СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) Інформаційних технологій та електроніки

Кафедра Інформаційних технологій та програмування
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
(бакалавр, магістр)
спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

на тему Інформаційна система виявлення підозрілих транзакцій у фінансових потоках

Виконав: студент групи ІПЗ-21з _____ А. О. Парамонов
(підпис) (ініціали і прізвище)

Керівник _____ В. Г. Іванов
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ О. І. Захожай
(підпис) (ініціали і прізвище)

Рецензент Ратов Д.В.

Київ – 2025

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) Інформаційних технологій та електроніки.
Кафедра Інформаційних технологій та програмування
(повна назва кафедри)
Освітній ступінь бакалавр
(бакалавр, магістр)
спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
спеціалізація Інженерія програмного забезпечення
(назва спеціалізації)

_____ Захожай О.І.
“___” _____ 2025 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

Парамонову Артему Олеговичу
(прізвище, ім’я, по-баткові)

1. Тема роботи Інформаційна система виявлення підозрілих транзакцій у фінансових потоках
_____
_____

Керівник роботи Іванов Віталій Геннадійович, Доц., к.т.н._____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “27” травня 2025 року № 67/15.15-С

2.Строк подання студентом роботи 20.06.25
3. Вихідні дані до роботи Об’єктом дипломної роботи є розробка та реалізація інформаційної системи виявлення підозрілих транзакцій у фінансових потоках
_____
_____
_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)Вступ.Аналітична частина, з висвітленням наступних питань: Актуальність теми створення інформаційної системи виявлення підозрілих транзакцій у фінансових потоках. Проблематика шахрайства у фінансовому секторі та наслідки для малого й середнього бізнесу. Обґрунтування необхідності використання програмного забезпечення для автоматизації контролю та аналізу транзакцій. Огляд методів виявлення аномалій у транзакційних даних. Аналіз сучасних інструментів та сервісів для виявлення фінансових порушень. Постановка задачі розробки системи, формулювання функціональних та нефункціональних вимог до застосунку.

Основна частина, в якій висвітлено**:** Розробку архітектури програмного забезпечення. Обґрунтування вибору технологій реалізації: Python, PyQt5, SQLite, Pandas, Matplotlib. Побудову модульної структури застосунку із виділенням окремих компонентів: інтерфейс користувача, логіка аналізу, візуалізація, логування, робота з базою даних, експорт звітів. Розробку моделі даних та структури таблиць у базі SQLite для збереження транзакцій. Реалізацію алгоритмів евристичного аналізу: виявлення великих сум, нічних переказів, переказів самому собі. Побудову зручного графічного інтерфейсу користувача з відображенням таблиці, кнопками керування, генерацією звітів у PDF та Excel. Здійснення логування дій користувача для журналу подій. Проведення ручного тестування на основі контрольних сценаріїв. Оцінку стабільності, зручності та ефективності роботи системи.Висновок.Перелік використаних джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників)

_____

_____

6. Консультанти розділів роботи

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

7.Дата видачі завдання 19.04.25

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Аналіз предметної області	19.04.25-27.04.25	Виконано
2	Аналіз існуючих рішень	28.04.25-04.05.25	Виконано
3	Визначення функціональних вимог до системи	05.05.25-12.05.25	Виконано
4	Проєктування інформаційної системи	13.05.25-19.05.25	Виконано
5	Реалізація інформаційної системи	20.05.25-05.06.25	Виконано
6	Тестування інформаційної системи	06.06.25-08.06.25	Виконано
7	Оформлення пояснювальної записки	09.06.25-19.06.25	Виконано
8	Підготовка та подання кваліфікаційної роботи	19.06.25-20.06.25	Виконано

Студент _____ А.О. Парамонов
(підпис) (ініціали і прізвище)

Керівник роботи _____ В. Г. Іванов
(підпис) (ініціали і прізвище)

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку цифрової економіки фінансові операції дедалі частіше здійснюються в електронному вигляді, що створює нові виклики у сфері інформаційної безпеки. Однією з найгостріших проблем є виявлення шахрайських або підозрілих транзакцій, які можуть завдати суттєвих збитків як для окремих користувачів, так і для фінансових установ. Ефективна система моніторингу та аналізу транзакцій є критично важливою для своєчасного реагування на потенційні загрози. Саме тому створення інформаційної системи, здатної виявляти підозрілі транзакції у фінансових потоках, є надзвичайно актуальним і практично значущим завданням.

Метою дипломної роботи є розробка інформаційної системи, яка дозволяє автоматично виявляти підозрілі транзакції на основі евристичних правил аналізу.

Для досягнення мети поставлено наступні завдання:

1. Дослідити методи виявлення аномалій у фінансових транзакціях
2. Спроєктувати структуру системи та обрати технології реалізації
3. Розробити програмне забезпечення з графічним інтерфейсом
4. Реалізувати механізми імпорту даних, аналізу та збереження результатів
5. Забезпечити функціональність експорту звітів та журналу подій
6. Провести тестування системи та оцінити її ефективність.

Об’єктом дослідження є фінансові транзакції, які підлягають автоматизованому аналізу.

Предметом дослідження є методи і засоби виявлення підозрілих операцій у фінансових потоках на основі заданих критеріїв та правил.

У роботі використано методи аналізу, моделювання, порівняння та евристичного оцінювання транзакцій. Технічна реалізація базується на мові програмування Python з використанням бібліотек Pandas, SQLite, PyQt5, Matplotlib та інших засобів візуалізації і звітності.

Результатом дипломної роботи є робоча десктопна програма, яка може бути використана для виявлення потенційно шахрайських транзакцій у невеликих фінансових структурах, бухгалтерських відділах, або як основа для побудови масштабованої аналітичної системи. Система дозволяє зручно завантажувати дані, аналізувати їх, отримувати візуальні результати та генерувати звіти, що є актуальним для підвищення безпеки фінансових операцій.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Інформаційні технології дедалі глибше проникають у всі сфери діяльності людини, зокрема у фінансову галузь. Сьогодні фінансові транзакції здійснюються майже виключно в електронному форматі, що дозволяє забезпечити зручність, швидкість і глобальний доступ до банківських послуг. Проте одночасно з ростом цифровізації зростає й рівень фінансових загроз, зокрема шахрайства у сфері онлайн-платежів[1]. За даними досліджень, кількість інцидентів фінансового шахрайства зростає щороку, що спричиняє як прямі матеріальні збитки, так і втрату довіри клієнтів до фінансових установ.

Для вчасного реагування на потенційні загрози банки, платіжні сервіси та інші фінансові організації впроваджують спеціальні інформаційні системи, які дозволяють виявляти аномальні чи потенційно шахрайські транзакції. Такі системи повинні забезпечувати автоматизований аналіз великих обсягів даних, зручний інтерфейс взаємодії з користувачем, можливість адаптації до нових типів загроз та забезпечення достовірного збереження результатів[2].

У цьому розділі здійснено огляд ключових понять і підходів, пов’язаних із виявленням підозрілих транзакцій у фінансових потоках, а також розглянуто сучасні рішення та методи аналізу, які можуть бути використані при створенні відповідної інформаційної системи.

1.1. Сутність та класифікація фінансових транзакцій

Фінансова транзакція є невід’ємною частиною сучасного економічного життя та являє собою процес передачі грошових коштів або фінансових активів між сторонами — юридичними або фізичними особами[3]. Транзакції можуть здійснюватися в різних формах: банківські перекази, оплата товарів чи послуг, отримання заробітної плати, нарахування бонусів.

В умовах цифрової трансформації значна частина транзакцій виконується у режимі онлайн, що забезпечується сучасними платіжними системами, банківськими API, мобільними додатками та платформами електронної комерції.

Залежно від ознак, транзакції класифікуються наступним чином:

1. **За напрямком операцій:**
   - Вхідні, а саме (отримання коштів)
   - Вихідні (перерахування коштів)
   - Внутрішні (перекази між рахунками одного користувача)
2. **За призначенням платежу:**
   - Комерційні платежі
   - Особисті перекази
   - Соціальні виплати
   - Транзакції у криптовалютах
3. **За обсягом:**
   - Мікротранзакції
   - Середні операції
   - Великі перекази
4. **За способом ініціювання:**
   - Транзакції
   - Ініційовані користувачем
   - Автоматичні списання
   - Системні перекази
5. **За часом:**
   - Звичайні (у робочий час)
   - Позарегламентні (вночі, у вихідні)

Кожен із зазначених типів має свій профіль ризику та може стати об’єктом контролю в системах виявлення шахрайства.

1.2. Проблема виявлення шахрайських операцій

Фінансове шахрайство — це умисні дії, спрямовані на неправомірне заволодіння грошовими коштами або активами шляхом підробки даних, маніпулювання системами або використання соціальної інженерії[3]. Проблема виявлення шахрайства полягає у тому, що шахрайські операції за

зовнішніми ознаками часто нічим не відрізняються від звичайних транзакцій. Це ускладнює їхню автоматичну ідентифікацію та потребує впровадження спеціальних механізмів аналізу поведінки.

До основних сценаріїв шахрайства належать:

- Багаторазове виконання однакових переказів із незначною затримкою
- Перекази на незвичних одержувачів
- Виконання операцій на великі суми поза межами звичних часових рамок
- Транзакції між пов’язаними особами без очевидної мети
- Використання викрадених платіжних даних

Класичні системи моніторингу не завжди справляються з такими випадками, що створює потребу в гнучкій інформаційній системі з можливістю налаштування правил виявлення аномалій[4].

1.3. Методи та підходи до виявлення аномалій

Залежно від рівня складності та доступних ресурсів, системи аналізу транзакцій використовують різні методи:

- **Евристичні правила**

Дозволяють швидко відстежувати закономірності, наприклад:

1. Сума перевищує встановлений поріг
2. Час виконання операції потрапляє у нічний інтервал
3. Відправник і одержувач однакові
4. Частота операцій за короткий проміжок часу перевищує норму

- **Машинне навчання**

Передбачає створення моделей класифікації (Decision Tree, Random Forest, SVM, нейромережі), які здатні виявляти шаблони поведінки на основі попередньо розмічених даних (шахрайські/нормальні).

Методи виявлення аномалій:

1. Кластеризація (DBSCAN, K-means)

2. Статистичне виявлення викидів (Z-score, IQR)
3. Ізоляційні дерева
4. Методи локальних відхилень (LOF).

- **Гібридні підходи:**

Поєднують переваги евристичних методів та ML-моделей, забезпечуючи як базову фільтрацію, так і глибокий аналіз[5].

У цій дипломній роботі реалізовано евристичний підхід як просту, але водночас найефективнішу базу для виявлення типових аномалій, що часто зустрічаються у фінансових потоках.

1.4. Аналіз існуючих систем виявлення шахрайства

На ринку існує велика кількість готових рішень, що дозволяють виявляти шахрайські транзакції. Вони можуть бути поділені на такі категорії:

1. Корпоративні системи:

SAS Fraud Management, FICO Falcon, Actimize — дорогі рішення, орієнтовані на великі банки, що мають складну структуру і вимагають значних ресурсів на інтеграцію та підтримку.

2. Комерційні SaaS-рішення:

Kount, ThreatMetrix, Signifyd — призначені для онлайн-магазинів та сервісів, мають API та дашборди для моніторингу транзакцій.

3. Програмні фреймворки:

Python-бібліотеки (PyOD, scikit-learn, Pandas, TensorFlow, CatBoost) — використовуються розробниками для побудови кастомних рішень, зокрема з використанням аномалій чи машинного навчання.

4. Відкриті системи аналізу:

OpenFDS, FraudLabs Pro (Free), Streamlit та Pandas — приклади з відкритим кодом для навчання, тестування або прототипування.

Порівняльний аналіз таких систем дозволяє зробити висновок: для локального, легкого використання найбільш доречно реалізувати власну евристичну систему у вигляді десктоп-додатка з простою структурою, зрозумілим інтерфейсом та можливістю подальшої модернізації.

У цьому розділі було розглянуто теоретичні аспекти виявлення підозрілих транзакцій, а також класифікацію фінансових операцій, типові сценарії шахрайства, методи виявлення аномалій та існуючі системи виявлення.

На основі аналізу встановлено, що:

- Найпростіші, але ефективні рішення можуть бути реалізовані за допомогою евристичних методів
- Більшість комерційних систем є складними у впровадженні для малого та середнього бізнесу
- Актуальність створення легкої, адаптивної системи на Python є виправданою як з технічної, так і з економічної точки зору.

Отримані висновки дають змогу чітко сформулювати вимоги до майбутньої системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Проєктування — це критично важливий етап розробки будь-якої інформаційної системи, який визначає не лише її архітектуру та функціональність, а й майбутню зручність використання, масштабованість та надійність[6]. Ефективне проєктування дозволяє уникнути багатьох помилок у реалізації, що особливо актуально для задач виявлення підозрілих транзакцій, де помилкові спрацювання чи пропуски можуть мати фінансові або репутаційні наслідки.

У цьому розділі здійснено постановку задачі розробки інформаційної системи виявлення підозрілих транзакцій, сформульовано вимоги до функціональності, обґрунтовано вибір технологій, розроблено логічну архітектуру та описано структуру зберігання даних. Також подано опис ключових компонентів системи та модель організації інформаційних потоків.

Інформаційна система виявлення підозрілих транзакцій — це спеціалізоване програмне забезпечення, призначене для автоматизованого аналізу фінансових потоків з метою виявлення аномальних, нетипових або потенційно шахрайських транзакцій[7]. Основною особливістю системи є використання евристичних методів аналізу, що дозволяють швидко й ефективно виявляти транзакції, які можуть потребувати додаткової перевірки або втручання людини.

Розроблена система є автономним десктопним застосунком, що не потребує підключення до мережі Інтернет або встановлення додаткових серверних компонентів. Це дозволяє забезпечити високий рівень безпеки даних і простоту встановлення на будь-якому персональному комп’ютері.

Система реалізує повний цикл обробки фінансових транзакцій, який включає:

1. Імпорт масиву транзакцій із зовнішнього джерела
2. Попередню обробку і збереження даних у локальну базу
3. Автоматичний аналіз із застосуванням набору евристичних правил
4. Виведення підозрілих транзакцій у зручному форматі

5. Побудову графіків для візуального аналізу активності
6. Формування фінального звіту у форматах PDF та Excel
7. Ведення журналу подій із реєстрацією ключових дій користувача.

Основні характеристики системи:

Тип системи: локальна десктопна програма з графічним інтерфейсом, що працює у середовищі операційної системи Windows.

