ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система автоматизації процесу обліку грошових витрат за допомогою програмного застосунку на основі Python»

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Євген Маламен
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Євген Маламен
група КНДМ-61

Керівник: Сєрих С.О.
науковий ступінь, вчене звання к.т.н.,доцент

Рецензент:
науковий ступінь, вчене звання
_____
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Маламену Євгену Миколайовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система автоматизації процесу обліку грошових витрат за допомогою програмного застосунку на основі Python»

керівник кваліфікаційної роботи Сєрих Сергій, к.т.н., доцент
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. №320

2. Строк подання кваліфікаційної роботи «15» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, документація мови програмування Python та її бібліотек.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   4.1. Огляд сучасних технологій та принципів для створення застосунку.
   4.2. Проектування та аналіз різних підходів для створення застосунку.
   4.3. Розробка застосунку для роботи з фінансами.
5. Перелік графічного матеріалу: *презентація*

5.1 Загальна характеристика дипломної роботи.
5.2 Актуальність теми.
5.3 Основні вимоги до застосунку.
5.4 Етапи створення
5.5 Аналіз програмного стеку
5.6.Розробка застосунку
5.7 Висновки
6. Дата видачі завдання«16» вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.09-27.09.24	Виконано
2	Аналіз сучасних методів для розробки	30.09-11.10.24	Виконано
3	Вивчення вибраних методів	14.10-25.10.24	Виконано
4	Розробка архітектури застосунку	28.10-08.11.24	Виконано
5	Реалізація функціоналу	11.11-22.12.24	Виконано
6	Тестування готового продукту	25.11-29.11.24	Виконано
7	Оформлення роботи: вступ, висновки, реферат	02.12-06.12.24	Виконано
8	Розробка демонстраційних матеріалів	09.12-13.12.24	Виконано

Здобувач вищої освіти _____ МАЛАМЕН Євген
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ СЄРИХ Сергій
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 63 стор., 28 рис., 12 джерел.

Наукове завдання – аналіз сучасних методів автоматизації фінансових процесів, огляд сучасних підходів до створення застосунків для обліку витрат, розробка програмного застосунку із використанням Python та його бібліотек.

Мета роботи – створення інтуїтивно зрозумілого, ефективного та модульного застосунку

Об'єкт дослідження – автоматизовані системи обліку фінансових витрат.

Предмет дослідження – розробка програмного застосунку для обліку витрат.

Короткий зміст роботи: У роботі проведено аналіз сучасних фінансових систем, проаналізовано актуальність автоматизації процесів обліку витрат, вивчено сучасні підходи до створення програмного забезпечення для фінансів. Для розробки використано Python, а саме його бібліотеки для роботи з базою даних, графічного інтерфейсу та візуалізації.

Розроблений застосунок було протестовано та вдосконалено, внаслідок чого він забезпечує зручний інтерфейс користувача та необхідну функціональність для обліку витрат. Програма відповідає сучасним вимогам безпеки та зручності користування.

Отже, у роботі описано створення програмного застосунку для автоматизації процесу обліку витрат за допомогою сучасних технологій програмування. У процесі виконання були використані методи аналізу сучасних технологій, документації бібліотек Python, а також практична реалізація та тестування розробленого застосунку.

Ключові слова: програмний застосунок, Python, Tkinter, SQLite, Pandas, Matplotlib, облік витрат, автоматизація, візуалізація даних.

ABSTRACT

The text part of the qualification work for obtaining the master's degree: 63 pages, 28 figures, 12 sources.

The scientific task is nalyzing modern methods of financial process automation, reviewing current approaches to creating expense-tracking applications, and developing a software application using Python and its libraries

The purpose of the work is creating an intuitive, efficient, and modular application

The object of research is automated systems for financial expense tracking

The subject of research is development of a software application for expense tracking

Summary of the work: The paper analyzes current financial systems, examines the relevance of automating expense-tracking processes, and explores modern approaches to creating finance-related software. Python was used for the development, in particular its libraries for database operations, graphical user interface creation, and data visualization.

The developed application was tested and improved, resulting in a user-friendly interface and the necessary functionality for expense tracking. The program meets modern requirements for security and ease of use.

Thus, the paper describes the creation of a software application for automating the expense-tracking process using modern programming technologies. The work involved the analysis of up-to-date technologies, the study of Python library documentation, as well as the practical implementation and testing of the developed application.

Keywords: software application, Python, Tkinter, SQLite, Pandas, Matplotlib, expense tracking, automation, data visualization.

ЗМІСТ

ВСТУП 10

1 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМ ОБЛІКУ ВИТРАТ ... 11

1.1 Важливість контролю витрат у сучасному світі 11

1.2 Перспективи та додаткові аспекти розвитку 12

1.3 Сучасні тренди створення застосунків для роботи з фінансами 19

1.4 Огляд аналогів 22

1.5 Етапи створення застосонку для обліку витрат 25

2 АНАЛІЗ ПРОГРАМНОГО СТЕКУ 29

2.1 Технології створення застосунку 29

2.2 Проектування інтерфейсу користувача за допомогою Tkinter 41

2.3 Збереження та управління даними в базі SQLite 45

2.4 Аналіз та обробка даних із використанням бібліотеки Pandas 48

2.5 Візуалізація витрат за допомогою Matplotlib 51

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ 55

3.1 Архітектура застосунку: структура та взаємодія модулів 55

3.2 Реалізація модуля введення та валідації даних витрат 58

3.3 Алгоритми обробки та аналізу фінансових даних 62

3.4 Генерація графіків і звітів для візуалізації результатів 63

3.5 Інтерфейс користувача: інтеграція функціоналу з Tkinter 65

3.6 Розробка та структура бази даних SQLite 68

3.7 Огляд кінцевого продукту 70

ВИСНОВКИ 77

ПЕРЕЛІК ПОСИЛАНЬ **78**

ДОДАТОК А **79**

ДОДАТОК Б **81**

ВСТУП

Сьогодні цифрові технології глибоко інтегровані в наше повсякденне життя, і особливо це стосується управління фінансами. Програмні застосунки для обліку витрат, бюджетування та аналізу фінансових даних набувають все більшої популярності серед індивідуальних користувачів і бізнесу. Розвиток фінансових технологій за останнє десятиліття демонструє значний прогрес у створенні інструментів, які дозволяють автоматизувати рутинні фінансові процеси та підвищити ефективність управління коштами.

Незважаючи на широкий вибір існуючих фінансових програм, залишається потреба у простих, функціональних і доступних рішеннях, які відповідають сучасним стандартам зручності та безпеки. Мова програмування Python та її бібліотеки Tkinter, SQLite, Pandas і Matplotlib відкривають великі можливості для створення інтуїтивно зрозумілих застосунків, які забезпечують як базові функції обліку, так і розширений аналіз даних із візуалізацією.

Метою даної роботи є створення програмного застосунку для обліку фінансів, що дозволяє користувачам легко відстежувати витрати, вести бюджети та аналізувати фінансову інформацію. Завдяки використанню сучасних підходів до розробки програмного забезпечення, реалізований застосунок буде відповідати потребам користувачів, забезпечуючи надійність, функціональність і простоту використання.

1 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМ ОБЛІКУ ВИТРАТ

1.1 Важливість контролю витрат у сучасному світі

У сучасному світі контроль витрат набув особливої важливості, оскільки ефективне управління фінансами є ключовим фактором успіху як для окремих осіб, так і для організацій. Зростаюча складність економічних відносин, глобалізація ринків та швидкий технологічний розвиток вимагають від суб'єктів господарювання постійного моніторингу та оптимізації своїх витрат.

