Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Проєктування та розробка інформаційної системи аналізу особистих витрат користувача»

Виконала ст. гр. КН-22-1 _____ Істоміна К. В.
Керівник _____ Тронь В. В.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2026

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *29* » *січня* 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-22-1 *Істоміної Карини Вікторівни*

1. Тема кваліфікаційної роботи: *«Проєктування та розробка інформаційної системи аналізу особистих витрат користувача»*

затверджено наказом по університету № *254с* від *13.05.2026 р.*

2. Термін здачі кваліфікаційної роботи: *10.06.2026 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 102с., додатки, презентація у Microsoft PowerPoint (19 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Тронь В. В.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*13.03.2026*
2	*Розділ 1*	*20.04.2026*
3	*Розділ 2*	*24.05.2026*
4	*Висновки*	*30.05.2026*
5	*Оформлення кваліфікаційної роботи*	*31.05.2026*
6	*Підготовка презентації та графічного матеріалу*	*01.06.2026*
7	*Підготовка доповіді до захисту*	*10.06.2026*

6. Дата видачі завдання: *28.01.2026р.*

Керівник _____ / Тронь В. В./

7. Запевнення: *Я, Істоміна Карина Вікторівна, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студентка _____ / Істоміна К. В.

АНОТАЦІЯ

Істоміна К. В. Проєктування та розробка інформаційної системи аналізу особистих витрат користувача: кваліфікаційна робота бакалавра : 122 – Комп'ютерні науки. Кривий Ріг. Криворізький національний університет, 2026. 102 с., 28 рис., 17 табл., 2 додатки та 21 джерело.

Актуальність теми обумовлена зростанням кількості електронних фінансових операцій та необхідністю ефективного контролю особистих доходів і витрат. Розроблена система забезпечує автономність роботи, швидкодію та конфіденційність фінансових даних завдяки локальному збереженню інформації.

Метою роботи є проєктування та розробка інформаційної системи аналізу особистих витрат користувача з використанням локального збереження даних, що забезпечує облік фінансових операцій, аналіз витрат, формування звітів, візуалізацію даних та автономну роботу без підключення до мережі Інтернет.

У першому розділі проаналізовано сучасні програмні рішення у сфері Personal Finance Management, досліджено предметну область, сформовано функціональні вимоги до системи, визначено архітектуру і принципи роботи аналітичного та візуалізаційного модулів.

У другому розділі описано трирівневу архітектуру застосунку, спроєктовано базу даних SQLite (таблиця Expenses із шістьма полями), реалізовано CRUD-операції через Microsoft.Data.Sqlite. Розроблено шість модулів: додавання витрат із валідацією, редагування, видалення, пошук, фільтрацію та статистику засобами LINQ. Інтерфейс реалізовано на WPF/XAML з інтеграцією LiveChartsCore. Тестування за 15 тест-кейсами підтвердило коректну роботу всіх модулів.

Ключові слова:

ІНФОРМАЦІЙНА СИСТЕМА, АНАЛІЗ ОСОБИСТИХ ВИТРАТ, УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ, WINDOWS PRESENTATION FOUNDATION, WPF, C#, .NET, SQLite, MVVM, XAML

ЗМІСТ

ВСТУП

У сучасному цифровому суспільстві управління особистими фінансами є важливою складовою фінансової грамотності та ефективного використання власних грошових ресурсів. Водночас значна кількість фінансових операцій, які виконуються щоденно, ускладнює ручний облік витрат та аналіз фінансової активності.

Останнім часом для автоматизації процесів фінансового обліку активно використовують інформаційні системи класу Personal Finance Management (PFM), які дозволяють здійснювати контроль витрат, планування бюджету, аналіз фінансових операцій та візуалізацію статистичних даних. Сучасні системи управління особистими фінансами використовують засоби аналітики, графічного представлення інформації та автоматичної категоризації витрат, що значно спрощує процес прийняття фінансових рішень користувачем.

Актуальність цієї роботи полягає у проєктуванні та розробці інформаційної системи аналізу особистих витрат користувача з використанням локального збереження даних, що забезпечить підвищений рівень конфіденційності фінансових даних користувача, швидкий доступ до інформації, автономність роботи та незалежність від хмарної інфраструктури.

Метою дослідження є проєктування та розробка інформаційної системи аналізу особистих витрат користувача з локальним збереженням даних, а об’єктом дослідження - процеси обліку та аналізу особистих фінансових витрат. Предметом дослідження є методи, моделі та програмні засоби реалізації інформаційної системи обліку й аналізу витрат із локальним збереженням даних.

Для досягнення цієї мети було вирішено такі завдання:

– аналіз предметної області управління особистими фінансами;

– дослідження сучасних програмних систем обліку та аналізу витрат користувача;

– формування функціональних та нефункціональних вимог до інформаційної системи аналізу витрат;

– проєктування архітектури програмної системи та структури локальної бази даних;

– розробка механізму локального збереження фінансової інформації з використанням SQLite;

– реалізація модулів аналізу витрат, формування статистики та побудови графіків;

– проведення тестування інформаційної системи та аналіз результатів її роботи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Аналіз предметної області управління особистими фінансами

У сучасному цифровому суспільстві процес управління особистими фінансами є важливою складовою фінансової грамотності користувача. Активний розвиток електронних платежів, онлайн-банкінгу, мобільних застосунків та цифрових сервісів призводить до збільшення кількості фінансових операцій, які користувач здійснює щоденно. У результаті виникає необхідність постійного контролю доходів і витрат, аналізу фінансової активності та ефективного планування бюджету [1].

У сучасних наукових дослідження системи класу Personal Finance Management (PFM) розглядаються як інструменти підтримки фінансового прийняття рішень (financial decision support systems), що допомагають користувачам контролювати власні фінансові ресурси, планувати витрати та формувати навички фінансової дисципліни [2]. Використання цифрових систем управління фінансами дозволяє автоматизувати процеси складання бюджету, аналізу витрат, прогнозування фінансової активності та контролю заощаджень.

Предметна область управління особистими фінансами охоплює широкий спектр процесів, пов’язаних із обліком та аналізом фінансової інформації користувача. До основних процесів належать облік доходів і витрат, категоризація транзакцій, формування статистичних звітів, аналіз фінансової активності, контроль бюджету та прогнозування майбутніх витрат. Основною метою систем управління особистими фінансами є підвищення ефективності використання фінансових ресурсів користувача та покращення фінансової дисципліни [3].

У сучасних умовах більшість користувачів здійснюють значну кількість фінансових операцій щоденно. Ручний контроль витрат є малоефективним, особливо при використанні декількох способів оплати та різних фінансових сервісів. Саме тому інформаційні системи управління особистими фінансами

набувають все більшої популярності серед користувачів мобільних та десктопних додатків.

Основною метою систем управління особистими фінансами є підвищення ефективності використання фінансових ресурсів користувача та покращення фінансової дисципліни. Основні процеси управління особистими фінансами наведено на рис. 1.1.

Рисунок 1.1 – Основні процеси управління особистими фінансами [4]

У процесі аналізу предметної області було визначено основні проблеми, які виникають під час управління особистими фінансами. Однією з головних проблем є відсутність систематизованого обліку витрат, через що користувачам складно аналізувати власну фінансову активність. Крім того, значна кількість користувачів не здійснює детального планування бюджету, що негативно впливає на ефективність використання фінансових ресурсів.

Ще однією важливою проблемою є складність аналізу фінансових операцій при великій кількості транзакцій. Без використання спеціалізованих програмних засобів користувачеві складно визначити основні категорії витрат, проаналізувати структуру фінансової активності та виявити тенденції зміни витрат у часі. Також у сучасних дослідженнях зазначається, що низький рівень фінансової грамотності часто пов’язаний із відсутністю зручних інструментів аналізу та візуалізації

фінансової інформації [5].

Окрему увагу необхідно приділити проблемам безпеки та конфіденційності фінансових даних. Фінансова інформація користувача є однією з найбільш чутливих категорій персональних даних, тому її збереження потребує використання надійних механізмів захисту [6]. У сучасних cloud-based системах існують ризики витоку інформації, несанкціонованого доступу та залежності від сторонніх сервісів. Більшість користувачів здійснюють значну кількість транзакцій щоденно, що ускладнює ручний контроль витрат.

У межах предметної області можна виділити основні сутності, які використовуються в системах управління особистими фінансами. Основні сутності предметної області наведені у табл. 1.1.

Таблиця 1.1 – Основні сутності предметної області

Сутність	**Опис**
User	Користувач системи
Expense	Витрата користувача
Income	Дохід користувача
Category	Категорія витрат
Budget	Запланований бюджет
Report	Аналітичний звіт
Statistics	Статистичний звіт

Сучасні системи *Personal Finance Management* надають користувачам інструменти аналітики, які дозволяють виявляти закономірності у витратах та покращувати процес прийняття фінансових рішень. Такі системи повинні забезпечувати реєстрацію фінансових операцій, класифікацію витрат, формування статистичних звітів, побудову графіків та контроль виконання бюджету.

Однією з ключових особливостей сучасних інформаційних систем є спосіб збереження даних. Більшість сучасних платформ використовують *cloud-based architecture*, при якій фінансова інформація користувача зберігається на віддалених

серверах. Такий підхід забезпечує синхронізацію між пристроями та доступ до інформації з будь-якого місця, однак створює ризики конфіденційності та залежність від постійного Інтернет-з'єднання [6].

Альтернативою cloud-oriented systems є використання локального збереження даних (*local-first approach*). При локальному підході інформація зберігається безпосередньо на пристрої користувача, що забезпечує вищий рівень конфіденційності та автономності роботи системи. Для систем аналізу особистих витрат це є особливо важливим, оскільки користувач повинен мати можливість працювати із фінансовою інформацією навіть без доступу до мережі Інтернет.

Крім того, локальне збереження забезпечує вищу швидкодію застосунку завдяки відсутності необхідності постійної синхронізації із серверною інфраструктурою. Важливою перевагою також є спрощення резервного копіювання, оскільки вся інформація може зберігатися у вигляді локального файлу бази даних.

Для реалізації локального збереження даних доцільним є використання SQLite. Згідно з офіційною документацією, SQLite є автономним, безсерверним механізмом транзакційної бази даних SQL, що не потребує налаштування (*serverless SQL database engine*) [7]. SQLite широко використовується у мобільних застосунках, десктопних програмах та вбудованих системах завдяки високій швидкодії, підтримці ACID транзакцій та мінімальному використанню ресурсів [7].

1.2 Огляд існуючих програмних систем обліку та аналізу витрат

На сучасному ринку програмного забезпечення існує велика кількість інформаційних систем для управління особистими фінансами. Розвиток мобільних технологій, онлайн-банкінгу та цифрових фінансових сервісів сприяв активному поширенню застосунків класу *Personal Finance Management (PFM)*, основною метою яких є автоматизація процесів обліку та аналізу фінансової активності користувача [3].

Сучасні системи управління особистими фінансами дозволяють користувачам

контролювати доходи та витрати, здійснювати планування бюджету, формувати статистичні звіти та аналізувати структуру фінансових операцій. Більшість сучасних платформ підтримують функції автоматичної категоризації витрат, синхронізації між пристроями та побудови графіків для візуалізації фінансових даних [1].

До найбільш популярних систем управління особистими фінансами належать:

– Mint;
– YNAB (You Need A Budget);
– Wallet;
– Monefy;
– Money Manager;
– Goodbudget;
– CoinKeeper.

У більшості випадків такі системи реалізуються у вигляді мобільних застосунків або web-oriented platforms, що забезпечують користувачам доступ до фінансової інформації з різних пристроїв. Основні програмні системи управління фінансами наведені на рис. 1.2.

System / Feature	intuit mint	YNAB	Wallet	Money Manager	Monefy
Analytics	Advanced	Advanced	Advanced	Medium	Basic
Budgeting	Yes	Yes	Yes	Yes	Yes
Offline Mode	No	Partial	Partial	Yes	Yes
Cloud Sync	Yes	Yes	Partial	No	No
Reports	Yes	Yes	Yes	Yes	Limited
Price Model	Free (Premium features)	Paid (Subscription)	Free (Premium features)	Free (In-app purchases)	Free (Premium features)

Рисунок 1.2 – Програми сучасних систем управління фінансами

Однією з найбільш відомих систем управління особистими фінансами є *Mint*, розроблена компанією Intuit Inc. Mint функціонував як веб-додаток і мобільний додаток для користувачів США і Канади. Основною особливістю системи була автоматична синхронізація з банківськими рахунками користувача, що дозволяло автоматизувати процес обліку фінансових операцій.

Система Mint підтримувала функції автоматичної категоризації витрат, формування бюджету, фінансової аналітики та генерації статистичних звітів. Крім того, застосунок забезпечував користувачів сповіщеннями про платежі та перевищення встановленого бюджету. Завдяки інтеграції з банківськими сервісами користувач отримував можливість автоматичного імпорту фінансових операцій без необхідності ручного введення даних.

Попри значний функціонал, система Mint мала низку недоліків. Основним недоліком була залежність від cloud-based інфраструктури, оскільки всі фінансові дані користувачів зберігалися на віддалених серверах. Це створювало потенційні ризики конфіденційності та вимагало постійного доступу до мережі Інтернет [6]. Крім того, система орієнтувалася переважно на користувачів банківських сервісів США та Канади, що обмежувало можливість її використання в інших країнах.

Іншою популярною системою є *YNAB (You Need A Budget)* - онлайн-платформа для управління особистим бюджетом, заснована на методології *envelope budgeting system* (система бюджетування за конвертами). Основна ідея системи полягає у розподілі коштів користувача між окремими категоріями бюджету, що дозволяє більш ефективно контролювати витрати та планувати фінансову діяльність.

Система YNAB підтримує функції планування бюджету, контролю фінансових цілей, аналізу витрат та формування статистичних звітів. Однією з переваг системи є сучасний UI/UX design та детальна фінансова аналітика. Крім того, YNAB активно використовує механізми візуалізації фінансової інформації, що дозволяє користувачам швидше аналізувати власну фінансову активність.

Проте використання YNAB має певні обмеження. Насамперед система працює за subscription-based model, тобто більшість функціоналу доступна лише після

оформлення платної підписки. Також система може бути складною для нових користувачів через специфічний підхід до планування бюджету та велику кількість аналітичних функцій.

Також, однією з популярних мобільних систем для обліку витрат є *Money Manager*. Основною особливістю цього застосунку є підтримка локального збереження даних та можливість роботи offline. Система дозволяє користувачам здійснювати облік витрат і доходів, формувати статистику та будувати графіки без необхідності постійного підключення до мережі Інтернет.

Money Manager характеризується простим та інтуїтивно зрозумілим інтерфейсом, що робить систему зручною для щоденного використання. На відміну від багатьох cloud-oriented платформ, застосунок забезпечує автономність роботи та швидкий доступ до інформації завдяки локальному збереженню даних. Це є важливою перевагою для систем аналізу особистих витрат, оскільки користувач отримує можливість працювати із фінансовими даними навіть без Інтернет-з’єднання.

Разом із тим Money Manager має менш розвинені аналітичні можливості порівняно із cloud-based системами. Система підтримує базові механізми побудови графіків та формування статистики, однак має обмежені можливості інтеграції із зовнішніми сервісами та недостатню масштабованість для складних фінансових сценаріїв.

Ще однією популярною системою є *Monefy* - мобільний застосунок для швидкого обліку особистих витрат. Основною особливістю системи є мінімалістичний інтерфейс та спрощений процес додавання фінансових операцій. Monefy орієнтований насамперед на користувачів, які потребують швидкого обліку витрат без використання складних аналітичних інструментів.

Система підтримує локальне збереження інформації, побудову базових графіків та категоризацію витрат. Основними перевагами застосунку є простота використання, висока швидкодія та підтримка offline mode. Проте аналітичні можливості системи є обмеженими, а функціонал статистичних звітів поступається

більш складним платформам.

Система *Wallet* також належить до сучасних платформ управління фінансами та підтримує синхронізацію із банківськими сервісами, аналітику витрат і планування бюджету. Основною особливістю Wallet є поєднання хмарної синхронізації та офлайн-функціональності. Система підтримує багатовалютність, фінансові звіти та механізми спільного доступу до бюджету.

Попри широкий функціонал, Wallet частково залежить від серверної інфраструктури та Інтернет-з'єднання. Крім того, окремі аналітичні функції доступні лише у преміум версії. Для більш наочного порівняння основних функціональних можливостей систем управління фінансами доцільно використати табл. 1.2.

Таблиця 1.2 – Порівняння існуючих систем управління фінансами

Система	**Локальне сховище**	**Офлайн-режим**	**Аналітика**	**Безкоштовна версія**
Mint	Ні	Ні	Розширена	Частково
YNAB	Ні	Частково	Розширена	Ні
Wallet	Частково	Частково	Розширена	Частково
Money Manager	Так	Так	Середня	Так
Monefy	Так	Так	Базова	Так

Проведений аналіз показує, що більшість сучасних систем управління фінансами орієнтовані на використання cloud-based architecture та інтеграцію із банківськими сервісами. Такі системи забезпечують потужні аналітичні можливості, проте створюють ризики конфіденційності та залежність від Інтернет-з'єднання [6].

У свою чергу, системи з локальним збереженням даних забезпечують вищий рівень автономності та конфіденційності, однак часто мають менш розвинений функціонал аналітики. Саме тому під час проєктування інформаційної системи аналізу особистих витрат доцільно поєднати переваги локального збереження даних із сучасними засобами фінансової аналітики та візуалізації інформації.

1.3 Аналіз функціональних можливостей та обмежень існуючих рішень

Аналіз сучасних систем управління особистими фінансами показує, що більшість програмних рішень орієнтовані на автоматизацію процесів фінансового обліку, бюджетування та аналітики. Такі системи дозволяють користувачам контролювати власні доходи та витрати, формувати фінансові звіти, аналізувати структуру витрат та прогнозувати майбутні фінансові операції [1].

У сучасній науковій літературі системи класу Personal Finance Management (PFM) розглядаються як інструменти підтримки фінансового прийняття рішень (*financial decision support systems*), які сприяють підвищенню рівня фінансової грамотності користувачів та покращенню контролю за особистими фінансами [3]. Дослідження показують, що використання цифрових систем аналізу витрат позитивно впливає на формування навичок бюджетування, дозволяє користувачам більш ефективно контролювати власні фінансові ресурси та приймати обґрунтовані фінансові рішення [5].

Більшість сучасних платформ мають схожий базовий функціонал, однак відрізняються реалізацією архітектури, способом збереження даних, рівнем безпеки, механізмами синхронізації та аналітичними можливостями. Основними функціями сучасних систем управління фінансами є реєстрація витрат і доходів, планування бюджету, побудова статистичних звітів, візуалізація фінансових даних та підтримка механізмів резервного копіювання. На основі аналізу існуючих платформ можна виділити найбільш поширені функціональні можливості які наведені в табл. 1.3.

Таблиця 1.3 – Основні функції сучасних систем управління фінансами

Функція	**Характеристика**
Expense Tracking	Реєстрація та збереження витрат
Income Management	Облік доходів
Budget Planning	Планування бюджету
Financial Analytics	Аналіз фінансових даних
Visualization	Побудова графіків та діаграм
Notifications	Нагадування про платежі
Data Synchronization	Синхронізація між пристроями
Export Reports	Експорт звітів
Multi-Currency Support	Підтримка декількох валют
Cloud Backup	Хмарне резервне копіювання

Сучасні системи зазвичай використовують механізми автоматичної категоризації витрат, що дозволяє користувачам швидше аналізувати структуру власних фінансових операцій. Наприклад, витрати можуть автоматично відноситися до категорій «харчування», «транспорт», «розваги», «здоров'я», «освіта» або «комунальні послуги». Використання категоризації суттєво спрощує подальший аналіз фінансової активності та дозволяє будувати статистичні звіти для окремих категорій витрат.

Однією з ключових особливостей сучасних PFM-систем є використання засобів візуалізації даних. Графічне представлення фінансової інформації дозволяє користувачам швидше сприймати статистичні дані, аналізувати структуру витрат та виявляти тенденції зміни фінансової активності. Найбільш поширеними засобами візуалізації є pie charts, bar charts, line graphs, dashboards та financial summaries [9].

Для систем аналізу особистих витрат візуалізація є важливою складовою інтерфейсу користувача, оскільки графічне представлення інформації дозволяє швидше оцінити фінансовий стан та виявити найбільш витратні категорії. Наприклад, секторні діаграми дозволяють визначити співвідношення між категоріями витрат, гістограми - порівнювати витрати за окремими періодами, а лінійні графіки - аналізувати динаміку зміни фінансової активності у часі.

Попри значний функціонал, більшість сучасних систем мають суттєві недоліки. Основною тенденцією є використання cloud-based architecture, (наведено

на рис. 1.3.) тобто збереження фінансової інформації на віддалених серверах [10].

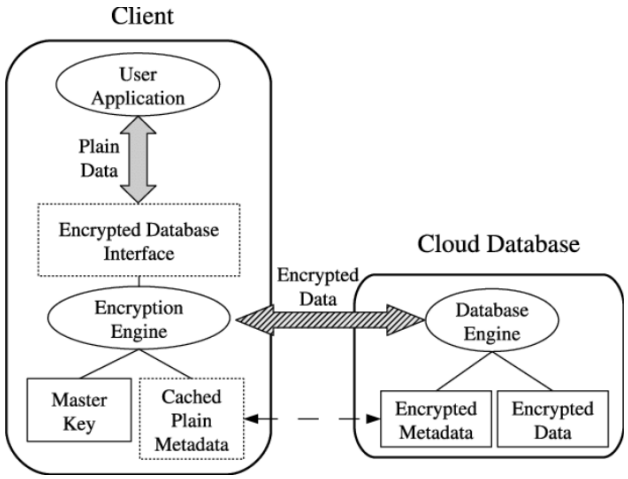


Рисунок 1.3 – Cloud-based архітектура

Cloud infrastructure забезпечує можливість синхронізації між пристроями, резервного копіювання та доступу до інформації з будь-якого місця. Завдяки цьому користувач може працювати із системою на декількох пристроях одночасно та не втрачати дані у випадку пошкодження локального пристрою.

Проте використання cloud-oriented architecture створює низку проблем, пов'язаних із конфіденційністю та безпекою персональних даних. Фінансова інформація належить до категорії чутливих персональних даних, тому її збереження у хмарному середовищі може призводити до ризиків витоку інформації або несанкціонованого доступу [6].

У сучасних дослідженнях з інформаційної безпеки зазначається, що фінансова галузь є однією з найбільш вразливих до інформаційних атак та витоків даних [11]. Основними ризиками cloud-based systems є:

- несанкціонований доступ до даних;
- витік фінансової інформації;
- кібератаки;
- залежність від сторонніх сервісів;
- зниження рівня контролю користувача над власними даними.

Крім того, централізоване збереження персональної інформації створює

додаткові загрози безпеці та ускладнює забезпечення конфіденційності даних користувача [6].

Ще одним недоліком сучасних cloud-oriented платформ є залежність від постійного підключення до мережі Інтернет. Більшість сучасних систем потребують стабільного Інтернет-з'єднання для синхронізації даних та роботи із серверною інфраструктурою. У результаті користувач може зіткнутися із затримками синхронізації, тимчасовою недоступністю сервісу або неможливістю повноцінної роботи в offline mode.

Для користувачів, які часто працюють без стабільного Інтернет-з'єднання, така залежність створює суттєві обмеження. Саме тому сучасні дослідження все частіше розглядають local first approach як альтернативу традиційній cloud architecture [12].

Окрему проблему становить використання subscription-based model. Більшість сучасних платформ надають лише базовий функціонал у безкоштовній версії, тоді як розширені аналітичні можливості, синхронізація між пристроями, резервне копіювання та експорт звітів доступні лише після оформлення платної підписки.

Ще одним недоліком сучасних систем є перевантаженість інтерфейсів користувача. Частина платформ містить надмірну кількість функцій, вкладених меню та аналітичних блоків, що ускладнює використання системи новими користувачами. У результаті складна навігація та перевантажений UI/UX негативно впливають на зручність роботи із системою.