Призначення: виявлення підозрілих транзакцій на основі заданих критеріїв, обробка великого обсягу фінансових операцій, формування звітності для подальшого аналізу або аудиту.

Кінцевий користувач: спеціаліст без глибоких технічних знань — бухгалтер, аудитор, фінансовий аналітик, співробітник відділу безпеки або керівник фінансового підрозділу.

Робота з даними: підтримка зчитування транзакцій з файлів CSV, обробка та збереження даних у базі SQLite, фільтрація та аналіз, експорт результатів у формати PDF та Excel. Крім того, реалізація візуалізації активності транзакцій на графіках — як загальної динаміки, так і окремо для виявлених підозрілих операцій.

Безпека та автономність: програмне забезпечення повинно функціонувати повністю автономно, без доступу до мережі Інтернет, що виключає можливість витоку чутливих фінансових даних. Всі операції повинні виконуватись локально, а дані користувача залишаються конфіденційними.

Інтерфейс користувача: система повинна мати інтуїтивно зрозумілий інтерфейс, реалізований за допомогою PyQt5, який дозволить завантажувати файли, переглядати таблиці, запускати аналіз, очищувати базу, будувати графіки та створювати звіти лише за допомогою кількох кліків.

2.2. Постановка задачі та вимоги до системи

Формулювання задачі

Метою є створення локальної десктопної інформаційної системи, яка дозволяє виявляти підозрілі транзакції на основі заздалегідь визначених

евристичних правил. Система має аналізувати імпортовані фінансові операції, зберігати їх у базі даних, здійснювати перевірку на наявність ознак шахрайства, формувати результати у зручному для користувача форматі та забезпечувати повну автономність використання[8].

Користувач повинен мати змогу швидко завантажити великий масив транзакцій (наприклад, з банківської системи або Excel-таблиці), переглянути інформацію у таблиці, провести автоматичний аналіз, побачити список потенційно небезпечних операцій та, за необхідності, створити офіційний звіт[9].

Функціональні вимоги

До основних функціональних можливостей системи належать:

- Імпорт транзакцій
- Зчитування CSV-файлів, експортованих із банківських систем або внутрішніх облікових програм
- Автоматичне перетворення типів даних (дата, сума, текст)
- Обробка файлів обсягом понад 10 000 записів
- Збереження транзакцій у БД
- Використання вбудованої SQLite-бази
- Виключення дублювання транзакцій за ідентифікатором
- Можливість очищення бази даних користувачем через інтерфейс

Аналіз підозрілих транзакцій:

1. **Фільтрація за критеріями:**

- Сума більша за встановлений поріг (наприклад, 10 000 грн)
- Транзакції, здійснені між 00:00 та 05:00
- Перекази самому собі (відправник і одержувач однакові)
- Транзакції, які повторюються більше 2 разів у межах 1 хвилини

2. **Відображення результатів:**

- Табличне представлення всієї бази даних
- Фільтр лише підозрілих операцій

- Окрема колонка з позначкою «Підозра»

3. **Візуалізація даних:**

- Побудова графіків динаміки транзакцій за датами
- Окремі графіки для підозрілих транзакцій
- Аналіз частоти за годинами, днями тижня

4. **Генерація звітів:**

- Експорт результатів аналізу у PDF із підсумками
- Формування таблиці з ключовими метриками
- Можливість експорту в Excel для подальшої роботи в бухгалтерських системах

5. **Логування дій:**

- Збереження історії запусків
- Запис часу імпорту, аналізу, експорту
- Простий лог-файл у форматі .txt

6. **Керування даними:**

- Кнопка очищення бази;
- Запобігання випадковому видаленню

Нефункціональні вимоги

- Продуктивність: програма має обробляти до 100 тис. записів без збоїв і зависань
- Безпека: усі дані зберігаються локально без мережевих підключень
- Інтерфейс: інтуїтивно зрозумілий інтерфейс, придатний для користувача без спеціальних IT-знань
- Надійність: система повинна коректно працювати навіть у разі помилок у CSV-файлі
- Портативність: програма має легко запускатися на будь-якому ПК з встановленим Python

2.3. Вибір засобів реалізації

Під час розробки інформаційної системи виявлення підозрілих транзакцій одним із ключових етапів стало обґрунтування та вибір технологій, інструментів і програмних засобів. Основними критеріями відбору були: простота використання, відповідність функціональним потребам, відкритість технологій, сумісність між компонентами, а також наявність якісної документації та активної підтримки спільноти[10].

Основною мовою програмування було обрано Python. Це високорівнева мова загального призначення, яка надає широкі можливості для створення як веб-, так і десктопних застосунків. Python вирізняється зручним синтаксисом, великою кількістю бібліотек для аналізу даних, побудови графічного інтерфейсу, обробки таблиць та роботи з базами даних, що і стало головним аргументом на її користь[11].

Графічний інтерфейс програми реалізовано за допомогою бібліотеки PyQt5 — одного з найпопулярніших інструментів для створення GUI-додатків на Python. PyQt5 забезпечує розробку віконних застосунків із підтримкою кнопок, меню, таблиць, діалогових вікон, а також дозволяє інтегрувати візуалізації та графіки без необхідності у використанні додаткових фреймворків[12].

Для зберігання транзакцій обрана вбудована база даних SQLite. Вона є надійною, простою в реалізації та не потребує встановлення окремого сервера, що ідеально підходить для локального застосування. Завдяки цьому рішення можна розгортати на будь-якому персональному комп’ютері без складної конфігурації[13]. У SQLite створено одну основну таблицю, яка зберігає інформацію про транзакції: дату, час, суму, відправника, одержувача та ознаку підозрілості.

Для імпорту та обробки даних із CSV-файлів використано бібліотеку Pandas — стандарт де-факто у світі обробки табличної інформації в Python. Вона дозволяє ефективно працювати з великими масивами даних, здійснювати фільтрацію, сортування, агрегацію та виконувати пошук аномалій за заданими

критеріями. Перевагою є також легкість експорту результатів у формати CSV або Excel[14].

Побудову графіків і візуалізацію динаміки транзакцій забезпечує бібліотека Matplotlib. Вона дозволяє створювати гістограми, діаграми, графіки активності за датами й часом, що є важливим для виявлення закономірностей у підозрілих операціях. Графіки інтегруються безпосередньо в інтерфейс користувача, що покращує візуальне сприйняття даних.

Окрему увагу приділено формуванню звітності. Для створення PDF-звітів використано бібліотеку ReportLab, яка дає змогу програмно створювати звіти з текстом, таблицями, заголовками та оформленням[15]. Це дозволяє користувачеві отримувати підсумки аналізу у форматі, придатному для друку чи архівування. Експорт у Excel реалізовано за допомогою функціоналу Pandas із використанням форматування для сумісності з табличними редакторами.

Ще одним важливим компонентом стала реалізація механізму логування дій. Було створено окремий модуль, що відповідає за збереження історії запусків системи, аналізу транзакцій, експорту результатів та інших ключових дій користувача у вигляді лог-файлу. Це дає змогу простежити хронологію використання програми та здійснити аудит дій у разі потреби.

Усі компоненти системи розроблялися у середовищі PyCharm — сучасній інтегрованій системі розробки для Python, яка забезпечує зручну роботу з файлами проєкту, підсвічування синтаксису, автоматичну перевірку помилок, інтеграцію з Git та інші інструменти, що значно прискорюють і полегшують процес розробки[16].

Таким чином, обраний набір технологій повністю відповідає поставленим завданням. Він дозволяє створити ефективну, стабільну, масштабовану та зручну для користувача інформаційну систему з повним циклом обробки, аналізу, візуалізації та документування фінансових транзакцій. Обґрунтований вибір інструментів є запорукою успішної реалізації наступного етапу — програмної реалізації інформаційної системи.

2.4. Архітектура програмного забезпечення

Розроблена інформаційна система побудована за принципами модульної архітектури, що передбачає поділ функціональності програми на окремі логічно незалежні частини. Такий підхід дає низку переваг: спрощення процесу розробки, тестування та супроводу, а також можливість повторного використання модулів у майбутніх проєктах[17].

Кожен модуль виконує чітко визначену роль у межах загальної логіки програми. Міжмодульна взаємодія відбувається за допомогою передавання структурованих об'єктів даних, що дозволяє уникнути жорсткого зв'язування компонентів і покращує масштабованість системи

Модулі та їх функціональність:

- main.py — головний запускний файл. Ініціалізує інтерфейс програми, обробляє помилки на рівні запуску, виступає точкою входу в застосунок. Саме звідси користувач починає роботу з програмою.
- gui.py — центральний модуль побудови графічного інтерфейсу користувача на базі PyQt5. Відповідає за створення основного вікна програми, відображення таблиць, кнопок, форм, підключення сигналів до відповідних функцій, реакцію на дії користувача. Координує роботу всіх інших модулів, виступаючи як «контролер» у шаблоні Model-View-Controller.
- analysis.py — модуль обробки та аналізу транзакцій. Містить набір евристичних правил, що застосовуються до кожного запису: виявлення транзакцій із завищеною сумою, транзакцій у нічний час, переказів самому собі, повторюваних операцій тощо. Повертає результати аналізу у вигляді міток або окремого DataFrame.
- database.py — реалізує збереження, оновлення та видалення даних у локальній базі SQLite. Забезпечує створення таблиць, імпорт CSV-файлів у базу, отримання записів для відображення, а також операцію повного очищення бази за запитом користувача.

- pdf_export.py — відповідає за створення структурованого звіту у форматі PDF із результатами аналізу. У звіті виводяться загальна кількість транзакцій, кількість підозрілих, список аномальних операцій, дата аналізу, а також заголовки та таблиці в зручному форматі.
- logger.py — реалізує систему логування дій користувача. Кожен запуск програми, імпорт транзакцій, запуск аналізу, створення звітів, очищення бази тощо фіксується у лог-файлі у вигляді текстових повідомлень із зазначенням дати й часу. Це дозволяє вести історію використання та забезпечує мінімальний аудит активності.
- graph.py — модуль побудови графіків на основі даних транзакцій. Генерує діаграми активності за днями, годинами, дозволяє побачити пік операцій або аномальні сплески. Працює у зв’язці з Matplotlib і відображається безпосередньо у GUI.

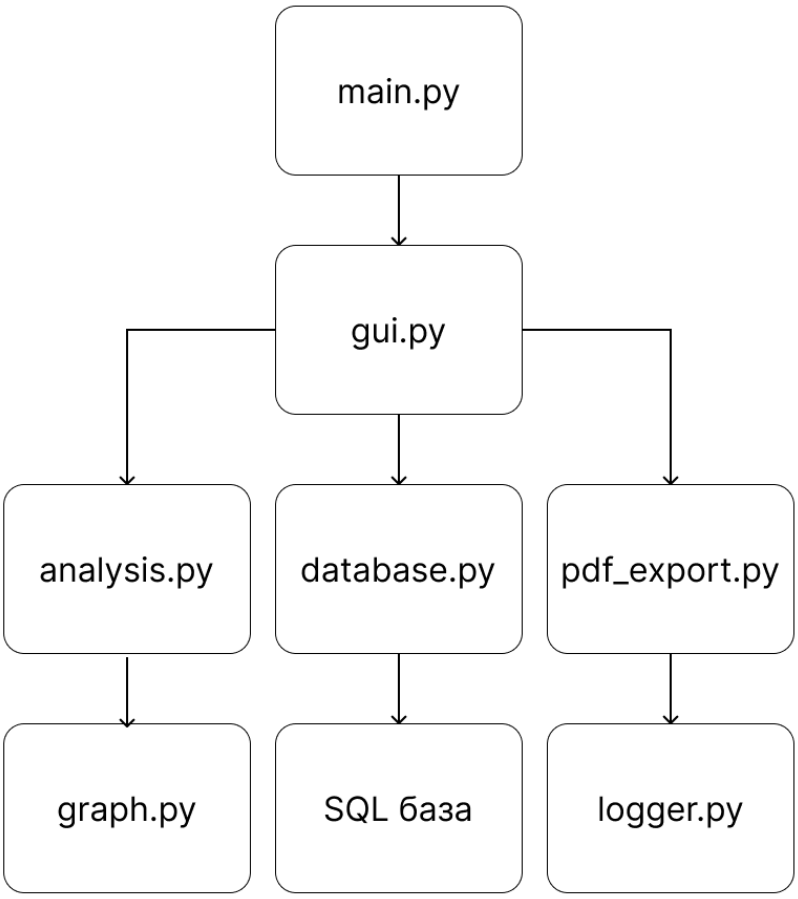

Рис. 2.1 — Взаємодія модулів

Архітектурна модель

Загалом, архітектура системи базується на класичному патерні Model-View-Controller (MVC):

1. Model: база даних SQLite та модуль analysis.py, що обробляє бізнес-логіку.
2. View: графічний інтерфейс (gui.py), який взаємодіє з користувачем.
3. Controller: модуль main.py і частково gui.py, що з'єднує логіку та візуальну частину.

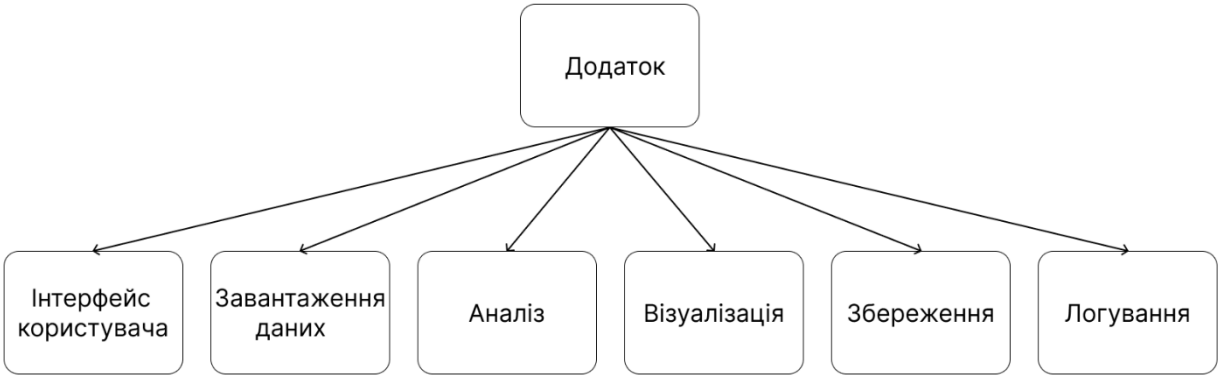

Рис. 2.2 — Архітектурна модель

Схема взаємодії

1. Користувач запускає програму через main.py.
2. Завантажується інтерфейс (gui.py), який показує кнопку для імпорту CSV.
3. CSV-файл зчитується, обробляється і передається в database.py для збереження.
4. Після імпорту — транзакції виводяться у таблиці інтерфейсу.
5. При натисканні кнопки "Аналіз" викликається analysis.py, який повертає список підозрілих транзакцій.
6. Дані позначаються у таблиці, графіки будуються через graph.py.
7. За потреби створюється PDF-звіт через pdf_export.py.
8. Усі дії користувача (імпорт, аналіз, звіти) записуються у logger.py.