Для підприємств контроль витрат є критичним елементом забезпечення конкурентоспроможності та фінансової стійкості. Ефективне управління витратами дозволяє компаніям знижувати собівартість продукції або послуг, що, у свою чергу, сприяє підвищенню прибутковості та можливості інвестування в розвиток. Зокрема, впровадження сучасних інструментів контролінгу витрат допомагає підприємствам адаптуватися до обмеженості ресурсів та мінливих умов господарювання

В умовах обмеженості ресурсів особливої актуальності набуває застосування сучасних методів управління витратами. Оптимізація та зменшення витрат на виробництві дозволяє підприємству регулювати ціни, встановлюючи їх на рівні, конкурентному на ринку. Розробка ефективної системи управління витратами включає встановлення взаємозв'язку між системою управління витратами та бюджетним управлінням, визначення напрямів зниження витрат у перспективі та розробку плану заходів щодо їх зменшення

Для індивідуальних споживачів контроль витрат є основою фінансової грамотності та стабільності. Розуміння того, куди спрямовуються кошти, дозволяє оптимізувати витрати, уникати непотрібних боргів та планувати майбутні фінансові цілі. Аналіз витрат допомагає виявити непотрібні або надмірні витрати,

що, у свою чергу, сприяє підвищенню якості життя та досягненню фінансової незалежності

У сфері бізнесу прогнозування витрат є стратегічним процесом, який дозволяє компаніям управляти бюджетом, раціоналізувати ресурси та забезпечити фінансову стабільність. Використання сучасних технологій аналізу даних та програмного забезпечення для прогнозування витрат допомагає бізнесам приймати обґрунтовані рішення та підвищувати рентабельність

Загалом, у сучасному динамічному середовищі контроль витрат є невід'ємною складовою успішного управління фінансами. Він забезпечує можливість адаптації до змін, підвищення ефективності використання ресурсів та досягнення стратегічних цілей як для окремих осіб, так і для організацій.

1.2 Перспективи та додаткові аспекти розвитку

Дослідження історії облікових систем свідчить, що перші спроби спростити процес управління фінансами належать ще давнім цивілізаціям, які використовували різноманітні інструменти для підрахунку й контролю товарно-грошових потоків. Так, у Месопотамії було розроблено систему глиняних табличок, у Стародавньому Єгипті – папірусні звіти, а в Римській імперії – ранні бухгалтерські книги, що описували рух коштів у домашніх господарствах. З плином часу потреба в більш точному й швидкому обліку зростала, що стимулювало розвиток різних технічних пристроїв: від рахівниць і перших механічних калькуляторів до сучасних комп’ютерних систем.

На початку XX століття, з появою електронних обчислювальних машин, підхід до обліку витрат у великих компаніях змінився кардинально. Виникла можливість обробляти значні обсяги даних, що, своєю чергою, спричинило формування нових методів і принципів фінансової аналітики. Застосунки для персонального користування стали особливо популярними вже з появою персональних комп’ютерів у другій половині століття. Нині, на основі мов

програмування високого рівня (зокрема Python), створюються платформи та застосунки, що дають змогу автоматизовано, швидко й гнучко управляти інформацією про витрати, використовуючи навіть смартфони й хмарні сховища.

Важливо відзначити, що процес контролю особистих витрат не можна розглядати суто в технічному або математичному аспекті. Він також має філософську та соціальну складову. Люди в різні епохи дивилися на гроші під різним кутом: як на засіб виживання, показник статусу, вираження особистої свободи чи обмеження. У сучасних умовах цифровізації суспільства фінансова грамотність стає не лише економічною категорією, а й питанням соціальної культури та відповідальності.

З одного боку, автоматизація процесів обліку витрат полегшує ведення бюджету, знижує ризик людських помилок і створює відчуття більшої впевненості в майбутньому. З іншого боку, психологічні фактори (наприклад, імпульсивні покупки, залежність від реклами) можуть ставати перепоною для об'єктивного аналізу даних. Існує навіть міждисциплінарна сфера досліджень – поведінкова економіка, що вивчає, як емоції та соціальні норми впливають на фінансові рішення індивідів.

Розробка системи обліку витрат із використанням мови Python виходить за рамки виключно програмної інженерії, адже вона поєднує кілька галузей знань:

- Економіка. Правильне розуміння механізмів формування бюджету, оптимізації витрат і планування фінансових показників передбачає глибоке знайомство з основами бухгалтерського обліку та фінансового аналізу.
- Психологія. Люди рідко ухвалюють абсолютно раціональні рішення, особливо у сфері особистих фінансів. Тому важливо враховувати поведінкові аспекти та розуміти, як можна впливати на користувача (через інтерфейс, нагадування, візуалізацію) для підвищення ефективності фінансового контролю.
- Інформаційні технології. Розробка високонавантажених сервісів і безпечних баз даних, застосування алгоритмів машинного навчання

для прогнозування витрат і рекомендацій – усе це вимагає відповідних знань та навичок програмування.

Однією з головних тенденцій сучасного розроблення програмного забезпечення є можливість масштабувати застосунок залежно від потреб користувача чи компанії. Для систем обліку витрат це означає:

- Потенційне збільшення кількості користувачів: від приватного використання до корпоративного застосування у великих компаніях.
- Інтеграція з іншими сервісами: онлайн-банкінгом, бухгалтерськими системами, сервісами інтернет-магазинів тощо.
- Потреба в захисті даних: із збільшенням кількості користувачів та обсягів інформації зростає й ризик витоків фінансових даних.

Недоліками масштабування можуть бути: збільшення складності інфраструктури, підвищення вимог до апаратного забезпечення, а також необхідність постійного оновлення системи безпеки та реагування на зміни в законодавстві.

Сучасні підходи до організації роботи над програмними застосунками (зокрема системами обліку витрат) передбачають використання гнучких методологій і культури DevOps, щоб прискорити випуск оновлень і покращити якість продукту. Тут можуть бути застосовані такі практики:

- Continuous Integration (CI): автоматизація процесу збирання та тестування коду при кожному внесенні змін у репозиторій.
- Continuous Delivery (CD): швидке розгортання нових версій застосунку на середовище розробки чи навіть у продуктивне середовище.
- Інфраструктура як код (Infrastructure as Code): опис налаштувань серверів та інших елементів інфраструктури у вигляді конфігураційних файлів, що полегшує управління та масштабування.

Загалом DevOps підхід дає змогу уникнути несумісності версій, оперативно усувати вразливості в коді та продумано розподіляти ресурси, зокрема й для безперебійної роботи автоматизованих систем обліку витрат.

Одним із перспективних напрямів розвитку фінансових застосунків є інтеграція алгоритмів машинного навчання (ML) і штучного інтелекту (AI). Наприклад, збирання великої кількості даних про витрати користувачів дає змогу навчати моделі прогнозування майбутніх витрат, виявляти підозрілі транзакції або формувати індивідуальні рекомендації щодо заощаджень. Переваги такого підходу:

- Покращення персоналізації: система може аналізувати звички користувача та підлаштовувати під них інтерфейс або функціонал.
- Аналітика на основі Big Data: за наявності великої бази користувачів і транзакцій, можна виявляти закономірності та тренди.
- Прогнозування можливих економічних ризиків: за допомогою аналітичних моделей можна попереджати про можливі витрати чи боргові зобов’язання, якщо раніше спостерігалися подібні ситуації в статистично схожих користувачів.

З іншого боку, повноцінне впровадження алгоритмів штучного інтелекту вимагає значних обчислювальних ресурсів і відповідної кваліфікації розробників, що впливає на собівартість і вартість володіння продуктом.

Автоматизація фінансових процесів пов’язана з низкою ризиків безпеки. Злочинці можуть спробувати викрасти конфіденційні дані, паролі чи інформацію про транзакції. Тому сучасні застосунки повинні передбачати:

- Шифрування: застосування криптографічних алгоритмів для зберігання даних у базі та передачі їх між клієнтом і сервером (HTTPS, SSL/TLS).
- Двофакторну автентифікацію (2FA): підвищення захищеності за рахунок додаткового коду, який надходить через SMS або генерується спеціальним застосунком.

- Резервне копіювання даних: регулярне збереження бекапів на віддалені сервери або в хмарні сховища для відновлення у випадку збою чи кібератаки.
- Аудит подій: реєстрацію всіх ключових подій у системі (авторизація, створення/видалення даних) з метою швидкого виявлення підозрілої активності.