Альтернативою cloud-based architecture є використання *local-first approach*, (наведено на рис. 1.4) при якому інформація зберігається безпосередньо на пристрої користувача [12]. У такій архітектурі користувач отримує повний контроль над власними даними та може працювати із системою незалежно від доступу до Інтернету. При цьому можна мати миттєвий доступ до своїх даних за будь-яких умов. Основні переваги локального збереження наведені в табл. 1.4.

Таблиця 1.4 – Переваги локального збереження даних

Перевага	Опис
Privacy	Дані не передаються стороннім серверам
Offline Access	Робота без Інтернету
High Performance	Висока швидкість доступу
Simplicity	Спрощена архітектура
Data Ownership	Повний контроль користувача

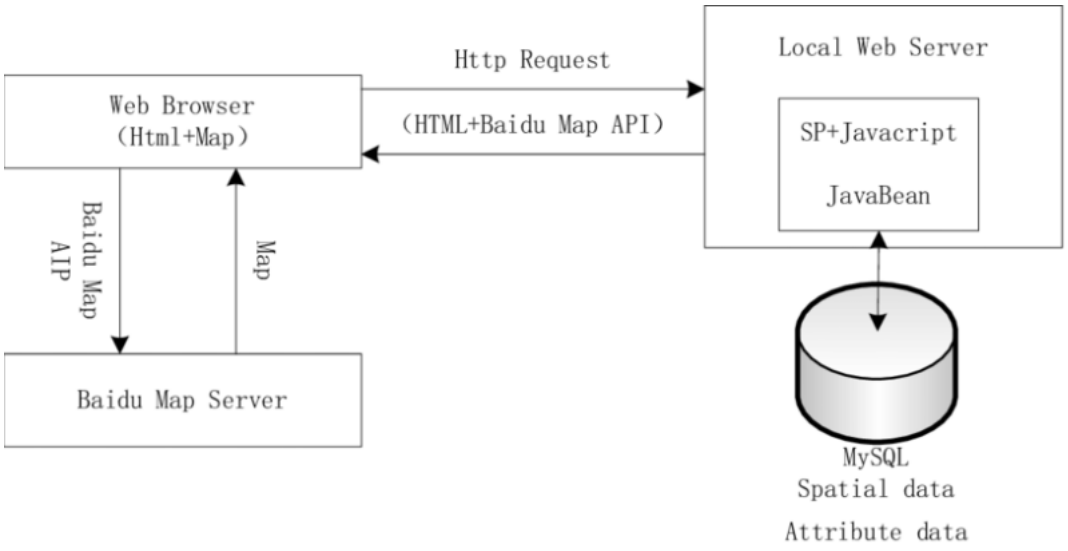


Рисунок 1.4 – Local-based архітектура [8]

Локальне збереження є особливо важливим для систем аналізу особистих витрат, оскільки фінансова інформація користувача не передається стороннім серверам та залишається під його повним контролем. Крім того, локальна архітектура забезпечує швидший доступ до інформації завдяки відсутності необхідності постійної синхронізації із серверною інфраструктурою.

Для реалізації локального збереження даних доцільним є використання SQLite. Відповідно до офіційної документації, SQLite є автономним, безсерверним механізмом транзакційної бази даних SQL, що не потребує налаштування (*self-contained, serverless, zero-configuration SQL database engine*) [7].

SQLite є однією з найбільш популярних embedded databases та активно використовується у мобільних застосунках, desktop applications, браузерах та фінансових інструментах [7]. Основною особливістю SQLite є те, що вся база даних зберігається у вигляді одного локального файлу, що значно спрощує архітектуру

інформаційної системи та процес резервного копіювання.

Перевагами SQLite є підтримка ACID transactions, висока швидкодія, кросплатформеність та мінімальне використання системних ресурсів [7]. Для систем аналізу особистих витрат це є важливою перевагою, оскільки дозволяє забезпечити швидку роботу системи навіть на малопотужних пристроях.

Попри значні переваги, SQLite також має певні обмеження. Основними недоліками є обмежена масштабованість, менша ефективність при великій кількості одночасних записів та відсутність повноцінної розподіленої архітектури [7].

Проте для системи аналізу особистих витрат ці недоліки не є критичними, оскільки система орієнтована переважно на однокористувацьке середовище, а обсяг фінансових даних є відносно невеликим. Саме тому використання SQLite є доцільним рішенням для реалізації локального збереження даних у межах даного дипломного проєкту.

1.4 Формування вимог до інформаційної системи аналізу витрат користувача

На основі проведеного аналізу предметної області та існуючих програмних рішень формується сукупність вимог до інформаційної системи аналізу особистих витрат користувача. Формування вимог є одним із ключових етапів проєктування програмного забезпечення, оскільки саме вимоги визначають майбутню архітектуру системи, її функціональні можливості, технологічний стек та особливості взаємодії користувача із програмою [13].

Сучасні системи класу Personal Finance Management (PFM) повинні забезпечувати не лише базовий облік фінансових операцій, а й підтримувати механізми аналітики, візуалізації та контролю бюджету. У наукових дослідженнях зазначається, що ефективні системи фінансового менеджменту дозволяють користувачам підвищувати рівень фінансової дисципліни, оптимізувати витрати та приймати більш обґрунтовані фінансові рішення [14].

Інформаційна система аналізу особистих витрат, яка розробляється у межах дипломного проєкту, повинна орієнтуватися на принцип local-first architecture, при якому основне збереження даних виконується локально на пристрої користувача. Такий підхід дозволяє підвищити конфіденційність фінансової інформації, забезпечити автономну роботу системи без постійного підключення до мережі Інтернет та мінімізувати залежність від зовнішніх серверних сервісів [12].

Основними вимогами до майбутньої системи є:

- ефективний облік доходів та витрат користувача;
- можливість швидкого аналізу фінансової активності;
- побудова статистичних звітів та графіків;
- підтримка локального збереження даних;
- стабільна робота без підключення до Інтернету;
- забезпечення конфіденційності персональної інформації;
- простий та інтуїтивно зрозумілий інтерфейс.

На відміну від багатьох сучасних cloud-oriented платформ, система повинна бути орієнтована на автономне використання. Це особливо важливо для користувачів, які не бажають передавати фінансову інформацію стороннім сервісам або працюють в умовах нестабільного Інтернет-з’єднання. Локальне збереження даних дозволяє зменшити ризики витоку персональної інформації та забезпечити повний контроль користувача над власними фінансовими даними [15].

Функціональні вимоги визначають основні можливості системи та сценарії взаємодії користувача із програмою. Однією з головних функцій є реєстрація фінансових операцій. Користувач повинен мати можливість додавати записи про витрати та доходи із зазначенням суми, категорії, дати, опису та типу операції. Для забезпечення зручності подальшого аналізу система повинна підтримувати редагування та видалення записів.

Кожна фінансова операція повинна містити набір атрибутів, необхідних для подальшої аналітичної обробки. Планується, що запис витрати міститиме такі поля:

- ID операції;
- тип операції (Expense / Income);
- сума;
- дата;
- категорія;
- опис;
- спосіб оплати;
- дата створення запису.

У системі планується використання декількох основних сутностей бази даних, між якими будуть встановлені логічні зв’язки. Основні таблиці надано в табл. 1.5.

Таблиця 1.5 – Основні таблиці бази даних системи

Таблиця	**Призначення**
Users	Збереження інформації про користувача
Expenses	Дані про витрати
Income	Дані про доходи
Categories	Категорії фінансових операцій
Budgets	Планування бюджету
Reports	Аналітичні звіти

Для забезпечення швидкого доступу до інформації система повинна підтримувати механізми пошуку та фільтрації даних. Користувач повинен мати можливість виконувати:

1. фільтрацію витрат за датою;
2. фільтрацію за категоріями;
3. пошук за описом операції;
4. сортування за сумою витрат;
5. перегляд статистики за певний період.

Аналітичний модуль системи є одним із ключових компонентів програмного забезпечення. Його основним призначенням є автоматичний аналіз фінансової активності користувача та формування статистичних звітів. У сучасних PFM-

системах аналітичні модулі дозволяють виявляти тенденції у витратах та оцінювати структуру фінансової поведінки користувача.

У межах проєкту планується реалізація таких аналітичних функцій, як: групування витрат за категоріями; розрахунок загальної суми витрат; побудова щомісячної статистики; аналіз динаміки витрат; визначення найбільш витратних категорій; розрахунок середніх витрат за період.

Для покращення сприйняття статистичної інформації система повинна підтримувати візуалізацію даних. Графічне представлення інформації дозволяє користувачу швидше аналізувати структуру витрат та виявляти фінансові тенденції [9].

У системі планується використання таких типів графіків:

- pie charts для відображення структури витрат;
- bar charts для порівняння витрат за категоріями;
- line charts для аналізу змін витрат у часі;
- dashboards для відображення загальної статистики.

Окрему увагу необхідно приділити нефункціональним вимогам. Однією з найважливіших вимог є надійність системи. Програмне забезпечення повинно забезпечувати стабільну роботу без втрати інформації навіть у випадку помилок або аварійного завершення роботи програми. Для реалізації цієї вимоги доцільним є використання SQLite, яка підтримує ACID-транзакції та забезпечує цілісність даних [16].

Важливою вимогою є також безпека персональних даних. Оскільки система працює із фінансовою інформацією користувача, необхідно мінімізувати ризики несанкціонованого доступу та витоку даних.

Крім цього, система повинна характеризуватися високою продуктивністю та низьким споживанням ресурсів. SQLite є ефективним рішенням для локальних інформаційних систем, оскільки не потребує окремого серверного програмного забезпечення та забезпечує швидке виконання SQL-запитів [16].

Інтерфейс користувача повинен бути інтуїтивно зрозумілим та адаптивним. Значна кількість сучасних фінансових систем має перевантажений інтерфейс, що ускладнює використання програмного забезпечення новими користувачами. Тому під час проєктування майбутньої системи необхідно забезпечити просту навігацію, логічне розташування елементів управління та зрозумілу структуру екранів.

Архітектура системи також повинна забезпечувати можливість подальшого розширення функціоналу. У перспективі система може бути доповнена:

1. модулем резервного копіювання;
2. підтримкою декількох валют;
3. автоматичною категоризацією витрат;
4. експортом звітів у PDF та Excel;
5. системою нагадувань про платежі.

Таблиця 1.6 – Основні вимоги до системи

Тип вимоги	**Опис**
Functional	Облік витрат та доходів
Security	Захист персональних даних
Performance	Висока швидкодія
Reliability	Надійність роботи
Offline Support	Робота без Інтернету
Analytics	Формування статистики
Visualization	Побудова графіків
Scalability	Можливість розширення

1.5 Постановка задачі проєктування системи

Метою дипломного проєкту є проєктування та розробка інформаційної системи аналізу особистих витрат користувача з використанням локального збереження даних. Основним призначенням системи є автоматизація процесів обліку фінансових операцій, аналізу витрат, формування статистичних звітів та візуалізації фінансових даних. У сучасних дослідженнях системи класу Personal Finance Management (PFM) розглядаються як інструменти підтримки прийняття фінансових рішень, які дозволяють підвищити ефективність управління

персональними фінансами та покращити фінансову дисципліну користувача [14].

У процесі проєктування система повинна забезпечувати не лише базову реєстрацію фінансових операцій, але й можливість подальшого аналізу структури витрат, контролю бюджету та формування візуальної статистики. Особливістю майбутньої системи є використання local-first architecture, при якій усі дані зберігаються безпосередньо на пристрої користувача без обов'язкової синхронізації із зовнішніми серверами [12]. Такий підхід дозволяє підвищити рівень конфіденційності фінансової інформації, забезпечити автономну роботу системи та мінімізувати ризики витоку персональних даних.

Для досягнення поставленої мети необхідно вирішити комплекс задач, пов'язаних із аналізом предметної області, проєктуванням архітектури системи, реалізацією локального збереження даних та створенням аналітичного модуля. На початковому етапі необхідно виконати аналіз існуючих програмних рішень та визначити їх переваги і недоліки. Особливу увагу доцільно приділити способам збереження даних, механізмам аналітики та рівню безпеки сучасних фінансових систем.

Наступним етапом є формування функціональних та нефункціональних вимог до системи. На основі проведеного аналізу визначено, що майбутня система повинна підтримувати сценарії створення, редагування та видалення фінансових операцій, здійснювати автоматичний розрахунок статистики витрат та забезпечувати візуалізацію даних у вигляді графіків і діаграм.

У межах дипломного проєкту також необхідно реалізувати механізм локального збереження даних із використанням SQLite. Вибір SQLite обумовлений її безсерверною архітектурою, підтримкою ACID-транзакцій та високою швидкодією при роботі з локальними застосунками [16]. Крім цього, SQLite не потребує встановлення окремого серверного програмного забезпечення, що значно спрощує структуру інформаційної системи.

Одним із ключових етапів проєктування є створення структури бази даних. База даних повинна забезпечувати швидкий доступ до фінансової інформації,

підтримку SQL-запитів та цілісність даних. У межах системи планується використання декількох основних таблиць, між якими будуть встановлені логічні зв’язки. Основні таблиці бази даних системи наведені у табл. 1.5.

Основною таблицею системи є таблиця Expenses (табл. 1.7) у якій буде зберігатися інформація про фінансові витрати користувача. Для забезпечення можливості аналітики та фільтрації даних запис витрати повинен містити декілька ключових полів.

Таблиця 1.7 – Основні поля таблиці Expenses

Поле	Тип даних	Призначення
ExpenseID	INTEGER	Унікальний ідентифікатор
Amount	REAL	Сума витрати
CategoryID	INTEGER	Категорія витрати
Date	DATE	Дата операції
Description	TEXT	Опис витрати
PaymentMethod	TEXT	Спосіб оплати
CreatedAt	DATETIME	Дата створення запису

Для підвищення зручності використання система повинна підтримувати різні сценарії роботи користувача. Основними сценаріями є:

- додавання нової фінансової операції;
- редагування запису;
- видалення витрати;
- перегляд статистики за певний період;
- аналіз витрат за категоріями;
- перегляд динаміки витрат;
- пошук фінансових операцій;
- фільтрація даних за датою або категорією.

Архітектура майбутньої системи повинна складатися з декількох основних компонентів, кожен із яких відповідатиме за окрему функціональність програмного

забезпечення. Саме такі компоненти показані в табл. 1.8.

Таблиця 1.8 – Основні компоненти інформаційної системи

Компонент	**Призначення**
User Interface	Взаємодія з користувачем
Business Logic Layer	Обробка фінансових операцій
Analytics Module	Аналіз та статистика
Database Layer	Збереження даних
SQLite Database	Локальна база даних

Інтерфейс користувача повинен забезпечувати просту та зрозумілу взаємодію із системою. Значна кількість сучасних PFM-систем має перевантажений інтерфейс та складну навігацію, що негативно впливає на користувацький досвід. Тому під час проєктування необхідно забезпечити логічне розташування елементів інтерфейсу, швидкий доступ до основних функцій та мінімізацію надлишкових елементів UI [17].

Окрему увагу необхідно приділити аналітичному модулю системи. Його основним призначенням є автоматичний аналіз фінансової активності користувача та формування статистичних звітів. У сучасних системах управління фінансами аналітичні інструменти дозволяють визначати структуру витрат, аналізувати тенденції та оцінювати ефективність бюджетування.

У межах проєкту планується реалізація таких видів статистики: щомісячна статистика витрат; статистика за категоріями; аналіз найбільших витрат; розрахунок середніх витрат та порівняння витрат за різні періоди.

Для візуалізації фінансових даних система повинна підтримувати побудову різних типів графіків та діаграм. Графічне представлення інформації дозволяє користувачу швидше аналізувати структуру витрат та виявляти фінансові тенденції [9].

У системі планується використання:

- pie charts для аналізу структури витрат;
- bar charts для порівняння категорій;
- line charts для відображення змін витрат у часі;

– dashboards для відображення загальної статистики.

Однією з ключових вимог до системи є підтримка автономної роботи без доступу до мережі Інтернет. Більшість сучасних фінансових платформ використовують cloud-based architecture, що створює залежність від серверної інфраструктури та мережевого з'єднання [10]. У запропонованій системі вся інформація буде зберігатися локально, що дозволить користувачу працювати із програмою незалежно від стану Інтернет-з'єднання.

Локальне збереження даних також позитивно впливає на рівень конфіденційності персональної інформації. Фінансові дані є чутливою категорією персональної інформації, тому передача таких даних стороннім сервісам створює ризики витоку та несанкціонованого доступу. Використання SQLite дозволяє забезпечити повний контроль користувача над власними фінансовими даними.

У результаті виконання дипломного проєкту повинна бути створена інформаційна система, яка дозволить вести облік доходів та витрат, здійснювати фінансову аналітику, формувати статистичні звіти та будувати графіки без необхідності використання хмарної інфраструктури. Система повинна забезпечувати високу швидкодію, автономність роботи та безпечне локальне збереження інформації.

Висновки до розділу

У першому розділі проведено аналіз предметної області управління особистими фінансами та досліджено сучасні підходи до організації систем класу Personal Finance Management (PFM). Розглянуто основні процеси управління особистими фінансами, зокрема облік доходів і витрат, бюджетування, аналіз фінансової активності, формування статистики та візуалізацію фінансових даних. Визначено, що ефективне використання інформаційних систем дозволяє підвищити рівень фінансової грамотності користувачів, покращити контроль за витратами та спростити процес прийняття фінансових рішень.

У роботі було виконано огляд існуючих програмних систем для обліку та аналізу особистих витрат, серед яких Mint, YNAB, Wallet, Monefy та Money Manager. Проведений аналіз дозволив визначити їх основні функціональні можливості, переваги та недоліки. Встановлено, що більшість сучасних систем використовують cloud-based architecture, яка забезпечує синхронізацію даних між пристроями та доступ до інформації через мережу Інтернет, проте водночас створює ризики втрати конфіденційності фінансових даних, залежність від серверної інфраструктури та обмеження автономної роботи системи.

Також у першому розділі було проаналізовано функціональні можливості сучасних систем управління фінансами, зокрема механізми категоризації витрат, побудови статистичних звітів, фінансової аналітики та візуалізації даних. Визначено, що важливою складовою таких систем є використання графічного представлення інформації у вигляді секторних діаграм, гістограм та аналітичних панелей, що дозволяє користувачам швидше аналізувати структуру власних витрат і виявляти фінансові тенденції.

У процесі дослідження обґрунтовано доцільність використання local-first approach та локального збереження даних із застосуванням SQLite. Визначено, що використання локальної бази даних забезпечує підвищений рівень конфіденційності, автономність роботи, швидкий доступ до інформації та спрощення архітектури програмної системи. Окремо розглянуто переваги SQLite як автономного безсерверного механізму з підтримкою ACID-транзакцій та однофайлової структури збереження даних.

На основі проведеного аналізу було сформовано функціональні та нефункціональні вимоги до інформаційної системи аналізу особистих витрат користувача. Визначено основні компоненти майбутньої системи, структуру бази даних, функції аналітичного модуля та модулів візуалізації статистичних даних. Отримані результати створюють основу для подальшого проєктування архітектури системи, розробки структури локальної бази даних та реалізації програмного забезпечення у наступних розділах дипломної роботи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ АНАЛІЗУ ОСОБИСТИХ ВИТРАТ

2.1 Архітектура та загальна структура інформаційної системи

Проєктування архітектури програмного забезпечення є одним із визначальних етапів розробки інформаційної системи, оскільки обрана архітектурна модель безпосередньо впливає на підтримуваність коду, масштабованість системи та якість взаємодії між її компонентами. Для розробленої інформаційної системи аналізу особистих витрат обрано трирівневу архітектуру із поділом відповідальностей між рівнями представлення, бізнес-логіки та даних. Така організація коду відповідає сучасним практикам розробки настільних застосунків і забезпечує незалежність кожного рівня від деталей реалізації суміжних [13].

Застосунок реалізовано на платформі .NET 8 із використанням технології Windows Presentation Foundation (WPF). WPF є частиною екосистеми .NET і забезпечує потужну систему зв'язування даних (data binding), декларативне визначення інтерфейсу за допомогою мови XAML та апаратне прискорення графічного відображення через DirectX. Вибір WPF обумовлений тим, що ця технологія надає зрілий інструментарій для побудови настільних застосунків із складним інтерфейсом, підтримує прив'язку даних до довільних джерел та дозволяє відокремити візуальну логіку від програмної. Альтернативи - WinForms та UWP - мають суттєві обмеження: WinForms не підтримує декларативної розмітки та має обмежені можливості стилізації, тоді як UWP вимагає специфічного середовища виконання та орієнтована переважно на застосунки з магазину Microsoft Store .

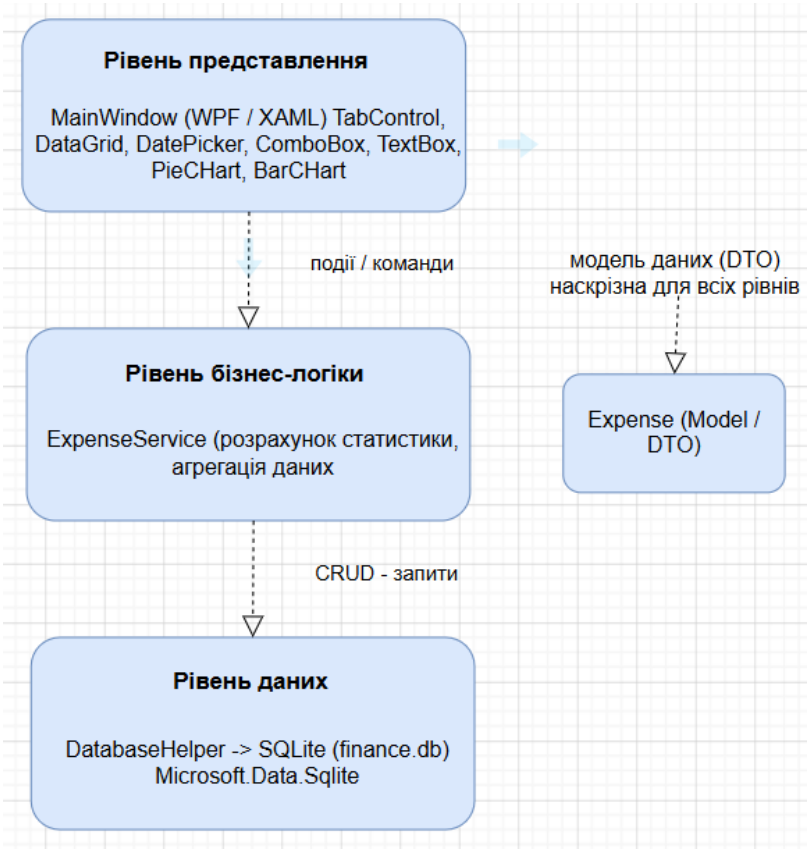


Рисунок 2.1 – Структурна схема трирівневої архітектури

Загальна структура системи складається з чотирьох взаємопов'язаних компонентів: вікна головного інтерфейсу *(MainWindow)*, допоміжного класу роботи з базою даних *(DatabaseHelper),* моделі предметної області *(Expense)* та сервісного класу обробки даних *(ExpenseService).* Взаємозв'язок між цими компонентами реалізовано таким чином, що *MainWindow* звертається до *DatabaseHelper* для виконання операцій читання та запису, а клас *Expense* слугує єдиним транспортним об'єктом - Data Transfer Object (DTO) - для передачі даних між рівнями. Такий підхід до організації коду забезпечує чіткість меж відповідальності та спрощує тестування окремих компонентів.

Клас *MainWindow* є центральним компонентом інтерфейсного рівня. Він успадковує базовий клас *Window* та реалізує інтерфейс *INotifyPropertyChanged,* що дозволяє системі прив'язки WPF автоматично оновлювати елементи інтерфейсу при зміні властивостей об'єкта. У *MainWindow* зосереджено основну логіку взаємодії з користувачем: обробники подій кнопок і полів введення, методи завантаження та

відображення даних, а також логіка ініціалізації графіків. Незважаючи на те що весь код взаємодії з інтерфейсом зосереджено в одному класі, структура методів дотримується принципу єдиної відповідальності: кожен метод відповідає за строго визначену дію - додавання, редагування, видалення, пошук або фільтрацію.