Переваги архітектури

- Зрозумілість і модульність: кожен компонент має чітко визначене завдання.
- Масштабованість: легко додати новий модуль
- Зручність тестування: модулі можна перевіряти окремо.
- Гнучкість: можлива подальша адаптація під Web або мобільну версію.

2.5. Моделювання структури даних

Для забезпечення збереження, обробки та аналізу транзакцій у межах інформаційної системи виявлення підозрілих транзакцій була розроблена структурована модель даних, яка реалізується у вигляді локальної реляційної бази даних на основі SQLite. Ця модель дозволяє ефективно зберігати всі необхідні атрибути транзакцій, забезпечувати швидкий доступ до інформації, здійснювати пошук, фільтрацію, обчислення та аналіз підозрілих операцій[18].

База даних побудована у вигляді однієї основної таблиці transactions, яка містить усю інформацію про фінансові операції. Така однотаблична структура обрана з міркувань простоти, автономності та відсутності потреби в складному реляційному моделюванні у межах локальної десктопної системи.

Основна таблиця: transactions

Таблиця містить такі поля:

- id — ціле число, первинний ключ, автоматично збільшується (INTEGER PRIMARY KEY AUTOINCREMENT);
- date — дата транзакції у форматі "РРРР-ММ-ДД" (TEXT);
- time — час транзакції у форматі "ГГ:ХХ:СС" (TEXT);
- amount — сума транзакції, десяткове число (REAL);
- sender — ідентифікатор або назва відправника (TEXT);
- receiver — ідентифікатор або назва отримувача (TEXT);

- is_suspicious — ознака підозрілості, булеве значення (INTEGER: 0 — ні, 1 — так);
- suspicion_reason — причина, через яку транзакція визнана підозрілою (TEXT).

Ці поля охоплюють основні атрибути, необхідні для аналітики: хронологічну інформацію, суму переказу, учасників операції та результат евристичного аналізу. Поле suspicion_reason є ключовим для пояснення, чому саме транзакція була позначена системою як потенційно небезпечна. Це може бути, наприклад, «Сума перевищує 10 000», «Операція вночі», «Переказ самому собі».

Переваги моделі даних:

- Стислість і зрозумілість — уся інформація знаходиться в одній таблиці, що спрощує роботу з нею
- Масштабованість — за потреби структура може бути розширена без переробки основного ядра системи
- Продуктивність — оптимізовано для десктопного використання без надмірного навантаження
- Універсальність — така структура сумісна з більшістю систем обробки даних (можна легко експортувати до Excel або BI-систем).

Модель даних є критичним компонентом архітектури програмного забезпечення, адже вона визначає спосіб збереження і представлення транзакцій для подальшого аналізу[19]. Обрана структура відповідає потребам проєкту: вона проста, надійна, ефективна і гнучка до подальших змін. Це дозволяє сконцентруватися на реалізації аналітичної логіки, не втрачаючи при цьому контроль над якістю і достовірністю даних.

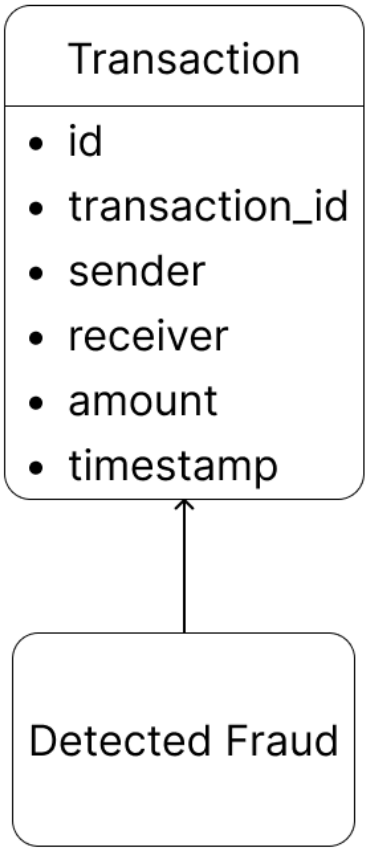

Рис. 2.3 — Основна таблиця

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Розробка програмного забезпечення — це практичний етап, який реалізує попередньо сформульовані вимоги, архітектуру та структуру даних. На цьому етапі відбувається безпосереднє написання коду, інтеграція компонентів, перевірка працездатності та підготовка до тестування. Реалізація інформаційної системи виявлення підозрілих транзакцій базується на обраній модульній архітектурі, засобах розробки на Python і використанні бібліотек для аналізу, графіки, побудови інтерфейсу та формування звітності.

У цьому розділі детально розглядається, як саме реалізовано ключові компоненти: завантаження та збереження транзакцій, виконання аналізу, роботу інтерфейсу, побудову графіків, генерацію звітів, логування дій користувача. Програмний код побудований у зручній для розширення формі, що дозволяє у майбутньому адаптувати систему під нові умови чи вдосконалити механізми аналізу.

3.1. Технології реалізації

Розробка системи здійснювалася мовою Python, що була обрана за її універсальність, простий синтаксис та розвинуту екосистему бібліотек. **Основними інструментами реалізації виступили:**

- PyQt5 — для побудови графічного інтерфейсу користувача
- SQLite — для локального збереження даних
- Pandas — для обробки, фільтрації та аналізу транзакцій
- Matplotlib — для побудови графіків
- ReportLab — для створення PDF-звітів
- стандартна бібліотека Python (os, datetime, csv) — для системних операцій, зчитування файлів, форматування дат та часу.

Проєкт реалізовувався у середовищі розробки PyCharm, що забезпечує зручне керування модулями, інтеграцію з Git, автодоповнення коду, відлагодження та швидке тестування. Вся логіка розподілена по функціональних модулях, описаних у попередньому розділі. Робота над застосунком проводилася поетапно — спочатку реалізовувалися базові

функції завантаження та аналізу транзакцій, далі — інтерфейс, звіти, графіки та ведення логування.

3.2. Опис головних модулів

Реалізація програмного забезпечення базується на поділі функціональності між кількома логічними модулями. Такий підхід дозволив із самого початку структурувати код, уникнути дублювання, а також спростити супровід і подальший розвиток системи. У цьому підрозділі розглянуто реалізацію основних функцій, які становлять ядро інформаційної системи.

Імпорт даних та робота з базою (модулі database.py, analysis.py)

Файл database.py відповідає за збереження транзакцій у локальній базі SQLite.

При імпорті CSV-файлу відбувається:

1. Відкриття та читання вмісту файлу
2. Валідація структури таблиці (наявність необхідних стовпців: дата, час, сума, відправник, отримувач)
3. Очищення попередніх даних (якщо викликано опцію очищення)
4. Збереження кожного запису у таблицю transactions

Дані зберігаються у форматі, який дозволяє подальший фільтр за сумою, датою, часом, а також пошук повторюваних операцій або транзакцій із однаковими учасниками.

```python
import sqlite3
from pathlib import Path
import pandas as pd


DB_PATH = Path("data/results.sqlite")

def create_connection():  4 usages
    conn = sqlite3.connect(DB_PATH)
    return conn

def create_tables():  4 usages
    conn = create_connection()
    cursor = conn.cursor()

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS transactions (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            transaction_id TEXT,
            sender TEXT,
            receiver TEXT,
            amount REAL,
            timestamp TEXT
        )
    ''')

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS suspicious (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            transaction_id TEXT,
            reason TEXT
        )
```

Рис. 3.1 — Модуль бази даних

Аналіз транзакцій (модуль analysis.py)

У модулі analysis.py реалізовано базовий евристичний аналіз транзакцій.

Для кожної транзакції система перевіряє низку умов:

1. Чи перевищує сума заданий поріг (10 000 грн)
2. Чи виконано переказ у нічний час (з 00:00 до 06:00)
3. Чи співпадають відправник і отримувач

4. Чи є повторення ідентичних транзакцій у межах короткого інтервалу

Кожна транзакція, яка задовольняє хоча б одну з умов, позначається як підозріла, а у відповідне поле suspicion_reason вноситься причина виявлення.

```python
import pandas as pd
from database import insert_transaction, insert_suspicious
from rules import SUSPICIOUS_AMOUNT_THRESHOLD, NIGHT_HOURS_START, NIGHT_HOURS_END



SUSPICIOUS_AMOUNT_THRESHOLD = 10000
NIGHT_HOURS = range(NIGHT_HOURS_START, NIGHT_HOURS_END)

def load_transactions_from_csv(file_path: str) -> pd.DataFrame:  2 usages
    df = pd.read_csv(file_path)
    df.columns = df.columns.str.lower()
    return df

def analyze_transactions(df: pd.DataFrame):  2 usages
    for _, row in df.iterrows():
        tx = {
            'transaction_id': row['transaction_id'],
            'sender': row['sender'],
            'receiver': row['receiver'],
            'amount': float(row['amount']),
            'timestamp': row['timestamp']
        }

        # Збереження в БД
        insert_transaction(tx)

        # Аналіз
        suspicious_reasons = []

        if tx['amount'] > SUSPICIOUS_AMOUNT_THRESHOLD:
            suspicious_reasons.append('Велика сума')
```

Рис. 3.2 — Модуль аналітики

Інтерфейс користувача (модуль gui.py)

Файл gui.py реалізує інтерфейс користувача з використанням PyQt5. **Інтерфейс має такі компоненти:**

1. Кнопки для завантаження CSV, запуску аналізу, побудови графіків, експорту звіту, очищення бази
2. Таблицю з усіма транзакціями, де підозрілі виділяються кольором
3. Вікна повідомлень для відображення підсумків
4. Інтеграцію із графіками та PDF-звітом

Програма надає повний цикл дій — від імпорту даних до створення звіту без необхідності звернення до зовнішніх сервісів або командного рядка.

```
import sys
from PyQt5.QtWidgets import (
    QApplication, QWidget, QPushButton, QLabel, QVBoxLayout,
    QFileDialog, QMessageBox, QTableWidget, QTableWidgetItem,
    QLineEdit, QDialog, QFormLayout, QSpinBox, QDialogButtonBox,
    QTextEdit
)
from analysis import load_transactions_from_csv, analyze_transactions
from database import (
    create_tables, get_all_suspicious, export_suspicious_to_csv, clear_database
)
from report import generate_pdf_report
from rules import (
    SUSPICIOUS_AMOUNT_THRESHOLD, NIGHT_HOURS_START, NIGHT_HOURS_END, update_rules
)
from logger import write_log, read_logs

class FraudDetectionApp(QWidget):  1 usage
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Система виявлення підозрілих транзакцій")
        self.setGeometry(100, 100, 800, 600)
        self.init_ui()

    def init_ui(self):  1 usage
        self.layout = QVBoxLayout()

        self.label_status = QLabel("Оберіть файл з транзакціями...")
        self.layout.addWidget(self.label_status)

        self.btn_load = QPushButton("Завантажити CSV")
        self.btn_load.clicked.connect(self.load_file)
```

Рис. 3.3 — Модуль інтерфейсу користувача

Генерація звітності (модуль pdf_export.py)

Для створення друкованого звіту реалізовано модуль pdf_export.py, який використовує бібліотеку ReportLab.

У звіт входить:

1. Заголовок з датою створення
2. Таблиця підозрілих транзакцій
3. Кількість записів та коротке зведення
4. Пояснення до правил виявлення

Звіт зберігається у форматі PDF і автоматично відкривається після збереження.

```python
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from database import get_all_suspicious
import sqlite3
import os
from pathlib import Path


def generate_pdf_report(output_path):  2 usages
    font_path = str(Path("assets/DejaVuSans.ttf").resolve())

    pdfmetrics.registerFont(TTFont( name: "DejaVu", font_path))

    conn = sqlite3.connect("data/results.sqlite")
    cursor = conn.cursor()
    cursor.execute("""
        SELECT suspicious.transaction_id, suspicious.reason,
               transactions.sender, transactions.receiver,
               transactions.amount, transactions.timestamp
        FROM suspicious
        JOIN transactions ON suspicious.transaction_id = transactions.transaction_id
    """)
    rows = cursor.fetchall()
    conn.close()

    c = canvas.Canvas(output_path, pagesize=A4)
    width, height = A4
    c.setFont( psfontname: "DejaVu",  size: 12)
    c.drawString( x: 50, height - 50,  text: "Звіт про підозрілі транзакції")
    y = height - 80
```

Рис. 3.4 — Модуль генерації .pdf звіту

Графіки активності (модуль graph.py)

Модуль graph.py дозволяє побудувати два типи графіків:

1. Загальну активність транзакцій за годинами;
2. Окрему активність підозрілих транзакцій.

Це дає змогу користувачеві візуально оцінити час пікової активності, а також порівняти звичайні й аномальні операції у часовому розрізі.

```python
import pandas as pd
import matplotlib.pyplot as plt
import sqlite3

def get_transaction_data():  2 usages
    conn = sqlite3.connect("data/results.sqlite")
    df = pd.read_sql_query( sql: "SELECT * FROM transactions", conn)
    conn.close()
    return df

def plot_amount_distribution():  2 usages
    df = get_transaction_data()
    plt.figure(figsize=(8, 5))
    plt.hist(df['amount'], bins=10, edgecolor='black')
    plt.title("Розподіл сум транзакцій")
    plt.xlabel("Сума")
    plt.ylabel("Кількість")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_time_distribution():  2 usages
    df = get_transaction_data()
    df['hour'] = pd.to_datetime(df['timestamp']).dt.hour
    plt.figure(figsize=(8, 5))
    df['hour'].value_counts().sort_index().plot(kind='bar')
    plt.title("Транзакції за годинами доби")
    plt.xlabel("Година")
    plt.ylabel("Кількість транзакцій")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Рис. 3.5 — Модуль побудови графіків

Журнал подій (модуль logger.py)

Кожна дія користувача (запуск, імпорт, аналіз, створення звіту, очищення бази) фіксується в окремому лог-файлі. Це дозволяє відслідковувати історію використання, а також виконувати аудит або аналіз випадків порушень.