Крім того, розробники повинні стежити за виконанням локальних та міжнародних стандартів (наприклад, GDPR у Європейському Союзі) щодо обробки персональних даних. Порушення цих правил може призвести до серйозних юридичних наслідків і втрати репутації.

Попри те, що блокчейн здебільшого асоціюють із криптовалютами, ця технологія набуває все більшого поширення у сфері фінансів загалом. Децентралізований характер зберігання даних може забезпечити:

- Необоротність записів: інформація, що потрапляє в блокчейн, не може бути змінена без узгодження з учасниками мережі.
- Прозорість: усі операції відстежуються, проте залишаються анонімними (якщо це передбачено конкретним протоколом).
- Зменшення посередників: у теорії користувач може передавати гроші чи активи безпосередньо іншому користувачеві без участі банку.

У контексті особистих фінансів або корпоративного обліку витрат блокчейн може пропонувати інноваційні підходи до верифікації операцій, проте потрібно брати до уваги технічну складність реалізації та нормативні обмеження в різних юрисдикціях.

Сучасний користувач очікує мати доступ до своїх фінансових даних у будь-який момент і з будь-якого пристрою. Тож мобільні застосунки для обліку витрат стають дедалі популярнішими. Вони пропонують:

- Швидкий запис транзакцій: скорочення часу на введення даних за рахунок інтерактивних форм та автозаповнення категорій витрат.

- Push-сповіщення: нагадування про майбутні платежі, перевищення лімітів витрат чи появу нових корисних функцій.
- Інтеграцію з іншими сервісами: календарями, завданнями, електронною поштою, соціальними мережами, що дає змогу збирати більш повні дані про активність користувача.

Усе це створює додаткові виклики для розробників у контексті UX/UI дизайну, продуктивності, безпеки та узгодженості з серверною частиною.

Одним зі способів стимулювати користувачів вести детальний облік власних витрат є впровадження елементів гейміфікації. Це можуть бути:

- Досягнення та нагороди: відкриття нових “рівнів” економії, якщо користувачеві вдалося зменшити витрати на певний відсоток упродовж місяця.
- Змагання з іншими (якщо йдеться про групу однодумців або корпоративний контекст): створення рейтингу чи турнірної таблиці найекономніших співробітників.
- Віртуальна валюта чи бали: накопичення внутрішніх “балів лояльності” за регулярне введення даних або досягнення фінансових цілей.

Такі мотиваційні механізми базуються на тому, що людина схильна приділяти більше уваги діяльності, яка приносить відчуття прогресу та винагороди. У рамках дипломного проєкту це може стати інструментом залучення й утримання активних користувачів.

У корпоративних сценаріях або навіть для просунутих домашніх користувачів може бути корисним впровадження так званих Dashboard-рішень. Це інтерактивні панелі з ключовими показниками (KPI), діаграмами, графіками та індикаторами динаміки витрат. Завдяки когортному аналізу можна розбивати витрати за конкретними проміжками часу, подіями, категоріями або групами користувачів (у разі корпоративної платформи).

Наприклад, керівник відділу фінансів у компанії може відстежувати, як змінюються витрати на відрядження з плином часу, або визначати частку витрат,

що припадає на окремі проєкти. Домашній користувач може відстежувати, як змінюється сума комунальних платежів залежно від пори року, порівнювати витрати на продукти з різними магазинами тощо.

Для тих розробників, які прагнуть виходити з програмними продуктами на глобальний ринок, актуальним стає питання відповідності міжнародним стандартам фінансової звітності (IFRS) чи локальним стандартам GAAP у США, а також іншим податковим нормативам. Хоча особисті фінанси переважно не вимагають суворого дотримання таких вимог, корпоративні рішення вимагають врахування:

- Структури податкової звітності.
- Норм і правил формування документів.
- Звірки даних у міжнародних транзакціях (наприклад, якщо працівники виконують роботу дистанційно з іншої країни).

Це може ускладнювати програмну логіку, але водночас відкриває ширші можливості для застосунку, що прагне вийти за межі локальних ринків.

Хоча сьогодні це може здатися футуристичним, проте деякі компанії вже експериментують із використанням квантових комп’ютерів для складних фінансових розрахунків і моделювання. У далекій перспективі квантові обчислення дозволять обробляти великі масиви даних у рази швидше, що може знайти застосування у динамічному перерахунку бюджету, аналізі інвестиційних портфелів чи навіть симуляції індивідуальних моделей споживання.

Щодо VR/AR, то тут можемо уявити ситуацію, коли користувач “поринає” у віртуальний офіс і бачить тривимірну модель власних витрат, може “переміщуватися” між категоріями як у грі. Хоча це, звісно, поки що екзотика для сфери обліку, досвід показує, що технології розвиваються стрімко і можуть пропонувати несподівані рішення навіть для, здавалося б, рутинних процесів.

1.3 Сучасні тренди створення застосунків для роботи з фінансами

Фінансові технології відіграють ключову роль у створенні сучасних застосунків для роботи з фінансами, які мають бути не лише функціональними, але й безпечними, інтуїтивно зрозумілими та персоналізованими. Розробка таких програм потребує врахування широкого спектра аспектів, що охоплюють інтерфейс користувача, безпеку даних, інтеграцію з іншими платформами, аналітику та прогнозування, а також забезпечення доступності й зручності для користувачів. Ефективний інтерфейс для фінансового застосунку повинен бути зрозумілим, естетично привабливим і зручним у використанні. Особливу увагу слід приділяти адаптивності дизайну, щоб забезпечити його коректну роботу на різних пристроях, від комп’ютерів до мобільних телефонів. Користувачі повинні легко знаходити необхідні функції, такі як додавання витрат чи перегляд звітів, що досягається завдяки продуманій навігації.

Захист даних є критично важливим, особливо у фінансових застосунках, де користувачі довіряють програмі чутливу інформацію. Шифрування даних, мультифакторна аутентифікація та токенізація платіжної інформації є стандартними методами забезпечення безпеки. Водночас авторські підходи, такі як поведінкова аналітика, що виявляє підозрілу активність, стають популярними інструментами для забезпечення додаткового рівня захисту. Крім того, регулярні перевірки на вразливості та постійне оновлення системи безпеки допомагають зменшити ризики кібератак.

Сучасні фінансові застосунки також мають забезпечувати інтеграцію з іншими платформами та сервісами. Це включає підтримку відкритого банкінгу для взаємодії з банківськими системами через API, а також інтеграцію з платформами для бухгалтерії та бізнес-аналітики, такими як QuickBooks або Tableau. Інтеграція допомагає створювати більш комплексні рішення, які відповідають сучасним потребам користувачів. Наприклад, застосунок може

автоматично імпортувати банківські транзакції, аналізувати їх і показувати користувачеві звіт про витрати та доходи в зручному форматі.

Аналітика та прогнозування є важливою складовою фінансових програм, оскільки дозволяють користувачам отримувати більше інформації про свої фінанси та робити обґрунтовані рішення. Сучасні технології, такі як машинне навчання, дозволяють аналізувати минулі дані для створення прогнозів, наприклад, виявлення ризику перевищення бюджету чи рекомендацій щодо заощаджень. Візуалізація даних у вигляді графіків і діаграм допомагає користувачам швидше розуміти динаміку своїх фінансів і приймати відповідні рішення.

Важливим аспектом розвитку систем обліку витрат є постійне розширення їх функціоналу шляхом інтеграції з іншими сервісами та платформами. На практиці це може реалізовуватися через підключення до банківських API, автоматичне імпортування даних з електронних гаманців або платіжних систем, а також завдяки обміну інформацією з бухгалтерськими застосунками. Зростає попит і на мобільні інструменти контролю витрат, які надають можливість оперативно фіксувати транзакції просто у телефоні та синхронізувати їх із десктопною версією через хмарні сховища. Крім того, користувачі часто очікують, що застосунок матиме дружній інтерфейс, який дозволить налаштовувати фінансові цілі та отримувати сповіщення про стан бюджету. Подальше розширення таких систем може передбачати впровадження елементів машинного навчання: наприклад, методи кластеризації допоможуть виявляти незвичні патерни витрат, а алгоритми прогнозування дозволять заздалегідь попереджати про можливі перевитрати.