Клас *DatabaseHelper* реалізує шаблон проєктування Repository і є єдиною точкою доступу до локальної бази даних SQLite. У його конструкторі виконується ініціалізація підключення та у разі відсутності файлу бази даних, автоматичне створення схеми таблиць. Усі методи класу: *AddExpense, GetExpenses, UpdateExpense, DeleteExpense* - виконують SQL-запити з використанням параметризованих виразів, що виключає можливість SQL-ін'єкцій і відповідає вимогам безпечного програмування [16]. Клас використовує бібліотеку *Microsoft.Data.Sqlite,* яка є офіційним провайдером доступу до SQLite в екосистемі .NET і забезпечує зручний API для виконання запитів.

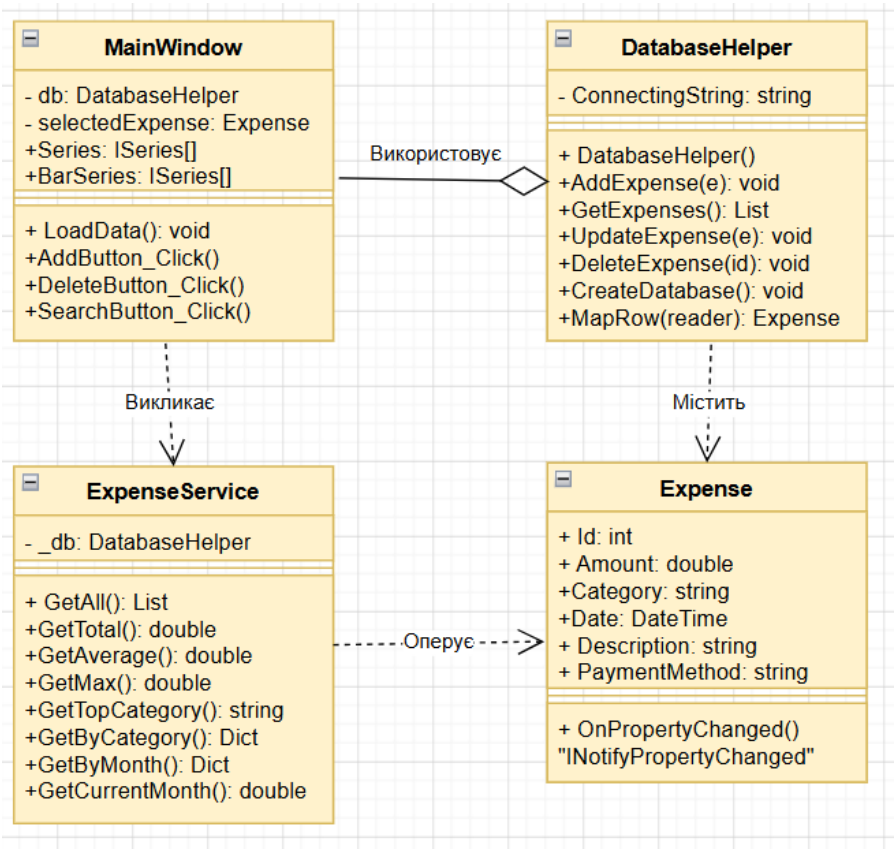


Рисунок 2.2 – UML-діаграма класів

Модель *Expense* є центральною сутністю предметної області. Вона описує структуру одного запису про фінансову витрату та містить такі властивості: унікальний ідентифікатор (*Id*), числова сума витрати (*Amount*), текстова назва категорії (*Category*), дата здійснення витрати (*Date*), текстовий опис (*Description*) та спосіб оплати (*PaymentMethod*). Клас не містить жодної бізнес-логіки - він слугує виключно носієм даних між рівнями системи. Реалізація *INotifyPropertyChanged* у цьому класі дозволяє елементам WPF автоматично реагувати на зміни значень при прив'язці даних через *DataGrid* та інші елементи управління.

Клас *ExpenseService* виконує роль сервісного рівня між інтерфейсом та сховищем даних. Він агрегує звернення до *DatabaseHelper* та реалізує методи обчислення статистичних показників: загальної суми витрат, середнього значення, максимальної витрати, суми за поточний місяць та розподілу витрат за категоріями. Винесення цих обчислень в окремий клас дозволяє уникнути дублювання коду та спростити тестування аналітичної логіки незалежно від рівня подання.

Взаємодія між компонентами системи організована у вигляді спрямованого графа залежностей: *MainWindow* залежить від *DatabaseHelper* та *ExpenseService; DatabaseHelper і ExpenseService* залежать від *Expense* як від моделі даних. Зворотних залежностей немає - модельний клас не знає нічого про інтерфейс або сховище. Такий однонаправлений потік залежностей відповідає принципу інверсії залежностей і спрощує подальше розширення системи, наприклад, у разі заміни SQLite на інше сховище або додавання нових аналітичних функцій.

Для підключення бібліотеки SQLite використано NuGet-пакет *Microsoft.Data.Sqlite* версії 8.x. Версія пакета узгоджується з цільовою платформою проєкту .NET 8, що забезпечує стабільність залежностей і доступність довгострокової підтримки (Long-Term Support). Бібліотека LiveCharts для побудови діаграм підключена через пакет *LiveChartsCore.SkiaSharpView.WPF*, що є офіційним провайдером LiveCharts 2 для WPF і використовує SkiaSharp як рушій відображення, незалежний від конкретної платформи UI [9].

Таблиця 2.1 – Основні компоненти інформаційної системи та їх відповідальність

Компонент	Тип	Відповідальність
MainWindow	WPF Window	Інтерфейс користувача, обробка подій, відображення даних
DatabaseHelper	Repository	Виконання CRUD-операцій з базою даних SQLite
Expense	Model / DTO	Транспортний об'єкт даних між рівнями системи
ExpenseService	Service	Обчислення статистичних показників, агрегація даних
Microsoft.Data.Sqlite	NuGet-пакет	Провайдер доступу до SQLite в екосистемі .NET
LiveChartsCore.SkiaSharpView.WPF	NuGet-пакет	Побудова інтерактивних діаграм у WPF-застосунку

Файл бази даних *finance.db* зберігається у поточній робочій директорії застосунку, що забезпечує переносність рішення: разом із виконуваним файлом може бути скопійовано на будь-який пристрій із встановленим середовищем .NET без потреби у налаштуванні серверного ПЗ. Рядок підключення є константою класу, що спрощує зміну шляху до файлу у разі потреби.

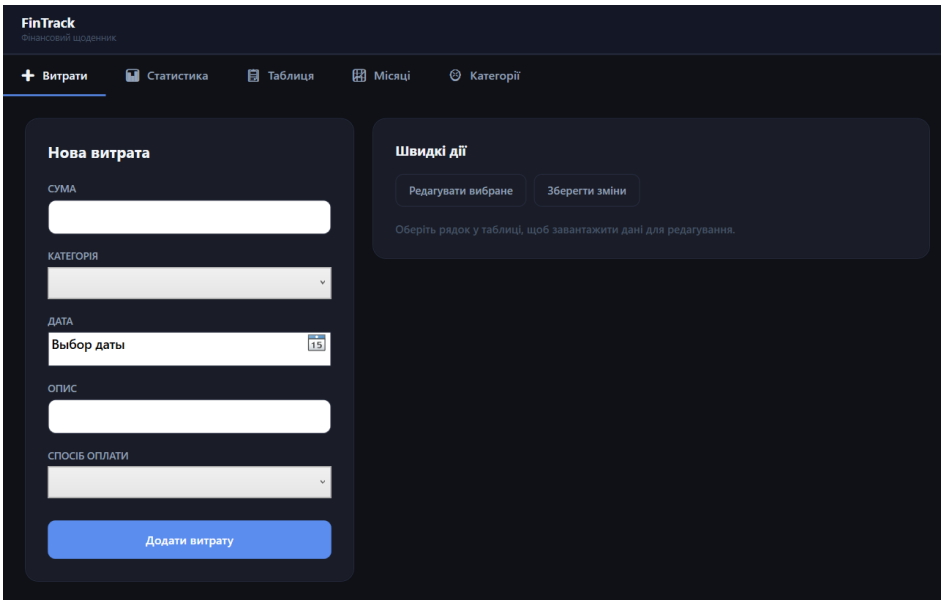


Рисунок 2.3 – Скріншот головного вікна застосунку

Процес запуску застосунку починається з точки входу класу *App*, що ініціалізує WPF-середовище та завантажує головне вікно. У конструкторі *MainWindow* виконується створення екземпляра *DatabaseHelper,* що автоматично ініціалізує базу даних, та перший виклик методу *LoadData(),* який завантажує всі записи із SQLite та заповнює таблицю інтерфейсу й статистичні блоки. Таким чином, при кожному запуску застосунку користувач одразу бачить актуальний стан своїх фінансових даних без необхідності виконання додаткових дій.

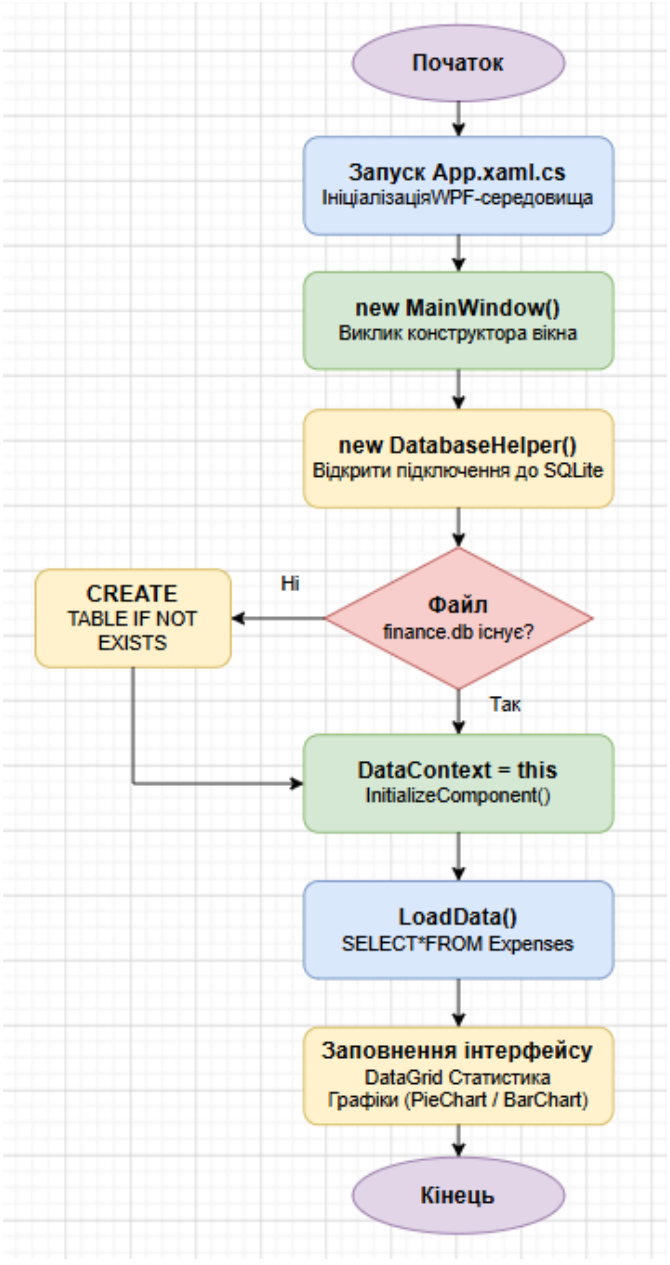


Рисунок 2.4 – Блок-схема ініціалізації застосунку

Отже, архітектура інформаційної системи побудована на принципах поділу відповідальностей, однонаправленої залежності між компонентами та централізованого управління доступом до даних. Застосування трирівневої структури з чітко визначеними класами *MainWindow, DatabaseHelper, Expense ma ExpenseService* забезпечує логічну організацію коду, полегшує подальший супровід системи та створює основу для розширення функціоналу у наступних версіях.

2.2 Проєктування структури даних та механізму локального збереження

Проєктування структури бази даних є одним із ключових етапів розробки інформаційної системи, оскільки від організації даних безпосередньо залежить ефективність виконання запитів, цілісність інформації та зручність подальшого розширення системи. У розробленій інформаційній системі аналізу особистих витрат для локального збереження даних використовується реляційна СКБД SQLite, яка забезпечує автономну роботу застосунку без необхідності встановлення окремого серверного програмного забезпечення [16].

Вибір SQLite обумовлений рядом технічних переваг, що є визначальними для настільного однокористувацького застосунку. По-перше, SQLite реалізує безсерверну архітектуру: уся база даних зберігається у єдиному файлі на диску, що спрощує розгортання, резервне копіювання та перенесення системи між пристроями. По-друге, SQLite підтримує повний набір ACID-транзакцій (Atomicity, Consistency, Isolation, Durability), що гарантує цілісність даних навіть у разі аварійного завершення роботи програми [7]. По-третє, бібліотека *Microsoft.Data.Sqlite* надає зручний та типобезпечний API для інтеграції SQLite з .NET-застосунками і є офіційно підтримуваним компонентом платформи .NET 8. Порівняння SQLite з альтернативними рішеннями за ключовими критеріями наведено у таблиці 2.2.

Таблиця 2.2 – Порівняння СКБД

Критерій	**SQLite**	**MS SQL Server LocalDB**	**LiteDB**
Серверна архітектура	Не потребує	LocalDB	Не потребує
Розмір бібліотеки	~1 МБ	~300 МБ	~450 МБ
ACID-транзакції	Так	Так	Так
Підтримка SQL	Повна	Повна	Обмежена
Інтеграція .NET	Microsoft.Data.Sqlite	Microsoft.Data.SqlClient	LiteDB NuGet
Однофайлове сховище	Так	Ні	Так
Використання в prod	Широке	Корпоративне	Обмежене
Підтримка платформи	Крос-платформна	Windows	Крос-платформна

Як видно з таблиці 2.2, SQLite є єдиним рішенням, яке одночасно відповідає вимогам безсерверної архітектури, підтримує повний синтаксис SQL та має офіційну інтеграцію з платформою .NET, що й обумовило його вибір для даної системи. Структура бази даних системи складається з однієї основної таблиці - *Expenses,* яка зберігає всі записи про фінансові витрати користувача. Рішення обмежитися однією таблицею обумовлене тим, що в межах поточної версії системи відсутня потреба у зберіганні окремих довідників категорій або користувачів - категорія зберігається безпосередньо у текстовому полі запису витрати, що спрощує структуру бази та пришвидшує виконання запитів. Детальний опис полів таблиці Expenses наведено у таблиці 2.3. У разі подальшого розширення системи, наприклад, для підтримки багатокористувацького режиму або ієрархічних категорій, структуру може бути нормалізовано до третьої нормальної форми шляхом виділення окремих таблиць.

Таблиця 2.3 – Структура таблиці Expenses

Поле	**Тип даних SQLite**	**Тип C#**	**Обмеження**	**Призначення**
Id	INTEGER	Int	PRIMARY KEY AUTOINCREMENT	Унікальний ідентифікатор запису
Amount	REAL	double	NOT NULL	Сума витрати
Category	TEXT	string	NOT NULL	Категорія витрати
Date	TEXT	DateTime	NOT NULL	Дата у форматі ISO 8601
Description	TEXT	string	DEFAULT ‘ ‘	Текстовий опис
PaymentMethod	TEXT	string	DEFAULT ‘ ‘	Спосіб оплати

Таблиця *Expenses* містить шість полів. Поле *Id* є цілочисельним первинним ключем із автоматичним збільшенням *(INTEGER PRIMARY KEY AUTOINCREMENT),* що забезпечує унікальну ідентифікацію кожного запису без ручного управління лічильником. Поле *Amount* зберігає суму витрати у форматі *REAL*, що відповідає типу *double* мови C# і забезпечує достатню точність для фінансових обчислень. Поле *Category* є текстовим *(TEXT NOT NULL)* і містить назву категорії витрати - наприклад, «Їжа», «Транспорт», «Розваги». Поле *Date* зберігає дату витрати у рядковому форматі ISO 8601 *(yyyy-MM-dd)*, що забезпечує коректне лексикографічне сортування дат та незалежність від локальних налаштувань системи. Поле *Description* містить довільний текстовий опис витрати, введений користувачем, а поле *PaymentMethod - спосіб оплати (наприклад, «Готівка», «Банківська картка», «Google Pay»).* Структурні зв’язки між сутністю та її атрибутами відображено на рис. 2.5.

Використання конструкції *CREATE TABLE IF NOT EXISTS* дозволяє виконувати цей запит при кожному запуску застосунку без ризику втрати даних: якщо таблиця вже існує, запит ігнорується. Це реалізовано у конструкторі класу *DatabaseHelper*, що забезпечує автоматичну ініціалізацію схеми при першому запуску програми.

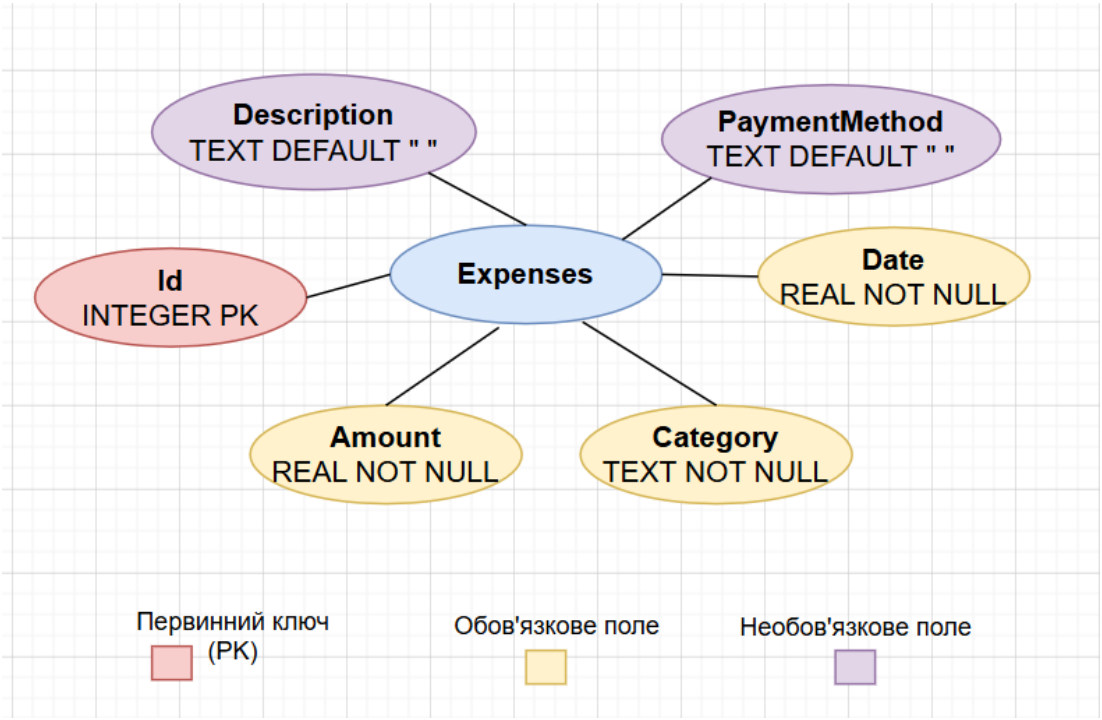


Рисунок 2.5 – ER-діаграма таблиці Expenses бази даних системи

Механізм роботи з даними реалізований через чотири CRUD-операції, кожна з яких відповідає окремому методу класу *DatabaseHelper*. Взаємозв'язок між операціями, методами репозиторію та таблицею бази даних наочно демонструє рис. 2.6.

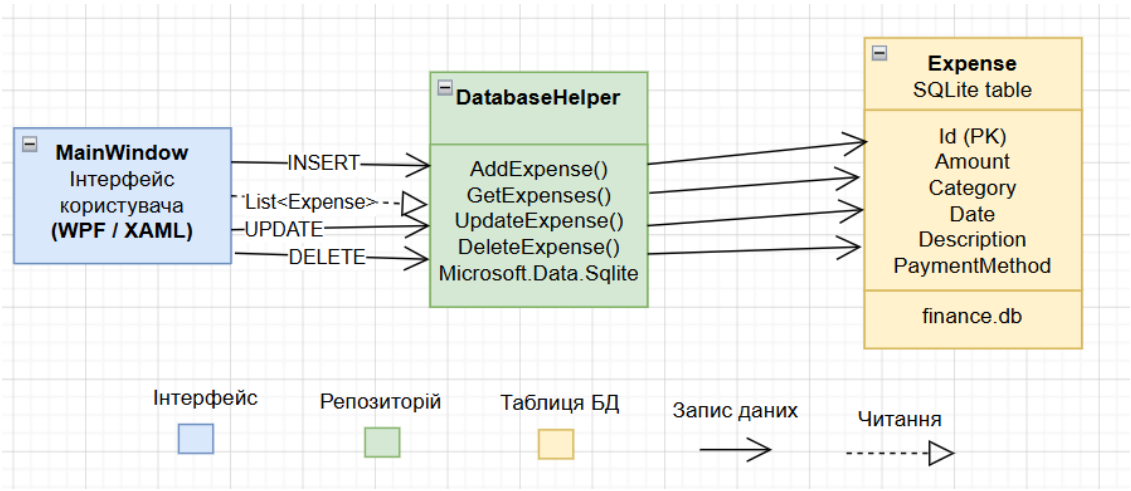


Рисунок 2.6 – Схема CRUD-операцій інформаційної системи з базою даних SQLite

Усі методи використовують параметризовані SQL-запити через об'єкт *SqliteCommand*, що унеможливлює SQL-ін'єкції та забезпечує коректну обробку спеціальних символів у текстових полях.

Операція додавання (*AddExpense)* формує запит *INSERT INTO Expenses* із п'ятьма параметрами та виконує його через *command.ExecuteNonQuery()*. Дата перетворюється на рядок формату *yyyy-MM-dd* перед збереженням, що гарантує однаковий формат незалежно від регіональних налаштувань операційної системи. Детально послідовність виконання цієї операції, включно з перевіркою введених даних, відображено на рис. 2.7.

