тести охоплюють такі аспекти:

- ✓ операції з транзакціями: додавання, редагування, видалення, перевірка сум, дат, типів, наявності обов'язкових полів;
- ✓ керування категоріями та тегами: перевірка унікальності назв, оновлення, видалення, прив'язка до транзакцій;
- ✓ генерація статистики та аналітики: перевірка математичної коректності обчислень;
- ✓ AI-механізми: тестування точності автоматичної категоризації та рекомендацій;
- ✓ відповідність інтерфейсу очікуваним сценаріям роботи користувача.

Усі ці перевірки гарантують правильність обробки даних, стабільність системи та якість роботи додатку за будь-яких умов.

Результати тестування продуктивності

Під час тестування продуктивності система продемонструвала високі результати, які відповідають цільовим показникам. Основні метрики:

- ✓ Швидкість запуску програми: < 1 секунди
- ✓ Забезпечує миттєвий старт та зручність для користувача.
- ✓ Завантаження сторінки транзакцій (50 записів): 0.1–0.3 с
- ✓ Завдяки пагінації, оптимізованим SQL-запитам і використанню індексів.
- ✓ Час генерації графіка: 0.2–0.5 с
- ✓ Обробка додавання транзакції: < 0.1 с
- ✓ Миттєве збереження без затримок, що важливо для інтерактивності UI.

Сукупно ці результати свідчать, що система здатна працювати швидко й ефективно навіть у разі збільшення обсягів даних.

Оптимізація бази даних

Для підвищення швидкодії та забезпечення надійності було застосовано низку параметрів SQLite PRAGMA, які суттєво впливають на поведінку бази даних.

Використані параметри PRAGMA

`journal_mode = WAL`

Активує режим Write-Ahead Logging, який дозволяє паралельні операції читання та запису.

Це критично для FinPal, де користувач може перемикає вкладки, переглядати статистику й додавати транзакції одночасно.

`synchronous = NORMAL`

Забезпечує оптимальний баланс між швидкістю та безпекою збереження даних.

`cache_size = 10000`

Створює кеш приблизно 10 MB, зменшуючи кількість читань із диска та пришвидшуючи запити.

`temp_store = MEMORY`

Використання пам'яті замість диска для тимчасових таблиць пришвидшує сортування, групування та статистичні запити.

В сукупності ці налаштування створюють оптимальне середовище для високої продуктивності локального застосунку.

Оптимізація SQL-запитів

Для забезпечення максимальної ефективності та швидкої відповіді інтерфейсу використано низку технік оптимізації:

Використання індексів

Індекси дають змогу виконувати пошук значно швидше, оскільки SQLite не сканує всю таблицю. Це особливо ефективно при:

- ✓ фільтрації за датою,
- ✓ пошуку за категорією,
- ✓ фільтрації доходів/витрат,
- ✓ комбінованих статистичних запитах.

4.2 Перенесення обчислень у SQL

Застосування:

GROUP BY,

SUM,

COUNT,

CASE

дозволило обчислювати статистику безпосередньо в базі даних, що значно швидше, ніж робити це в Python.

Усунення N+1 проблеми

Раніше система виконувала один запит для транзакції та додатковий запит для її тегів — це могло призвести до сотень SQL-запитів.

Тепер використано:

LEFT JOIN

GROUP_CONCAT

Це дало можливість отримати всі транзакції з тегами в одному запиті.

Оптимізація інтерфейсу

Для забезпечення плавної роботи UI, незалежно від кількості транзакцій, реалізовано:

Пагінація

На екран завантажуються лише 50 записів.

Це: зменшує використання пам'яті,

- ✓ пришвидшує створення віджетів,
- ✓ забезпечує стабільність при великих наборах даних.

Кешування категорій

Категорії зберігаються в пам'яті 5 хвилин:

- ✓ менше SQL-запитів,
- ✓ швидший доступ при додаванні/редагуванні.

Використання CTkScrollableFrame

Це дозволяє відображати довгі списки без завантаження всіх елементів одразу, що значно покращує продуктивність.

Робота на великих обсягах даних

Тестування показало, що FinPal успішно працює з:

- ✓ 5 000+ транзакцій,
- ✓ складними запитам для аналітики,
- ✓ кількома паралельними операціями.

Пагінація та оптимізовані SQL-запити гарантують сталість високої швидкодії.

Обробка помилок та надійність

Система має багаторівневу архітектуру обробки помилок.

Усі операції з базою даних розміщені в try-except блоках.