Ще один перспективний напрямок пов’язаний із гейміфікацією. У сучасному насиченому інформаційному просторі користувачів дедалі складніше мотивувати щодня вносити дані про витрати. Тому запровадження ігрових механік, наприклад, “досягнень” за економію коштів або можливості змагатися з друзями у певних категоріях витрат, може сприяти підвищенню регулярності ведення обліку та покращенню фінансової дисципліни. Водночас інтерфейс має

бути простим і не відволікати користувача від основних завдань, адже складні й надмірно деталізовані форми часто демотивують оновлювати дані.

Персоналізація є ще одним важливим аспектом. Сучасні застосунки дозволяють налаштовувати категорії витрат, інтегрувати функціонал із календарем для нагадувань про рахунки чи фінансові цілі, а також створювати індивідуальні бюджети. Крім того, застосунки можуть використовувати рекомендаційні системи для автоматичного надання порад на основі поведінки користувача, наприклад, оптимальні стратегії економії чи інвестицій.

Реалізація обробки даних у реальному часі є викликом для фінансових застосунків. Це необхідно для відображення актуальних залишків на рахунках, обробки транзакцій і оновлення прогнозів. Використання потокової обробки даних, хмарних платформ та кешування забезпечує високу продуктивність і швидкість доступу до інформації. Одним із нових трендів є вбудовані фінанси, які інтегрують фінансові функції в нефінансові продукти або платформи. Наприклад, це може бути інтеграція функції оплати безпосередньо в застосунок для замовлення товарів чи послуг, що робить користування такими послугами значно зручнішим.

Фінансові застосунки також можуть відігравати освітню роль, підвищуючи фінансову грамотність користувачів. Наприклад, програми можуть пропонувати навчальні матеріали, демонструвати прогрес у досягненні фінансових цілей або надавати інтерактивні поради щодо покращення фінансового стану.

Таким чином, створення фінансових застосунків є складним процесом, який потребує врахування багатьох технічних, безпекових і користувацьких аспектів. Інноваційні підходи, такі як інтеграція штучного інтелекту, візуалізація даних і поведінкова аналітика, дозволяють розробникам створювати якісні та конкурентоспроможні продукти. Усі ці аспекти разом формують комплексний підхід до розробки, орієнтований на задоволення потреб сучасних користувачів.

1.4 Огляд аналогів

На сьогодні ринок програмних рішень для обліку грошових витрат та управління фінансами є надзвичайно різноманітним та динамічним. Існує чимало як комерційних онлайн-сервісів та мобільних застосунків, що надають швидкий доступ до даних та інтерактивну аналітику, так і десктопних програм з відкритим вихідним кодом, які дають змогу гнучко налаштовувати систему під власні потреби та повністю контролювати конфіденційність інформації. Ключовою тенденцією є автоматизація та централізація фінансових даних: більшість популярних сервісів дозволяють синхронізуватися з банківськими рахунками або імпортувати витрати за допомогою файлів у різних форматах, що мінімізує ручне введення даних.

Поряд із цим, зростає попит на мобільні додатки та «хмарні» рішення, які надають можливість оперативно відстежувати витрати, аналізувати категорії затрат і отримувати статистику в режимі реального часу. Усе більшого значення набувають питання безпеки та захисту особистих даних, що стимулює розробників запроваджувати такі функції, як двофакторна аутентифікація та шифрування транзакцій. Для частини користувачів важливим є також фактор локалізації: підтримка української мови та інтеграція з місцевими банками часто визначають їхній вибір.

Загалом, поточна ситуація на ринку рішень для ведення особистих та корпоративних фінансів демонструє баланс між простими у використанні мобільними інструментами для персональних потреб та більш потужними, універсальними системами, орієнтованими на малий бізнес чи спеціалізовані завдання. Такий широкий вибір дає можливість кожному користувачеві знайти оптимальний варіант під власні вимоги — чи то швидка й зручна реєстрація витрат, чи то глибокий бухгалтерський аналіз із розширеною звітністю.

Mint. Один із найпопулярніших сервісів для контролю особистих фінансів у Північній Америці. Він надає можливість з’єднуватися з банківськими рахунками

і платіжними картками, автоматично зчитує транзакції та формує звіти про доходи й витрати.

Переваги:

- Автоматична синхронізація з банками: мінімізує ручне введення даних.
- Зручний та візуально привабливий інтерфейс з діаграмами та категоріями витрат.
- Безкоштовний функціонал за рахунок реклами і пропозицій фінансових продуктів.

Недоліки:

- Обмежена інтеграція для України: сервіс орієнтований переважно на США та Канаду.
- Безпекові ризики: необхідно надавати дані для підключення до банку.

You Need A Budget (YNAB). Це веб- та мобільний застосунок, створений із фокусом на плануванні особистих фінансів. Основна ідея полягає в тому, щоби «дати кожному долару роботу» та розподілити всі доходи по категоріях ще до того, як кошти будуть витрачені.

Переваги:

- Методологія управління бюджетом (4 Правила YNAB), що спонукає корисувачів до більш дисциплінованого планування.
- Дружній інтерфейс з підказками, навчальними матеріалами та прикладами.
- Доступні мобільні додатки, які синхронізуються в реальному часі.

Недоліки:

- Щомісячна/річна підписка, що може бути невигідно для користувачів, які шукають безкоштовне рішення.
- Орієнтованість переважно на особисті фінанси, для бізнес-завдань функціонал обмежений.

GnuCash. Безкоштовна та відкрита програма для управління персональними й малими бізнес-фінансами. Працює на різних операційних системах (Windows, macOS, Linux). Інтерфейс традиційний, схожий на бухгалтерські програми, підтримує подвійний запис (double-entry accounting).

Переваги:

- Відкритий вихідний код (Open Source), безкоштовне використання та активна спільнота.
- Можливості для малого бізнесу: підтримка рахунків, фактур, клієнтів, постачальників тощо.
- Функціональність імпорту/експорту у різних форматах (CSV, QIF, OFX тощо).

Недоліки:

- Відносно складний інтерфейс для новачків. Потрібен час, щоб розібратися з основами бухгалтерського обліку.
- Досить обмежена мобільна версія (існують сторонні додатки, але офіційний функціонал не такий гнучкий, як у Mint чи YNAB).

HomeBank. Ще один безкоштовний додаток із відкритим вихідним кодом, орієнтований на управління персональними фінансами. Підтримує імпорт файлів з банківських систем, гнучке налаштування категорій витрат/доходів та надає базову аналітику.

Переваги:

- Легкий у використанні (більш спрощений інтерфейс у порівнянні з GnuCash).
- Відкритий вихідний код, можливість кастомізації.
- Детальна статистика з інтерактивними графіками та звітами.

Недоліки:

- Обмежений функціонал для бізнесу: застосунок орієнтований переважно на приватний облік.

- Відсутність інтегрованих онлайн-сервісів для автоматичної синхронізації транзакцій.

Незважаючи на наявність численних комерційних та безкоштовних продуктів для обліку витрат, у кожному з них користувачі можуть знайти певні обмеження — від складного інтерфейсу до відсутності вбудованої аналітики. Особливо актуальною виявляється потреба кастомізації: можливості додавати власні категорії витрат, змінювати налаштування відображення або підключати додаткові модулі. Під час етапу планування створення авторського програмного рішення важливо заздалегідь визначити, який стек технологій забезпечить потрібний рівень гнучкості. Якщо йдеться про довгостроковий проєкт, у якому зростатиме кількість користувачів, слід звернути увагу на масштабованість бази даних, інтеграцію з хмарними сервісами й реалізацію багатоетапного тестування.