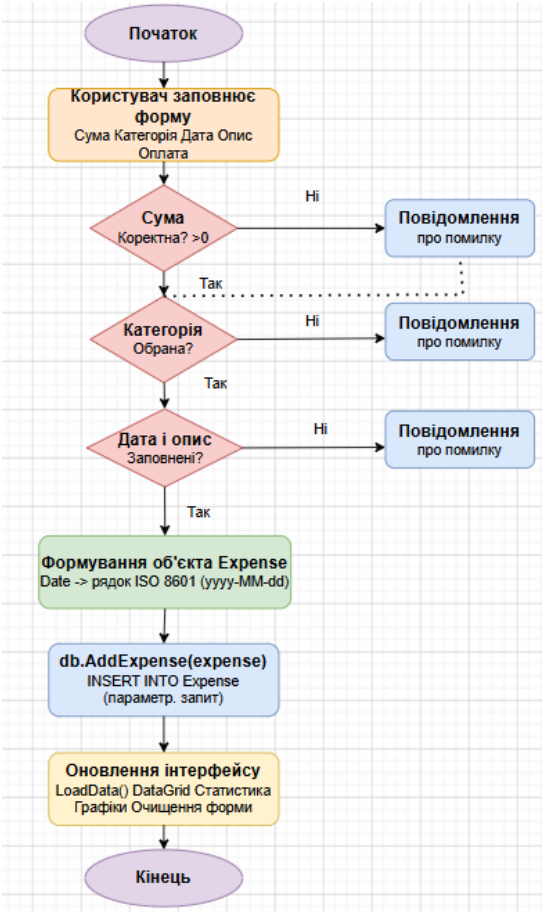


Рисунок 2.7 – Блок-схема операцій додавання витрати з валідацією

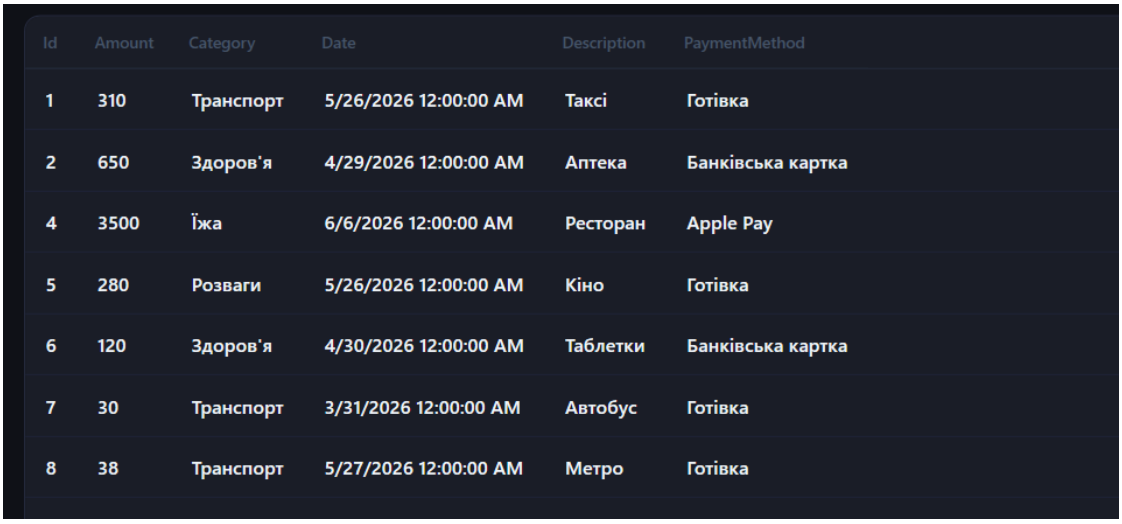
Операція читання *(GetExpenses)* виконує запит *SELECT* із сортуванням за датою у спадному порядку та повертає типізований список об'єктів *List<Expense>*. Дані зчитуються через *SqliteDataReader* рядково, і для кожного рядка формується новий екземпляр класу *Expense* з відповідними значеннями полів. Рядкове значення дати перетворюється назад у *DateTime через DateTime.Parse.*

Операція оновлення *(UpdateExpense)* формує запит *UPDATE Expenses SET ... WHERE Id = $id,* де як умова фільтрації використовується первинний ключ

запису. Це гарантує, що зміни застосовуються виключно до обраного запису, не торкаючись інших рядків таблиці.

Операція видалення *(DeleteExpense)* приймає лише цілочисельний ідентифікатор запису та виконує запит *DELETE FROM Expenses WHERE Id = $id*. Перед фактичним видаленням у шарі інтерфейсу відображається діалогове вікно підтвердження, що захищає від випадкової втрати даних.

Управління підключенням реалізовано з використанням конструкції *using var connection = new SqliteConnection(ConnectionString),* яка гарантує звільнення ресурсів після завершення операції незалежно від того, чи виникло виключення під час виконання запиту. Такий підхід відповідає рекомендаціям Microsoft щодо роботи з керованими ресурсами у .NET і запобігає витокам з'єднань при інтенсивному використанні застосунку. Практичний результат роботи механізму збереження даних можна спостерігати на скріншоті вкладки “Таблиця”, зображеному на рис. 2.8.



Id	Amount	Category	Date	Description	PaymentMethod
1	310	Транспорт	5/26/2026 12:00:00 AM	Таксі	Готівка
2	650	Здоров'я	4/29/2026 12:00:00 AM	Аптека	Банківська картка
4	3500	Їжа	6/6/2026 12:00:00 AM	Ресторан	Apple Pay
5	280	Розваги	5/26/2026 12:00:00 AM	Кіно	Готівка
6	120	Здоров'я	4/30/2026 12:00:00 AM	Таблетки	Банківська картка
7	30	Транспорт	3/31/2026 12:00:00 AM	Автобус	Готівка
8	38	Транспорт	5/27/2026 12:00:00 AM	Метро	Готівка

Рисунок 2.8 – Вкладка “Таблиця” інформаційної системи з переліком збережених витрат

Таким чином, структура бази даних спроєктована відповідно до принципів мінімальної достатності та практичної ефективності: єдина таблиця *Expenses* з шістьма полями повністю задовольняє функціональні вимоги системи, забезпечує швидке виконання аналітичних запитів і не ускладнює архітектуру надмірною

нормалізацією. Механізм CRUD-операцій через параметризовані запити гарантує безпеку та цілісність даних на рівні сховища.

2.3 Розробка функціональних модулів системи

Функціональна структура інформаційної системи аналізу особистих витрат складається з шести взаємопов'язаних модулів: додавання витрат, редагування записів, видалення записів, пошуку, фільтрації та формування статистики. Кожен модуль реалізований як окремий набір методів у класі *MainWindow* та відповідних методів класу *DatabaseHelper*, що забезпечує чіткий поділ відповідальності між рівнями системи. Загальну структуру функціональних модулів та їх взаємозв'язки відображено на рис. 2.9.

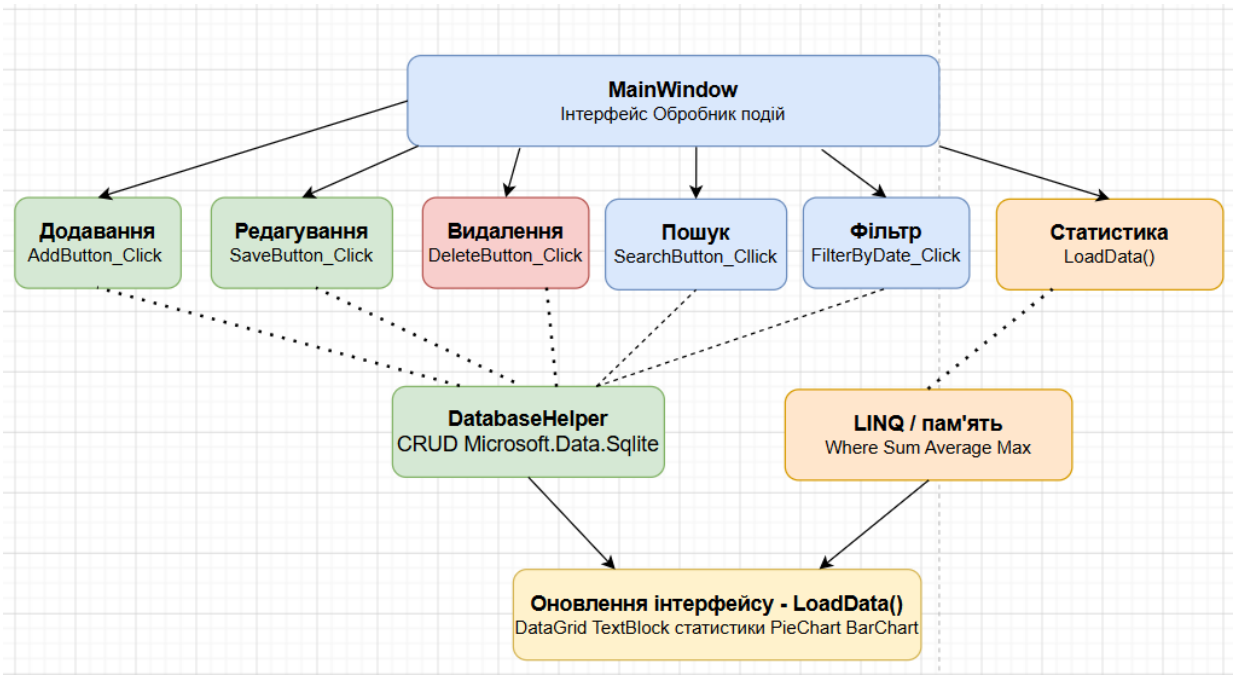


Рисунок 2.9 – Структура функціональних модулів інформаційної системи

Видно, що всі шість модулів взаємодіють із єдиним репозиторієм даних через клас *DatabaseHelper*, а результати їх роботи відображаються в інтерфейсі через метод *LoadData(),* який викликається після кожної модифікуючої операції.

Модуль додавання витрат. Логіка додавання нового запису зосереджена в

обробнику події *AddButton_Click.* При натисканні кнопки «Додати витрату» виконується послідовна валідація введених даних: перевіряється, чи є значення поля суми числовим і більшим за нуль, чи обрано категорію у *ComboBox*, чи вказано дату у *DatePicker*, чи заповнено текстове поле опису та чи обрано спосіб оплати. У разі порушення будь-якої умови виконання методу переривається і користувачу відображається повідомлення *MessageBox* із описом конкретної помилки. Якщо всі перевірки пройдено успішно, формується об'єкт класу *Expense* зі значеннями з відповідних полів форми, після чого викликається метод *db.AddExpense(expense)*. Після успішного збереження викликається *LoadData()* для оновлення таблиці та статистики, а поля форми очищаються методом *ClearFields().* Детальну послідовність виконання модуля додавання відображено на рис. 2.7 (підрозділ 2.2).

Модуль редагування записів. Редагування реалізовано через механізм завантаження даних обраного запису у форму введення. При виборі рядка у *DataGrid* спрацьовує обробник події *ExpensesGrid_SelectionChanged*, який зчитує об'єкт типу *Expense* із властивості *SelectedItem* та заповнює відповідні поля форми: *AmountTextBox*, *DescriptionTextBox*, *CategoryComboBox*, *ExpenseDatePicker* та *PaymentComboBox*. Обраний запис зберігається у приватному полі *selectedExpense* класу *MainWindow*. Після внесення змін користувач натискає кнопку «Зберегти», що викликає обробник *SaveButton_Click*. У цьому методі перевіряється, чи об'єкт *selectedExpense* не є null, після чого оновлені значення присвоюються полям об'єкта та викликається *db.UpdateExpense(selectedExpense)*. Алгоритм роботи модуля редагування наведено на рис. 2.10.

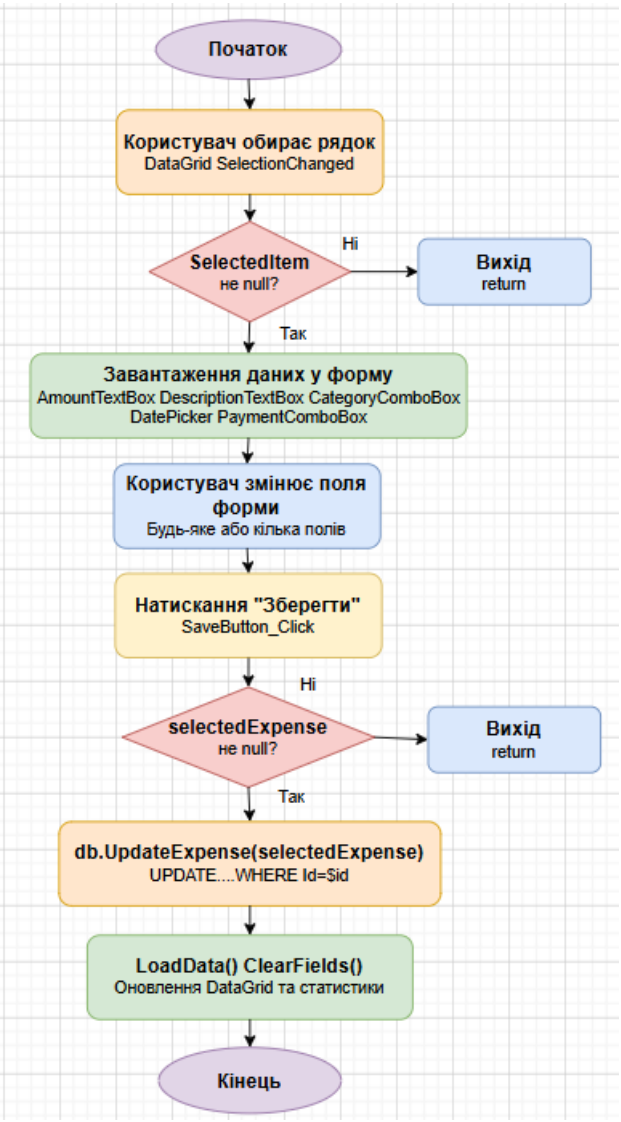


Рисунок 2.10 – Блок-схема модуля редагування запису про витрату

Модуль видалення записів. Видалення реалізовано в обробнику *DeleteButton_Click.* На першому кроці перевіряється наявність обраного рядка у *DataGrid* - якщо жодного рядка не обрано, відображається відповідне повідомлення. Далі об'єкт обраної витрати отримується через *ExpensesGrid.SelectedItem,* після чого відображається діалогове вікно підтвердження *MessageBox* з варіантами «Так» і «Ні». Видалення виконується лише у разі підтвердження *(MessageBoxResult.Yes)*, що захищає від випадкової втрати даних. Після підтвердження викликається *db.DeleteExpense(expense.Id)* із передаванням первинного ключа запису, а потім - *LoadData()* для оновлення інтерфейсу.

Модуль пошуку. Пошук реалізовано як клієнтську фільтрацію - усі записи завантажуються з бази даних одним запитом через *db.GetExpenses()*, а потім фільтруються у пам'яті засобами LINQ. Обробник *SearchButton_Click* отримує рядок пошуку з поля *SearchTextBox*, перетворює його у нижній регістр через *ToLower()* та відфільтровує список витрат за трьома полями одночасно: *Category*, *Description* та *PaymentMethod*. Результуючий список присвоюється властивості *ItemsSource* елемента *DataGrid*. Такий підхід забезпечує пошук за частковим збігом у будь-якому з трьох полів, що підвищує зручність використання системи. Алгоритм роботи модулів пошуку та фільтрації наведено на рис. 2.11.

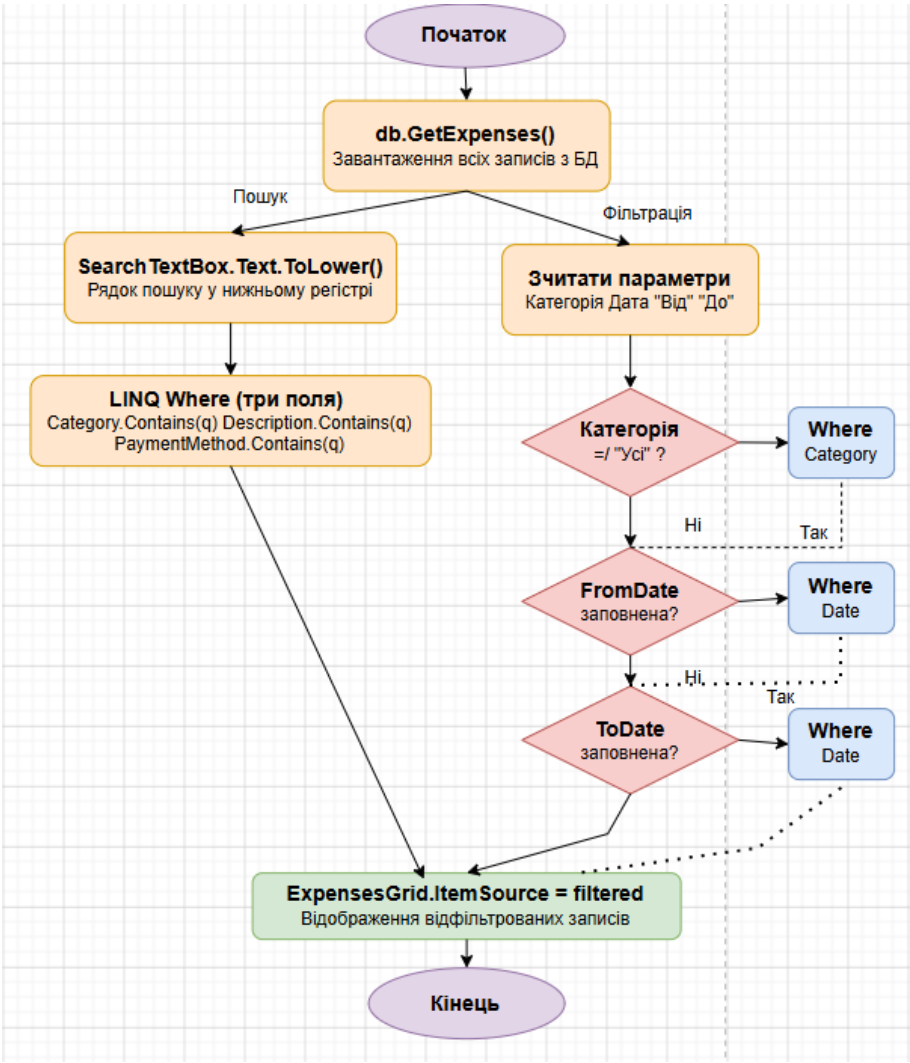


Рисунок 2.11 – Блок-схема модулів пошуку та фільтрації витрат

З рис. 2.11 видно, що обидва модулі використовують однотипну структуру:

завантаження повного набору даних, застосування умови фільтрації в пам'яті та передавання результату до *DataGrid*. Такий підхід обґрунтований невеликим очікуваним обсягом даних і дозволяє уникнути складних параметризованих SQL-запитів із динамічними умовами *WHERE*.

Модуль фільтрації. Система підтримує два незалежних типи фільтрації: за категорією та за діапазоном дат. Фільтрація за категорією реалізована в обробнику *FilterCategory_Click:* обране значення зчитується з *FilterCategoryComboBox*, і якщо воно не дорівнює «Усі», список витрат фільтрується за точним збігом поля *Category*. Фільтрація за діапазоном дат реалізована в обробнику *FilterByDate_Click:* якщо у полі *FromDatePicker* обрано дату, список обмежується записами з датою, більшою або рівною вказаній; аналогічно для *ToDatePicker* застосовується умова меншої або рівної дати. Обидва фільтри можуть застосовуватися незалежно: якщо поле дати не заповнено, відповідна умова не застосовується. Кнопка «Усі записи» скидає будь-яку активну фільтрацію, повторно викликаючи *LoadData().*

Модуль статистики. Статистичні показники обчислюються у методі *LoadData()* безпосередньо після завантаження повного списку витрат з бази даних. Усі обчислення виконуються засобами LINQ над колекцією об'єктів *List<Expense>* у пам'яті застосунку. Метод обчислює шість показників: загальну суму всіх витрат (*Sum*), середнє значення витрати (*Average*), кількість записів (*Count*), максимальну витрату (*Max*), категорію з найбільшою сумою витрат (групування за *Category* із сортуванням за спаданням) та суму витрат за поточний місяць (фільтрація за *Date.Month* та *Date.Year*). Результати присвоюються властивості Text відповідних елементів TextBlock у інтерфейсі. Після обчислення статистики метод *LoadData()* також ініціалізує серії даних для графіків *PieChart* та *BarChart* через *LiveCharts*. Логіку обчислення статистичних показників відображено на рис. 2.12.

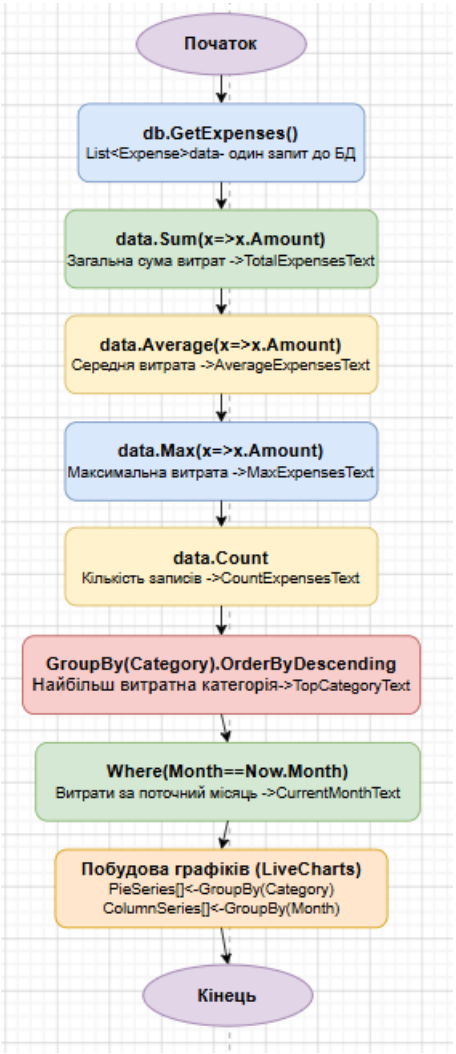


Рисунок 2.12 – Блок-схема модуля формування статистики

Модуль статистики послідовно обчислює всі шість показників з одного завантаженого набору даних, що виключає повторні звернення до бази даних та забезпечує узгодженість відображуваних значень.

Практичний результат роботи модулів фільтрації та пошуку можна спостерігати на рис. 2.13, де зображено інтерфейс панелі фільтрів застосунку.

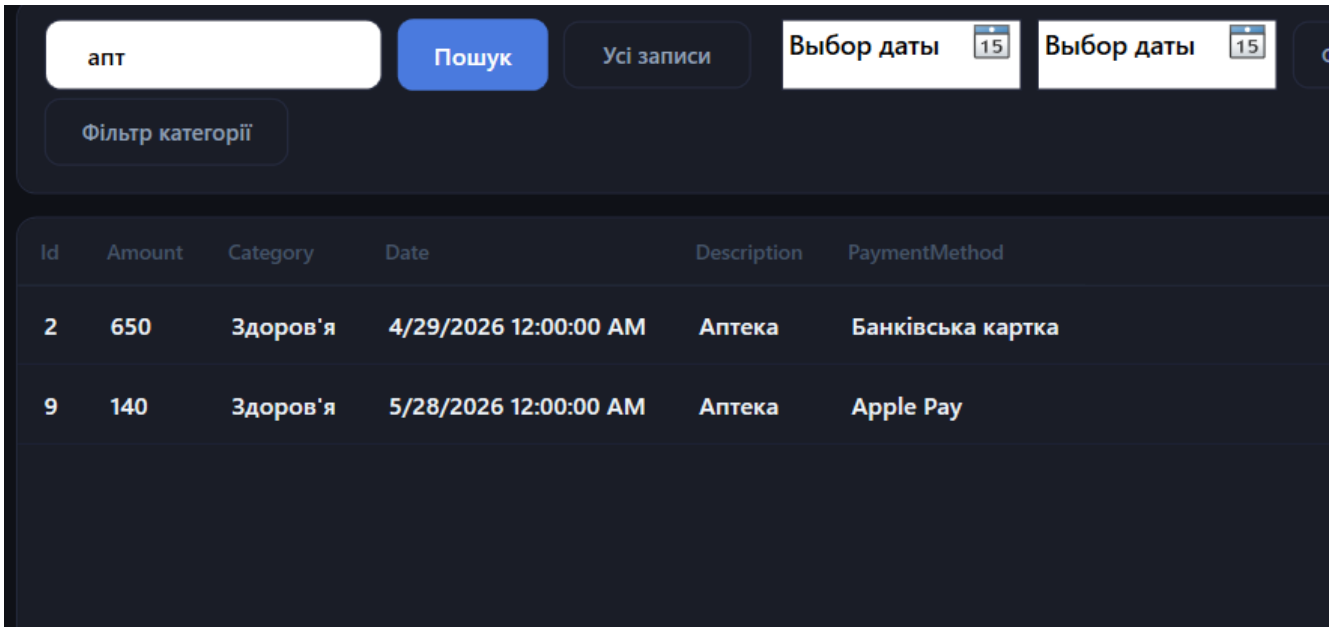


Рисунок 2.13 – Панель фільтрів та результати пошуку у вкладці «Таблиця»

Таким чином, усі шість функціональних модулів системи реалізовані у вигляді чітко розмежованих обробників подій та методів, що відповідають принципу єдиної відповідальності. Клієнтська фільтрація засобами LINQ забезпечує достатню продуктивність для очікуваного обсягу даних і спрощує логіку формування запитів до бази даних. Механізм послідовної валідації перед кожною модифікуючою операцією унеможливлює збереження некоректних даних і підвищує надійність системи в цілому.