```python
from datetime import datetime
from pathlib import Path

LOG_FILE = Path("data/logs.txt")

def write_log(message: str):  7 usages
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open(LOG_FILE, "a", encoding="utf-8") as f:
        f.write(f"[{now}] {message}\n")

def read_logs() -> str:  2 usages
    if LOG_FILE.exists():
        return LOG_FILE.read_text(encoding="utf-8")
    return "Лог-файл порожній."
```

Рис. 3.6 — Модуль журналу подій

Головний запуск (main.py)

Головний файл main.py відповідає лише за запуск інтерфейсу користувача. Завдяки цьому логіка застосунку чітко відокремлена від точки входу, що відповідає принципам чистої архітектури.

```python
from gui import launch_app
from database import create_tables

if __name__ == '__main__':
    create_tables()
    launch_app()
```

Рис. 3.7 — Модуль головного файлу запуску

3.3. Взаємодія модулів у процесі виконання програми

Ефективна робота інформаційної системи виявлення підозрілих транзакцій базується на злагодженій взаємодії між її функціональними модулями. Завдяки чітко структурованій архітектурі, кожен модуль виконує свою роль, а обмін інформацією між ними відбувається за допомогою

передавання даних (наприклад, у вигляді структур Pandas DataFrame або SQL-запитів). У цьому підрозділі наведено опис логіки послідовної роботи системи, починаючи від запуску інтерфейсу та завершуючи формуванням звіту й логуванням подій.

1. Запуск програми

Після запуску файлу main.py викликається функція launch_app(), яка створює екземпляр головного вікна інтерфейсу — FraudDetectionApp() з модуля gui.py. Водночас створюється вікно з усіма необхідними елементами: кнопками, таблицями, діалоговими вікнами, підключеними до відповідних функцій.

Модулі, що взаємодіють: main.py та gui.py.

```
class FraudDetectionApp(QWidget):  1 usage
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Система виявлення підозрілих транзакцій")
        self.setGeometry(100, 100, 800, 600)
        self.init_ui()

    def init_ui(self):  1 usage
        self.layout = QVBoxLayout()

        self.label_status = QLabel("Оберіть файл з транзакціями...")
        self.layout.addWidget(self.label_status)

        self.btn_load = QPushButton("Завантажити CSV")
        self.btn_load.clicked.connect(self.load_file)
        self.layout.addWidget(self.btn_load)

        self.btn_analyze = QPushButton("Запустити аналіз")
        self.btn_analyze.clicked.connect(self.run_analysis)
        self.btn_analyze.setEnabled(False)
        self.layout.addWidget(self.btn_analyze)

        self.btn_plot = QPushButton("Показати графіки")
        self.btn_plot.clicked.connect(self.show_graphs)
        self.btn_plot.setEnabled(False)
        self.layout.addWidget(self.btn_plot)

        self.btn_export = QPushButton("Експортувати результати (CSV)")
        self.btn_export.clicked.connect(self.export_results)
        self.btn_export.setEnabled(False)
        self.layout.addWidget(self.btn_export)
```

Рис. 3.8 — Функція створення графічного інтерфейсу

2. Завантаження CSV-файлу

Користувач натискає кнопку "Завантажити CSV". Через діалогове вікно PyQt5 обирається файл. Файл читається за допомогою Pandas (read_csv()), після чого всі записи передаються до функції export_suspicious_to_csv(output_path) із модуля database.py.

Взаємодія: gui.py та database.py.

У базі transactions.db автоматично створюється таблиця (якщо ще не існує), і всі транзакції з CSV-файлу зберігаються. Якщо під час зчитування або структурування виникає помилка — система повідомляє користувача через pop-up повідомлення.

```
def export_suspicious_to_csv(output_path):  2 usages
    conn = sqlite3.connect("data/results.sqlite")
    query = """
        SELECT suspicious.id, suspicious.transaction_id, suspicious.reason,
               transactions.sender, transactions.receiver, transactions.amount, transactions.timestamp
        FROM suspicious
        JOIN transactions ON suspicious.transaction_id = transactions.transaction_id
    """
    df = pd.read_sql_query(query, conn)
    conn.close()
    df.to_csv(output_path, index=False, encoding='utf-8-sig')
```

Рис. 3.9 — Функція завантаження файлу

3. Відображення транзакцій у таблиці

Після збереження даних у базу, викликається функція insert_transaction(tx) з модуля database.py, яка витягує всі записи з таблиці transactions. Дані виводяться у вигляді табличного компонента інтерфейсу (QTableWidget), де користувач може переглядати транзакції, їх суми, учасників і дати.

```
def insert_transaction(tx):  2 usages
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute( sql: '''
        INSERT INTO transactions (transaction_id, sender, receiver, amount, timestamp)
        VALUES (?, ?, ?, ?, ?)
    ''',  parameters: (tx['transaction_id'], tx['sender'], tx['receiver'], tx['amount'], tx['timestamp']))
    conn.commit()
    conn.close()
```

Рис. 3.10 — Функція відображення транзакцій

4. Аналіз підозрілих транзакцій

При натисканні кнопки "Аналізувати" викликається функція analyze_transactions() з модуля analysis.py. Ця функція отримує всі транзакції з бази, проводить послідовну перевірку за заданими евристичними критеріями (сума, нічний час, співпадіння відправника й отримувача тощо), після чого оновлює базу даних, встановлюючи мітку is_suspicious = 1 та додаючи suspicion_reason.

Взаємодія: gui.py та database.py + analysis.py.

Результати аналізу одразу оновлюються у таблиці інтерфейсу: підозрілі операції виділяються іншим кольором, а користувач отримує статистичне повідомлення про кількість виявлених записів.

```python
def analyze_transactions(df: pd.DataFrame):  2 usages
    for _, row in df.iterrows():
        tx = {
            'transaction_id': row['transaction_id'],
            'sender': row['sender'],
            'receiver': row['receiver'],
            'amount': float(row['amount']),
            'timestamp': row['timestamp']
        }

        # Збереження в БД
        insert_transaction(tx)

        # Аналіз
        suspicious_reasons = []

        if tx['amount'] > SUSPICIOUS_AMOUNT_THRESHOLD:
            suspicious_reasons.append('Велика сума')

        if tx['sender'] == tx['receiver']:
            suspicious_reasons.append('Відправник = Отримувач')

        try:
            hour = int(tx['timestamp'].split(' ')[1].split(':')[0])
            if hour in NIGHT_HOURS:
                suspicious_reasons.append('Нічна транзакція')
        except Exception as e:
            print(f"Помилка у timestamp: {tx['timestamp']} – {e}")

        for reason in suspicious_reasons:
            insert_suspicious(tx['transaction_id'], reason)
```

Рис. 3.11 — Функція аналізу транзакцій

5. Побудова графіків

За запитом користувача (натискання кнопки "Побудувати графіки") дані передаються в graph.py. За допомогою бібліотеки Matplotlib формуються дві гістограми:

1. Активність транзакцій за годинами доби
2. Активність підозрілих транзакцій за годинами

Графіки виводяться у новому вікні.

Взаємодія: gui.py та graph.py та database.py.

```
def get_transaction_data():  2 usages
    conn = sqlite3.connect("data/results.sqlite")
    df = pd.read_sql_query( sql: "SELECT * FROM transactions", conn)
    conn.close()
    return df

def plot_amount_distribution():  2 usages
    df = get_transaction_data()
    plt.figure(figsize=(8, 5))
    plt.hist(df['amount'], bins=10, edgecolor='black')
    plt.title("Розподіл сум транзакцій")
    plt.xlabel("Сума")
    plt.ylabel("Кількість")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_time_distribution():  2 usages
    df = get_transaction_data()
    df['hour'] = pd.to_datetime(df['timestamp']).dt.hour
    plt.figure(figsize=(8, 5))
    df['hour'].value_counts().sort_index().plot(kind='bar')
    plt.title("Транзакції за годинами доби")
    plt.xlabel("Година")
    plt.ylabel("Кількість транзакцій")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

Рис. 3.12 — Функція побудови графіків

6. Генерація звіту

Користувач натискає кнопку "Експорт у PDF". Система передає підозрілі транзакції до модуля pdf_export.py, де на основі шаблону формується звіт з заголовками, таблицею, зведенням, датою. Файл зберігається у папці reports/ і автоматично відкривається після завершення.

Взаємодія: gui.py та pdf_export.py + database.py.

```python
def generate_pdf_report(output_path):  2 usages
    font_path = str(Path("assets/DejaVuSans.ttf").resolve())

    pdfmetrics.registerFont(TTFont( name: "DejaVu", font_path))

    conn = sqlite3.connect("data/results.sqlite")
    cursor = conn.cursor()
    cursor.execute("""
        SELECT suspicious.transaction_id, suspicious.reason,
               transactions.sender, transactions.receiver,
               transactions.amount, transactions.timestamp
        FROM suspicious
        JOIN transactions ON suspicious.transaction_id = transactions.transaction_id
    """)
    rows = cursor.fetchall()
    conn.close()

    c = canvas.Canvas(output_path, pagesize=A4)
    width, height = A4
    c.setFont( psfontname: "DejaVu",  size: 12)
    c.drawString( x: 50, height - 50,  text: "Звіт про підозрілі транзакції")
    y = height - 80

    headers = ["Transaction ID", "Причина", "Відправник", "Одержувач", "Сума", "Час"]
    c.drawString( x: 50, y, " | ".join(headers))
    y -= 20

    for row in rows:
        row_str = " | ".join(str(cell) for cell in row)
        c.drawString( x: 50, y, row_str[:150])  # truncate long rows
        y -= 20
```

Рис. 3.13 — Функція генерації .pdf звіту

7. Очищення бази

При натисканні кнопки "Очистити базу" викликається функція clear_database() з модуля database.py, яка видаляє всі записи з таблиці transactions. Інтерфейс автоматично оновлюється, і таблиця стає порожньою.

```
def clear_database():  2 usages
    conn = sqlite3.connect("data/results.sqlite")
    cursor = conn.cursor()
    cursor.execute("DELETE FROM suspicious")
    cursor.execute("DELETE FROM transactions")
    conn.commit()
    conn.close()
```

Рис. 3.14 — Функція очистки бази даних

8. Логування подій

На кожному з кроків система викликає функцію write_log() з модуля logger.py, яка додає запис у лог-файл.

Кожен лог включає:

1. Тип дії
2. Дату та час події
3. Додаткову інформацію

Взаємодія: gui.py та logger.py.

```
def write_log(message: str):  7 usages
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open(LOG_FILE, "a", encoding="utf-8") as f:
        f.write(f"[{now}] {message}\n")
```

Рис. 3.15 — Функція логування подій

Завдяки модульній структурі система працює послідовно, прозоро та стабільно. Кожен етап виконується незалежно, а компоненти взаємодіють через чітко визначені функції та дані. Це дозволяє ефективно обробляти великі масиви транзакцій, виявляти аномалії та формувати звіти без участі спеціаліста-програміста.

3.4. Інтерфейс користувача

Графічний інтерфейс користувача (GUI) — це ключовий елемент системи, що забезпечує зручну взаємодію користувача з усіма функціями програми без потреби звертатися до командного рядка чи редагування коду.

Інтерфейс розроблено за допомогою бібліотеки PyQt5, яка дозволяє створювати сучасні, інтерактивні та кросплатформенні віконні застосунки.

Інтерфейс є інтуїтивно зрозумілим і структурованим у вигляді одного головного вікна, яке містить набір функціональних елементів.

Головні компоненти інтерфейсу

1. **Головне меню / кнопки:**
   - «Завантажити CSV» – відкриває діалогове вікно для вибору файлу транзакцій
   - «Проаналізувати» – запускає евристичну перевірку всіх записів
   - «Побудувати графіки» – виводить діаграми активності транзакцій
   - «Експортувати у PDF» – генерує звіт про підозрілі операції
   - «Експортувати в Excel» – формує таблицю результатів у форматі .xlsx
   - «Очистити базу» – видаляє всі дані з локальної бази
   - «Вихід» – закриває програму
2. **Таблиця транзакцій (QTableWidget):**
   - Відображає всі записи, імпортовані з CSV або витягнуті з бази
   - Колонки: дата, час, сума, відправник, отримувач, підозрілість, причина
   - Підозрілі транзакції автоматично виділяються
   - Підтримується горизонтальне та вертикальне прокручування
3. **Системні повідомлення:**
   - Після кожної дії (імпорт, аналіз, звіт) з’являється діалогове вікно з результатом операції
   - Повідомлення допомагають користувачу уникати помилок
4. **Вікна підтвердження:**
   - Перед очищенням бази чи виходом з програми система запитує підтвердження дій

Принципи UX/UI

Інтерфейс реалізовано з урахуванням таких принципів:

- Мінімалізм: усе необхідне розміщено в одному вікні, без перевантаження зайвими елементами
- Інтуїтивність: назви кнопок чітко відповідають діям, які вони виконують
- Зворотний зв’язок: після кожної дії користувач отримує підтвердження або повідомлення про помилку
- Кольорові підказки: підозрілі транзакції підсвічуються, що дозволяє миттєво оцінити ситуацію

Переваги реалізації через PyQt5

- Гнучкість компонування: можна легко додавати нові елементи без переробки всього вікна
- Кросплатформеність: програма працює як у Windows, так і в Linux чи macOS
- Розширюваність: у разі потреби можна реалізувати вкладки, фільтри, сортування або темну тему

Інтерфейс користувача — це міст між користувачем та функціональністю системи. Завдяки PyQt5, візуальна частина програми є зручною, функціонально повною та орієнтованою на практичне використання. Це особливо важливо для користувачів без технічної освіти, які можуть з легкістю завантажити дані, провести аналіз і сформувати звіт без залучення спеціаліста.

РОЗДІЛ 4. ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

Після завершення реалізації програмного продукту критично важливою стадією життєвого циклу є тестування, що дозволяє переконатися у працездатності, стабільності, зручності та надійності інформаційної системи. Метою даного етапу є виявлення помилок або недоліків, які могли бути допущені на попередніх стадіях проєктування та програмування, а також об'єктивна оцінка якості та ефективності реалізованого функціоналу в умовах, максимально наближених до реального використання.

Тестування проводилося за допомогою спеціально підготовлених тестових сценаріїв, з використанням ручних маніпуляцій в інтерфейсі програми, а також з аналізом логів та звітів, що формуються системою.