Крім того, у процесі створення застосунку для обліку витрат слід завчасно попрацювати над планом розширень. Наприклад, після першого релізу може з’явитися ідея додати функцію складання сімейного бюджету, де кілька членів родини спільно ведуть облік. Іншим логічним кроком стає впровадження різних типів доступу: адміністратор, звичайний користувач, гість із обмеженими правами. Етап “мозкового штурму” перед написанням коду дає змогу визначити такі вимоги та зорієнтуватися, наскільки вони впливають на архітектуру застосунку і його базу даних.

1.5 Етапи створення застосонку для обліку витрат

Процес створення фінансового застосунку є багатокроковим і вимагає ретельного планування та реалізації. Він охоплює такі основні етапи:

1. Збір вимог: На початковому етапі визначаються цілі та функціональні вимоги до фінансового застосунку. Розробники проводять аналіз цільової аудиторії, з’ясовують, які функції будуть затребуваними, наприклад, облік витрат, бюджетування, звіти або аналітика. Також визначаються вимоги до безпеки

даних, які є особливо важливими для фінансових програм, та платформи, на яких застосунок буде працювати (Windows, macOS, мобільні пристрої). Результатом цього етапу є створення документа специфікацій.

2. Планування: Після збору вимог створюється план розробки застосунку. Це включає розробку архітектури програми: поділ на модулі (наприклад, база даних, інтерфейс, обробка даних) та визначення взаємодії між ними. Також на цьому етапі створюється схема бази даних, яка визначає, як зберігатимуться фінансові дані (витрати, доходи, категорії). Для візуалізації інтерфейсу створюються прототипи або макети.

3. Дизайн: Дизайн фінансового застосунку має бути зрозумілим, естетичним і функціональним. Підбирається кольорова палітра, шрифти та стилі елементів інтерфейсу. Важливо, щоб дизайн враховував зручність навігації, а також був адаптований для різних пристроїв. Наприклад, мобільні версії повинні мати спрощений інтерфейс із великими кнопками для легшого користування.

4. Розробка: Етап розробки програмного забезпечення є одним із найскладніших і найважливіших у створенні фінансового застосунку. Він охоплює написання коду, реалізацію бази даних, бізнес-логіки та створення інтерфейсу користувача. Спочатку створюється структура бази даних, яка відповідає вимогам до зберігання даних. У нашому випадку використовується SQLite, оскільки ця база даних проста у використанні та не потребує окремого серверного середовища. Створюються таблиці для зберігання інформації про витрати, доходи, бюджети та інші фінансові дані. Наступним кроком є розробка бізнес-логіки, яка включає функції для обчислення залишків бюджету, аналізу витрат за категоріями та побудови графіків для візуалізації даних. Потім реалізується інтерфейс користувача, що створюється за допомогою бібліотеки Tkinter. Цей інтерфейс дозволяє користувачам легко взаємодіяти з програмою: вводити дані, переглядати звіти, змінювати налаштування бюджету тощо. Завершальним етапом є інтеграція всіх компонентів — бази даних, бізнес-логіки та інтерфейсу — для забезпечення коректної взаємодії між ними. Такий підхід

дозволяє створити цілісний продукт, який відповідає вимогам і очікуванням користувачів.

5. Тестування: Тестування є невід'ємною частиною процесу розробки фінансового застосунку, оскільки саме на цьому етапі перевіряється якість роботи програми. Починається тестування з перевірки роботи окремих модулів. Наприклад, тестується коректність функцій, які обчислюють бюджет, зберігають дані в базі чи будують графіки. Потім проводиться інтеграційне тестування, яке перевіряє, як різні модулі взаємодіють між собою. Це особливо важливо для програм із кількома компонентами, адже будь-яка помилка у взаємодії може вплинути на роботу всієї системи. Функціональне тестування перевіряє, чи відповідає програма заданим вимогам, а юзабіліті-тестування оцінює зручність використання інтерфейсу. Особлива увага приділяється тестуванню безпеки, щоб упевнитися, що конфіденційні дані захищені від несанкціонованого доступу. Тестування проводиться не лише в лабораторних умовах, але й із залученням реальних користувачів, які допомагають виявити потенційні проблеми або недоліки у функціоналі. Завершується етап тестування створенням звіту, який фіксує всі знайдені помилки та заходи, вжиті для їх усунення.

6. Реліз та розгортання: Після успішного тестування застосунок готовий до релізу. Його пакують у вигляді інсталяційного файлу для Windows або публікують у магазинах додатків для мобільних платформ. На цьому етапі забезпечується розгортання програми на живому сервері або хмарному середовищі, якщо застосунок має функції онлайн-синхронізації. Також підготовлюється документація для користувачів, яка пояснює, як користуватися програмою.

7. Підтримка та післярелізний догляд: Після релізу програми починається етап її підтримки, який є не менш важливим, ніж попередні кроки розробки. Підтримка включає моніторинг продуктивності застосунку, щоб забезпечити його стабільну роботу. Розробники регулярно аналізують відгуки користувачів, щоб зрозуміти, які функції потребують покращення або додаткового розвитку. Виправлення багів, які можуть бути виявлені після випуску, виконується

оперативно, щоб користувачі не стикалися з незручностями. Окрім цього, програма оновлюється для додавання нових функцій і адаптації до змін у фінансовій сфері чи законодавстві. Наприклад, можуть впроваджуватися нові алгоритми аналізу даних або інтеграція з новими фінансовими платформами. Важливо також забезпечувати сумісність програми з новими версіями операційних систем або пристроїв. Етап підтримки завершується підготовкою звітів про проведені оновлення, які фіксують зміни та вдосконалення програми, що допомагає визначити її подальший розвиток.

Ці етапи є загальними та можуть бути адаптовані відповідно до конкретного проекту. Крім того, важливо брати до уваги такі аспекти, як оптимізація для пошукових систем (SEO), мобільна сумісність та безпека веб-сайту.

2 АНАЛІЗ ПРОГРАМНОГО СТЕКУ

2.1 Технології створення застосунку

Мова програмування Python займає провідні позиції серед інструментів для створення програмного забезпечення. Вона відома своєю універсальністю, легкістю у використанні та можливістю застосування в різних галузях — від веб-розробки до автоматизації бізнес-процесів. У цьому розділі буде детально розглянуто характеристики, переваги та практичне використання Python у розробці програмних систем, зокрема для обліку витрат.

Python був створений Гвідо ван Россумом у 1991 році, і відтоді мова постійно розвивалася, отримуючи оновлення, які роблять її ще більш функціональною та зручною. Із самого початку основною ідеєю Python була простота використання та читабельність коду. Це робить його доступним навіть для новачків у програмуванні. Завдяки активній спільноті розробників Python отримав тисячі додаткових бібліотек, що значно розширили його можливості.

Python традиційно вважають одним із найбільш гнучких інструментів для реалізації фінансового програмного забезпечення, оскільки він має велике ком’юніті та бібліотеки, які прискорюють розробку. Наприклад, інтеграція з веб-фреймворками (Flask, Django) відкриває шлях до створення веб-версій застосунку з віддаленим доступом до бази даних. Цікаво, що Python однаково зручний для написання простих скриптів аналізу (де застосунок виконується лише на локальній машині) та для побудови комплексних клієнт-серверних систем, здатних обробляти великий потік одночасних звернень.

Ще одна важлива перевага — кросплатформеність. Розробник може створювати застосунок на Windows, а потім без значних змін запустити його на Linux або macOS. Це суттєво спрощує процес тестування й розгортання системи, особливо коли в команді працюють фахівці з різними операційними системами. До того ж можливість швидкої інтеграції з мовами C/C++ робить Python

привабливим для завдань, де потрібна висока продуктивність для обробки великих обсягів даних.

Ця мова програмування є інтерпретованою з динамічною типізацією. Це означає, що код виконується безпосередньо, без попередньої компіляції, а типи даних визначаються під час виконання програми. Іншою важливою особливістю є простий синтаксис, який нагадує структуру англійської мови, завдяки чому код легко читається та пишеться.