2.4 Реалізація інтерфейсу користувача та засобів візуалізації даних

Інтерфейс користувача інформаційної системи реалізовано засобами технології Windows Presentation Foundation із використанням декларативної мови розмітки XAML. WPF надає потужну систему прив'язки даних, шаблони елементів керування та централізований механізм стилізації, що дозволяє відокремити візуальне представлення від програмної логіки та забезпечити однорідний зовнішній вигляд усіх елементів застосунку. Загальну структуру інтерфейсу з розподілом елементів по вкладках відображено на рис. 2.14.

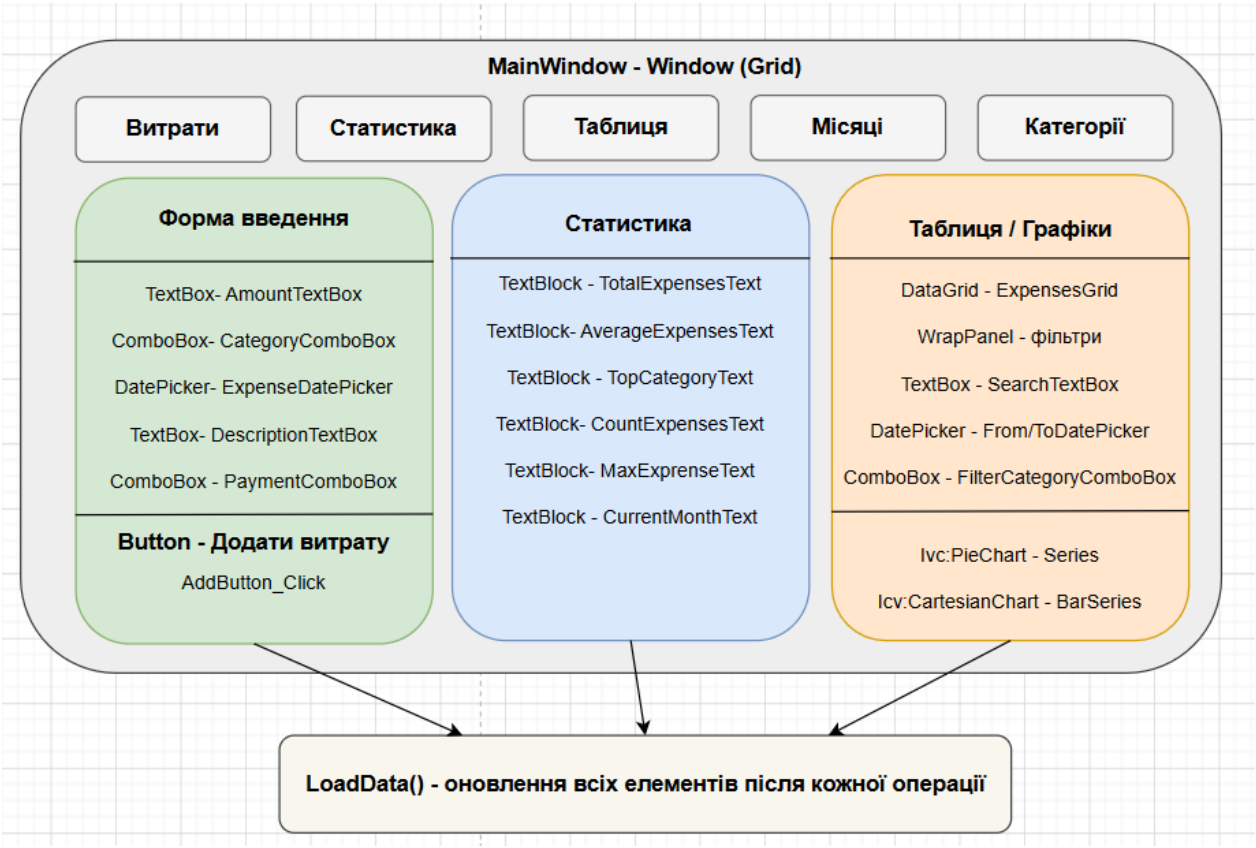


Рисунок 2.14 – Структура інтерфейсу користувача інформаційної системи

Інтерфейс організовано у вигляді п'ятивкладкової структури на основі елемента *TabControl*, де кожна вкладка відповідає окремій функціональній зоні системи: введення витрат, перегляд статистики, управління таблицею записів, стовпчикова діаграма та кругова діаграма.

Головний контейнер інтерфейсу - елемент *Grid* - організовує розмітку головного вікна *MainWindow*. Усі стилі елементів керування визначено безпосередньо у XAML-розмітці через властивості *Background, Foreground, BorderBrush, CornerRadius та FontFamily.* Система стилів WPF дозволяє визначати стилі як ресурси (*ResourceDictionary*) на рівні вікна або застосунку, що забезпечує їх повторне використання без дублювання коду розмітки. Ієрархію елементів керування головного вікна наведено у табл. 2.4.

Таблиця 2.4 – Елементи керування інтерфейсу та їх призначення

Елемент	**Тип WPF**	**Ім’я (x:Name)**	**Вкладка**	**Призначення**
Поле суми	TextBox	AmountTextBox	Витрати	Введення числової суми витрати
Категорія	ComboBox	CategoryComboBox	Витрати	Вибір категорії з фіксованого списку
Дата витрати	DatePicker	ExpenseDatePicker	Витрати	Вибір дати здійснення витрати
Опис	TextBox	DescriptionTextBox	Витрати	Введення текстового опису
Спосіб оплати	ComboBox	PaymentComboBox	Витрати	Вибір способу оплати
Загальна сума	TextBlock	TotalExpensesText	Статистика	Відображення загальної суми
Середня витрата	TextBlock	AverageExpensesText	Статистика	Відображення середнього значення
Топ категорія	TextBlock	TopCategoryText	Статистика	Назва найбільш витратної категорії
Кількість	TextBlock	CountExpensesText	Статистика	Загальна кількість записів
Максимум	TextBlock	MaxExpenseText	Статистика	Найбільша одинична витрата
Поточний місяць	TextBlock	CurrentMonthText	Статистика	Сума витрат за поточний місяць
Таблиця записів	DataGrid	ExpensesGrid	Таблиця	Перегляд, вибір записів для операцій
Пошуковий рядок	TextBox	SearchTextBox	Таблиця	Введення пошукового запису
Фільтр “Від”	DatePicker	FromDatePicker	Таблиця	Початок діапазону дат фільтрації

Продовження табл. 2.4.

Елемент	Тип WPF	Ім’я (x:Name)	Вкладка	Призначення
Фільтр “До”	DatePicker	ToDatePicker	Таблиця	Кінець діапазону дат фільтрації
Фільтр категорій	ComboBox	FilterCategoryComboBox	Таблиця	Фільтрація за обраною категорією
Кругова діаграма	Lvc:PieChart	-	Категорії	Розподіл витрат за категоріями
Стовпчикова діаграма	Lvc:CartesianChart	-	Місяці	Витрати за категоріями / місяцями

Вкладка «Витрати». Перша вкладка містить форму введення нового запису та блок швидкого редагування. Форма організована у вигляді вертикальної *StackPanel* і включає такі елементи: *TextBox* для введення суми *(AmountTextBox), ComboBox* для вибору категорії *(CategoryComboBox)* з чотирма фіксованими варіантами, *DatePicker* для вибору дати *(ExpenseDatePicker)*, *TextBox* для введення опису *(DescriptionTextBox)* та *ComboBox* для вибору способу оплати *(PaymentComboBox)* з п'ятьма варіантами. Кнопка «Додати витрату» розміщена внизу форми та ініціює обробник *AddButton_Click*. Праворуч від форми розміщено блок швидких дій із кнопками редагування та збереження, що дозволяє вносити зміни до обраного запису без переходу до іншої вкладки. Зовнішній вигляд вкладки введення витрат відображено на рис. 2.15.

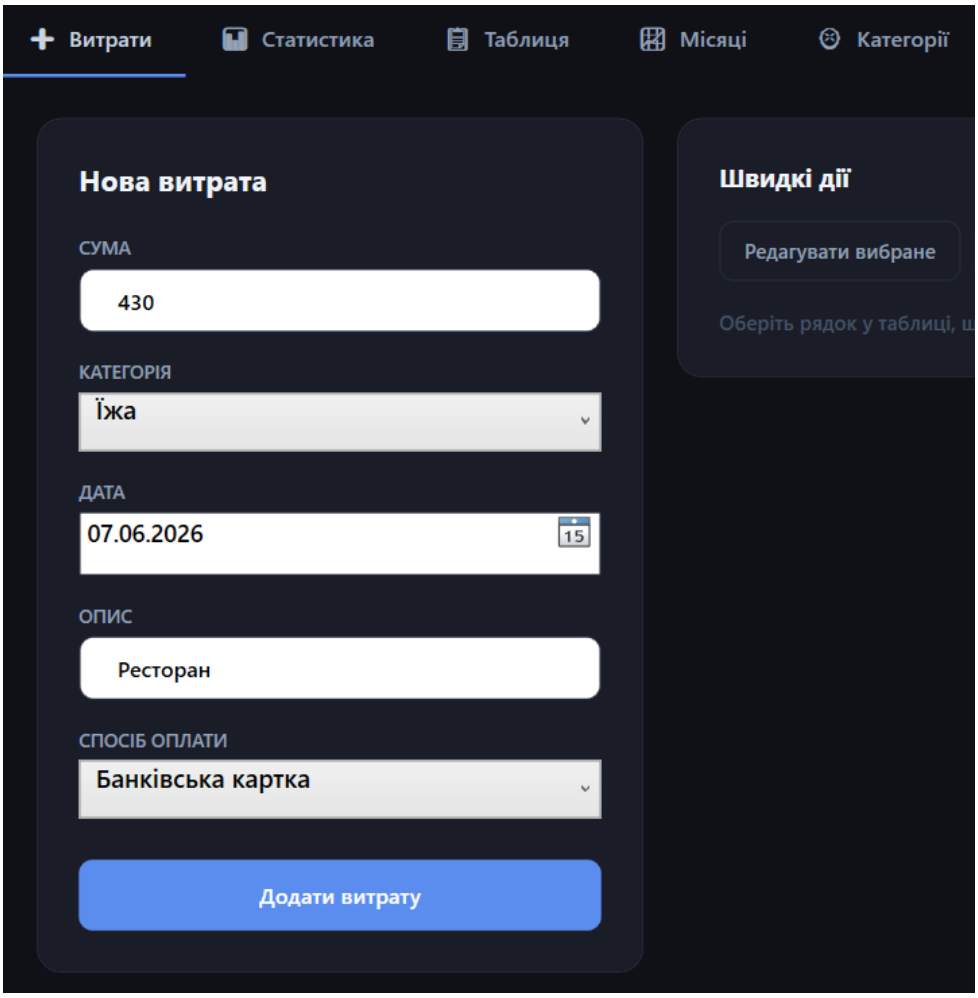


Рисунок 2.15 – Вкладка “Витрати” з формою введення нового запису

Вкладка «Статистика». Друга вкладка містить шість інформаційних блоків типу *Border* із вкладеними елементами *TextBlock*, які відображають ключові показники фінансового аналізу. Блоки розміщені у вертикальному *StackPanel* із чергуванням кольорового фону, що забезпечує візуальну диференціацію показників. Кожен блок містить один *TextBlock* із відповідним текстовим полем: *TotalExpensesText, AverageExpensesText, TopCategoryText, CountExpensesText, MaxExpenseText* та *CurrentMonthText*. Значення цих полів оновлюються програмно після кожної операції з даними через метод *LoadData()*. Зовнішній вигляд вкладки статистики наведено на рис. 2.16.

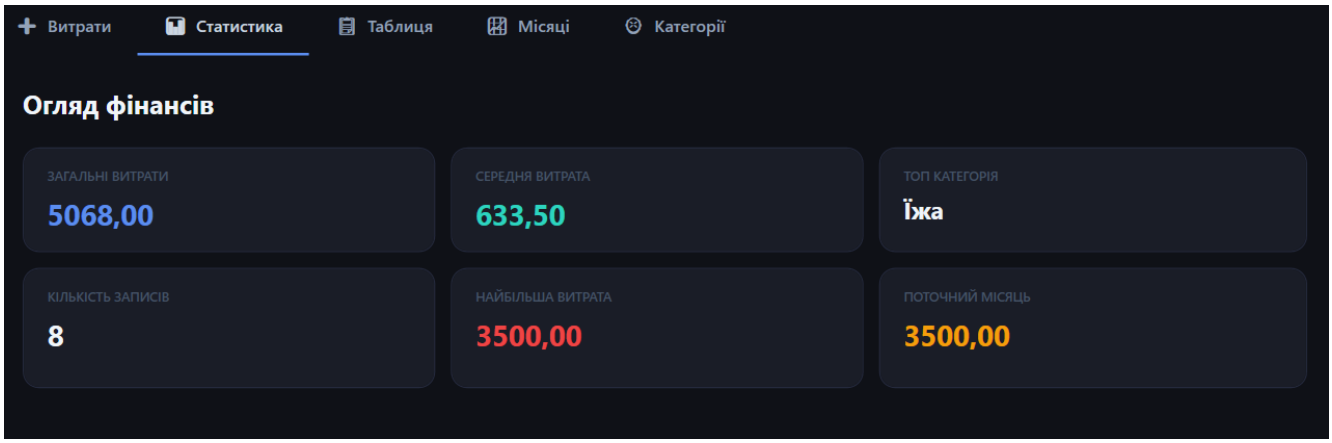


Рисунок 2.16 – Вкладка “Статистика” з ключовими фінансовими показниками

Вкладка «Таблиця». Третя вкладка є основним робочим простором для управління записами. Вона побудована на елементі *Grid* із трьома рядками: панель фільтрів *(WrapPanel)*, основна таблиця *(DataGrid)* та рядок кнопок керування. Елемент *DataGrid* налаштований з властивостями *AutoGenerateColumns="True", IsReadOnly="True", CanUserAddRows="False"* та *CanUserResizeRows="False"*, що забезпечує режим лише для читання з автоматичним формуванням стовпців на основі властивостей класу *Expense*. Обробник *SelectionChanged* відстежує вибір рядка для подальшого редагування або видалення. Панель фільтрів містить *TextBox* для пошуку, дві кнопки (*SearchButton* та *ShowAllButton*), два *DatePicker* для діапазону дат, *ComboBox* для фільтрації за категорією та відповідні кнопки застосування фільтрів.

Вкладки «Місяці» та «Категорії». Четверта та п'ята вкладки містять відповідно стовпчикову та кругову діаграми, реалізовані засобами бібліотеки LiveCharts 2. Кожна діаграма розміщена у власному *Grid*-контейнері та прив'язана до властивостей *BarSeries* та *Series* класу *MainWindow* через механізм *Binding*. Структуру прив'язки даних діаграм до властивостей ViewModel відображено на рис. 2.17.

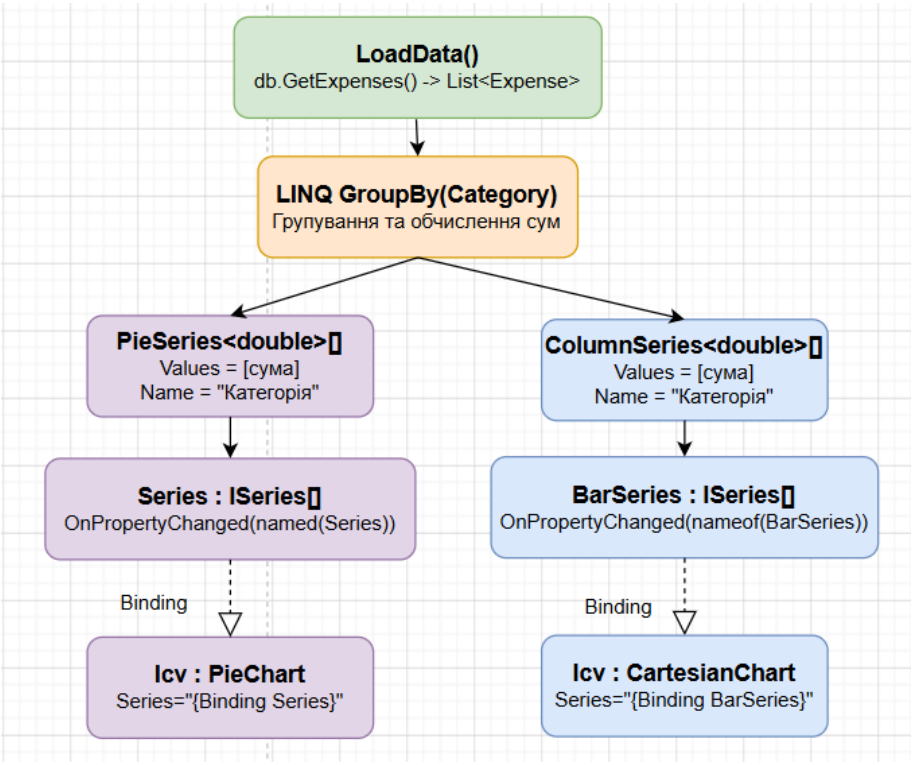


Рисунок 2.17 – Схема прив’язки даних між властивостями MainWindow та елементами LiveCharts

Обидва графіки отримують дані через стандартний механізм *Binding* WPF: зміна властивостей *Series* та *BarSeries* у коді автоматично оновлює відображення діаграм без явного звернення до елементів інтерфейсу. Це стає можливим завдяки реалізації інтерфейсу *INotifyPropertyChanged* у класі *MainWindow*. Для побудови діаграм використовується бібліотека *LiveChartsCore.SkiaSharpView.WPF* версії 2.x, яка є наступним поколінням LiveCharts із повністю переписаним рушієм відображення на основі *SkiaSharp* [9]. На відміну від першої версії бібліотеки, LiveCharts 2 підтримує прив'язку даних через стандартні інтерфейси *ISeries[]* та *IAxis[]*, що забезпечує сумісність із шаблонами MVVM та WPF-стилями прив'язки.

Кругова діаграма реалізована через елемент *lvc:PieChart* з властивістю *Series*, прив'язаною до масиву *ISeries[]* типу *PieSeries<double>*. Кожен сектор діаграми відповідає одній категорії витрат. Серії формуються у методі *LoadData()* шляхом групування списку витрат за полем *Category* із використанням LINQ-оператора *GroupBy* та подальшим обчисленням суми витрат для кожної групи. Кожна

отримана пара «категорія - сума» перетворюється на об'єкт *PieSeries<double>* із присвоєнням назви категорії властивості *Name* та масиву з одним елементом властивості *Values*.

Стовпчикова діаграма реалізована через елемент *lvc:CartesianChart* з властивістю *Series*, прив'язаною до масиву *ISeries[]* типу *ColumnSeries<double>*. На відміну від кругової діаграми, стовпчикова відображає суму витрат у розрізі категорій: кожна категорія представлена окремою серією з одним стовпцем. Для коректного відображення осей та підписів у розробленій системі використовується масив *XAxes* типу *Axis[]*, прив'язаний до тієї самої властивості *BarSeries*. Зовнішній вигляд обох діаграм із реальними даними наведено на рис. 2.18.

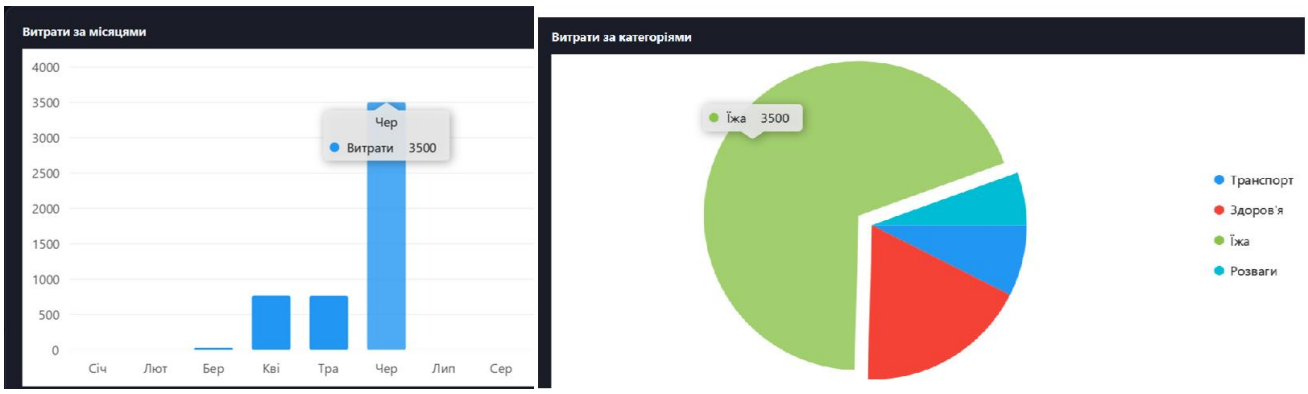


Рисунок 2.18 – Вкладки “Місяці” та “Категорії” з діаграмами розподілу витрат

Важливою особливістю реалізації є те, що масиви серій *Series* та *BarSeries* оголошені як властивості з реалізацією *INotifyPropertyChanged.* При виклику *OnPropertyChanged(nameof(Series))* WPF автоматично сигналізує елементу *PieChart* про необхідність перемалювання, що забезпечує актуальність графіків після кожної операції додавання, редагування або видалення записів без явного звернення до елементів інтерфейсу з коду.

Таким чином, інтерфейс інформаційної системи побудований на принципах чіткого функціонального розподілу між вкладками, мінімізації переходів між екранами та автоматичного оновлення всіх елементів відображення через механізм прив'язки даних WPF. Використання бібліотеки LiveCharts 2 забезпечує інтерактивну та актуальну візуалізацію фінансових даних без необхідності

реалізації власного рушія відображення.

2.5 Реалізація модуля аналізу та формування рекомендацій

Аналітична складова інформаційної системи реалізована як сукупність методів обчислення статистичних показників та механізму їх інтерпретації для формування практичних рекомендацій користувачу. На відміну від простого відображення даних, аналітичний модуль перетворює сирі записи про витрати на змістовну інформацію, яка дозволяє оцінити фінансову поведінку та виявити потенційні проблемні зони в структурі витрат [4]. Загальну схему аналітичного модуля наведено на рис. 2.19.

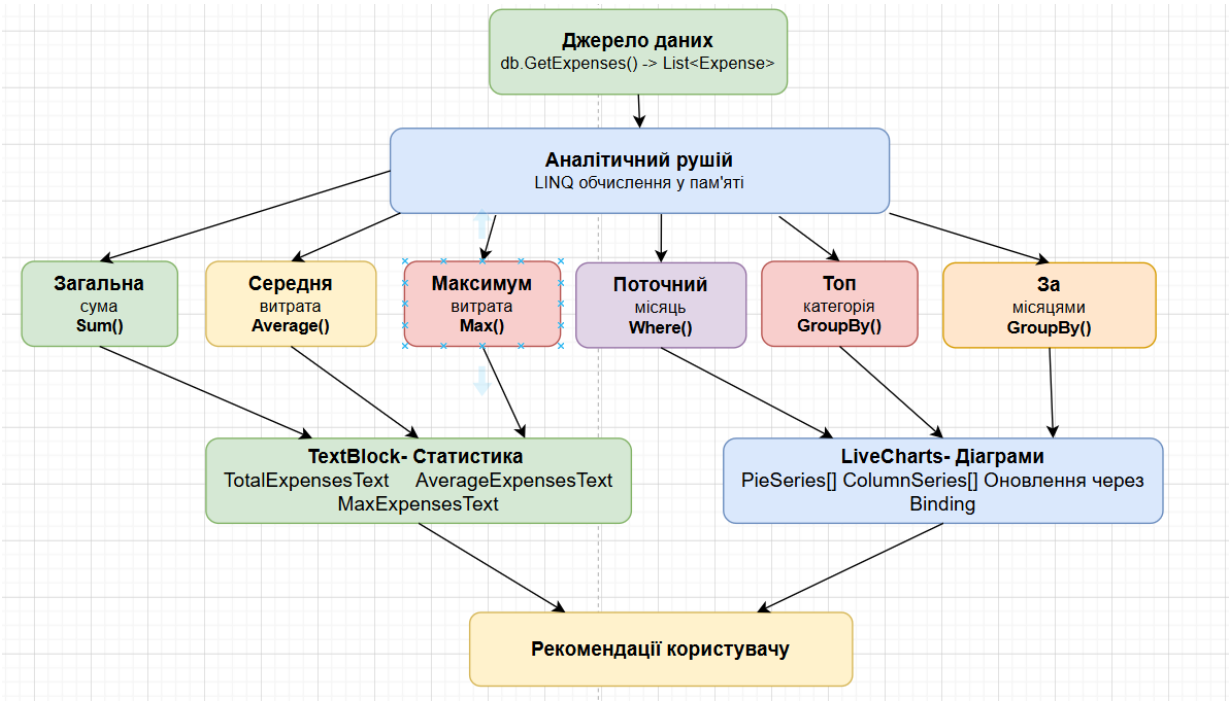


Рисунок 2.19 – Загальна схема аналітичного модуля інформаційної системи

З рис. 2.19 видно, що аналітичний модуль виконує послідовне перетворення даних: від сирих записів бази даних через статистичні обчислення до формування показників та інтерпретаційних висновків.