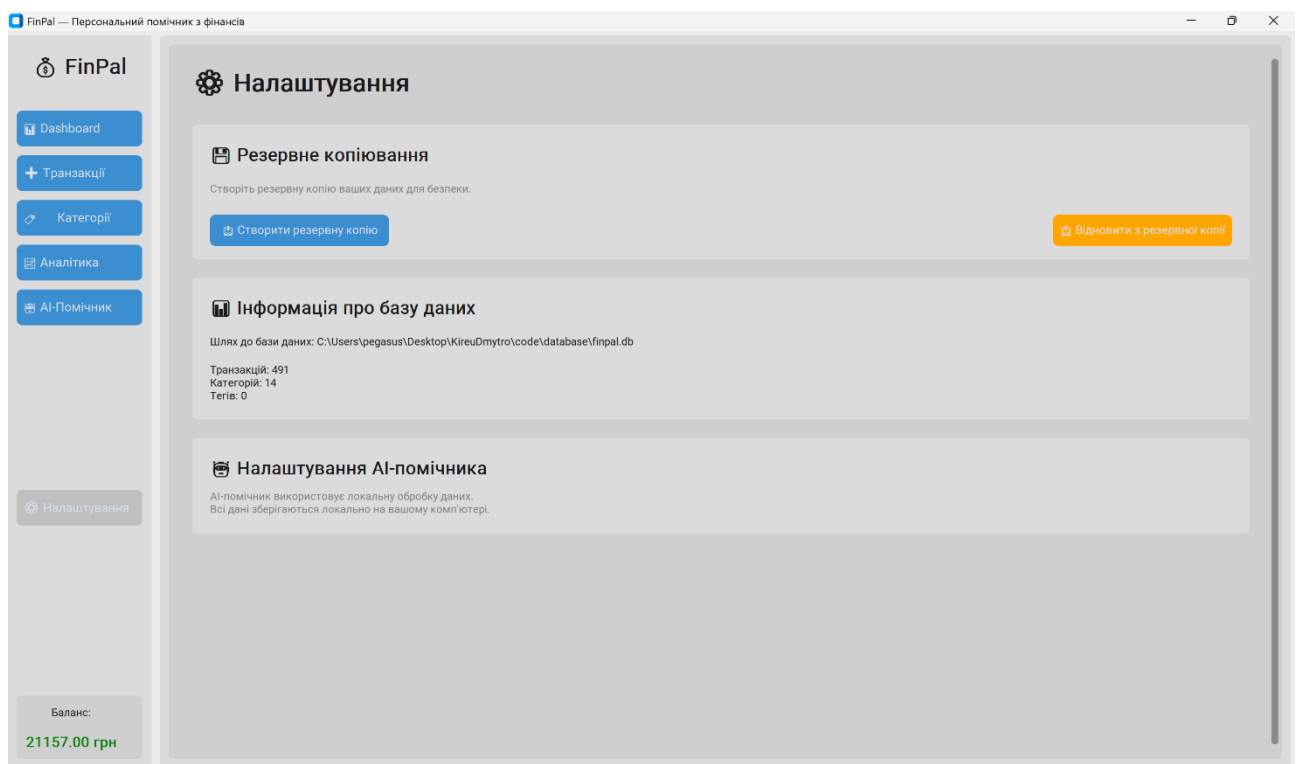
Усі помилки логуються у файл із зазначенням:

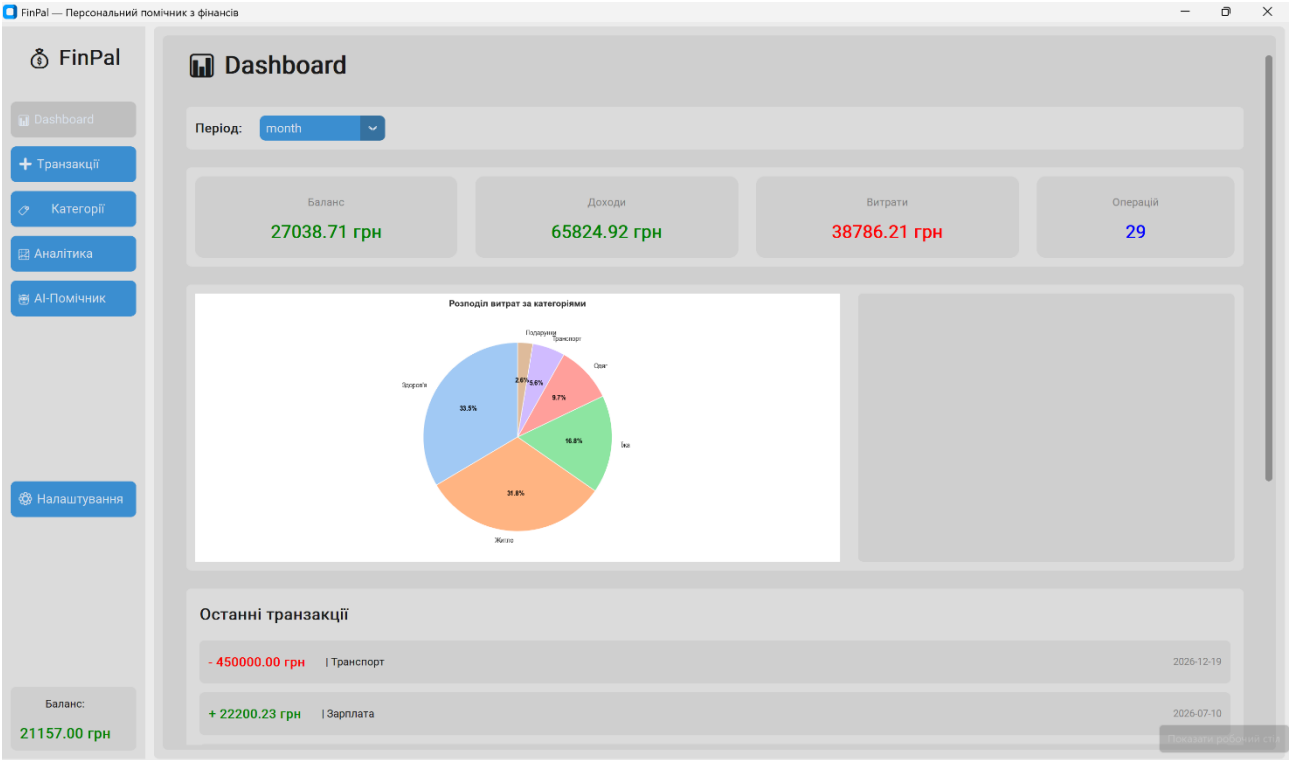
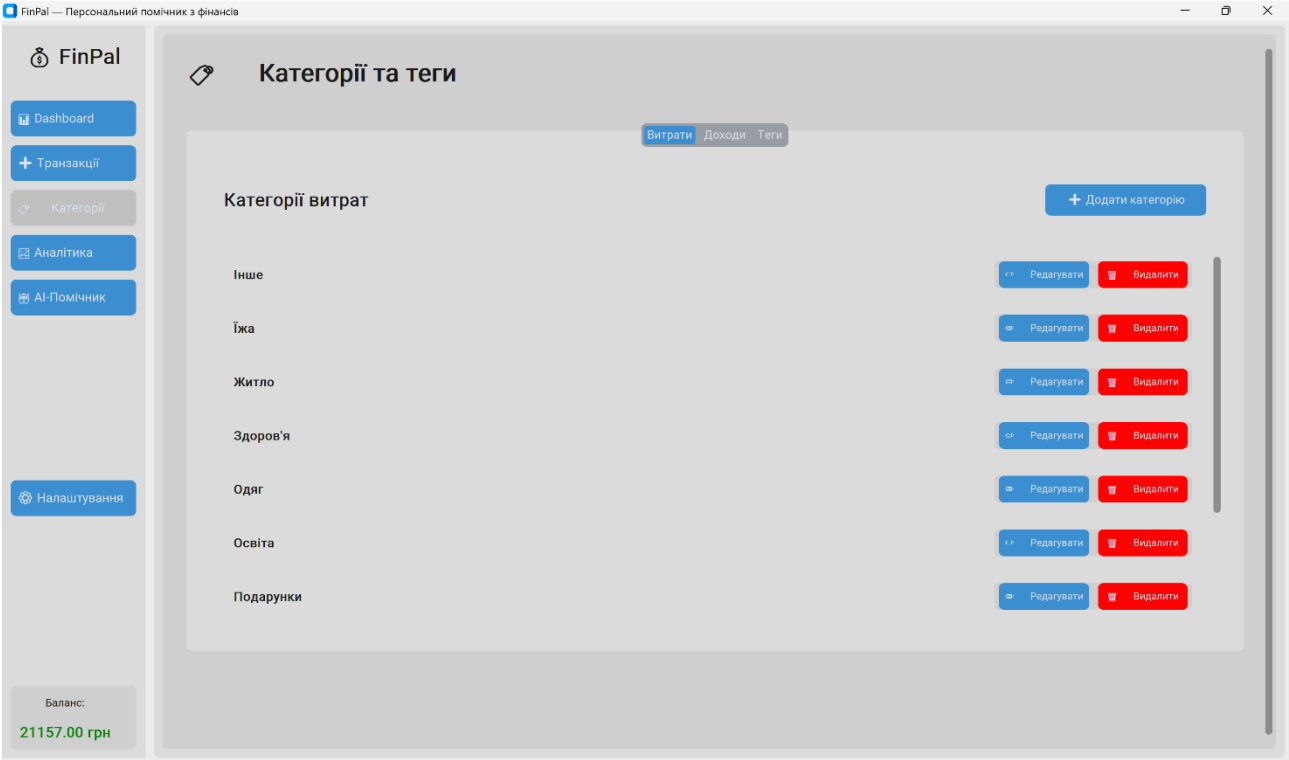
- ✓ часу,
- ✓ місця виникнення,