Один із основних факторів, який сприяв популярності Python, — це широкий набір стандартних бібліотек. Вони дозволяють швидко реалізовувати різноманітні функції, такі як робота з файлами, обробка даних, створення мережевих додатків тощо. Крім того, Python підтримує інтеграцію з іншими мовами програмування, такими як C, C++ і Java, що робить його надзвичайно гнучким інструментом. Велика спільнота розробників активно створює нові бібліотеки та інструменти, що дозволяє Python залишатися актуальним у різних сферах програмування.

Він пропонує значний набір інструментів для роботи з даними, що робить його ідеальним вибором для створення систем обліку витрат. Наприклад, бібліотека Pandas дозволяє працювати з фінансовими даними у вигляді таблиць, здійснювати сортування, фільтрацію та агрегацію. Завдяки бібліотеці Matplotlib можна створювати візуалізації витрат у вигляді графіків і діаграм. SQLite забезпечує зручне зберігання фінансових транзакцій у локальній базі даних. Усі ці інструменти легко інтегруються один з одним, що дозволяє створити повноцінний програмний продукт.

Фінансові програми часто потребують обробки великих обсягів даних. Наприклад, система обліку витрат може аналізувати транзакції за певний період, виявляти закономірності у витратах або навіть прогнозувати майбутні витрати. Завдяки бібліотекам, таким як NumPy і Pandas, Python дозволяє реалізувати ці функції з мінімальними витратами часу.

Ще однією важливою складовою фінансових систем є візуалізація. Графічне представлення даних полегшує їх аналіз. Наприклад, кругова діаграма допомагає

побачити, які категорії витрат є найбільш значущими. Python також дозволяє інтегрувати ці візуалізації у графічний інтерфейс користувача за допомогою бібліотеки Tkinter.

Tkinter: Особливості використання. Tkinter — це стандартна бібліотека Python, яка надає розробникам можливість створювати графічний інтерфейс користувача (GUI) для програм різного рівня складності. Основна ідея цієї бібліотеки полягає у спрощенні створення візуальних елементів, таких як кнопки, поля вводу, меню та діалогові вікна, шляхом надання доступу до високорівневих інструментів. Tkinter базується на інструментарії Tcl/Tk, що є надійною платформою для розробки інтерфейсів і працює однаково на різних операційних системах, таких як Windows, macOS та Linux.

Однією з найважливіших переваг Tkinter є його простота у використанні. Для початку роботи з цією бібліотекою достатньо базових знань мови Python. Tkinter є інтегрованою частиною стандартної бібліотеки Python, тому не потребує додаткового встановлення. Це робить його ідеальним вибором для новачків, які хочуть створювати графічні програми без складної підготовки або інсталяції сторонніх бібліотек.

Tkinter забезпечує великий вибір стандартних віджетів для створення GUI. До них належать мітки для відображення тексту або зображень, кнопки для виконання дій, текстові поля для введення даних, прапорці для вибору опцій та навіть полотно для створення малюнків або графіків. Наприклад, мітка (Label) може бути використана для відображення статичного тексту, а текстове поле (Entry) дозволяє користувачеві вводити інформацію. Ці елементи забезпечують базову функціональність, необхідну для більшості застосунків, але водночас вони дуже гнучкі й можуть бути налаштовані під конкретні потреби.

Для управління розташуванням елементів у вікні Tkinter пропонує три основні менеджери геометрії: pack, grid і place. Кожен із них має свої особливості й використовується залежно від складності макета. Менеджер pack дозволяє автоматично розташовувати елементи зверху вниз або зліва направо. Grid більш гнучкий і дозволяє створювати таблиці для розміщення віджетів у вигляді рядків і

стовпців. Place використовується для точного позиціонування елементів за координатами, що дозволяє створювати складні макети з високим рівнем контролю.

Крім віджетів, Tkinter підтримує події, які дозволяють реалізувати інтерактивність у програмах. Наприклад, натискання кнопки або введення тексту може викликати певну дію, таку як обробка введених даних або відображення повідомлення. Події обробляються за допомогою "зворотних викликів" (callback), які є функціями, пов'язаними з певними діями користувача. Це дозволяє створювати динамічні програми, які реагують на дії користувача в реальному часі.

Одним із ключових аспектів Tkinter є його підтримка кросплатформенності. Програми, створені за допомогою цієї бібліотеки, працюють однаково на різних операційних системах, що робить Tkinter надійним вибором для розробників, які прагнуть забезпечити універсальність своїх застосунків. Завдяки цьому розробник може створити програму на одному комп'ютері, але бути впевненим, що вона працюватиме без змін на інших пристроях.

Tkinter також підтримує інтеграцію з іншими бібліотеками Python, що значно розширює його можливості. Наприклад, бібліотека sqlite3 дозволяє працювати з базами даних, зберігаючи й обробляючи дані, тоді як matplotlib дає змогу створювати графіки й діаграми, які можуть бути відображені у вікні Tkinter. Це робить його універсальним інструментом для створення комплексних програм.

Ще однією перевагою Tkinter є можливість налаштування інтерфейсу. Розробник може змінювати кольори, шрифти, розміри елементів і навіть додавати зображення, щоб зробити інтерфейс привабливішим і зручнішим для користувача. Крім того, бібліотека підтримує створення багатовіконних програм, що дозволяє розробляти складні системи з меню, панелями інструментів і діалоговими вікнами.

Tkinter надає інструменти для побудови не лише графічних інтерфейсів, але й логіки взаємодії між різними частинами програми. Наприклад, програми для обліку витрат можуть використовувати Tkinter для введення даних про витрати, sqlite3 для їхнього зберігання й обробки, а matplotlib для візуалізації у вигляді

графіків і діаграм. Усе це інтегрується в єдине середовище, що спрощує роботу як для розробників, так і для користувачів.

SQLite: Особливості використання. SQLite — це легка реляційна система управління базами даних (СУБД), яка відома своєю простотою та універсальністю. Вона є однією з найпоширеніших баз даних у світі, оскільки вбудована в більшість сучасних операційних систем, мобільних пристроїв і програмного забезпечення. Основна перевага SQLite полягає в тому, що для її використання не потрібно встановлювати окремий сервер чи додаткове програмне забезпечення. Вона зберігає всі дані у вигляді одного файлу на локальному пристрої, що робить її ідеальним вибором для невеликих програм.

Однією з головних причин популярності SQLite є її автономність. База даних працює без необхідності підключення до сервера, що значно спрощує процес налаштування й управління. Це особливо зручно для невеликих застосунків, таких як мобільні додатки, системи обліку витрат чи персональні фінансові менеджери. У разі використання SQLite для програми всі дані зберігаються у вигляді єдиного файлу з розширенням .db, який можна переносити між пристроями.

SQLite інтегрується в Python за допомогою стандартної бібліотеки sqlite3, яка дозволяє працювати з базою даних без необхідності встановлення сторонніх модулів. Це забезпечує швидкий початок роботи навіть для новачків. У Python підключення до бази даних здійснюється лише кількома рядками коду, а створення таблиць, вставка, оновлення та вибірка даних реалізуються за допомогою стандартних SQL-запитів.

На відміну від складніших СУБД, таких як MySQL або PostgreSQL, SQLite не вимагає попереднього налаштування користувачів, прав доступу чи серверних конфігурацій. Це дозволяє розробникам зосередитися на логіці програми, а не на адмініструванні бази даних.

SQLite підтримує повноцінну реляційну модель, яка дозволяє організувати дані у вигляді таблиць, що складаються з рядків і стовпців. Кожна таблиця має

унікальне ім'я та структуру, яка визначається під час її створення. Наприклад, таблиця для зберігання витрат може мати такі поля:

- id — унікальний ідентифікатор витрати.
- date — дата здійснення витрати.
- category — категорія витрати (наприклад, "Їжа", "Транспорт").
- description — опис витрати.
- amount — сума витрати.

Ця структура дозволяє зберігати дані у впорядкованому вигляді, що полегшує їхню вибірку та аналіз.