Розрахунок загальної суми витрат. Загальна сума обчислюється як проста агрегація значень поля *Amount* по всьому завантаженому набору даних засобами

LINQ-оператора *Sum*. Цей показник відображає абсолютний обсяг витрат за весь час використання системи і слугує базовим орієнтиром для оцінки динаміки витрат. Формально обчислення виражається як:

$$S = \sum_{i=1}^{n} a_i \tag{2.1}$$

де S — загальна сума витрат, — сума і-го запису, n - загальна кількість записів. У коді системи це відповідає виразу *data.Sum(x => x.Amount)*, результат якого присвоюється текстовому полю *TotalExpensesText* з форматуванням у вигляді рядка.

Розрахунок середньої витрати. Середнє значення витрати обчислюється за умови ненульової кількості записів через LINQ-оператор *Average*. Для захисту від ділення на нуль у разі порожньої бази даних використовується умовна перевірка *data.Any()* перед обчисленням. Показник середньої витрати є важливим аналітичним індикатором: значне відхилення окремих транзакцій від середнього вказує на нерівномірність витрат або наявність аномально великих одноразових платежів. Математично:

$$\overline{a} = \frac{1}{n}\sum_{i=1}^{n} a_i\,,\ n > 0 \tag{2.2}$$

де $\overline{a}$— середнє значення витрати на одну транзакцію, грн; n — загальна кількість записів у базі даних, $n>0$; a_i — сума і-го запису про витрату, грн; і — порядковий номер запису у вибірці.

Розрахунок максимальної витрати. Максимальне значення серед усіх записів обчислюється через *data.Max(x => x.Amount)* аналогічно із захисною перевіркою *data.Any()*. Найбільша витрата ідентифікується разом із категорією, до якої вона належить, що реалізовано через *data.OrderByDescending(x => x.Amount).First().Category.* Цей показник відображається у полі *TopCategoryText та MaxExpenseText* відповідно.

Розрахунок витрат за поточний місяць. Фільтрація записів за поточним місяцем виконується через LINQ-умову *Where(x => x.Date.Month ==*

DateTime.Now.Month && x.Date.Year == DateTime.Now.Year) із подальшим підсумовуванням відфільтрованих значень. Подвійна умова з перевіркою року є обов'язковою - без неї система некоректно включала б записи за однойменний місяць попередніх років. Результат присвоюється полю *CurrentMonthText.*

Визначення найбільш витратної категорії. Визначення категорії з найбільшою сукупною сумою витрат є найскладнішою аналітичною операцією модуля. Вона реалізована через послідовне застосування LINQ-операторів: групування всіх записів за полем *Category (GroupBy),* обчислення суми витрат для кожної групи (*Sum*), сортування груп за спаданням суми *(OrderByDescending)* та вибір першого елемента *(First)*. Детальну послідовність обчислення цього показника наведено на рис. 2.20.

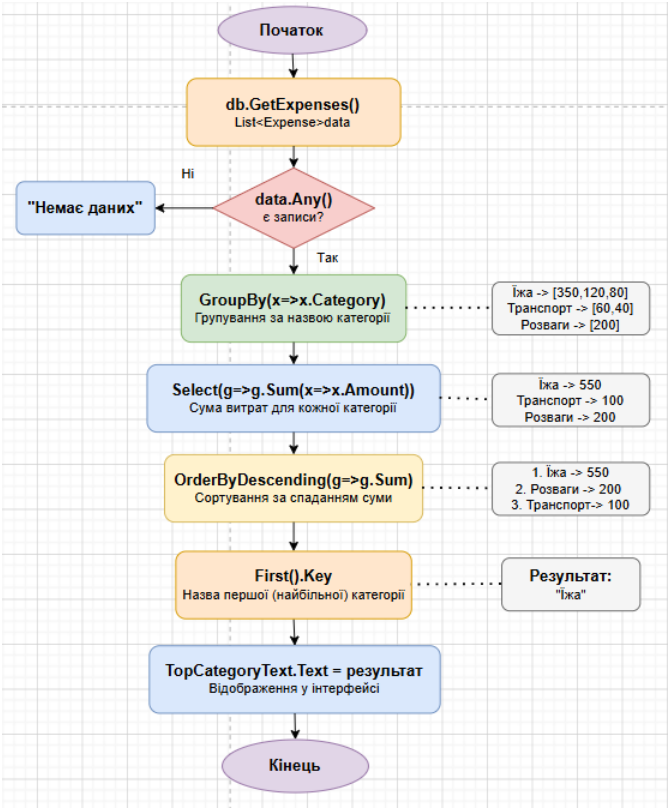


Рисунок 2.20 – Алгоритм визначення найбільш витратної категорії

Алгоритм виконує чотири послідовні перетворення вхідного набору даних і є детерміністичним: за однакових вхідних даних завжди повертає однаковий результат. У разі, якщо база даних не містить жодного запису, обчислення захищено

попередньою перевіркою.

Аналіз витрат за місяцями. Для побудови стовпчикової діаграми виконується групування записів за комбінованим ключем рік-місяць. Отримані групи сортуються у хронологічному порядку та перетворюються на серії даних для бібліотеки LiveCharts. Такий підхід забезпечує коректне відображення хронології навіть за наявності пропущених місяців - у цьому разі стовпці для відсутніх місяців просто не відображаються.

Таблиця 2.5 – Статистичні показники аналітичного модуля та методи їх обчислення

Показник	**Поле інтерфейсу**	**LINQ-вираз**	**Захист від помилок**	**Призначення**
Загальна сума	TotalExpensesText	Data.Sum(x=>x.Amount)	Повертає 0 для порожньої колекції	Абсолютний обсяг витрат
Середня витрата	AverageExpensesText	Data.Average(x=>x.Amount)	Data.Any() перед викликом	Типовий розмір витрат
Максимальна витрата	MaxExpenseText	Data.Max(x=>x.Amount)	Data.Any() перед викликом	Виявлення аномальних транзакцій
Кількість записів	CountExpensesText	Data.Count	-	Обсяг вибірки
Топ-категорія	TopCategoryText	Data.GroupBy(). OrderByDescending() First()	Data.Any() перед викликом	Домінуюча стаття витрат
Поточний місяць	CurrentMonthText	Data.Where(Month && Year) .Sum()	Повертає 0 без записів місяця	Витрати у поточному місяці

Формування рекомендацій. На основі обчислених статистичних показників система здатна формувати практичні рекомендації, які відображаються користувачу у вигляді текстових повідомлень або підсвічених індикаторів. Рекомендації формуються за набором правил, кожне з яких перевіряє певну умову на основі статистики. Алгоритм формування рекомендацій наведено на рис. 2.21.

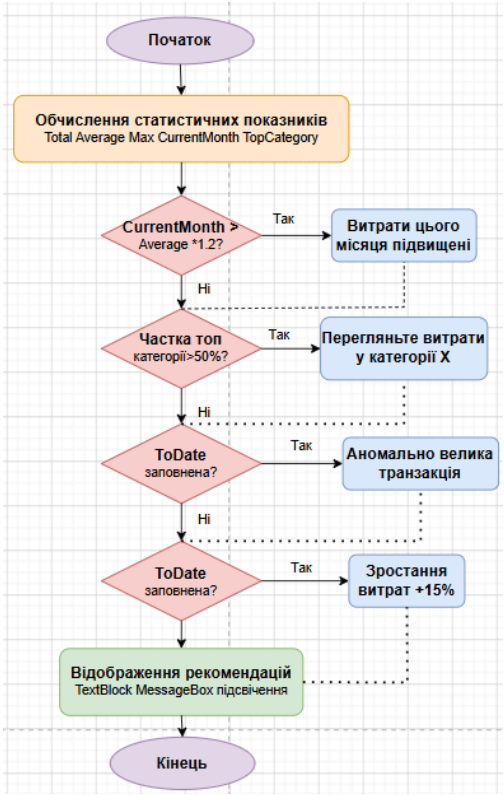


Рисунок 2.21 – Алгоритм формування рекомендацій на основі статистичних показників

Система перевіряє чотири незалежні умови й формує рекомендації відповідно до кожної з них. Перша умова стосується перевищення витрат поточного місяця відносно середньомісячного значення: якщо поточний місяць перевищує середнє більш ніж на 20 відсотків, система повідомляє про підвищений рівень витрат. Друга умова визначає домінуючу категорію: якщо частка однієї категорії перевищує 50 відсотків від загальної суми, система рекомендує переглянути витрати в цій категорії. Третя умова виявляє аномально великі одиничні транзакції: якщо максимальна витрата перевищує середню більш ніж у три рази, система

звертає увагу користувача на цю транзакцію. Четверта умова фіксує зростання витрат: якщо сума поточного місяця перевищує попередній місяць більш ніж на 15 відсотків, система повідомляє про тенденцію до зростання.

Практичний результат роботи аналітичного модуля - відображення статистичних показників у вкладці «Статистика» з реальними обчисленими значеннями — наведено на рис. 2.22.

Рисунок 2.22 – Результати роботи аналітичного модуля у вкладці “Статистика”

Таким чином, аналітичний модуль системи реалізує повний цикл обробки даних: від завантаження сирих записів через статистичні обчислення до формування інтерпретованих показників і практичних рекомендацій. Застосування LINQ забезпечує лаконічну та ефективну реалізацію всіх аналітичних операцій безпосередньо в пам'яті застосунку без додаткових запитів до бази даних.

2.6 Тестування програмного застосунку та аналіз результатів

Тестування є невід'ємною складовою процесу розробки програмного забезпечення, оскільки дозволяє верифікувати відповідність реалізованого функціоналу сформульованим вимогам та виявити дефекти до введення системи в експлуатацію [11]. Для розробленої інформаційної системи аналізу особистих витрат проведено функціональне тестування методом «чорної скриньки» - без аналізу внутрішньої структури коду, виключно на основі перевірки вхідних і

вихідних даних. Такий підхід дозволяє оцінити систему з позиції кінцевого користувача та перевірити коректність реалізації кожної задокументованої функції. Загальний план тестування із розподілом тест-кейсів за функціональними модулями наведено у табл. 2.6.

Таблиця 2.6 – План функціонального тестування інформаційної системи

Модуль	**К-сть тест-кейсів**	**Позитивних**	**Негативних**	**Ідентифікатори**
Додавання витрат	3	1	2	ТК-01, ТК-02, ТК-03
Редагування записів	2	1	1	ТК-04, ТК-05
Видалення записів	2	1	1	ТК-06,ТК-07
Пошук	2	1	1	ТК-08, ТК-09
Фільтрація	3	3	0	ТК-10, ТК-11, ТК-12
Статистика та діаграми	3	3	0	ТК-13, ТК-14, ТК-15
Разом	**15**	**10**	**5**	**-**

Як видно з табл. 2.6, тестування охоплює всі шість функціональних модулів системи та включає як позитивні перевірки коректних сценаріїв, так і негативні перевірки поведінки системи за некоректних вхідних даних. Загалом було розроблено та виконано п'ятнадцять тест-кейсів.

Тестування модуля додавання витрат. Перевірка модуля додавання проводилась за трьома тест-кейсами. Перший позитивний тест (ТК-01) передбачав заповнення всіх п'яти полів форми коректними значеннями та натискання кнопки «Додати витрату». Очікуваний результат - поява нового рядка у *DataGrid* та оновлення статистичних показників. Результат виконання відповідав очікуваному. Другий тест (ТК-02) перевіряв поведінку системи у разі введення нечислового значення у поле суми - символів замість цифр. Система відобразила повідомлення про помилку та не виконала запис до бази даних, що відповідає очікуваній поведінці. Третій тест (ТК-03) перевіряв спробу збереження запису з незаповненим полем категорії. Система також коректно відобразила відповідне попередження.

Детальні результати тестування модуля додавання наведено у табл. 2.7.

Таблиця 2.7 – Результати тестування модуля додавання витрат

ID	Опис тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
ТК-01	Додавання коректного запису	Сума:350, Категорія: “Їжа”, Дата: поточна, Опис: “Продукти”, Оплата: “Картка”	Новий рядок у DataGrid, оновлення статистики	Запис з’явився, статистика оновлена	Пройдено
ТК-02	Введення нечислової суми	Сума: “abc”, інші поля заповнені	MessageBox із повідомленням про помилку, запис не збережено	Відображено попередження, запис не збережено	Пройдено
ТК-03	Відсутність категорії	Сума: 200, Категорія: не обрано, інші поля заповнені	MessageBox“Оберіть категорію”, запис не збережено	Відображено відповідне попередження	Пройдено

Тестування модуля редагування. Перевірка редагування включала два тест-кейси. Позитивний тест (ТК-04) передбачав вибір існуючого рядка у *DataGrid*, зміну значення поля суми та натискання «Зберегти». Перевірялось автоматичне завантаження даних у форму при виборі рядка, коректне збереження змін до бази даних та оновлення відображення у таблиці. Усі три складові пройшли перевірку. Негативний тест (ТК-05) перевіряв натискання «Зберегти» без попереднього вибору рядка в таблиці - система коректно проігнорувала дію завдяки перевірці *selectedExpense != null*. Результати перевірок відображено у табл. 2.8.

Таблиця 2.8 – Результати тестування модуля редагування та видалення

ID	Опис тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
ТК-04	Редагування обраного запису	Вибір рядка - зміна суми з 350 на 500 - “Зберегти”	Дані у БД DataGrid оновлено, відображає нове значення	Сума оновлена у таблиці та БД	Пройдено
ТК-05	Збереження без вибору запису	Натискання “Зберегти” без вибору рядка	Жодних дій, метод завершується через Null-guard	Жодних змін не відбулося	Пройдено
ТК-06	Видалення з підтвердженням	Вибір рядка - “Видалити” - “Так” у діалозі	Запис видалено з БД та DataGrid	Рядок зник із таблиці	Пройдено
ТК-07	Відмова від видалення	Вибір рядка - “Видалити” - “Ні” у діалозі	Запис залишається незмінним	Рядок присутній у таблиці	Пройдено

Тестування модуля видалення. Позитивний тест (ТК-06) перевіряв повний сценарій видалення: вибір рядка, натискання «Видалити», підтвердження у діалозі та перевірка відсутності запису у *DataGrid* після операції. Результат відповідав очікуваному. Негативний тест (ТК-07) перевіряв відмову від видалення у діалозі підтвердження - натискання кнопки «Ні». Запис залишився у таблиці незмінним, що підтверджує коректність реалізації захисту від випадкового видалення.

Тестування модулів пошуку та фільтрації. Тестування пошуку (ТК-08, ТК-09) включало перевірку пошуку за частковим збігом рядка у полях опису та категорії, а також пошуку за рядком, що не відповідає жодному запису - у другому випадку *DataGrid* відображав порожню таблицю без помилок. Тестування фільтрації за датою (ТК-10, ТК-11) перевіряло коректність застосування одностороннього фільтра (лише «Від» або лише «До») та двостороннього діапазону одночасно. В усіх чотирьох тест-кейсах результати відповідали очікуваним.

Тестування фільтрації за категорією (ТК-12) підтвердило коректне відображення лише записів обраної категорії та повернення до повного списку при виборі значення «Усі». Скріншот інтерфейсу під час виконання тесту пошуку наведено на рис. 2.23.

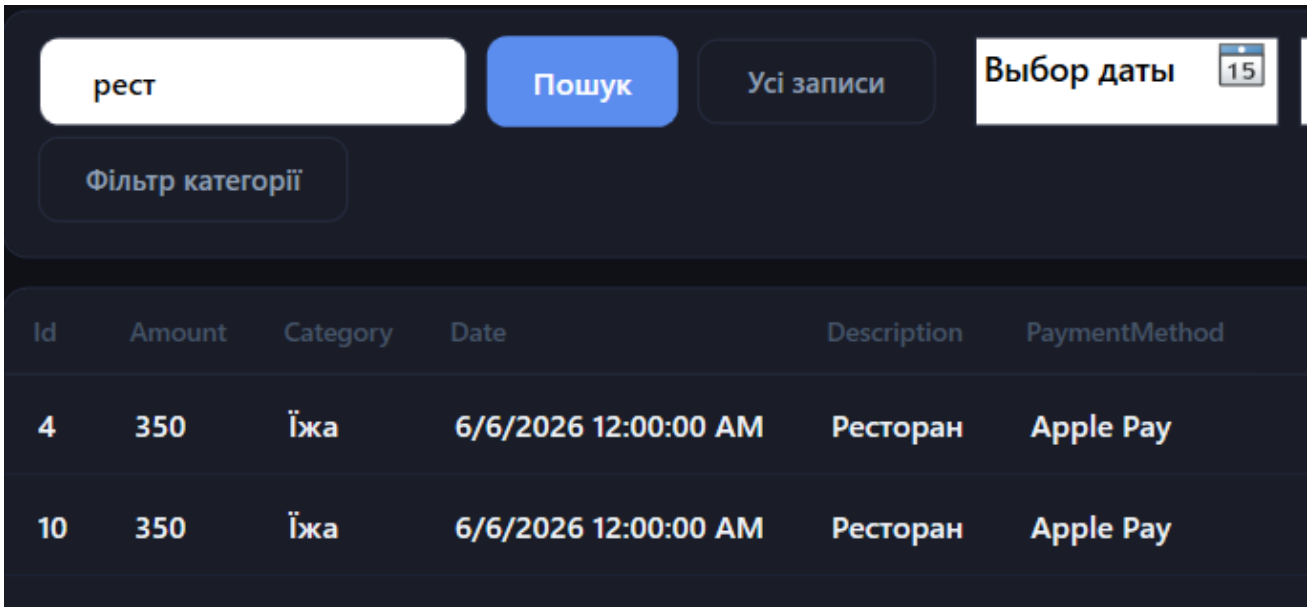


Рисунок 2.23 – Результат виконання тест-кейсу ТК-08: пошук за частковим збігом

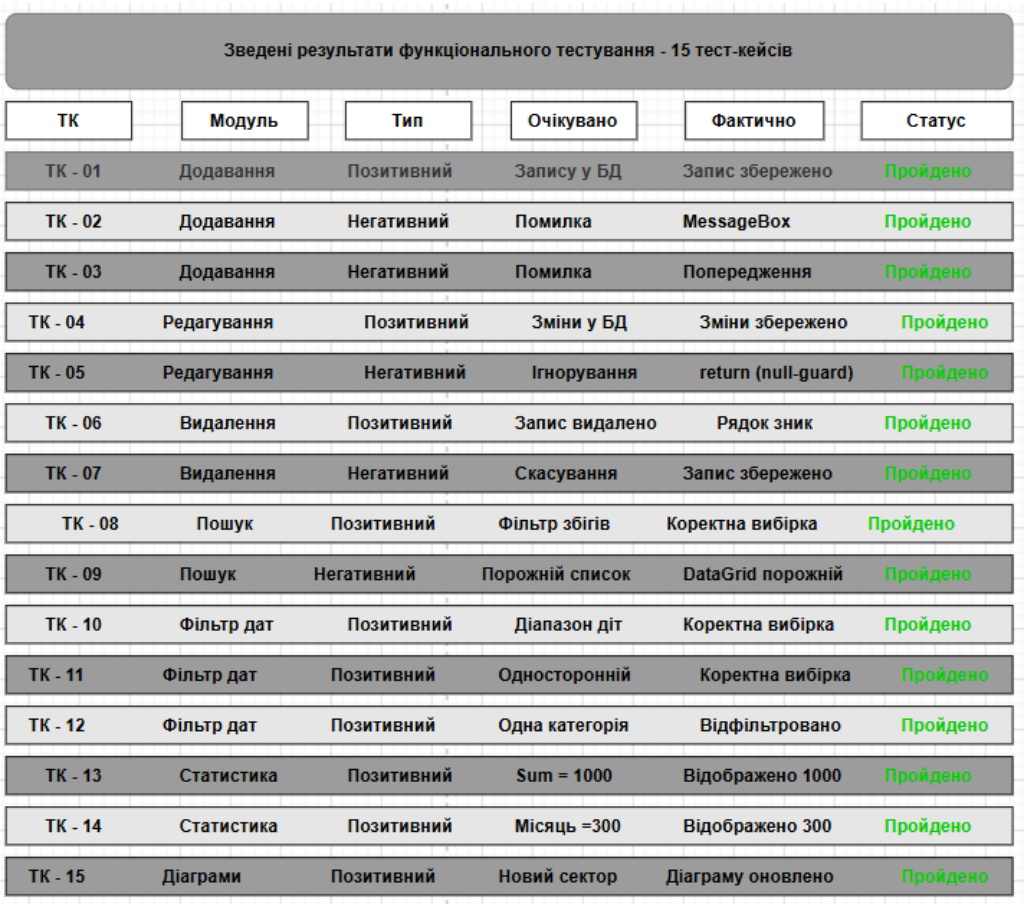
Тестування модуля статистики. Перевірка точності обчислень (ТК-13) виконувалась на контрольному наборі даних із відомими значеннями: до бази були внесені рівно десять записів по 100 гривень кожен. Після завантаження система відобразила загальну суму 1000 грн, середню витрату 100 грн, максимум 100 грн та кількість записів 10, що повністю відповідало очікуваним значенням. Точність обчислень підтверджено без похибок округлення. Тест ТК-14 перевіряв коректність показника «Поточний місяць» - серед внесених записів лише три були датовані поточним місяцем і роком, тому очікувана сума становила 300 грн. Система відобразила значення 300 грн, що відповідало очікуваному. Результати тестування модуля статистики наведено у табл. 2.9.

Таблиця 2.9 – Результати тестування модуля статистики та точності обчислень

ID	Опис тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
ТК-13	Точність загальної суми	10 записів по 100	Sum =100, Average = 100, Count =10	Sum = 1000, Average = 100, Count =10	Пройдено
ТК-14	Точність суми за місяць	10 записів, 3 з них поточного місяця по 100	CurrentMonth = 300	Відображено 300	Пройдено
ТК-15	Оновлення діаграм	Додано запис нової категорії “Освіта”	Новий сектор у PieChart, оновлений	Обидві діаграми оновлено коректно	Пройдено

Тестування модуля візуалізації. Тест ТК-15 перевіряв коректність оновлення діаграм після додавання нового запису. До бази була внесена витрата нової категорії «Освіта», якої раніше не існувало. Після виконання *LoadData()* кругова діаграма відобразила новий сектор із відповідною назвою та відсотковою часткою. Стовпчикова діаграма оновила висоту відповідного стовпця. Прив'язка через *OnPropertyChanged* спрацювала коректно в обох випадках.

Аналіз результатів тестування. Зведені результати всіх п'ятнадцяти тест-кейсів наведено на рис. 2.24 у вигляді діаграми успішності тестів.