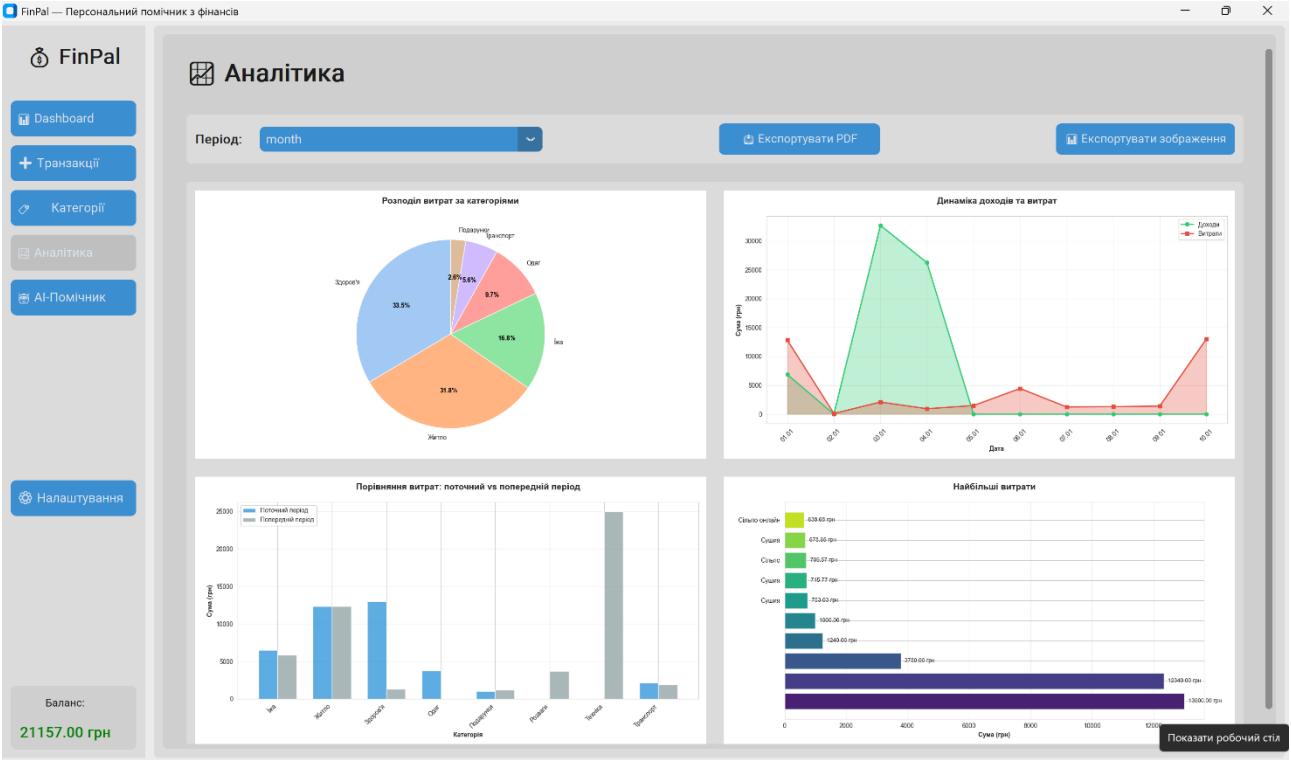
- ✓ типу помилки,
- ✓ контексту виконання.