SQLite підтримує транзакції, що забезпечує цілісність даних у разі виконання кількох взаємопов’язаних операцій. Наприклад, якщо програма записує кілька витрат одночасно, транзакції гарантують, що всі зміни будуть збережені лише у разі успішного виконання кожного кроку. У разі помилки транзакція може бути скасована, що запобігає пошкодженню даних.

SQLite підтримує стандартний SQL, що дозволяє виконувати різноманітні операції з базою даних. Наприклад, ви можете:

- Додавати дані за допомогою операції INSERT.
- Оновлювати дані за допомогою UPDATE.
- Видаляти записи за допомогою DELETE.
- Фільтрувати та вибирати дані за допомогою SELECT із умовами.

SQL-запити легко інтегруються в програму за допомогою Python, що робить взаємодію з базою даних інтуїтивно зрозумілою.

SQLite використовує ефективні механізми зберігання даних. Наприклад, вона автоматично оптимізує використання місця на диску, стискаючи дані під час їхнього запису. Крім того, SQLite дозволяє розробникам визначати індекси для пришвидшення пошуку, що є корисним у програмах із великим обсягом даних.

Переваги SQLite:

- Легка вага: База даних зберігається у вигляді одного файлу, що спрощує її перенесення.

- Відсутність сервера: Для роботи не потрібне налаштування серверного середовища.
- Швидкодія: SQLite оптимізована для читання та запису даних, що забезпечує високу продуктивність навіть на слабких пристроях.
- Кросплатформенність: Дані SQLite можуть використовуватися на будь-якій платформі без змін.
- Сумісність із Python: Використання стандартної бібліотеки sqlite3 забезпечує повну інтеграцію з Python.

Попри численні переваги, SQLite має деякі обмеження, які варто враховувати:

- Вона не підходить для програм, які потребують масштабованості або високої кількості одночасних користувачів.
- Відсутність механізмів керування правами доступу робить її менш підходящою для великих корпоративних систем.
- SQLite не підтримує деякі складні SQL-операції, такі як частина розподілених запитів.

SQLite часто використовується в поєднанні з іншими бібліотеками Python, такими як Tkinter і Pandas. Наприклад, дані, що зберігаються в SQLite, можуть бути легко виведені у графічний інтерфейс за допомогою Tkinter або проаналізовані за допомогою Pandas. Це забезпечує зручність роботи з базою даних навіть для складних застосунків. і стилізаційні ефекти для сторінки, розділити стиль від змісту, що спрощує зміну дизайну та забезпечує єдність стилю на всьому веб-сайті.

Pandas: Особливості використання. Pandas — це бібліотека Python, розроблена для обробки та аналізу структурованих даних. Її основними характеристиками є зручність використання, висока продуктивність і багатий набір функцій. Pandas часто використовується у фінансових розрахунках, наукових дослідженнях і бізнес-аналітиці. Для кращого розуміння її функціональних можливостей розглянемо детальніше основні аспекти бібліотеки.

DataFrame є основною структурою даних у Pandas. Це двовимірна таблиця, яка складається з рядків і стовпців. Кожен стовпець може мати свій тип даних, наприклад, числовий, текстовий або дата. DataFrame схожий на таблицю в реляційній базі даних або електронній таблиці Excel. Його основна перевага полягає в тому, що він дозволяє легко маніпулювати даними: сортувати, фільтрувати, групувати тощо.

DataFrame підтримує індексацію, що дозволяє швидко доступатися до потрібних рядків або стовпців. Наприклад, ви можете працювати з окремими значеннями, використовуючи індекси рядків і назв стовпців, що робить структуру гнучкою для складних операцій.

Одним із ключових етапів роботи з даними є їхній імпорт із зовнішніх джерел. Pandas дозволяє імпортувати дані з файлів різних форматів, таких як CSV, Excel, JSON і SQL. Процес імпорту максимально спрощений: функція read_csv() або read_excel() автоматично розпізнає структуру файлу та створює DataFrame. Pandas також дозволяє налаштувати обробку специфічних випадків, таких як відсутні значення або інші неточності у вихідних даних.

Підготовка даних після імпорту — це очищення та обробка, що включає видалення пропущених значень, зміну типів даних, об’єднання таблиць та інші операції. Наприклад, для роботи з датами Pandas автоматично конвертує їх у формат datetime, що дозволяє проводити операції сортування та групування.

Pandas забезпечує інструменти для вибору підмножин даних, що базуються на умовах. Наприклад, якщо у вас є таблиця витрат, ви можете вибрати всі записи за певною категорією або за періодом. Фільтрація даних здійснюється шляхом використання умовних операторів, таких як >, <, ==, що робить її дуже простою у виконанні.

Ця функціональність дозволяє знаходити закономірності в даних, усувати дублікати або аналізувати конкретні сегменти таблиці. Фільтрація може комбінуватися із сортуванням, що спрощує пошук найважливіших записів.

Pandas є потужним інструментом для аналізу даних завдяки вбудованим функціям для виконання статистичних і математичних операцій. Ви можете

обчислювати середні значення, знаходити мінімальні та максимальні значення, підраховувати кількість унікальних значень і багато іншого.

Наприклад, функції агрегації, такі як sum(), mean(), count(), дозволяють проводити глибокий аналіз даних без необхідності використання складних обчислень. Pandas також підтримує створення зведених таблиць (pivot tables), які є незамінними для аналізу великих масивів даних із багатьма змінними.

Робота з даними, прив’язаними до часу, є однією з найважливіших особливостей Pandas. Бібліотека дозволяє конвертувати текстові значення у формат дати, сортувати дані за часом, групувати записи за днями, тижнями чи місяцями. Це особливо корисно у фінансових додатках, де важливо аналізувати зміни даних у часі.

Наприклад, Pandas може обчислювати зміну витрат за місяць, виявляти пікові значення у витратах або створювати графіки трендів для аналізу динаміки даних.

Пандемійна інтеграція з бібліотекою Matplotlib дозволяє створювати різноманітні графіки та діаграми для візуалізації даних. Наприклад, ви можете побудувати стовпчасту діаграму витрат за категоріями або лінійний графік, що демонструє зміну витрат у часі.

Візуалізація даних допомагає зробити висновки більш зрозумілими та зрозумілими для користувачів програми. Це особливо корисно для презентацій або аналітичних звітів.

Однією з ключових переваг Pandas є можливість автоматизувати повторювані операції. Наприклад, якщо вам потрібно регулярно очищувати дані від дублікатів або створювати зведені звіти, Pandas дозволяє налаштувати ці операції один раз і виконувати їх автоматично.

Ця функціональність значно зменшує витрати часу на обробку даних і мінімізує помилки, що виникають під час ручної роботи.

Для роботи з великими наборами даних Pandas має вбудовані інструменти оптимізації. Наприклад, бібліотека підтримує використання індексів для прискорення пошуку, а також конвертацію числових даних у компактні формати.

Це дозволяє ефективно працювати навіть із великими таблицями, не знижуючи продуктивності.

Pandas інтегрується з багатьма іншими бібліотеками Python, такими як NumPy для виконання складних математичних обчислень, Matplotlib для створення графіків, Scikit-learn для машинного навчання. Це робить Pandas універсальним інструментом для аналітичних та наукових задач.

Matplotlib: Особливості використання. Matplotlib — це одна з найпотужніших бібліотек Python для візуалізації даних, яка забезпечує широкі можливості для створення статичних, інтерактивних і навіть анімаційних графіків. Вона використовується для аналізу великих обсягів даних у фінансових, наукових і бізнесових проєктах, надаючи розробникам інструменти для перетворення числових даних у зрозумілі графічні представлення. Основним компонентом Matplotlib є модуль pyplot, який пропонує простий синтаксис для швидкого створення графіків. Концептуально бібліотека базується на об’єктно-орієнтованій структурі, де графік представляється як вікно (figure), яке може містити один або кілька підграфіків (axes). Такий підхід забезпечує модульність і дозволяє створювати складні багатошарові графіки.