Зведені результати функціонального тестування - 15 тест-кейсів

ТК	Модуль	Тип	Очікувано	Фактично	Статус
ТК - 01	Додавання	Позитивний	Запису у БД	Запис збережено	Пройдено
ТК - 02	Додавання	Негативний	Помилка	MessageBox	Пройдено
ТК - 03	Додавання	Негативний	Помилка	Попередження	Пройдено
ТК - 04	Редагування	Позитивний	Зміни у БД	Зміни збережено	Пройдено
ТК - 05	Редагування	Негативний	Ігнорування	return (null-guard)	Пройдено
ТК - 06	Видалення	Позитивний	Запис видалено	Рядок зник	Пройдено
ТК - 07	Видалення	Негативний	Скасування	Запис збережено	Пройдено
ТК - 08	Пошук	Позитивний	Фільтр збігів	Коректна вибірка	Пройдено
ТК - 09	Пошук	Негативний	Порожній список	DataGrid порожній	Пройдено
ТК - 10	Фільтр дат	Позитивний	Діапазон діт	Коректна вибірка	Пройдено
ТК - 11	Фільтр дат	Позитивний	Односторонній	Коректна вибірка	Пройдено
ТК - 12	Фільтр дат	Позитивний	Одна категорія	Відфільтровано	Пройдено
ТК - 13	Статистика	Позитивний	Sum = 1000	Відображено 1000	Пройдено
ТК - 14	Статистика	Позитивний	Місяць =300	Відображено 300	Пройдено
ТК - 15	Діаграми	Позитивний	Новий сектор	Діаграму оновлено	Пройдено

Рисунок 2.24 – Зведені результати функціонального тестування інформаційної системи

Всі п'ятнадцять виконаних тест-кейсів отримали статус «Пройдено». Жодного критичного дефекту виявлено не було. У процесі тестування було виявлено та усунено один незначний недолік: у початковій реалізації логіка видалення містила інвертовану умову перевірки результату діалогу - *if (result == MessageBoxResult.Yes) return замість if (result != MessageBoxResult.Yes) return*, що призводило до видалення запису при натисканні «Ні» та збереження при «Так». Після виявлення помилку було виправлено, і повторний тест підтвердив коректну поведінку.

Таким чином, проведене функціональне тестування підтверджує працездатність усіх реалізованих модулів інформаційної системи. Всі перевірені позитивні сценарії відповідають очікуваній поведінці, а негативні перевірки

підтверджують наявність та коректність механізмів захисту від некоректних вхідних даних. Виявлений у процесі тестування дефект усунено, що додатково підтверджує цінність систематичного підходу до перевірки програмного забезпечення.

Висновки до розділу

У даному розділі було проведено повний цикл проєктування та реалізації інформаційної системи аналізу особистих витрат користувача. Архітектуру системи побудовано на основі трирівневої моделі з чітким розмежуванням рівнів представлення, бізнес-логіки та даних. Реалізовано чотири основні програмні компоненти. Для локального зберігання даних обрано СКБД SQLite. Спроєктовано реляційну схему бази даних, що складається з однієї таблиці *Expenses* із шістьма полями, достатніми для повного опису фінансової транзакції. Усі операції з базою даних виконуються через параметризовані SQL-запити, що виключає можливість SQL-ін'єкцій. Реалізовано повний набір CRUD-операцій. Розроблено шість функціональних модулів системи. Кожен модуль реалізовано у вигляді окремих методів-обробників із єдиною відповідальністю. Інтерфейс користувача реалізовано засобами WPF із використанням декларативної XAML-розмітки. Застосунок організовано у вигляді п'ятивкладкової структури *TabControl*, де кожна вкладка відповідає окремій функціональній зоні. Для побудови діаграм інтегровано бібліотеку *LiveChartsCore.SkiaSharpView.WPF,* яка забезпечує відображення кругової та стовпчикової діаграм з автоматичним оновленням через механізм *INotifyPropertyChanged* та прив'язку *Binding*. Аналітичний модуль реалізує обчислення шести статистичних показників із єдиного завантаженого набору даних. Проведено функціональне тестування методом «чорної скриньки» за п'ятнадцятьма тест-кейсами, що охоплюють усі реалізовані модулі та включають як позитивні, так і негативні перевірки. Усі тест-кейси отримали статус «Пройдено».

ВИСНОВКИ

У ході дослідження було проведено аналіз предметної області, виявлено стійку потребу у спеціалізованих інструментах особистого фінансового обліку. Встановлено, що переважна більшість наявних рішень є хмарними або мобільними, що пов'язане з постійним інтернет-з'єднанням та залежністю від стороннього серверного програмного забезпечення. Виявлено відсутність простих автономних десктопних застосунків для ведення персонального фінансового щоденника, що визначило актуальність і практичну цінність розробки.

У результаті аналізу технологічного стеку обрано оптимальне поєднання інструментів: WPF як платформу для побудови настільних застосунків із потужним механізмом прив'язки даних; SQLite як безсерверну реляційну СКБД з підтримкою ACID-транзакцій; LiveCharts 2 як сучасну бібліотеку інтерактивної візуалізації, нативно інтегровану з WPF через механізм Binding. Обґрунтовано переваги обраних технологій над альтернативами за критеріями автономності, продуктивності та зручності інтеграції.

Спроєктовано трирівневу архітектуру системи з чотирма основними програмними компонентами та однонаправленим потоком залежностей між ними. Розроблено структуру бази даних у вигляді єдиної таблиці Expenses із шістьма полями, достатніми для повного опису фінансової витрати. Усі звернення до бази даних реалізовано через параметризовані запити, що забезпечує захист від SQL-ін'єкцій.

Реалізовано тринадцять функціональних можливостей системи, що охоплюють повний цикл управління даними: додавання, редагування та видалення записів з обов'язковою валідацією та підтвердженням; пошук за частковим збігом у текстових полях; фільтрацію за категорією та діапазоном дат; обчислення шести статистичних показників; побудову кругової діаграми розподілу витрат за категоріями та стовпчикової діаграми динаміки витрат за місяцями.

Розроблено інтерфейс користувача у вигляді п'ятивкладкової структури з відокремленими функціональними зонами для введення витрат, перегляду

статистики, управління таблицею записів та аналізу діаграм. Застосовано систему стилізації XAML для забезпечення єдиного візуального вигляду всіх елементів керування. Проведено функціональне тестування методом «чорної скриньки» за п'ятнадцятьма тест-кейсами, що охоплюють усі реалізовані модулі системи та включають як позитивні сценарії коректної роботи, так і негативні перевірки поведінки системи за некоректних вхідних даних. Усі тест-кейси успішно пройдено. У процесі тестування виявлено та усунено один логічний дефект у модулі видалення записів.

Розроблена інформаційна система повністю відповідає сформульованим функціональним вимогам, забезпечує коректне збереження та обробку даних і може бути використана як самостійний продукт для ведення персонального фінансового обліку.

Практична цінність роботи полягає у створенні готового до використання автономного застосунку, що не потребує встановлення додаткового серверного програмного забезпечення, зберігає всі дані локально та забезпечує повну конфіденційність фінансової інформації користувача. Перспективами подальшого розвитку системи є реалізація автентифікації користувача та підтримки кількох профілів, додавання функції експорту даних у формати Excel та PDF, розробка мобільної версії на платформі .NET MAUI, інтеграція хмарного резервного копіювання бази даних та впровадження модуля прогнозування витрат на основі методів машинного навчання.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Xiao J. J. Personal Finance and Consumer Behavior. New York : Springer, 2016. 312 p.

2. Oliveira T., Thomas M., Baptista G. Mobile payment: Understanding the determinants of customer adoption and intention to recommend the technology. Computers in Human Behavior. 2016. Vol. 61. P. 404–414.

3. Personal Finance Management Applications and Services [Electronic resource].URL:https://www.researchgate.net/publication/383469539_Personal_Finance_Management_Application (date of access: 19.05.2026).

4. Financial Management. EDUCBA. URL: https://www.educba.com/financial-management/ (date of access: 18.05.2026).

5. Lusardi A. Financial Literacy and Financial Decision-Making in Older Adults. Generations. 2012. Vol. 36, No. 2. P. 25–32.

6. Gholami A., Laure E. Security and Privacy of Sensitive Data in Cloud Computing: A Survey of Recent Developments. arXiv preprint. 2016. arXiv:1601.01498.

7. About SQLite. SQLite Home Page. URL: https://www.sqlite.org/about.html (date of access: 19.05.2026).

8. Research of E-map System Based on Baidu Map API. ResearchGate. URL: https://www.researchgate.net/figure/The-system-architecture-As-is-shown-in-Figure-1-MySQL-is-the-local-database-The_fig1_340490518 (date of access: 19.05.2026).

9. Few S. Information Dashboard Design: Displaying Data for At-a-Glance Monitoring. Burlingame : Analytics Press, 2013. 223 p.

10. Erl T. Cloud Computing: Concepts, Technology & Architecture. Upper Saddle River : Prentice Hall, 2013. 528 p.

11. The Global Cyber Threat to Financial Systems. International Monetary Fund. URL: https://www.imf.org/external/pubs/ft/fandd/2021/03/global-cyber-threat-to-financial-systems-maurer.htm (date of access: 17.05.2026).

12. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 611 p.

13. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.

14. Kapoor J. R., Dlabay L. R., Hughes R. J. Personal Finance. 13th ed. New York : McGraw-Hill Education, 2018. 544 p.

15. Roman R., Zhou J., Lopez J. Securing the Internet of Things. Computer. 2013. Vol. 44, No. 9. P. 51–58.

16. SQLite Documentation. SQLite Home Page. URL: https://www.sqlite.org/docs.html (date of access: 20.05.2026).

17. Tidwell J. Designing Interfaces. 3rd ed. Sebastopol : O'Reilly Media, 2020. 600 p.

18. Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 122 "Комп'ютерні науки". Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.

19. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ : ДП «УкрНДНЦ», 2015. 26 с. (Інформація та документація).

20. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання. Київ : ДП «УкрНДНЦ», 2016. 16 с. (Інформація та документація).

21. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ : ДП «УкрНДНЦ», 2013. 23 с. (Інформація та документація).

ДОДАТОК А – Лістинги програмних класів

Лістинг А.1 - Клас моделі даних Expense

```
namespace PersonalFinanceManager.Models;
public class Expense
{
    public int Id { get; set; }
    public double Amount { get; set; }
    public string Category { get; set; }
    public DateTime Date { get; set; }
    public string Description { get; set; }
    public string PaymentMethod { get; set; }
}
```

Лістинг А.2 - Клас репозиторію DatabaseHelper

```
using Microsoft.Data.Sqlite;
using PersonalFinanceManager.Models;
using System.Collections.Generic;
namespace PersonalFinanceManager.Database;
public class DatabaseHelper
{
    private const string ConnectionString = "Data Source=finance.db";
    public DatabaseHelper()
    {
        CreateDatabase();
    }
    private void CreateDatabase()
    {
        using var connection = new SqliteConnection(ConnectionString);
        connection.Open();
        var command = connection.CreateCommand();
        command.CommandText =
        @"
            CREATE TABLE IF NOT EXISTS Expenses (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                Amount REAL,
```

```
            Category TEXT,
            Date TEXT,
            Description TEXT,
            PaymentMethod TEXT
        );
        ";
    command.ExecuteNonQuery();
}
public void AddExpense(Expense expense)
{
    using var connection = new SqliteConnection(ConnectionString);
    connection.Open();
    var command = connection.CreateCommand();
    command.CommandText =
    @"
        INSERT INTO Expenses (Amount, Category, Date, Description, PaymentMethod)
        VALUES ($amount, $category, $date, $description, $payment);
        ";
    command.Parameters.AddWithValue("$amount", expense.Amount);
    command.Parameters.AddWithValue("$category", expense.Category);
    command.Parameters.AddWithValue("$date", expense.Date);
    command.Parameters.AddWithValue("$description", expense.Description);
    command.Parameters.AddWithValue("$payment", expense.PaymentMethod);
    command.ExecuteNonQuery();
}
public List<Expense> GetExpenses()
{
    var list = new List<Expense>();
    using var connection = new SqliteConnection(ConnectionString);
    connection.Open();
    var command = connection.CreateCommand();
    command.CommandText = "SELECT * FROM Expenses";
    using var reader = command.ExecuteReader();
    while (reader.Read())
    {
        list.Add(new Expense
        {
```

```
            Id = reader.GetInt32(0),
            Amount = reader.GetDouble(1),
            Category = reader.GetString(2),
            Date = DateTime.Parse(reader.GetString(3)),
            Description = reader.GetString(4),
            PaymentMethod = reader.GetString(5)
        });
    }
    return list;
}
public void UpdateExpense(Expense expense)
{
    using var connection = new SqliteConnection(ConnectionString);
    connection.Open();
    var command = connection.CreateCommand();
    command.CommandText =
    @"
    UPDATE Expenses
    SET Amount = $amount,
        Category = $category,
        Date = $date,
        Description = $description,
        PaymentMethod = $payment
    WHERE Id = $id;
";
    command.Parameters.AddWithValue("$id", expense.Id);
    command.Parameters.AddWithValue("$amount", expense.Amount);
    command.Parameters.AddWithValue("$category", expense.Category);
    command.Parameters.AddWithValue("$date", expense.Date);
    command.Parameters.AddWithValue("$description", expense.Description);
    command.Parameters.AddWithValue("$payment", expense.PaymentMethod);
    command.ExecuteNonQuery();
}
public void DeleteExpense(int id)
{
    using var connection = new SqliteConnection(ConnectionString);
    connection.Open();
```

```
        var command = connection.CreateCommand();
        command.CommandText =
        @"
        DELETE FROM Expenses
        WHERE Id = $id;
    ";
        command.Parameters.AddWithValue("$id", id);
        command.ExecuteNonQuery();
    }
}
```

Лістинг А.3 - Клас сервісного рівня ExpenseService

```
using System;
using System.Collections.Generic;
using Microsoft.Data.Sqlite;
using PersonalFinanceManager.Models;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace PersonalFinanceManager.Services
{
    internal class ExpenseService
    {
        private const string ConnectionString =
        "Data Source=finance.db";
        public void AddExpense(Expense expense)
        {
            using var connection =
                new SqliteConnection(ConnectionString);
            connection.Open();
            var command = connection.CreateCommand();
            command.CommandText =
            @"
        INSERT INTO Expenses
        (
            Amount,
            Category,
```

```
            Date,
            Description,
            PaymentMethod
        )
        VALUES
        (
            $amount,
            $category,
            $date,
            $description,
            $paymentMethod
        );
        ";
            command.Parameters.AddWithValue(
                "$amount",
                expense.Amount);

            command.Parameters.AddWithValue(
                "$category",
                expense.Category);
            command.Parameters.AddWithValue(
                "$date",
                expense.Date);
            command.Parameters.AddWithValue(
                "$description",
                expense.Description);
            command.Parameters.AddWithValue(
                "$paymentMethod",
                expense.PaymentMethod);
            command.ExecuteNonQuery();
        }
    }
}
```