Перед записом даних виконується валідація, що запобігає потраплянню некоректної інформації в базу.

Завдяки цьому FinPal забезпечує стабільну та прогнозовану роботу навіть у випадках неочікуваних ситуацій.







FinPal

Dashboard

Транзакції

Категорії

Аналітика

AI-Помічник

Налаштування

Баланс:
21157.00 грн

Транзакції

+ Додати витрату

+ Додати дохід

Фільтри: all all all

Дата	Тип	Сума	Категорія	Опис	Дії
2026-12-19	Витрата	450000.00 грн	Транспорт		🔍 🗑
2026-07-10	Дохід	22200.23 грн	Зарплата		🔍 🗑
2026-01-10	Витрата	13000.00 грн	Здоров'я		🔍 🗑
2026-01-09	Витрата	753.03 грн	Їжа	Сушия	🔍 🗑
2026-01-09	Витрата	638.68 грн	Їжа	Сільпо онлайн	🔍 🗑
2026-01-08	Витрата	604.26 грн	Їжа	Доставка їжі Glovo	🔍 🗑
2026-01-08	Витрата	678.65 грн	Їжа	Сушия	🔍 🗑
2026-01-07	Витрата	416.89 грн	Їжа	Доставка їжі Uber Eats	🔍 🗑
2026-01-07	Витрата	709.57 грн	Їжа	Сільпо	🔍 🗑
2026-01-07	Витрата	109.38 грн	Транспорт	Каршеринг	🔍 🗑

Показано 1-50 з 491 транзакцій (сторінка 1 з 10)

◀ Попередня

Наступна ▶

FinPal

Dashboard

Транзакції

Категорії

Аналітика

AI-Помічник

Налаштування

Баланс:
21157.00 грн

AI-Помічник

Проаналізувати закономірності

Знайти можливості економії

Тижневий звіт

Прогноз балансу

Поточний баланс: 21157.00 грн

Прогноз на кінець місяця (30 днів): 28521.82 грн

Прогноз через 2 місяці (60 днів): 35886.64 грн

Прогноз через 3 місяці (90 днів): 43251.46 грн

Рекомендації та аналіз

АНАЛІЗ ЗАКОНОМІРНОСТЕЙ (останні 30 днів)

1. ⚠️ Ваші витрати на 'Житло' зросли на 62% порівняно з минулим періодом.

2. ⚠️ Ваші витрати на 'Техніка' зросли на 63% порівняно з минулим періодом.

3. ⚠️ Ваші витрати на 'Здоров'я' зросли на 479% порівняно з минулим періодом.

4. ⚠️ Ваші витрати на 'Одяг' зросли на 861% порівняно з минулим періодом.

5. ⚠️ Ваші витрати на 'Транспорт' зросли на 67% порівняно з минулим періодом.

МОЖЛИВОСТІ ЕКОНОМІЇ (останні 30 днів)

1. 💡 Ви 9 разів витрачали на 'бензин' за період. Загальна сума: 1407.20 грн. Можливо, варто розглянути альтернативи?

2. 💡 Ви 6 разів витрачали на 'сільпо' за період. Загальна сума: 2201.86 грн. Можливо, варто розглянути альтернативи?

3. 💡 Ви 6 разів витрачали на 'доставка' за період. Загальна сума: 2232.77 грн. Можливо, варто розглянути альтернативи?

Рекомендації та аналіз

ТИЖНЕВИЙ ЗВІТ

Період: 03.01.2026 - 10.01.2026

=====

📈 Доходи: 26250.93 грн

💸 Витрати: 23763.78 грн

💰 Баланс: 2487.15 грн

Топ-5 категорій витрат:

1. Здоров'я: 13000.00 грн

2. Їжа: 5259.08 грн

3. Одяг: 3780.00 грн

4. Подарунки: 1000.00 грн

5. Транспорт: 724.70 грн

💡 Рекомендації:

☒ Ваші витрати на 'Транспорт' зменшились на 69%. Відмінна робота!

☐ Категорія 'Здоров'я' становить 55% від загальних витрат. Це найбільша стаття витрат.