4.1. Методика тестування

Процес тестування передбачав перевірку кожної ключової функціональної частини програми відповідно до її призначення. Застосовувалась методика чорного ящика, коли тестувальник взаємодіє з системою як кінцевий користувач, не вдаючись у внутрішню реалізацію коду. Такий підхід особливо корисний у контексті GUI-застосунків.

Тестові дані були підготовлені у вигляді CSV-файлів, які моделювали:

- Звичайні транзакції — без ознак аномалій;
- Підозрілі операції — з навмисно заданими аномаліями (великі суми, нічний час, перекази самому собі);
- Помилкові файли — з пропущеними полями або неправильним форматом.

Також було протестовано програму на великих обсягах даних (понад 1000 транзакцій), щоб перевірити стабільність і продуктивність.

4.2. Функціональне тестування

Функціональне тестування охопило перевірку кожного модуля на відповідність очікуваній поведінці:

- **Тестування головного меню**

Завантаження системи відбувається швидко, всі функції працюють коректно.

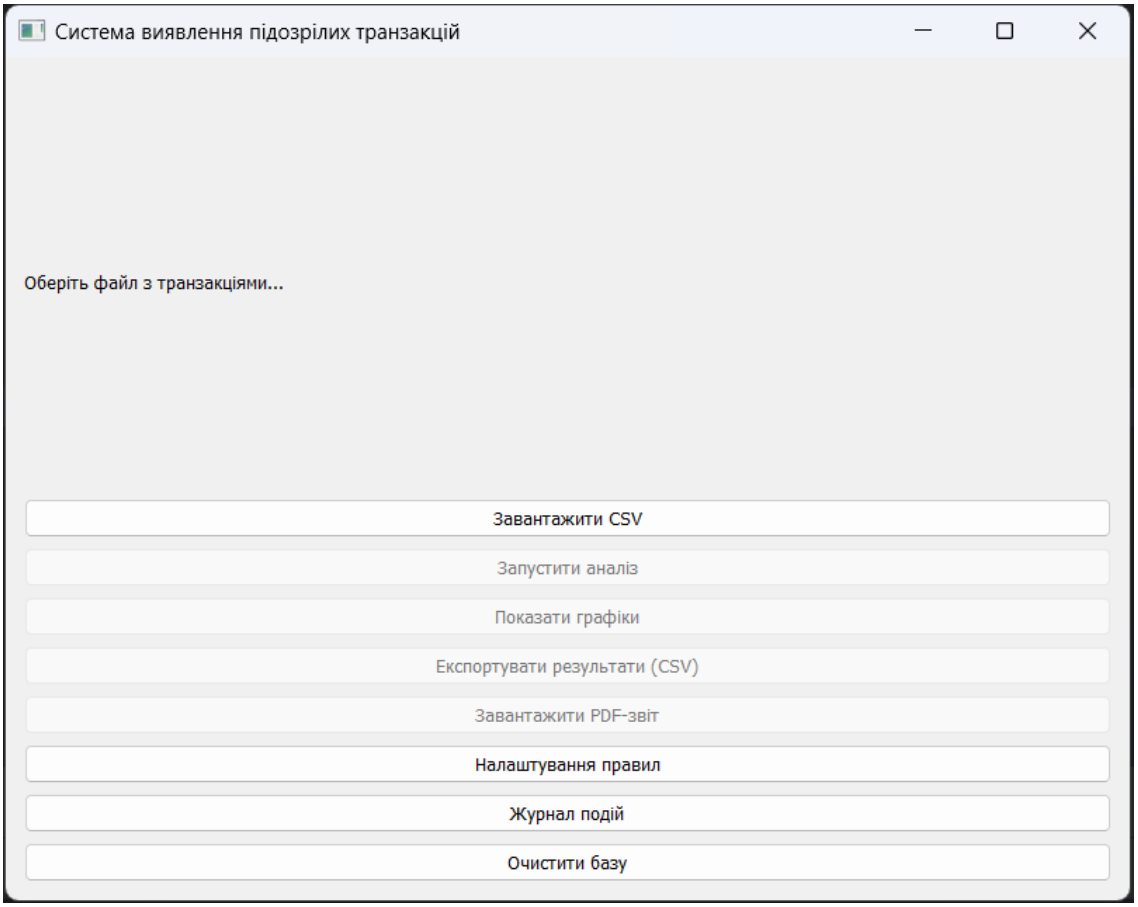

Рис. 4.1 — Головне меню

- **Завантаження тестового файлу з транзакціями**

Файл швидко завантажився та знаходиться в системі

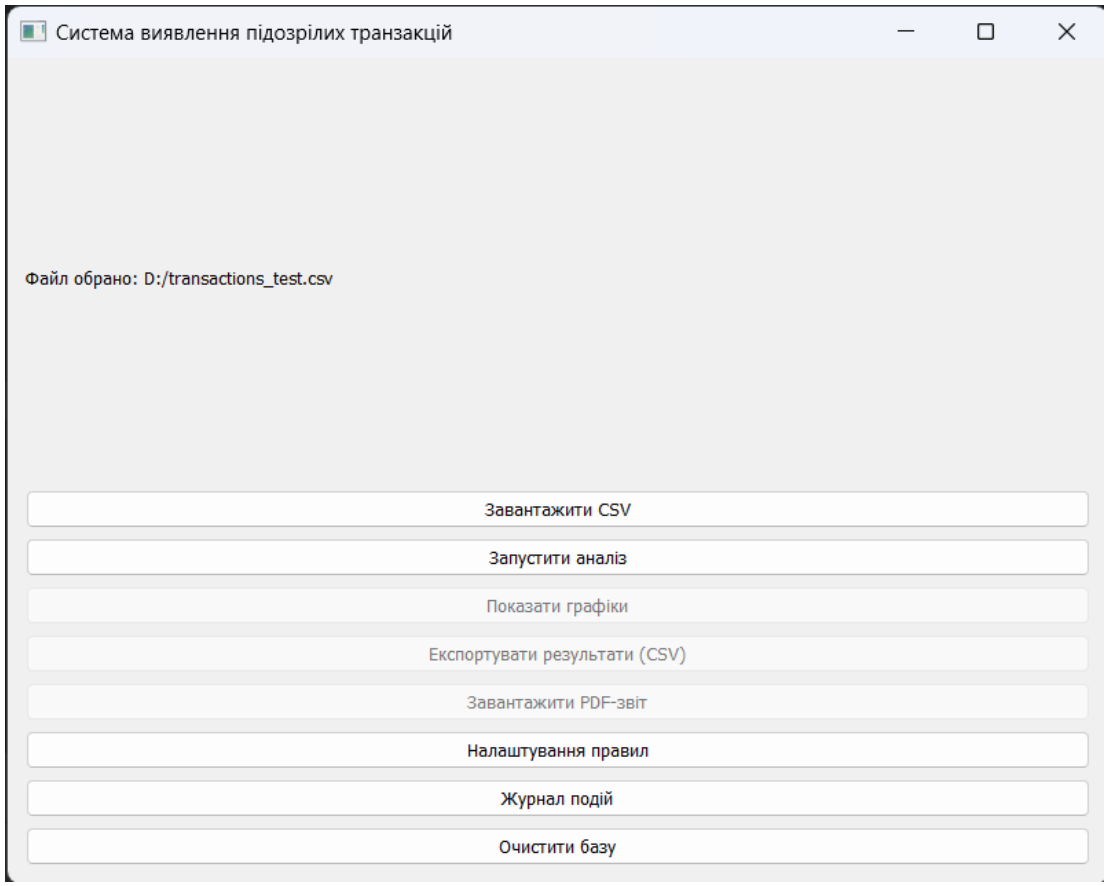

Рис. 4.2 — Завантаження файлу транзакцій

- **Запуск аналізу файлу**

Система швидко просканувала файл та виявила підозрілі транзакції

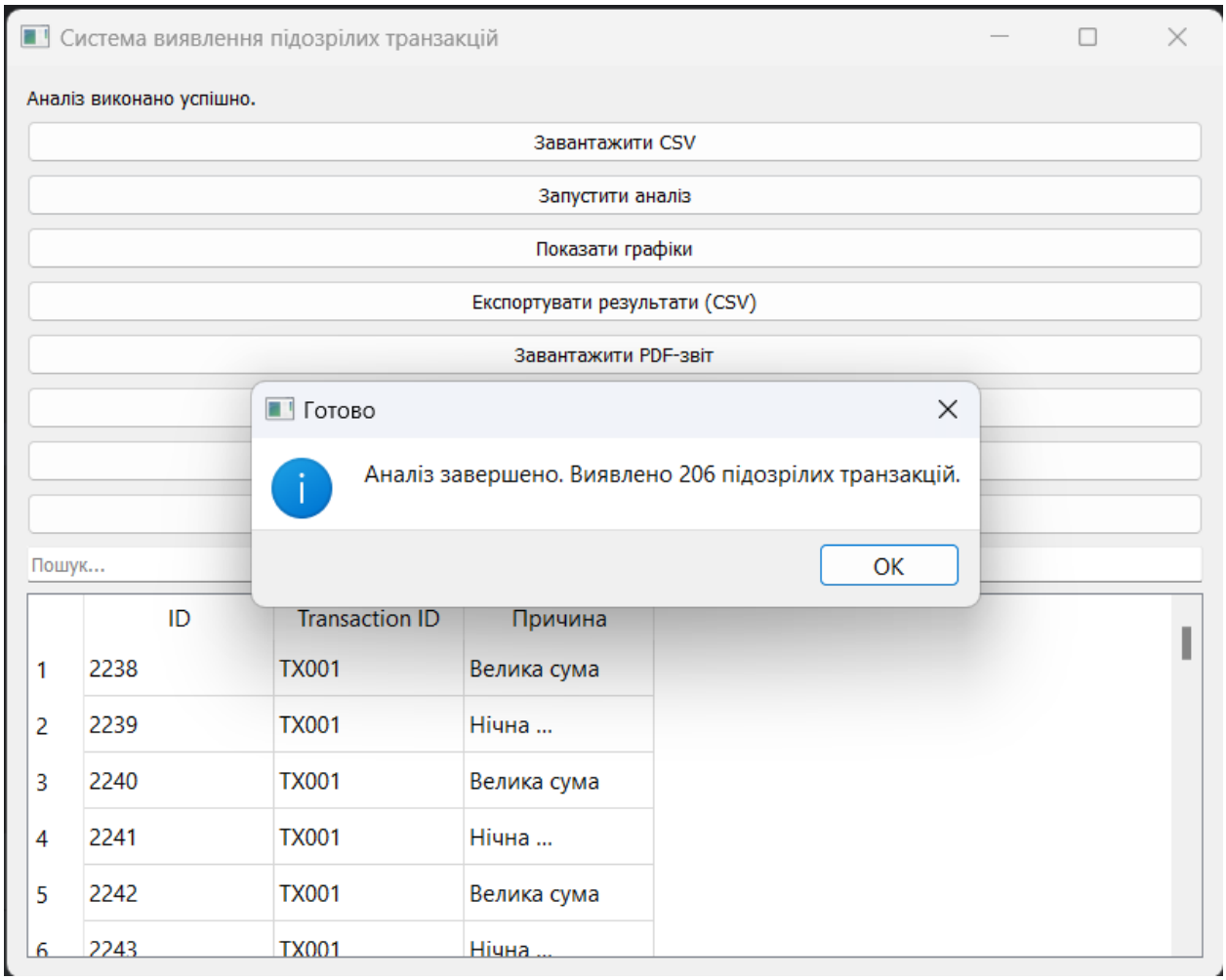

Рис. 4.3 — Запуск аналізу та результат

- **Перегляд звіту у форматі графіку**

Графік вивів детальну та достовірну інформацію

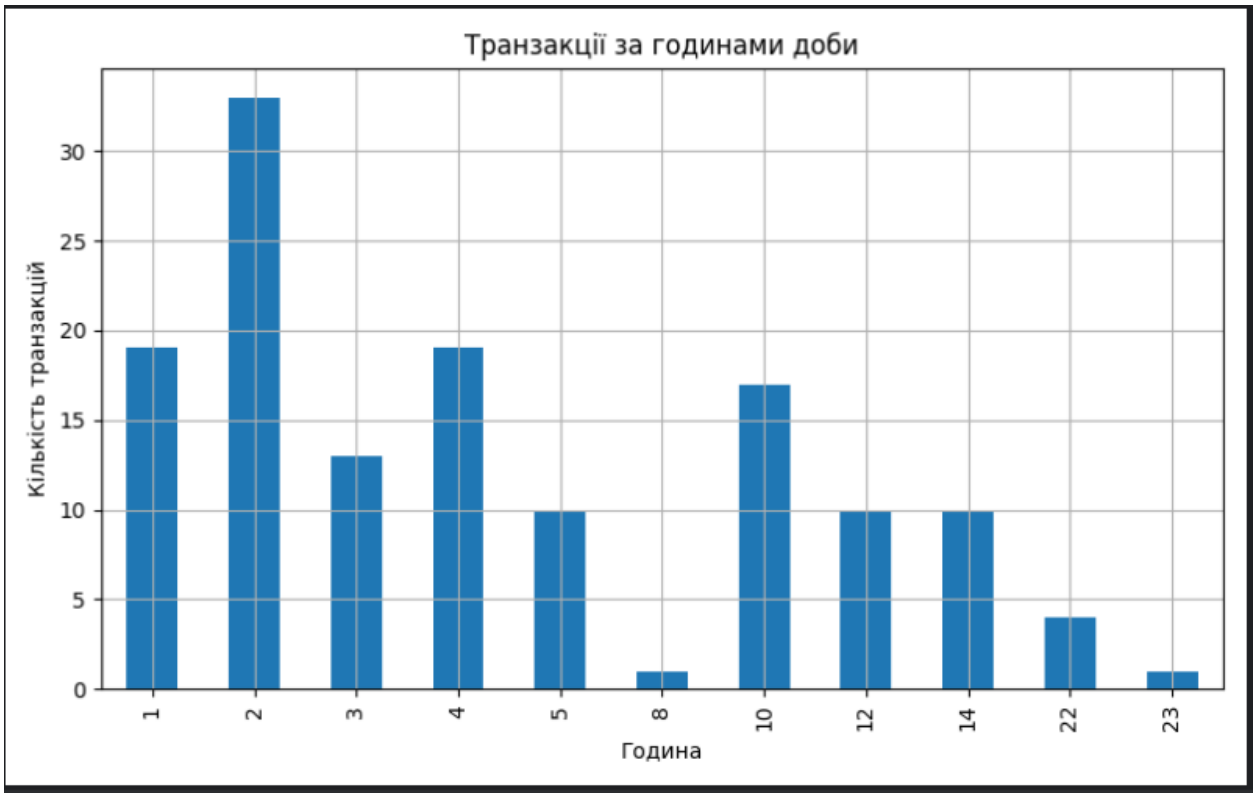

Рис. 4.4 — Графік погодинних транзакцій

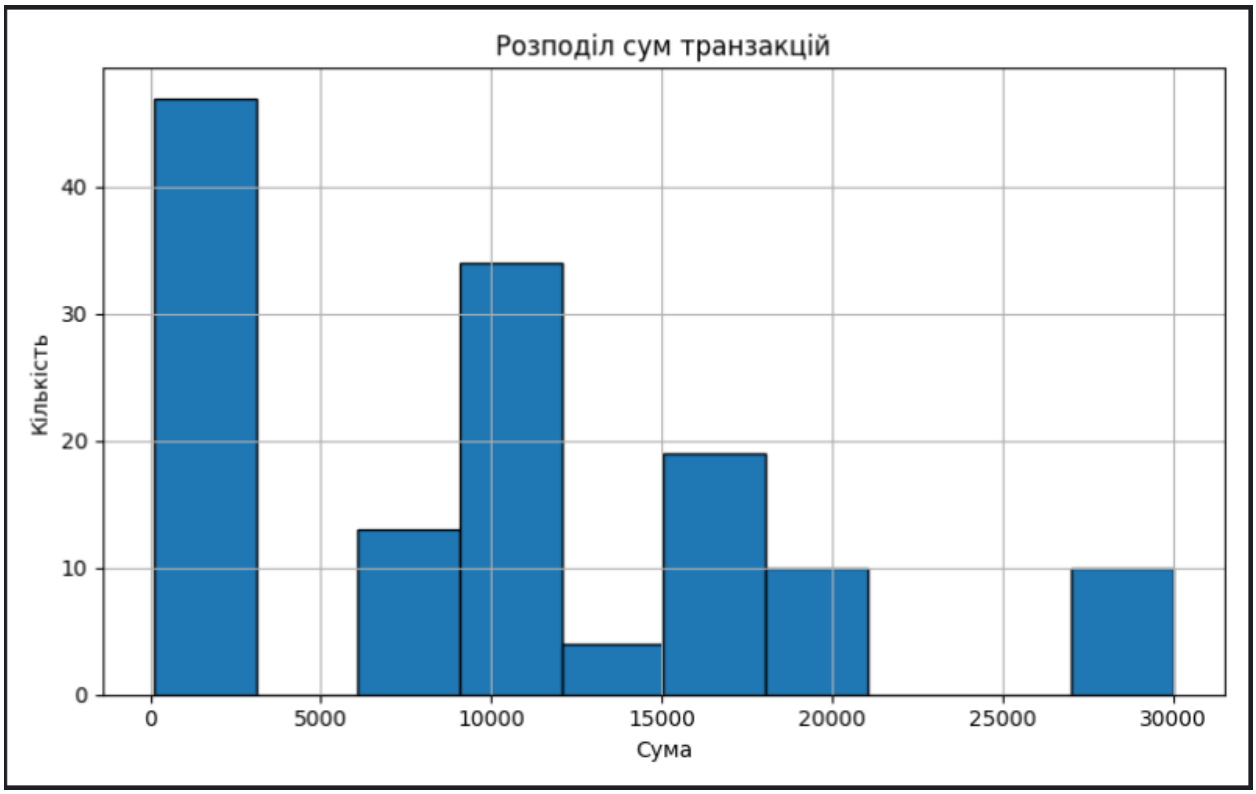

Рис. 4.5 — Графік розподілу сум транзакцій

- **Збереження звіту у форматі .csv**

Файл збережено у відповідній папці та коректно відкривається

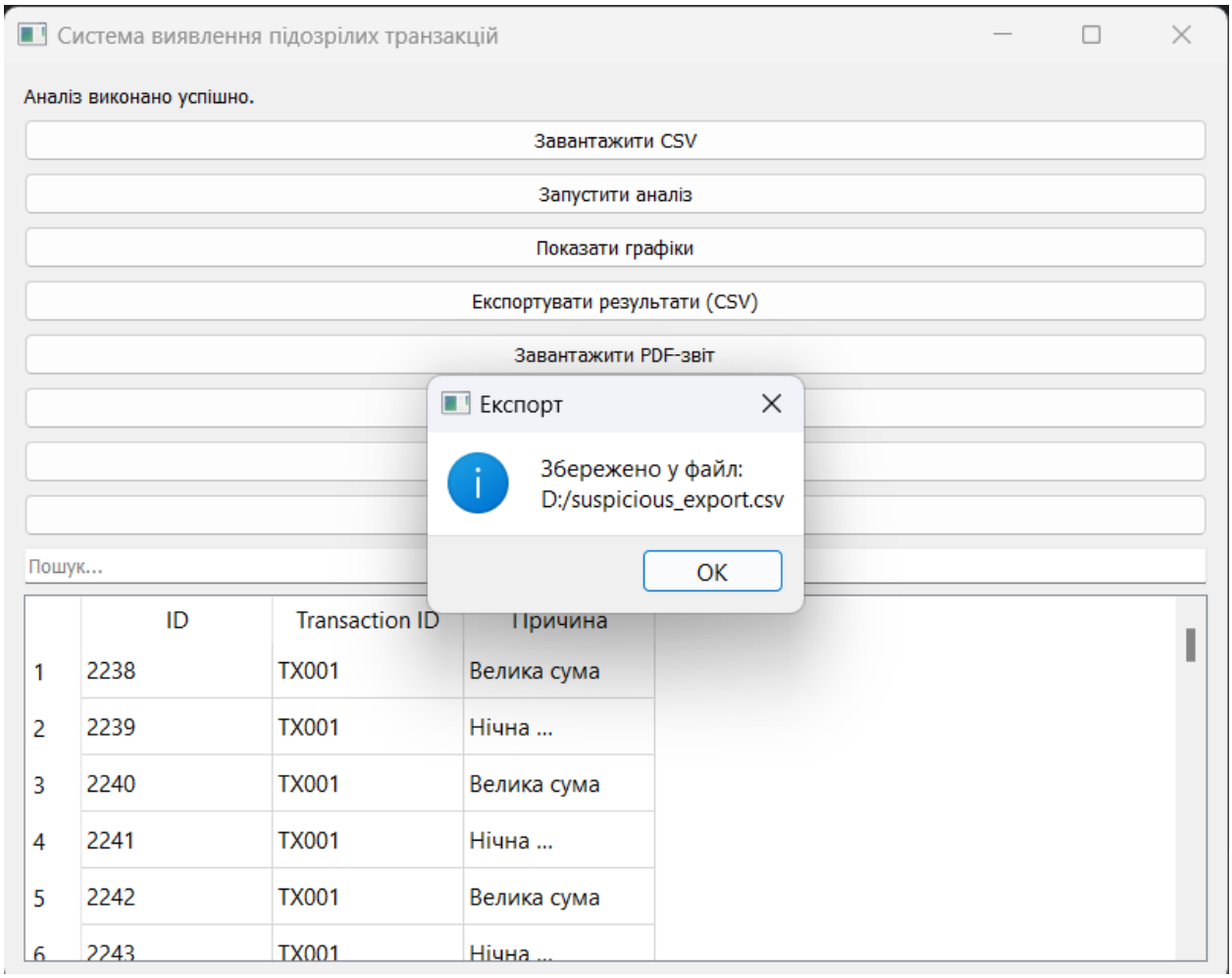

Рис. 4.6 — Збереження результатів у форматі .csv

- **Збереження звіту у форматі .pdf**

Файл збережено у відповідній папці та коректно відкривається

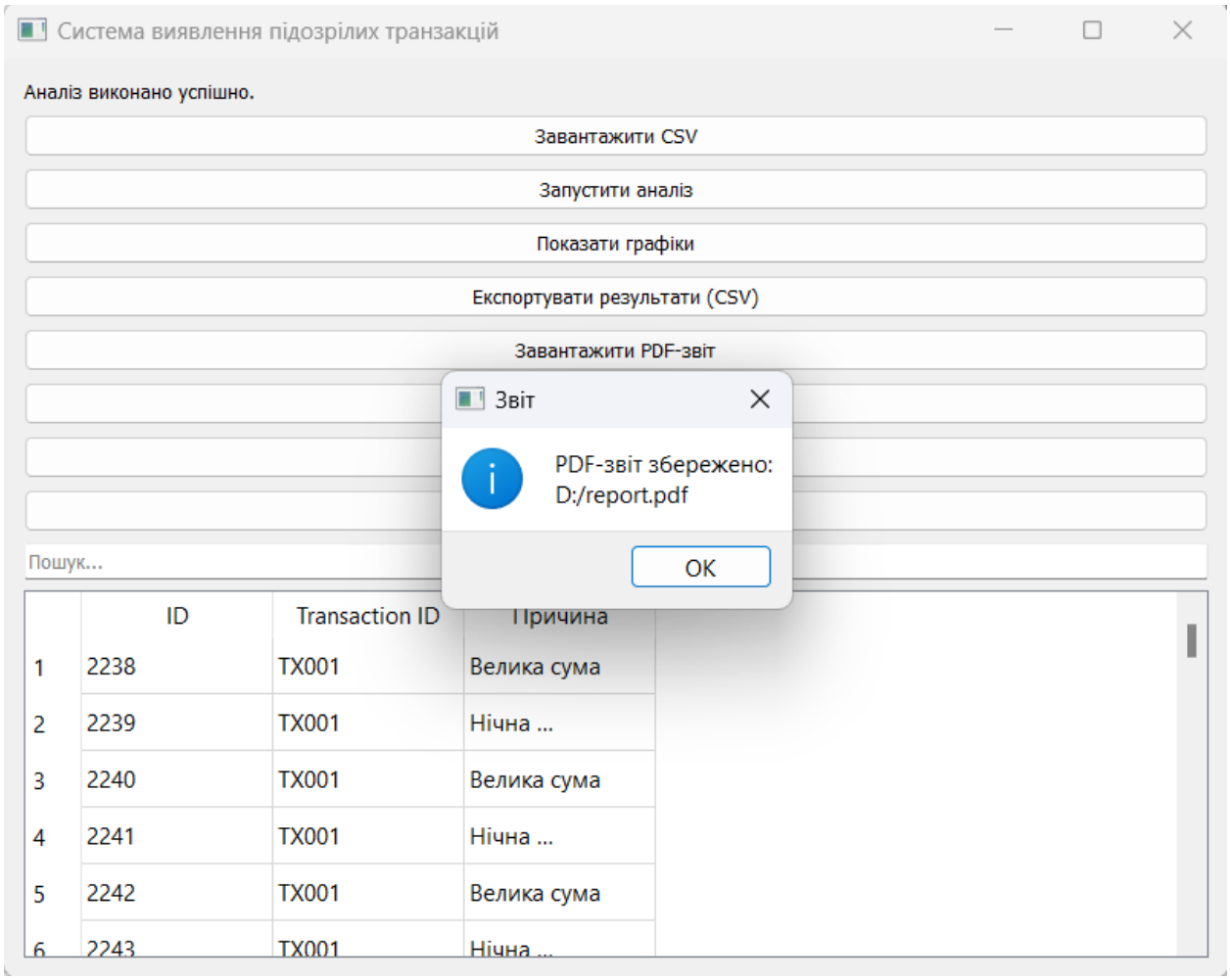

Рис. 4.7 — Збереження результатів у форматі .pdf

- **Тестування кастомних правил пошуку**

Зміни успішно прийняті та застосовані

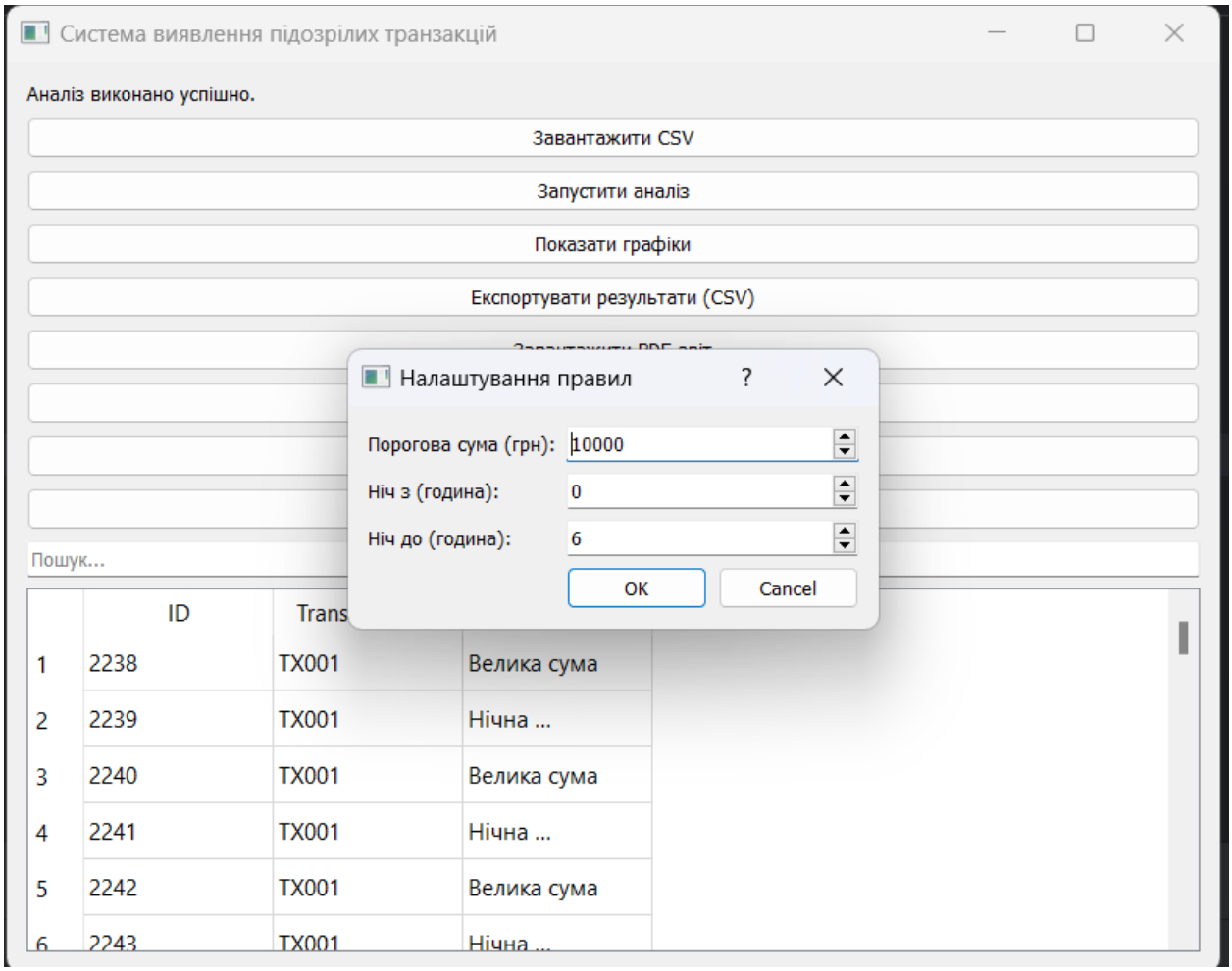

Рис. 4.8 — Налаштування кастомних правил аналізу

- **Перевірка журналу логів**

Логи успішно зберігаються та показують детальні події

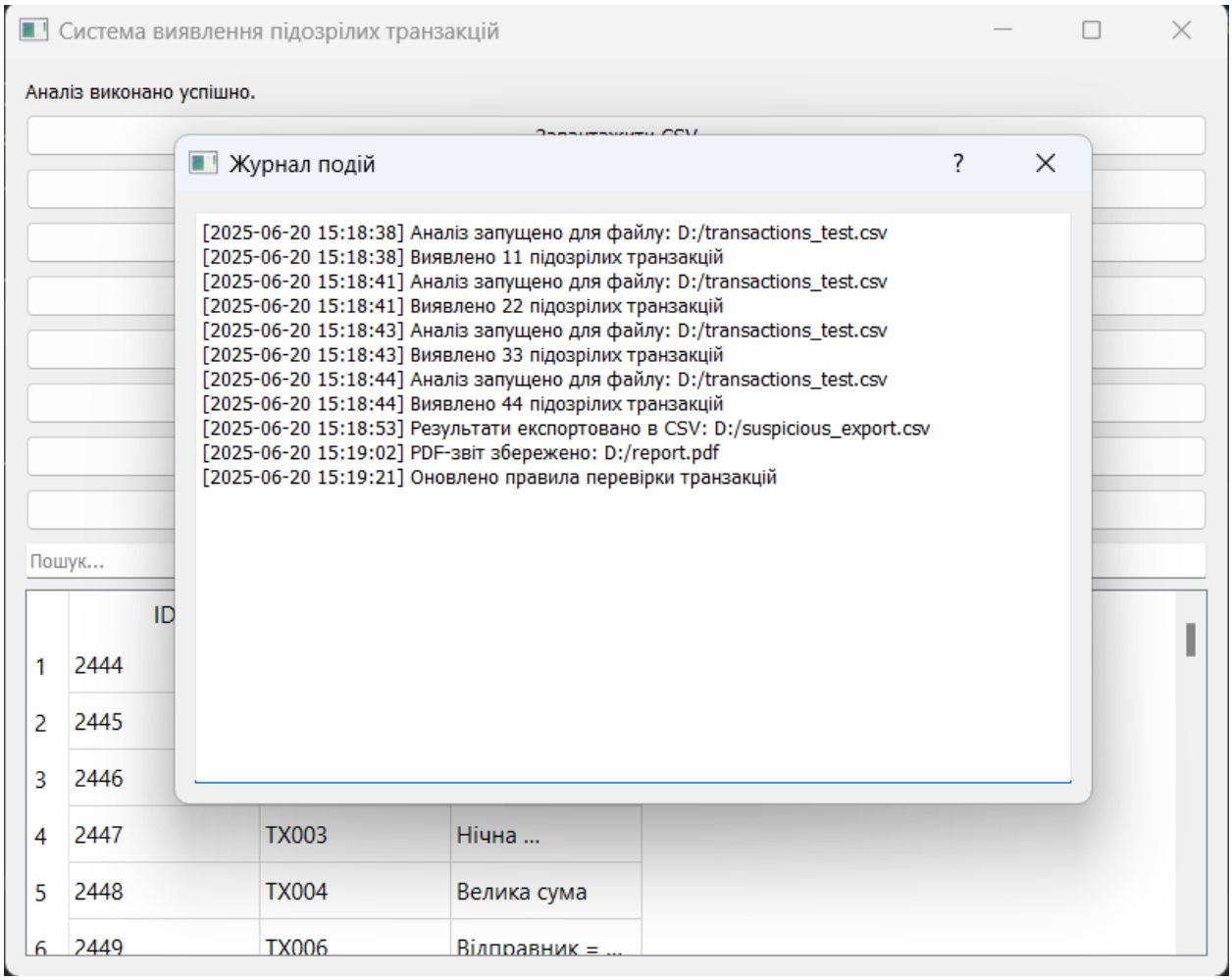