ДОДАТОК Б – XAML-розмітка та код головного вікна

Лістинг Б.1 - XAML-розмітка головного вікна

```
<Window x:Class="PersonalFinanceManager.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:lvc="clr-
namespace:LiveChartsCore.SkiaSharpView.WPF;assembly=LiveChartsCore.SkiaSharpView.WPF"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        Title="Personal Finance Manager"
        Height="700" Width="1060"
        WindowStartupLocation="CenterScreen"
        Background="#0F1117"
        WindowStyle="SingleBorderWindow">
    <Window.Resources>
        <!--  COLOURS  -->
        <SolidColorBrush x:Key="BgPage"       Color="#0F1117"/>
        <SolidColorBrush x:Key="BgCard"       Color="#1A1D27"/>
        <SolidColorBrush x:Key="BgInput"      Color="#12151F"/>
        <SolidColorBrush x:Key="BgTab"        Color="#1A1D27"/>
        <SolidColorBrush x:Key="BgTabSel"     Color="#0F1117"/>
        <SolidColorBrush x:Key="AccentBlue"   Color="#5B8DEF"/>
        <SolidColorBrush x:Key="AccentTeal"   Color="#2DD4BF"/>
        <SolidColorBrush x:Key="AccentAmber"  Color="#F59E0B"/>
        <SolidColorBrush x:Key="AccentRed"    Color="#EF4444"/>
        <SolidColorBrush x:Key="TxtPrimary"   Color="#F1F5F9"/>
        <SolidColorBrush x:Key="TxtSecondary" Color="#94A3B8"/>
        <SolidColorBrush x:Key="TxtMuted"     Color="#475569"/>
        <SolidColorBrush x:Key="Border"       Color="#252A3D"/>
        <!-- LABEL -->
        <Style x:Key="Lbl" TargetType="TextBlock">
            <Setter Property="Foreground" Value="#94A3B8"/>
            <Setter Property="FontSize"   Value="11"/>
```

```
    <Setter Property="FontWeight" Value="SemiBold"/>
    <Setter Property="Margin"     Value="0,0,0,5"/>
</Style>
<!--  TEXTBOX  -->
<Style x:Key="Inp" TargetType="TextBox">
    <Setter Property="Background"              Value="White"/>
    <Setter Property="Foreground"              Value="Black"/>
    <Setter Property="CaretBrush"              Value="#5B8DEF"/>
    <Setter Property="BorderBrush"             Value="#252A3D"/>
    <Setter Property="BorderThickness"         Value="1"/>
    <Setter Property="FontSize"                Value="13"/>
    <Setter Property="Height"                  Value="38"/>
    <Setter Property="Padding"                 Value="10,0"/>
    <Setter Property="VerticalContentAlignment" Value="Center"/>
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate TargetType="TextBox">
          <Border x:Name="bd" Background="{TemplateBinding Background}"
                  BorderBrush="{TemplateBinding BorderBrush}"
                  BorderThickness="{TemplateBinding BorderThickness}"
                  CornerRadius="8">
            <ScrollViewer x:Name="PART_ContentHost"
                          Margin="{TemplateBinding Padding}"
                          VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>
          </Border>
          <ControlTemplate.Triggers>
            <Trigger Property="IsFocused" Value="True">
              <Setter TargetName="bd" Property="BorderBrush" Value="#5B8DEF"/>
            </Trigger>
          </ControlTemplate.Triggers>
        </ControlTemplate>
      </Setter.Value>
    </Setter>
</Style>
<!--  COMBOBOX  -->
```

```
<Style x:Key="Cmb" TargetType="ComboBox">
    <Setter Property="Background"      Value="#12151F"/>
    <Setter Property="Foreground"      Value="Black"/>
    <Setter Property="BorderBrush"     Value="#252A3D"/>
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="FontSize"        Value="15"/>
    <Setter Property="Height"          Value="35"/>
    <Setter Property="Padding"         Value="10,0"/>
</Style>
<!--  DATE   -->
<Style x:Key="Dp" TargetType="DatePicker">
    <Setter Property="Background"      Value="White"/>
    <Setter Property="Foreground"      Value="Black"/>
    <Setter Property="BorderBrush"     Value="#252A3D"/>
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="FontSize"        Value="14"/>
    <Setter Property="Height"          Value="38"/>
</Style>
<!--  PRIMARY КНОПКА   -->
<Style x:Key="BtnPrimary" TargetType="Button">
    <Setter Property="Background"      Value="#5B8DEF"/>
    <Setter Property="Foreground"      Value="White"/>
    <Setter Property="BorderThickness" Value="0"/>
    <Setter Property="FontSize"        Value="13"/>
    <Setter Property="FontWeight"      Value="SemiBold"/>
    <Setter Property="Height"          Value="38"/>
    <Setter Property="Padding"         Value="18,0"/>
    <Setter Property="Cursor"          Value="Hand"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Border x:Name="bd" Background="{TemplateBinding Background}"
                        CornerRadius="8" Padding="{TemplateBinding Padding}">
                    <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
                </Border>
```

```
            <ControlTemplate.Triggers>
              <Trigger Property="IsMouseOver" Value="True">
                <Setter TargetName="bd" Property="Background" Value="#4A7ADE"/>
              </Trigger>
              <Trigger Property="IsPressed" Value="True">
                <Setter TargetName="bd" Property="Background" Value="#3A68C8"/>
              </Trigger>
            </ControlTemplate.Triggers>
          </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
<!-- ПРОЗРАЧ. КНОПКА -->
<Style x:Key="BtnGhost" TargetType="Button">
    <Setter Property="Background"      Value="Transparent"/>
    <Setter Property="Foreground"      Value="#94A3B8"/>
    <Setter Property="BorderBrush"     Value="#252A3D"/>
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="FontSize"        Value="12"/>
    <Setter Property="Height"          Value="36"/>
    <Setter Property="Padding"         Value="14,0"/>
    <Setter Property="Cursor"          Value="Hand"/>
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate TargetType="Button">
          <Border x:Name="bd" Background="{TemplateBinding Background}"
                  BorderBrush="{TemplateBinding BorderBrush}"
                  BorderThickness="{TemplateBinding BorderThickness}"
                  CornerRadius="8" Padding="{TemplateBinding Padding}">
            <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
          </Border>
          <ControlTemplate.Triggers>
            <Trigger Property="IsMouseOver" Value="True">
              <Setter TargetName="bd" Property="Background" Value="#1E2235"/>
              <Setter Property="Foreground" Value="#F1F5F9"/>
```

```
                </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
<!-- УДАЛЕНИЕ КНОРКА -->
<Style x:Key="BtnDanger" TargetType="Button" BasedOn="{StaticResource BtnPrimary}">
  <Setter Property="Background" Value="#EF4444"/>
  <Setter Property="FontSize"   Value="12"/>
  <Setter Property="Height"     Value="36"/>
</Style>
<!-- KPI КАРТ -->
<Style x:Key="KpiCard" TargetType="Border">
  <Setter Property="Background"       Value="#1A1D27"/>
  <Setter Property="CornerRadius"     Value="12"/>
  <Setter Property="Padding"          Value="18,14"/>
  <Setter Property="BorderBrush"      Value="#252A3D"/>
  <Setter Property="BorderThickness"  Value="1"/>
</Style>
<!-- DATAGRID -->
<Style x:Key="Dg" TargetType="DataGrid">
  <Setter Property="Background"             Value="Transparent"/>
  <Setter Property="Foreground"             Value="#F1F5F9"/>
  <Setter Property="BorderThickness"        Value="0"/>
  <Setter Property="GridLinesVisibility"    Value="None"/>
  <Setter Property="RowBackground"          Value="Transparent"/>
  <Setter Property="AlternatingRowBackground" Value="#161A28"/>
  <Setter Property="HeadersVisibility"      Value="Column"/>
  <Setter Property="CanUserAddRows"         Value="False"/>
  <Setter Property="CanUserResizeRows"      Value="False"/>
  <Setter Property="AutoGenerateColumns"    Value="True"/>
  <Setter Property="IsReadOnly"             Value="True"/>
  <Setter Property="FontSize"               Value="13"/>
  <Setter Property="ColumnHeaderStyle">
```

```
    <Setter.Value>
      <Style TargetType="DataGridColumnHeader">
        <Setter Property="Background"       Value="Transparent"/>
        <Setter Property="Foreground"       Value="#475569"/>
        <Setter Property="FontSize"         Value="11"/>
        <Setter Property="FontWeight"       Value="SemiBold"/>
        <Setter Property="Height"           Value="36"/>
        <Setter Property="Padding"          Value="12,0"/>
        <Setter Property="BorderThickness"  Value="0,0,0,1"/>
        <Setter Property="BorderBrush"      Value="#252A3D"/>
      </Style>
    </Setter.Value>
  </Setter>
  <Setter Property="RowStyle">
    <Setter.Value>
      <Style TargetType="DataGridRow">
        <Setter Property="Height"          Value="42"/>
        <Setter Property="Background"      Value="Transparent"/>
        <Setter Property="BorderThickness" Value="0,0,0,1"/>
        <Setter Property="BorderBrush"     Value="#1E2235"/>
        <Style.Triggers>
          <Trigger Property="IsMouseOver" Value="True">
            <Setter Property="Background" Value="#1E2235"/>
          </Trigger>
          <Trigger Property="IsSelected" Value="True">
            <Setter Property="Background" Value="#1A2D52"/>
          </Trigger>
        </Style.Triggers>
      </Style>
    </Setter.Value>
  </Setter>
  <Setter Property="CellStyle">
    <Setter.Value>
      <Style TargetType="DataGridCell">
        <Setter Property="BorderThickness" Value="0"/>
```

```
            <Setter Property="Padding"         Value="12,0"/>
            <Setter Property="Background"      Value="Transparent"/>
            <Setter Property="Template">
              <Setter.Value>
                <ControlTemplate TargetType="DataGridCell">
                  <Border Background="{TemplateBinding Background}"
                          Padding="{TemplateBinding Padding}">
                    <ContentPresenter VerticalAlignment="Center"/>
                  </Border>
                </ControlTemplate>
              </Setter.Value>
            </Setter>
          </Style>
        </Setter.Value>
    </Setter>
</Style>
<!--  TAB CONTROL  -->
<Style TargetType="TabControl">
    <Setter Property="Background"      Value="#0F1117"/>
    <Setter Property="BorderThickness" Value="0"/>
    <Setter Property="Padding"         Value="0"/>
</Style>
<Style TargetType="TabItem">
    <Setter Property="Foreground"      Value="#94A3B8"/>
    <Setter Property="FontSize"        Value="13"/>
    <Setter Property="FontWeight"      Value="SemiBold"/>
    <Setter Property="Height"          Value="44"/>
    <Setter Property="Padding"         Value="20,0"/>
    <Setter Property="Background"      Value="Transparent"/>
    <Setter Property="BorderThickness" Value="0"/>
    <Setter Property="Cursor"          Value="Hand"/>
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate TargetType="TabItem">
          <Border x:Name="bd" Padding="{TemplateBinding Padding}"
```

```
                Background="{TemplateBinding Background}"
                BorderThickness="0,0,0,2"
                BorderBrush="Transparent">
              <ContentPresenter x:Name="cp"
                        ContentSource="Header"
                        VerticalAlignment="Center"/>
            </Border>
            <ControlTemplate.Triggers>
              <Trigger Property="IsSelected" Value="True">
                <Setter TargetName="bd" Property="BorderBrush" Value="#5B8DEF"/>
                <Setter Property="Foreground" Value="#F1F5F9"/>
              </Trigger>
              <Trigger Property="IsMouseOver" Value="True">
                <Setter Property="Foreground" Value="#CBD5E1"/>
              </Trigger>
            </ControlTemplate.Triggers>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </Window.Resources>
  <Grid Background="#0F1117">
    <Grid.RowDefinitions>
      <!-- header bar  (BEPX)-->
      <RowDefinition Height="56"/>
      <!-- Tab strip -->
      <RowDefinition Height="Auto"/>
      <!-- Заголов -->
      <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <!-- ═══ НАЗВА ═ -->
    <Border Grid.Row="0" Background="#141726"
        BorderBrush="#252A3D" BorderThickness="0,0,0,1">
      <Grid Margin="24,0">
        <Grid.ColumnDefinitions>
```

```
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
          </Grid.ColumnDefinitions>
          <!-- Logo -->
          <StackPanel Grid.Column="0" Orientation="Horizontal"
                      VerticalAlignment="Center">
            <StackPanel VerticalAlignment="Center">
              <TextBlock Text="FinTrack"
                         Foreground="#F1F5F9"
                         FontSize="15" FontWeight="Bold"/>
              <TextBlock Text="Фінансовий щоденник"
                         Foreground="#475569"
                         FontSize="10"/>
            </StackPanel>
          </StackPanel>
        </Grid>
      </Border>
      <!--  ── TAB STRIP ──────────────────────────────────────────────────────── -->
      <Border Grid.Row="1" Background="#141726"
              BorderBrush="#252A3D" BorderThickness="0,0,0,1">
      </Border>
  <!--  ── TAB ОСНОВА ──────────────────────────────────────────────────────── -->
      <TabControl Grid.Row="0" Grid.RowSpan="3" Margin="0,56,0,0">
        <!-- TAB 1 — ДОДАТИ ВИТРАТУ -->
        <TabItem Header="Витрати">
          <ScrollViewer Background="#0F1117"
                        VerticalScrollBarVisibility="Auto">
            <Grid Margin="28,24">
              <Grid.ColumnDefinitions>
                <ColumnDefinition Width="360"/>
                <ColumnDefinition Width="*"/>
              </Grid.ColumnDefinitions>
              <!-- Form card -->
```

```
<Border Grid.Column="0" Background="#1A1D27"
        CornerRadius="14" Padding="24"
        BorderBrush="#252A3D" BorderThickness="1"
        VerticalAlignment="Top">
    <StackPanel>
        <TextBlock Text="Нова витрата"
                   Foreground="#F1F5F9"
                   FontSize="17" FontWeight="Bold"
                   Margin="0,0,0,22"/>
        <!-- Amount -->
        <TextBlock Text="СУМА" Style="{StaticResource Lbl}"/>
        <TextBox x:Name="AmountTextBox"
                 Style="{StaticResource Inp}"
                 Margin="0,0,0,16"/>
        <!-- Category -->
        <TextBlock Text="КАТЕГОРІЯ" Style="{StaticResource Lbl}"/>
        <ComboBox x:Name="CategoryComboBox"
                  Style="{StaticResource Cmb}"
                  Margin="0,0,0,16">
            <ComboBoxItem Content="Їжа"/>
            <ComboBoxItem Content="Транспорт"/>
            <ComboBoxItem Content="Розваги"/>
            <ComboBoxItem Content="Здоров'я"/>
        </ComboBox>
        <!-- Date -->
        <TextBlock Text="ДАТА" Style="{StaticResource Lbl}"/>
        <DatePicker x:Name="ExpenseDatePicker"
                    Style="{StaticResource Dp}"
                    Margin="0,0,0,16"/>
        <!-- ОПИС -->
        <TextBlock Text="ОПИС" Style="{StaticResource Lbl}"/>
        <TextBox x:Name="DescriptionTextBox"
                 Style="{StaticResource Inp}"
                 Margin="0,0,0,16"/>
        <!-- ПЛАТА -->
```

```
        <TextBlock Text="СПОСІБ ОПЛАТИ" Style="{StaticResource Lbl}"/>
        <ComboBox x:Name="PaymentComboBox"
                  Style="{StaticResource Cmb}"
                  Margin="0,0,0,24">
          <ComboBoxItem Content="Готівка"/>
          <ComboBoxItem Content="Банківська картка"/>
          <ComboBoxItem Content="Google Pay"/>
          <ComboBoxItem Content="Apple Pay"/>
          <ComboBoxItem Content="Банківський переказ"/>
        </ComboBox>
        <!-- ДОДАТИ button -->
        <Button Content="Додати витрату"
                Style="{StaticResource BtnPrimary}"
                Height="42"
                Click="AddButton_Click"/>
      </StackPanel>
    </Border>
    <!-- РЕдагування -->
    <Border Grid.Column="1" Margin="20,0,0,0"
            Background="#1A1D27" CornerRadius="14"
            BorderBrush="#252A3D" BorderThickness="1"
            VerticalAlignment="Top" Padding="24">
      <StackPanel>
        <TextBlock Text="Швидкі дії"
                   Foreground="#F1F5F9"
                   FontSize="15" FontWeight="Bold"
                   Margin="0,0,0,16"/>
        <StackPanel Orientation="Horizontal" Margin="0,0,0,8">
          <Button Content="Редагувати вибране"
                  Style="{StaticResource BtnGhost}"
                  Click="EditButton_Click"
                  Margin="0,0,8,0"/>
          <Button Content="Зберегти зміни"
                  Style="{StaticResource BtnGhost}"
                  Click="SaveButton_Click"/>
```

```
            </StackPanel>
            <TextBlock Text="Оберіть рядок у таблиці, щоб завантажити дані для редагування."
                       Foreground="#475569"
                       FontSize="12" TextWrapping="Wrap"
                       Margin="0,8,0,0"/>
          </StackPanel>
        </Border>
      </Grid>
    </ScrollViewer>
  </TabItem>
  <!--  TAB 2 — СТАТИСТИКА -->
  <TabItem Header=" Статистика">
    <ScrollViewer Background="#0F1117"
                  VerticalScrollBarVisibility="Auto">
      <StackPanel Margin="28,24">
        <TextBlock Text="Огляд фінансів"
                   Foreground="#F1F5F9"
                   FontSize="20" FontWeight="Bold"
                   Margin="0,0,0,20"/>
        <UniformGrid Columns="3" Rows="2">
          <Border Style="{StaticResource KpiCard}" Margin="0,0,12,12">
            <StackPanel>
              <TextBlock Text="ЗАГАЛЬНІ ВИТРАТИ"
                         Foreground="#475569"
                         FontSize="10" FontWeight="SemiBold"
                          Margin="0,0,0,8"/>
              <TextBlock x:Name="TotalExpensesText"
                         Foreground="#5B8DEF"
                         FontSize="22" FontWeight="Bold"/>
            </StackPanel>
          </Border>
          <Border Style="{StaticResource KpiCard}" Margin="0,0,12,12">
            <StackPanel>
              <TextBlock Text="СЕРЕДНЯ ВИТРАТА"
                         Foreground="#475569"
```

```
                FontSize="10" FontWeight="SemiBold"
                 Margin="0,0,0,8"/>
      <TextBlock x:Name="AverageExpensesText"
                Foreground="#2DD4BF"
                FontSize="22" FontWeight="Bold"/>
   </StackPanel>
</Border>
<Border Style="{StaticResource KpiCard}" Margin="0,0,0,12">
   <StackPanel>
      <TextBlock Text="ТОП КАТЕГОРІЯ"
                Foreground="#475569"
                FontSize="10" FontWeight="SemiBold"
                 Margin="0,0,0,8"/>
      <TextBlock x:Name="TopCategoryText"
                Foreground="#F1F5F9"
                FontSize="18" FontWeight="Bold"/>
   </StackPanel>
</Border>
<Border Style="{StaticResource KpiCard}" Margin="0,0,12,0">
   <StackPanel>
      <TextBlock Text="КІЛЬКІСТЬ ЗАПИСІВ"
                Foreground="#475569"
                FontSize="10" FontWeight="SemiBold"
                 Margin="0,0,0,8"/>
      <TextBlock x:Name="CountExpensesText"
                Foreground="#F1F5F9"
                FontSize="22" FontWeight="Bold"/>
   </StackPanel>
</Border>
<Border Style="{StaticResource KpiCard}" Margin="0,0,12,0">
   <StackPanel>
      <TextBlock Text="НАЙБІЛЬША ВИТРАТА"
                Foreground="#475569"
                FontSize="10" FontWeight="SemiBold"
                 Margin="0,0,0,8"/>
```

```
                <TextBlock x:Name="MaxExpenseText"
                           Foreground="#EF4444"
                           FontSize="22" FontWeight="Bold"/>
              </StackPanel>
            </Border>
            <Border Style="{StaticResource KpiCard}" Margin="0,0,0,0">
              <StackPanel>
                <TextBlock Text="ПОТОЧНИЙ МІСЯЦЬ"
                           Foreground="#475569"
                           FontSize="10" FontWeight="SemiBold"
                           Margin="0,0,0,8"/>
                <TextBlock x:Name="CurrentMonthText"
                           Foreground="#F59E0B"
                           FontSize="22" FontWeight="Bold"/>
              </StackPanel>
            </Border>
          </UniformGrid>
        </StackPanel>
      </ScrollViewer>
    </TabItem>
    <!--       ТАВ 3 — ТАБЛИЦЯ   -->
    <TabItem Header="  Таблиця">
      <Grid Background="#0F1117" Margin="28,16,28,16">
        <Grid.RowDefinitions>
          <RowDefinition Height="Auto"/>
          <RowDefinition Height="*"/>
          <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <!-- ФИЛЬТР -->
        <Border Grid.Row="0" Background="#1A1D27"
                CornerRadius="10" Padding="14,10"
                BorderBrush="#252A3D" BorderThickness="1"
                Margin="0,0,0,12">
          <WrapPanel>
            <TextBox x:Name="SearchTextBox"
```

```
                Style="{StaticResource Inp}"
                Width="180" Margin="0,0,8,4"/>
        <Button Content="Пошук"
                Style="{StaticResource BtnPrimary}"
                Width="80" Margin="0,0,8,4"
                Click="SearchButton_Click"/>
        <Button Content="Усі записи"
                Style="{StaticResource BtnGhost}"
                Width="100" Margin="0,0,16,4"
                Click="ShowAllButton_Click"/>
        <DatePicker x:Name="FromDatePicker"
                    Style="{StaticResource Dp}"
                    Width="128" Margin="0,0,8,4"/>
        <DatePicker x:Name="ToDatePicker"
                    Style="{StaticResource Dp}"
                    Width="128" Margin="0,0,8,4"/>
        <Button Content="Фільтр дат"
                Style="{StaticResource BtnGhost}"
                Width="90" Margin="0,0,16,4"
                Click="FilterByDate_Click"/>
        <ComboBox x:Name="FilterCategoryComboBox"
                  Style="{StaticResource Cmb}"
                  Width="140" Margin="0,0,8,4">
          <ComboBoxItem Content="Усі"/>
          <ComboBoxItem Content="Їжа"/>
          <ComboBoxItem Content="Транспорт"/>
          <ComboBoxItem Content="Розваги"/>
          <ComboBoxItem Content="Здоров'я"/>
        </ComboBox>
        <Button Content="Фільтр категорії"
                Style="{StaticResource BtnGhost}"
                Width="130" Margin="0,0,0,4"
                Click="FilterCategory_Click"/>
      </WrapPanel>
    </Border>
```

```
        <!-- DataGrid -->
        <Border Grid.Row="1" Background="#1A1D27"
                CornerRadius="10"
                BorderBrush="#252A3D" BorderThickness="1"
                Padding="0">
          <DataGrid x:Name="ExpensesGrid"
                    Style="{StaticResource Dg}"
                    SelectionChanged="ExpensesGrid_SelectionChanged"/>
        </Border>
        <!-- Action buttons -->
        <StackPanel Grid.Row="2" Orientation="Horizontal"
                    HorizontalAlignment="Right"
                    Margin="0,12,0,0">
          <Button Content="Видалити вибране"
                    Style="{StaticResource BtnDanger}"
                    Width="148" Margin="0,0,8,0"
                    Click="DeleteButton_Click"/>
          <Button Content="Зберегти зміни"
                    Style="{StaticResource BtnPrimary}"
                    Width="140"
                    Click="SaveButton_Click"/>
        </StackPanel>
      </Grid>
    </TabItem>
    <!--            TAB 4 — BAR CHART     -->
    <TabItem Header=" Місяці">
      <Grid Background="#0F1117" Margin="28,24">
        <Border Background="#1A1D27" CornerRadius="14"
                BorderBrush="#252A3D" BorderThickness="2"
                Padding="20">
          <StackPanel>
            <TextBlock Text="Витрати за місяцями"
                       Foreground="#F1F5F9"
                       FontSize="15" FontWeight="Bold"
                       Margin="0,0,0,12"/>
```

```
                <lvc:CartesianChart
                    Series="{Binding BarSeries}"
                    XAxes="{Binding XAxes}"
                    Height="420">
                </lvc:CartesianChart>
            </StackPanel>
        </Border>
    </Grid>
</TabItem>
<!--  TAB 5 — PIE CHART  -->
<TabItem Header="Категорії" Cursor="Hand">
    <Grid Background="#0F1117" Margin="28,24">
        <Border Background="#1A1D27" CornerRadius="14"
                BorderBrush="#252A3D" BorderThickness="2"
                Padding="20">
            <StackPanel>
                <TextBlock Text="Витрати за категоріями"
                           Foreground="#F1F5F9"
                           FontSize="15" FontWeight="Bold"
                           Margin="0,0,0,12"/>
                <lvc:PieChart Series="{Binding Series}"
                              Height="420"
                              LegendPosition="Right"/>
            </StackPanel>
        </Border>
    </Grid>
</TabItem>
</TabControl>
</Grid>
</Window>
```

Лістинг Б.2 - Код головного вікна

```
using LiveChartsCore;
using LiveChartsCore.SkiaSharpView;
```

```
using LiveChartsCore.SkiaSharpView.WPF;
using PersonalFinanceManager.Database;
using PersonalFinanceManager.Models;
using System.Collections.Generic;
using LiveChartsCore.Measure;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace PersonalFinanceManager
{
    public partial class MainWindow : Window, INotifyPropertyChanged
    {
        private ISeries[] _series;
        private Expense selectedExpense;
        public ISeries[] BarSeries { get; set; }
        public Axis[] XAxes { get; set; }
        public ISeries[] Series
        {
            get => _series;
            set
            {
                _series = value;
                OnPropertyChanged(nameof(Series));
            }
        }
```

```
public event PropertyChangedEventHandler PropertyChanged;
protected void OnPropertyChanged(string name)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
}
//   Підключення БД
private DatabaseHelper db = new DatabaseHelper();
public MainWindow()
{
    InitializeComponent();
    //ХАМЛ бачить граф.
    DataContext = this;
    LoadData();
}
//ДОДАВАННЯ ВИТРАТИ
private void AddButton_Click(object sender, RoutedEventArgs e)
{
    if (!double.TryParse(AmountTextBox.Text, out double amount) || amount <= 0)
    {
        MessageBox.Show("Некоректна сума");
        return;
    }
    if (CategoryComboBox.SelectedItem == null ||
        PaymentComboBox.SelectedItem == null ||
        ExpenseDatePicker.SelectedDate == null)
    {
        MessageBox.Show("Заповніть всі поля");
        return;
    }
    var expense = new Expense
    {
        Amount = amount,
        Category = (CategoryComboBox.SelectedItem as ComboBoxItem)?.Content.ToString(),
        PaymentMethod = (PaymentComboBox.SelectedItem as ComboBoxItem)?.Content.ToString(),
        Date = ExpenseDatePicker.SelectedDate.Value,
```

```
        Description = DescriptionTextBox.Text
    };
    db.AddExpense(expense);
    LoadData();
    ClearFields();
    MessageBox.Show("Додано!");
}
private void DeleteButton_Click(object sender, RoutedEventArgs e)
{
    if (ExpensesGrid.SelectedItem == null)
    {
        MessageBox.Show("Оберіть запис");
        return;
    }
    var result = MessageBox.Show("Видалити?", "Confirm",
        MessageBoxButton.YesNo);
    if (result != MessageBoxResult.Yes)
        return;
    var expense = (Expense)ExpensesGrid.SelectedItem;
    db.DeleteExpense(expense.Id);
    LoadData();
}
private void EditButton_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Редагування");
}
private void LoadData()
{
    var data = db.GetExpenses();
    ExpensesGrid.ItemsSource = data;
    TotalExpensesText.Text = data.Sum(x => x.Amount).ToString("0.00");
    AverageExpensesText.Text = data.Any() ? data.Average(x => x.Amount).ToString("0.00") : "0";
    CountExpensesText.Text = data.Count.ToString();
    MaxExpenseText.Text = data.Any()
        ? data.Max(x => x.Amount).ToString("0.00")
```

```
    : "0";
TopCategoryText.Text = data.Any()
    ? data.GroupBy(x => x.Category)
          .OrderByDescending(x => x.Sum(y => y.Amount))
          .First().Key
    : "-";
CurrentMonthText.Text = data
    .Where(x => x.Date.Month == DateTime.Now.Month)
    .Sum(x => x.Amount)
    .ToString("0.00");
Series = data
    .GroupBy(x => x.Category)
    .Select(x => new PieSeries<double>
    {
        Values = new[] { x.Sum(y => y.Amount) },
        Name = x.Key
    })
    .ToArray();
double[] monthValues = new double[12];
for(int i =1; i<=12; i++)
{
    monthValues[i - 1] = data
        .Where(x => x.Date.Month == i)
        .Sum(x => x.Amount);
}
BarSeries = new ISeries[]
{
    new ColumnSeries<double>
    {
        Values = monthValues,
        Name = "Витрати"
    }
};
MonthLabels = new[]
{
```

```
            "Січ",
            "Лют",
            "Бер",
            "Кві",
            "Тра",
            "Чер",
            "Лип",
            "Сер",
            "Вер",
            "Жов",
            "Лис",
            "Гру",
        };
        XAxes = new Axis[]
        {
            new Axis
            {
                Labels = MonthLabels
            }
        };
        OnPropertyChanged(nameof(Series));
        OnPropertyChanged(nameof(BarSeries));
        OnPropertyChanged(nameof(MonthLabels));
        OnPropertyChanged(nameof(XAxes));
    }
    public string[] MonthLabels { get; set; }
    private void SearchButton_Click(object sender, RoutedEventArgs e)
    {
        string search = SearchTextBox.Text.ToLower();
        var data = db.GetExpenses();
        var filtered = data
            .Where(x =>
                x.Category.ToLower().Contains(search)
                || x.Description.ToLower().Contains(search)
                || x.PaymentMethod.ToLower().Contains(search))
```

```
        .ToList();
    ExpensesGrid.ItemsSource = filtered;
}
private void ShowAllButton_Click(object sender, RoutedEventArgs e)
{
    LoadData();
}
private void FilterByDate_Click(object sender, RoutedEventArgs e)
{
    var data = db.GetExpenses();
    if (FromDatePicker.SelectedDate != null)
    {
        data = data
            .Where(x => x.Date >= FromDatePicker.SelectedDate.Value)
            .ToList();
    }
    if (ToDatePicker.SelectedDate != null)
    {
        data = data
            .Where(x => x.Date <= ToDatePicker.SelectedDate.Value)
            .ToList();
    }
    ExpensesGrid.ItemsSource = data;
}
private void FilterCategory_Click(object sender, RoutedEventArgs e)
{
    var data = db.GetExpenses();
    string category =
        (FilterCategoryComboBox.SelectedItem as ComboBoxItem)?.Content.ToString();
    if (category == "Усі")
    {
        ExpensesGrid.ItemsSource = data;
        return;
    }
    var filtered = data
```

```
        .Where(x => x.Category == category)
        .ToList();
    ExpensesGrid.ItemsSource = filtered;
}
private void ExpensesGrid_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    selectedExpense = ExpensesGrid.SelectedItem as Expense;
    if (selectedExpense == null) return;
    AmountTextBox.Text = selectedExpense.Amount.ToString();
    DescriptionTextBox.Text = selectedExpense.Description;
    ExpenseDatePicker.SelectedDate = selectedExpense.Date;
    CategoryComboBox.SelectedItem =
        CategoryComboBox.Items
            .OfType<ComboBoxItem>()
            .FirstOrDefault(x => x.Content.ToString() == selectedExpense.Category);
    PaymentComboBox.SelectedItem =
        PaymentComboBox.Items
            .OfType<ComboBoxItem>()
            .FirstOrDefault(x => x.Content.ToString() == selectedExpense.PaymentMethod);
}
private void SaveButton_Click(object sender, RoutedEventArgs e)
{
    if (selectedExpense == null)
        return;
    if (!double.TryParse(AmountTextBox.Text, out double amount))
    {
        MessageBox.Show("Некоректна сума");
        return;
    }
    selectedExpense.Amount = amount;
    selectedExpense.Description = DescriptionTextBox.Text;
    selectedExpense.Date = ExpenseDatePicker.SelectedDate ?? DateTime.Now;
    selectedExpense.Category =
        (CategoryComboBox.SelectedItem as ComboBoxItem)?.Content.ToString();
    selectedExpense.PaymentMethod =
```

```
            (PaymentComboBox.SelectedItem as ComboBoxItem)?.Content.ToString();
        db.UpdateExpense(selectedExpense);
        LoadData();
        ClearFields();
        MessageBox.Show("Збережено!");
    }
    private void ClearFields()
    {
        AmountTextBox.Clear();
        DescriptionTextBox.Clear();
        CategoryComboBox.SelectedIndex = -1;
        PaymentComboBox.SelectedIndex = -1;
        ExpenseDatePicker.SelectedDate = null;
    }
  }
}
```