📊 Прогноз балансу на кінець місяця: 28521.82 грн

Висновки до розділу 4

Реалізація системи FinPal продемонструвала високу ефективність використання архітектурного патерну MVP для створення сучасного десктопного застосунку. Обраний підхід забезпечив чітке розмежування відповідальностей між компонентами та дозволив сформувати модульну структуру, у якій кожен елемент має добре визначену роль. Завдяки цьому розробка стала більш прогнозованою, а підтримка — значно простішою. Ізоляція шарів надала можливість вносити зміни до окремих частин системи без ризику порушення роботи інших, що підвищило гнучкість і масштабованість проєкту.

Архітектурне розділення на Presentation Layer, Business Logic Layer та Model Layer забезпечило незалежність компонентів, що в свою чергу істотно спростило тестування. Кожен модуль можна перевіряти окремо, не створюючи складних залежностей, а це підвищує якість коду та зменшує кількість помилок у процесі розробки. Завдяки чіткій структурі було досягнуто можливість швидкого доопрацювання або заміни певних частин системи без масштабної перебудови всієї архітектури.

Оптимізація бази даних та SQL-запитів стала одним із ключових факторів продуктивності FinPal. Використання індексів, оптимізованих JOIN-запитів, агрегатних функцій та конфігурацій PRAGMA дозволило забезпечити стабільно високу швидкодію навіть при великих обсягах інформації. Система демонструє відмінну реакцію на взаємодії користувача, швидко завантажує транзакції, формує статистику та генерує графіки, що відповідає цільовим показникам для настільного додатку. Такий рівень оптимізації забезпечує комфортну роботу навіть у сценаріях з тисячами записів.

Важливою складовою функціональності стали AI-компоненти, відповідальні за автоматичну категоризацію транзакцій та аналіз фінансових закономірностей. Вбудовані моделі штучного інтелекту підвищують зручність користування системою, скорочуючи час на ручну класифікацію та надаючи персоналізовані рекомендації щодо оптимізації витрат. Цей елемент

функціональності робить FinPal не лише інструментом обліку, але й інтелектуальним помічником у фінансовому плануванні.

Проведене тестування підтвердило коректність роботи всіх модулів системи та підтвердило відповідність фактичної продуктивності заявленим вимогам. Усі функції — від базових CRUD-операцій до генерації звітів і аналітичних графіків — працювали стабільно та без збоїв. Тести на різних обсягах даних засвідчили, що система зберігає високу швидкість незалежно від масштабів навантаження. Це свідчить про надійну реалізацію як архітектурних рішень, так і внутрішньої логіки системи.

У цілому розробка FinPal довела, що поєднання гнучкої архітектури, оптимізованої бази даних, сучасних інструментів візуалізації та AI-механізмів дозволяє створити ефективний, продуктивний та зручний фінансовий застосунок. Система відзначається стабільністю, високою якістю реалізації та потенціалом для подальшого розширення, що робить її надійним рішенням для управління особистими фінансами.

ВИСНОВКИ

У рамках виконаної роботи було проведено комплексне дослідження, проєктування та реалізацію системи управління особистими фінансами FinPal, яка поєднує в собі сучасні технології розробки програмного забезпечення, раціональні архітектурні принципи та інтелектуальні компоненти для автоматизації ключових операцій. У процесі дослідження проаналізовано підходи до створення десктопних додатків, вивчено актуальні інструменти та технології для побудови фінансових систем, після чого реалізовано повнофункціональний застосунок, здатний забезпечувати користувача інтуїтивним інтерфейсом, високою продуктивністю та глибокими аналітичними можливостями.

Основні результати та досягнення роботи

1. Аналіз інструментів та обґрунтування вибору технологій

Одним із ключових етапів дослідження став порівняльний аналіз засобів для створення фінансових застосунків. На основі отриманих результатів визначено, що Python є найбільш збалансованим вибором завдяки:

- ✓ простому та виразному синтаксису,
- ✓ розвинутій екосистемі бібліотек,
- ✓ вбудованій підтримці числової обробки, аналітики та машинного навчання,
- ✓ можливості швидкої інтеграції AI-компонентів.

Python дозволив реалізувати складний функціонал у стислий термін, не витрачаючи час на низькорівневу конфігурацію або надлишковий шаблонний код.

Для розробки інтерфейсу користувача обрано CustomTkinter, який надає:

- ✓ сучасний і приємний дизайн,
- ✓ підтримку тем та адаптивних компонентів,
- ✓ високу продуктивність у десктопному середовищі.

Як основну СКБД обрано SQLite, що завдяки файловій архітектурі, надійності й повній підтримці SQL повністю відповідає вимогам локальних застосунків: швидке завантаження, автономна робота, просте резервне копіювання в один файл.