Рис. 4.9 — Журнал логування

- **Тестування очищення бази**

База успішно очищена

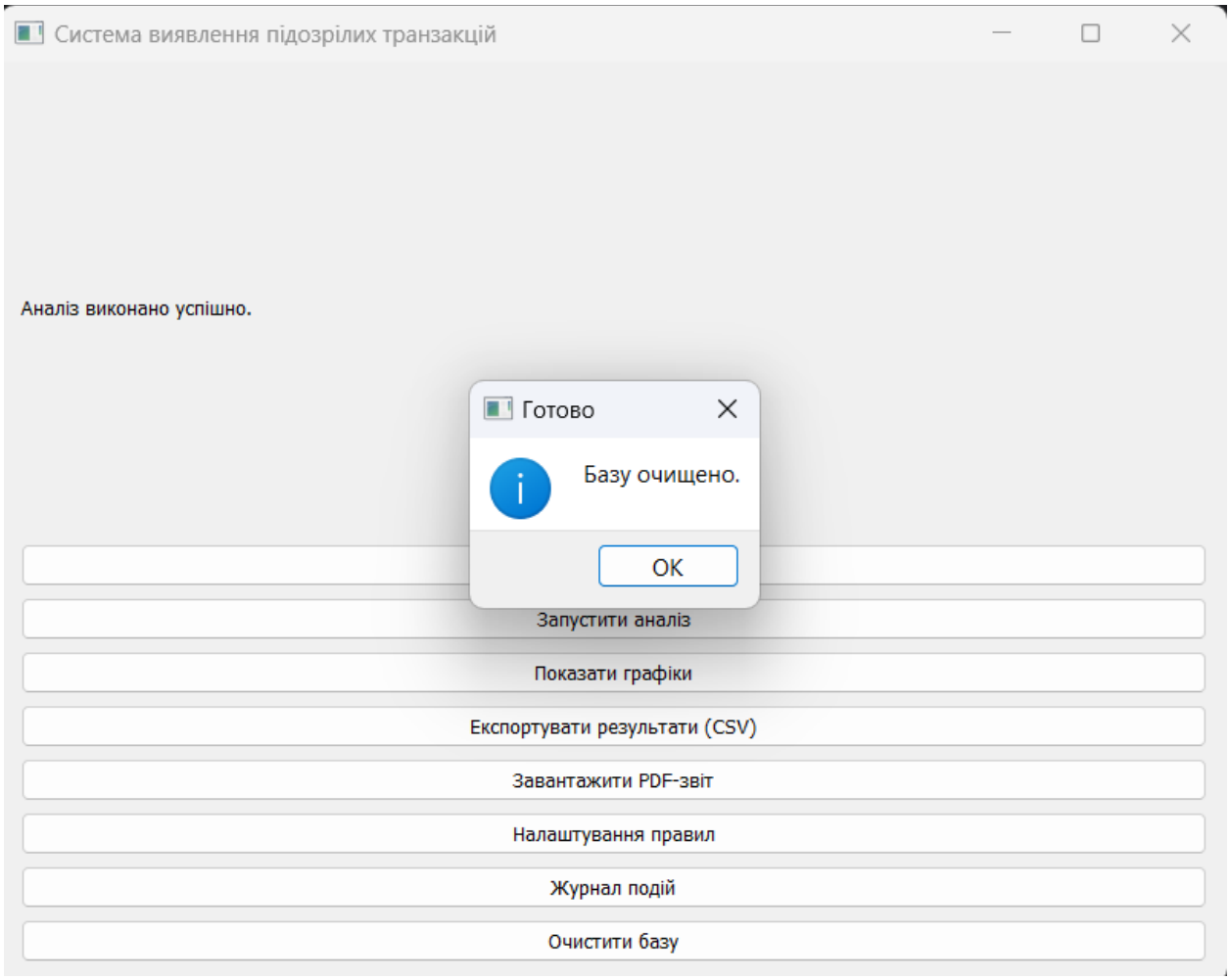

Рис. 4.10 — Очищення бази даних

4.3. Оцінка ефективності

Результати функціонального тестування дали змогу зробити наступні висновки:

Критерій	Оцінка
Стабільність	Не зафіксовано збоїв або помилок виконання
Швидкодія	Імпорт і аналіз до 1000 записів — < 1 сек
Якість аналізу	Висока точність виявлення аномалій
Зручність інтерфейсу	Інтуїтивне розташування елементів
Звітність	Створюється чітка документація для перегляду
Логування	Кожна дія користувача зберігається з датою

Таб 4.1 — Оцінка ефективності

Таким чином, програмна система успішно пройшла всі заплановані тести, продемонструвавши свою придатність до використання у реальних умовах фінансового моніторингу.

ВИСНОВОК

В результаті виконання дипломної роботи було розроблено, реалізовано та протестовано програмне забезпечення, що дозволяє виявляти аномальні транзакції на основі евристичного аналізу.

У процесі дослідження були досягнуті такі основні результати:

1. Проведено аналіз предметної області, розглянуто типові фінансові схеми, пов’язані з шахрайством, а також сформульовано критерії, за якими можна визначати підозрілі транзакції.
2. Спроєктовано структуру інформаційної системи, визначено функціональні вимоги, обрано модульну архітектуру, що забезпечує гнучкість і масштабованість рішення.
3. Реалізовано повноцінний десктопний застосунок на мові Python із використанням таких технологій, як PyQt5, SQLite, Pandas, Matplotlib, ReportLab.
4. Забезпечено зручний графічний інтерфейс, що дозволяє завантажувати дані, запускати аналіз, формувати звіти, візуалізувати активність транзакцій та зберігати лог дій користувача.
5. Проведено тестування системи, що підтвердило її функціональність, стабільність, швидкодію та готовність до використання в умовах малого або середнього бізнесу.

Практична цінність реалізованої системи полягає в її простоті, автономності (можливість роботи без доступу до Інтернету), а також у потенціалі адаптації під конкретні потреби користувачів або компаній, що мають справу з обробкою великої кількості фінансових операцій.

Отже, розроблена інформаційна система повністю відповідає поставленій меті та завданням дипломної роботи. Вона може бути використана як основа для подальшої розробки комерційних або корпоративних систем фінансового моніторингу, а також як навчальний інструмент у підготовці спеціалістів з інформаційної безпеки та аналізу транзакцій.