2. Архітектура MVP та модульність системи

FinPal реалізовано за архітектурним патерном MVP (Model–View–Presenter), що забезпечило:

- ✓ чітке розділення відповідальностей,
- ✓ високу модульність,
- ✓ легкість підтримки та розширення,
- ✓ незалежність компонентів під час тестування.

Архітектура охоплює:

Model — робота з даними: база даних, AI-компоненти, статистика;

View — інтерфейсні компоненти, які відповідають за взаємодію з користувачем;

Presenter / MainController — обробка логіки, координація запитів і реакція на події.

Така структура дала можливість динамічно масштабувати систему, додавати нові модулі та легко адаптувати програму під нові функціональні вимоги без модифікації існуючого ядра.

3. Реалізація AI-компонентів та інтелектуальної аналітики

Важливою частиною системи стало впровадження інтелектуальних модулів:

CategoryPredictor — автоматично визначає категорію транзакції, аналізуючи поведінку користувача, текст опису та історичні дані;

FinancialAdvisor — виконує аналіз закономірностей витрат, виявляє тенденції, прогнозує можливі фінансові стани та надає рекомендації щодо оптимізації.

AI-компоненти працюють локально, без використання зовнішніх сервісів чи API, що гарантує:

- ✓ повну приватність фінансових даних,
- ✓ відсутність потреби в інтернет-з'єднанні,
- ✓ високу швидкодію аналізу.

4. Оптимізація продуктивності та забезпечення швидкої роботи

У процесі реалізації особливу увагу приділено оптимізації продуктивності:

Індексація таблиць забезпечила моментальний пошук та фільтрацію.

Оптимізовані SQL-запити, включаючи JOIN та агрегати, мінімізували навантаження на Python-логіку.

Виправлення N+1 проблеми скоротило кількість запитів у десятки разів.

Пагінація в інтерфейсі забезпечила швидке відображення навіть за наявності кількох тисяч транзакцій.

PRAGMA-параметри SQLite (WAL, NORMAL, cache_size, temp_store) забезпечили стабільну швидкодію під час інтенсивної роботи.

Досягнуто наступних показників продуктивності:

- ✓ запуск додатку — < 1 секунди,
- ✓ завантаження списку транзакцій — 0.1–0.3 с,
- ✓ побудова графіків — 0.2–0.5 с,
- ✓ додавання транзакції — < 0.1 с.

Такі результати підтверджують, що система відповідає всім вимогам до швидкодії настільного фінансового застосунку.

5. Повнофункціональність та зручність використання

FinPal реалізує весь спектр можливостей, необхідних для комплексного управління фінансами:

- ✓ додавання та редагування транзакцій із валідацією та AI-підказками;
- ✓ гнучке керування категоріями та тегами;
- ✓ статистика за періодами та динамічні графіки;
- ✓ виявлення закономірностей і рекомендації щодо економії;
- ✓ повноцінне резервне копіювання даних;
- ✓ інтуїтивний інтерфейс на основі сучасного дизайну.

Система орієнтована на максимальну зручність та адаптивність до потреб користувача.

Перспективи розвитку

Створена архітектура дозволяє розширювати систему без суттєвих змін у базовій логіці. У майбутньому можливе впровадження:

- ✓ інтеграції з банківськими API для автоматичного імпорту транзакцій,
- ✓ багатовалютності,
- ✓ механізмів планування бюджету,
- ✓ цілей та лімітів витрат,
- ✓ мобільної версії з синхронізацією між пристроями,
- ✓ розширених AI-можливостей (глибинне прогнозування, рейтинги фінансової активності, пошук аномалій).

Завдяки модульній структурі додавання нових функцій не потребуватиме переписування існуючого коду.

Проект FinPal є комплексним рішенням для управління особистими фінансами, яке:

- ✓ побудоване на сучасних технологіях,
- ✓ має продуману модульну архітектуру,

- ✓ демонструє високу продуктивність,
- ✓ включає AI-компоненти для інтелектуальної підтримки,
- ✓ забезпечує зручний досвід користувача,
- ✓ готове до масштабування та подальшого розвитку.

Система повністю відповідає поставленим цілям і може бути використана як основа для створення комерційного продукту або масштабного фінансового сервісу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SQLite Documentation: <https://www.sqlite.org/docs.html>
2. Python 3.10 Documentation: <https://docs.python.org/3.10/>
3. CustomTkinter Documentation: <https://customtkinter.tomschimansky.com/documentation/>
4. Matplotlib Documentation: <https://matplotlib.org/stable/index.html>
5. Seaborn: Statistical Data Visualization: <https://seaborn.pydata.org/>
6. NumPy Documentation: <https://numpy.org/doc/stable/>
7. Pandas Documentation: <https://pandas.pydata.org/docs/>
8. Scikit-learn: Machine Learning in Python: <https://scikit-learn.org/stable/documentation.html>
9. ReportLab PDF Library Documentation: <https://www.reportlab.com/documentation/>
10. Pillow (PIL Fork) Documentation: <https://pillow.readthedocs.io/en/stable/>
11. openpyxl Documentation: <https://openpyxl.readthedocs.io/en/stable/>
12. CTkMessagebox Documentation: <https://github.com/Akascape/CTkMessagebox>
13. Learning Python: Powerful Object-Oriented Programming: <https://www.oreilly.com/library/view/learning-python-5th/9781449355722/>
14. Fluent Python: Clear, Concise, and Effective Programming: <https://www.oreilly.com/library/view/fluent-python-2nd/9781492056348/>
15. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython: <https://www.oreilly.com/library/view/python-for-data/9781491957653/>
16. Python Data Science Handbook: Essential Tools for Working with Data: <https://jakevdp.github.io/PythonDataScienceHandbook/>
17. Matplotlib: A 2D Graphics Environment: <https://doi.org/10.1109/MCSE.2007.55>
18. seaborn: statistical data visualization: <https://doi.org/10.21105/joss.03021>
19. Data Visualization: A Practical Introduction: <https://socviz.co/>

- 20.Using SQLite: <https://www.oreilly.com/library/view/using-sqlite/9781449394592/>
- 21.The Definitive Guide to SQLite: <https://doi.org/10.1007/978-1-4302-3226-1>
- 22.Learning SQL: Generate, Manipulate, and Retrieve Data: <https://www.oreilly.com/library/view/learning-sql-3rd/9781492057604/>
- 23.Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: <https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/>
- 24.Introduction to Machine Learning with Python: A Guide for Data Scientists: <https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/>
- 25.Artificial Intelligence: A Modern Approach: <http://aima.cs.berkeley.edu/>
- 26.Design Patterns: Elements of Reusable Object-Oriented Software: <https://www.oreilly.com/library/view/design-patterns-elements/0201633612/>
- 27.Clean Code: A Handbook of Agile Software Craftsmanship: <https://www.oreilly.com/library/view/clean-code-a/9780136083238/>
- 28.Patterns of Enterprise Application Architecture: <https://martinfowler.com/books/eaa.html>
- 29.Clean Architecture: A Craftsman's Guide to Software Structure and Design: <https://www.oreilly.com/library/view/clean-architecture-a/9780134494272/>
- 30.The Total Money Makeover: A Proven Plan for Financial Fitness: <https://www.ramseysolutions.com/store/books/the-total-money-makeover>
- 31.Rich Dad Poor Dad: What the Rich Teach Their Kids About Money: <https://www.richdad.com/>
- 32.Personal Finance For Dummies: <https://www.wiley.com/en-us/Personal+Finance+For+Dummies-p-9781119517894>
- 33.Stack Overflow Python Questions: <https://stackoverflow.com/questions/tagged/python>
- 34.CustomTkinter GitHub Repository: <https://github.com/TomSchimansky/CustomTkinter>

- 35.Real Python Tutorials: <https://realpython.com/>
- 36.Python Package Index (PyPI): <https://pypi.org/>
- 37.Refactoring.Guru Design Patterns: <https://refactoring.guru/design-patterns>
- 38.SQLite Tutorial: <https://www.sqlitetutorial.net/>
- 39.W3Schools Python Tutorial: <https://www.w3schools.com/python/>
- 40.GeeksforGeeks Python Programming: <https://www.geeksforgeeks.org/python-programming-language/>
- 41.Towards Data Science: <https://towardsdatascience.com/>
- 42.Python Developer's Guide: <https://devguide.python.org/>
- 43.Database Normalization Guide: <https://www.guru99.com/database-normalization.html>
- 44.PEP 8 Style Guide for Python Code: <https://peps.python.org/pep-0008/>
- 45.PEP 257 Docstring Conventions: <https://peps.python.org/pep-0257/>
- 46.Matplotlib Example Gallery: <https://matplotlib.org/stable/gallery/index.html>
- 47.Seaborn Example Gallery: <https://seaborn.pydata.org/examples/index.html>