СПИСОК ЛІТЕРАТУРИ

1. Закон України «Про бухгалтерський облік та фінансову звітність в Україні» від 16.07.1999 № 996-XIV. – Режим доступу: https://zakon.rada.gov.ua/laws/show/996-14
2. Національний банк України. Положення про здійснення фінансового моніторингу. – Режим доступу: https://bank.gov.ua/
3. Міністерство фінансів України. Методичні рекомендації щодо внутрішнього аудиту. – Режим доступу: https://mof.gov.ua
4. Rouse, M. (2020). What is fraud detection? – TechTarget. – Режим доступу: https://www.techtarget.com/searchsecurity/definition/fraud-detection
5. Financial Action Task Force (FATF). (2023). Guidance for Risk-Based Approach to Virtual Assets and VASPs. – Режим доступу: https://www.fatf-gafi.org/
6. James, G., Witten, D., Hastie, T., Tibshirani, R. (2021). An Introduction to Statistical Learning. Springer.
7. Aggarwal, C.C. (2017). Outlier Analysis. Springer.
8. Provost, F., Fawcett, T. (2013). Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. O’Reilly Media.
9. Bishop, C. (2006). Pattern Recognition and Machine Learning. Springer.
10. Anderson, R.J. (2020). Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley.
11. Pandas Documentation. (2024). pandas.pydata.org. – Режим доступу: https://pandas.pydata.org/docs/
12. PyQt5 Documentation. (2023). pyqt.readthedocs.io. – Режим доступу: https://www.riverbankcomputing.com/static/Docs/PyQt5/
13. ReportLab User Guide. (2024). – Режим доступу: https://www.reportlab.com/docs/reportlab-userguide.pdf
14. Matplotlib Documentation. (2024). – Режим доступу: https://matplotlib.org/stable/contents.html

15. Python Software Foundation. (2024). Official Python Documentation. – Режим доступу: https://docs.python.org/3/
16. Kravets, R., Sokol, I. (2022). Реалізація евристичних алгоритмів для виявлення аномальних транзакцій. // Інформаційні технології і системи. – № 1. – С. 15–21.
17. Степаненко, О.М., Яковенко, І.В. (2021). Програмне забезпечення для моніторингу фінансових потоків. // Вісник НТУУ «КПІ». Серія: Інформатика та обчислювальна техніка. – № 3. – С. 45–52.
18. Шевченко, Т.О. (2023). Аналіз сучасних засобів виявлення фінансового шахрайства. // Фінанси, облік і аудит. – № 4. – С. 33–39.
19. European Central Bank. (2022). Guide on internal control frameworks. – Режим доступу: https://www.ecb.europa.eu/

ДОДАТОК А

```
from gui import launch_app
from database import create_tables

if __name__ == '__main__':
    create_tables()
    launch_app()

from datetime import datetime
from pathlib import Path

LOG_FILE = Path("data/logs.txt")

def write_log(message: str):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open(LOG_FILE, "a", encoding="utf-8") as f:
        f.write(f"[{now}] {message}\n")

def read_logs() -> str:
    if LOG_FILE.exists():
        return LOG_FILE.read_text(encoding="utf-8")
    return "Лог-файл порожній."
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from database import get_all_suspicious
import sqlite3
import os
from pathlib import Path
```

```
def generate_pdf_report(output_path):
    font_path = str(Path("assets/DejaVuSans.ttf").resolve())

    pdfmetrics.registerFont(TTFont("DejaVu", font_path))

    conn = sqlite3.connect("data/results.sqlite")
    cursor = conn.cursor()
    cursor.execute("""
        SELECT suspicious.transaction_id, suspicious.reason,
               transactions.sender, transactions.receiver,
               transactions.amount, transactions.timestamp
        FROM suspicious
        JOIN transactions ON suspicious.transaction_id = transactions.transaction_id
    """)
    rows = cursor.fetchall()
    conn.close()

    c = canvas.Canvas(output_path, pagesize=A4)
    width, height = A4
    c.setFont("DejaVu", 12)
    c.drawString(50, height - 50, "Звіт про підозрілі транзакції")
    y = height - 80

    headers = ["Transaction ID", "Причина", "Відправник", "Одержувач",
"Сума", "Час"]
    c.drawString(50, y, " | ".join(headers))
    y -= 20
```

```
    for row in rows:
        row_str = " | ".join(str(cell) for cell in row)
        c.drawString(50, y, row_str[:150])  # truncate long rows
        y -= 20
        if y < 50:
            c.showPage()
            c.setFont("DejaVu", 12)
            y = height - 50

    c.save()

# rules.py

# Стандартні значення
SUSPICIOUS_AMOUNT_THRESHOLD = 10000
NIGHT_HOURS_START = 0
NIGHT_HOURS_END = 6

def update_rules(amount_threshold, night_start, night_end):
    global SUSPICIOUS_AMOUNT_THRESHOLD, NIGHT_HOURS_START,
NIGHT_HOURS_END
    SUSPICIOUS_AMOUNT_THRESHOLD = amount_threshold
    NIGHT_HOURS_START = night_start
    NIGHT_HOURS_END = night_end

import sys
from PyQt5.QtWidgets import (
    QApplication, QWidget, QPushButton, QLabel, QVBoxLayout,
    QFileDialog, QMessageBox, QTableWidget, QTableWidgetItem,
    QLineEdit, QDialog, QFormLayout, QSpinBox, QDialogButtonBox,
```

```
    QTextEdit
)
from analysis import load_transactions_from_csv, analyze_transactions
from database import (
    create_tables, get_all_suspicious, export_suspicious_to_csv, clear_database
)
from report import generate_pdf_report
from rules import (
    SUSPICIOUS_AMOUNT_THRESHOLD, NIGHT_HOURS_START,
NIGHT_HOURS_END, update_rules
)
from logger import write_log, read_logs

class FraudDetectionApp(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Система виявлення підозрілих транзакцій")
        self.setGeometry(100, 100, 800, 600)
        self.init_ui()

    def init_ui(self):
        self.layout = QVBoxLayout()

        self.label_status = QLabel("Оберіть файл з транзакціями...")
        self.layout.addWidget(self.label_status)

        self.btn_load = QPushButton("Завантажити CSV")
        self.btn_load.clicked.connect(self.load_file)
        self.layout.addWidget(self.btn_load)
```

```
self.btn_analyze = QPushButton("Запустити аналіз")
self.btn_analyze.clicked.connect(self.run_analysis)
self.btn_analyze.setEnabled(False)
self.layout.addWidget(self.btn_analyze)

self.btn_plot = QPushButton("Показати графіки")
self.btn_plot.clicked.connect(self.show_graphs)
self.btn_plot.setEnabled(False)
self.layout.addWidget(self.btn_plot)

self.btn_export = QPushButton("Експортувати результати (CSV)")
self.btn_export.clicked.connect(self.export_results)
self.btn_export.setEnabled(False)
self.layout.addWidget(self.btn_export)

self.btn_pdf = QPushButton("Завантажити PDF-звіт")
self.btn_pdf.clicked.connect(self.export_pdf)
self.btn_pdf.setEnabled(False)
self.layout.addWidget(self.btn_pdf)

self.btn_settings = QPushButton("Налаштування правил")
self.btn_settings.clicked.connect(self.open_settings)
self.layout.addWidget(self.btn_settings)

self.btn_log = QPushButton("Журнал подій")
self.btn_log.clicked.connect(self.show_log)
self.layout.addWidget(self.btn_log)

self.btn_clear = QPushButton("Очистити базу")
self.btn_clear.clicked.connect(self.clear_db)
```

```
        self.layout.addWidget(self.btn_clear)

        self.search_box = QLineEdit()
        self.search_box.setPlaceholderText("Пошук...")
        self.search_box.textChanged.connect(self.filter_table)
        self.search_box.hide()
        self.layout.addWidget(self.search_box)

        self.table = QTableWidget()
        self.table.setColumnCount(3)
        self.table.setHorizontalHeaderLabels(['ID', 'Transaction ID', 'Причина'])
        self.table.hide()
        self.layout.addWidget(self.table)

        self.setLayout(self.layout)

    def load_file(self):
        file_path, _ = QFileDialog.getOpenFileName(self, "Оберіть CSV-файл", "",
"CSV files (*.csv)")
        if file_path:
            self.file_path = file_path
            self.label_status.setText(f"Файл обрано: {file_path}")
            self.btn_analyze.setEnabled(True)

    def run_analysis(self):
        try:
            df = load_transactions_from_csv(self.file_path)
            if df.empty:
                QMessageBox.warning(self, "Помилка", "Файл порожній або
некоректний.")
```

```
                return
            analyze_transactions(df)
            suspicious = get_all_suspicious()
            self.show_table(suspicious)
            self.label_status.setText("Аналіз виконано успішно.")
            self.btn_plot.setEnabled(True)
            self.btn_export.setEnabled(True)
            self.btn_pdf.setEnabled(True)
            write_log(f"Аналіз запущено для файлу: {self.file_path}")
            write_log(f"Виявлено {len(suspicious)} підозрілих транзакцій")
            QMessageBox.information(self, "Готово", f"Аналіз завершено.
Виявлено {len(suspicious)} підозрілих транзакцій.")
        except Exception as e:
            QMessageBox.critical(self, "Помилка", f"Щось пішло не так:\n{e}")

    def show_table(self, data):
        self.original_data = data
        self.table.setRowCount(len(data))
        for row_idx, row_data in enumerate(data):
            for col_idx, value in enumerate(row_data):
                self.table.setItem(row_idx, col_idx, QTableWidgetItem(str(value)))
        self.table.show()
        self.search_box.show()

    def filter_table(self, text):
        text = text.lower()
        filtered_data = [
            row for row in self.original_data
            if any(text in str(cell).lower() for cell in row)
        ]
```

```
        self.table.setRowCount(len(filtered_data))
        for row_idx, row_data in enumerate(filtered_data):
            for col_idx, value in enumerate(row_data):
                self.table.setItem(row_idx, col_idx, QTableWidgetItem(str(value)))

    def show_graphs(self):
        from visualizations import plot_amount_distribution, plot_time_distribution
        plot_amount_distribution()
        plot_time_distribution()

    def export_results(self):
        output_path, _ = QFileDialog.getSaveFileName(self, "Зберегти результати",
"suspicious_export.csv", "CSV files (*.csv)")
        if output_path:
            try:
                export_suspicious_to_csv(output_path)
                write_log(f"Результати експортовано в CSV: {output_path}")
                QMessageBox.information(self, "Експорт", f"Збережено у
файл:\n{output_path}")
            except Exception as e:
                QMessageBox.critical(self, "Помилка", f"Не вдалося
експортувати:\n{e}")

    def export_pdf(self):
        output_path, _ = QFileDialog.getSaveFileName(self, "Зберегти звіт",
"report.pdf", "PDF files (*.pdf)")
        if output_path:
            try:
                generate_pdf_report(output_path)
                write_log(f"PDF-звіт збережено: {output_path}")
```

```
            QMessageBox.information(self, "Звіт", f"PDF-звіт збережено:\n{output_path}")
        except Exception as e:
            QMessageBox.critical(self, "Помилка", f"Не вдалося створити звіт:\n{e}")

    def clear_db(self):
        reply = QMessageBox.question(self, "Підтвердження", "Ви точно хочете очистити всі дані з бази?",
                                     QMessageBox.Yes | QMessageBox.No)
        if reply == QMessageBox.Yes:
            clear_database()
            write_log("Базу даних очищено вручну")
            self.table.setRowCount(0)
            self.table.hide()
            self.search_box.hide()
            QMessageBox.information(self, "Готово", "Базу очищено.")

    def open_settings(self):
        dialog = QDialog(self)
        dialog.setWindowTitle("Налаштування правил")
        form = QFormLayout(dialog)

        spin_amount = QSpinBox()
        spin_amount.setMaximum(1000000)
        spin_amount.setValue(SUSPICIOUS_AMOUNT_THRESHOLD)
        form.addRow("Порогова сума (грн):", spin_amount)

        spin_start = QSpinBox()
        spin_start.setRange(0, 23)
```

```
        spin_start.setValue(NIGHT_HOURS_START)
        form.addRow("Ніч з (година):", spin_start)

        spin_end = QSpinBox()
        spin_end.setRange(1, 24)
        spin_end.setValue(NIGHT_HOURS_END)
        form.addRow("Ніч до (година):", spin_end)

        buttons = QDialogButtonBox(QDialogButtonBox.Ok |
QDialogButtonBox.Cancel)
        buttons.accepted.connect(dialog.accept)
        buttons.rejected.connect(dialog.reject)
        form.addWidget(buttons)

        if dialog.exec_() == QDialog.Accepted:
            update_rules(spin_amount.value(), spin_start.value(), spin_end.value())
            write_log("Оновлено правила перевірки транзакцій")
            QMessageBox.information(self, "Застосовано", "Правила оновлено.")

    def show_log(self):
        dialog = QDialog(self)
        dialog.setWindowTitle("Журнал подій")
        layout = QVBoxLayout(dialog)

        text_area = QTextEdit()
        text_area.setReadOnly(True)
        text_area.setText(read_logs())

        layout.addWidget(text_area)
        dialog.setLayout(layout)
```

```
        dialog.resize(600, 400)
        dialog.exec_()

def launch_app():
    app = QApplication(sys.argv)
    create_tables()
    window = FraudDetectionApp()
    window.show()
    sys.exit(app.exec_())

import sqlite3
from pathlib import Path
import pandas as pd


DB_PATH = Path("data/results.sqlite")

def create_connection():
    conn = sqlite3.connect(DB_PATH)
    return conn

def create_tables():
    conn = create_connection()
    cursor = conn.cursor()

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS transactions (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            transaction_id TEXT,
            sender TEXT,
```

```
            receiver TEXT,
            amount REAL,
            timestamp TEXT
        )
    ''')

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS suspicious (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            transaction_id TEXT,
            reason TEXT
        )
    ''')

    conn.commit()
    conn.close()

def insert_transaction(tx):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO transactions (transaction_id, sender, receiver, amount,
timestamp)
        VALUES (?, ?, ?, ?, ?)
    ''', (tx['transaction_id'], tx['sender'], tx['receiver'], tx['amount'], tx['timestamp']))
    conn.commit()
    conn.close()

def insert_suspicious(tx_id, reason):
    conn = create_connection()
```

```
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO suspicious (transaction_id, reason)
        VALUES (?, ?)
    ''', (tx_id, reason))
    conn.commit()
    conn.close()

def get_all_suspicious():
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM suspicious')
    results = cursor.fetchall()
    conn.close()
    return results

def export_suspicious_to_csv(output_path):
    conn = sqlite3.connect("data/results.sqlite")
    query = """
        SELECT suspicious.id, suspicious.transaction_id, suspicious.reason,
               transactions.sender, transactions.receiver, transactions.amount,
transactions.timestamp
        FROM suspicious
        JOIN transactions ON suspicious.transaction_id = transactions.transaction_id
    """
    df = pd.read_sql_query(query, conn)
    conn.close()
    df.to_csv(output_path, index=False, encoding='utf-8-sig')

def clear_database():
```

```
    conn = sqlite3.connect("data/results.sqlite")
    cursor = conn.cursor()
    cursor.execute("DELETE FROM suspicious")
    cursor.execute("DELETE FROM transactions")
    conn.commit()
    conn.close()

import pandas as pd
from database import insert_transaction, insert_suspicious
from rules import SUSPICIOUS_AMOUNT_THRESHOLD,
NIGHT_HOURS_START, NIGHT_HOURS_END



SUSPICIOUS_AMOUNT_THRESHOLD = 10000
NIGHT_HOURS = range(NIGHT_HOURS_START, NIGHT_HOURS_END)

def load_transactions_from_csv(file_path: str) -> pd.DataFrame:
    df = pd.read_csv(file_path)
    df.columns = df.columns.str.lower()
    return df

def analyze_transactions(df: pd.DataFrame):
    for _, row in df.iterrows():
        tx = {
            'transaction_id': row['transaction_id'],
            'sender': row['sender'],
            'receiver': row['receiver'],
            'amount': float(row['amount']),
            'timestamp': row['timestamp']
```

```
        }

        # Збереження в БД
        insert_transaction(tx)

        # Аналіз
        suspicious_reasons = []

        if tx['amount'] > SUSPICIOUS_AMOUNT_THRESHOLD:
            suspicious_reasons.append('Велика сума')

        if tx['sender'] == tx['receiver']:
            suspicious_reasons.append('Відправник = Отримувач')

        try:
            hour = int(tx['timestamp'].split(' ')[1].split(':')[0])
            if hour in NIGHT_HOURS:
                suspicious_reasons.append('Нічна транзакція')
        except Exception as e:
            print(f"Помилка у timestamp: {tx['timestamp']} — {e}")

        for reason in suspicious_reasons:
            insert_suspicious(tx['transaction_id'], reason)

import pandas as pd
import matplotlib.pyplot as plt
import sqlite3

def get_transaction_data():
    conn = sqlite3.connect("data/results.sqlite")
```

```
    df = pd.read_sql_query("SELECT * FROM transactions", conn)
    conn.close()
    return df

def plot_amount_distribution():
    df = get_transaction_data()
    plt.figure(figsize=(8, 5))
    plt.hist(df['amount'], bins=10, edgecolor='black')
    plt.title("Розподіл сум транзакцій")
    plt.xlabel("Сума")
    plt.ylabel("Кількість")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

def plot_time_distribution():
    df = get_transaction_data()
    df['hour'] = pd.to_datetime(df['timestamp']).dt.hour
    plt.figure(figsize=(8, 5))
    df['hour'].value_counts().sort_index().plot(kind='bar')
    plt.title("Транзакції за годинами доби")
    plt.xlabel("Година")
    plt.ylabel("Кількість транзакцій")
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```