Візуалізація даних у Matplotlib починається зі створення основної структури графіка. Це може бути лінійний графік для демонстрації змін у часі, стовпчаста діаграма для порівняння категорій, кругова діаграма для відображення часток або точковий графік для виявлення залежностей між змінними. Бібліотека підтримує налаштування кожного елемента графіка, включаючи стилі ліній, кольори, маркери, підписи осей, заголовки, легенди та інші елементи. Наприклад, графіки можна адаптувати для лінійної або логарифмічної шкали, змінювати стиль ліній на суцільний, пунктирний або штрихпунктирний, а також додавати текстові мітки для поліпшення читабельності.

Matplotlib підтримує імпорт даних із різних джерел, таких як масиви NumPy, таблиці Pandas або дані з файлів CSV і баз даних. Завдяки цьому графіки можуть будуватися безпосередньо з підготовлених даних. Перед побудовою графіка важливо забезпечити очищення та впорядкування даних, що спрощується

завдяки інтеграції Matplotlib із бібліотеками Pandas та NumPy. Це забезпечує безшовну взаємодію між обробкою й візуалізацією даних. Наприклад, у фінансових застосунках можна зчитати дані про витрати за категоріями з бази даних, упорядкувати їх за зростанням або зменшенням і відобразити результати у вигляді кругової діаграми.

Однією з найсильніших сторін Matplotlib є її універсальність у підтримці різних типів графіків. Лінійні графіки дозволяють відстежувати зміну показників у часі, наприклад, витрат за місяцями. Стовпчасті діаграми підходять для порівняння значень, наприклад, витрат за різними категоріями, а кругові діаграми демонструють частки у загальному обсязі, наприклад, розподіл бюджету. Крім того, точкові графіки використовуються для аналізу залежностей між змінними, наприклад, витратами і доходами, а гістограми — для вивчення розподілу даних.

Гнучкість Matplotlib дозволяє користувачам створювати кілька графіків в одному вікні, що є зручним для порівняння різних наборів даних. Наприклад, у рамках одного звіту можна побудувати лінійний графік змін витрат і стовпчасту діаграму для доходів. Такий підхід забезпечує більш комплексний аналіз. Крім того, бібліотека підтримує інтерактивність: користувач може збільшувати або зменшувати масштаб графіка, переглядати координати точок або змінювати вигляд графіка безпосередньо у вікні програми. Це особливо важливо для інтерактивних застосунків, де користувачам потрібен швидкий доступ до детальної інформації.

Ще однією унікальною особливістю Matplotlib є підтримка анімацій, які дозволяють візуалізувати динамічні зміни даних. Наприклад, можна створити анімацію, яка демонструє, як змінювалися витрати за місяцями протягом року. Анімації можуть бути збережені у вигляді відеофайлів або GIF для подальшого використання у презентаціях чи звітах. Однак, незважаючи на потужні функції візуалізації, Matplotlib може бути складною для новачків через велику кількість доступних параметрів і налаштувань. Наприклад, створення складного графіка із кількома підписами може вимагати багато часу через необхідність налаштування кожного елемента.

Matplotlib також ефективно працює з великими наборами даних. Наприклад, якщо потрібно візуалізувати зміни витрат за кілька років, бібліотека здатна обробити сотні тисяч точок без значної втрати продуктивності. Для оптимізації роботи з великими даними можна використовувати методи вибірки або агрегування, що дозволяє зменшити кількість даних, які відображаються на графіку, зберігаючи їхню інформативність.

Збереження графіків є важливою частиною роботи з Matplotlib. Бібліотека підтримує експорт у різні формати, такі як PNG, SVG або PDF, що дозволяє використовувати графіки в документах, презентаціях або веб-застосунках. Завдяки інтеграції з Tkinter графіки можна відображати безпосередньо у вікні програми, що робить Matplotlib ідеальним вибором для інтерактивних застосунків. Наприклад, у фінансових системах можна створити графік витрат, який буде автоматично оновлюватися відповідно до вибраного користувачем періоду.

Попри свої переваги, Matplotlib має обмеження. Наприклад, для створення інтерактивних графіків сучасніші бібліотеки, такі як Plotly або Seaborn, можуть бути кращим вибором. Проте для статичних візуалізацій, які потребують високого рівня деталізації, Matplotlib залишається незамінною. Бібліотека ідеально підходить для наукових досліджень, бізнес-аналізу, а також для фінансових додатків, де потрібно створювати комплексні графічні представлення даних.

Matplotlib є універсальним інструментом для візуалізації даних у Python. Її широкі можливості дозволяють створювати графіки будь-якого рівня складності, а підтримка інтеграції з іншими бібліотеками Python робить її важливим елементом багатьох сучасних програм. Завдяки простоті використання, гнучкості налаштувань і підтримці інтерактивності ця бібліотека залишається однією з найкращих для аналізу та візуалізації даних у багатьох сферах діяльності.

2.2 Проектування інтерфейсу користувача за допомогою Tkinter

Оскільки веб-сайт в першу чергу служить для залучення нових клієнтів, то інформація, яка необхідна клієнту для ухвалення рішення скористатися наданими послугами, повинна знаходитися вгорі сторінки. Цією інформацією є короткий огляд діяльності компанії, її логотип та послуги, що надаються. Також на першому екрані сторінки мають бути контактні дані компанії та форма зворотного зв'язку. Для чинних клієнтів компанії має бути форма входу в особистий кабінет користувача, де відображається список бажань та історія замовлень. Також має бути надана інформація про способи оплати та посилання на платіжні послуги. Також на першій сторінці має бути посилання на інформацію про саму компанію та її поточні новини.

Щоб інтерфейс був інтуїтивним, рекомендується розділяти робочі зони: наприклад, зліва — панель навігації з категоріями витрат, а справа — область для детального перегляду та редагування записів. Також слід дбати про те, щоб кількість кроків для виконання основної дії (ввести нову витрату або переглянути статистику) була якомога меншою. При цьому розвиток інтерфейсу може йти за методом ітерацій: спочатку створюється мінімально робочий макет, а потім поступово додаються нові компоненти (наприклад, вкладка для побудови графіків чи кнопка “Імпортувати з файлу”).

Отже, коли ми визначилися з тим, що має бути на сторінці, можна розпочати прототипування веб-сайту. Спочатку визначимося із загальною структурою веб-сайту. Це потрібно, щоб спочатку зрозуміти, яким буде Створення графічного інтерфейсу користувача є важливим етапом у розробці будь-якого застосунку, оскільки саме через нього відбувається основна взаємодія з користувачем. У Python бібліотека Tkinter пропонує зручний і універсальний інструмент для побудови графічних інтерфейсів різного рівня складності. Основною перевагою Tkinter є його інтеграція до стандартної бібліотеки Python, що позбавляє розробників необхідності встановлювати додаткові компоненти.

Tkinter працює на основі бібліотеки Tcl/Tk і дозволяє створювати інтерфейси, які виглядають нативно на більшості операційних систем.

При використанні Tkinter першим кроком завжди є створення головного вікна застосунку, яке служить контейнером для всіх елементів інтерфейсу. Основою кожного застосунку є головний цикл (mainloop), який забезпечує постійну активність вікна, реагуючи на дії користувача. Без запуску цього циклу вікно не буде відображатися, а інтерактивність елементів стане неможливою.

Однією з основних переваг Tkinter є можливість комбінування широкого набору графічних елементів: міток, кнопок, текстових полів, випадаючих списків, шкал тощо. Ці елементи дозволяють створювати як прості, так і багатофункціональні інтерфейси. Приклад створення вікна для введення тексту зображено на рис. 2.1.

```
import tkinter as tk

root = tk.Tk()
root.title("Простий інтерфейс")

label = tk.Label(root, text="Введіть ваше ім'я:")
label.pack()

entry = tk.Entry(root)
entry.pack()

button = tk.Button(root, text="Підтвердити", command=lambda: print(f"Привіт, {entry.get()}!"
button.pack()

root.mainloop()
```

Рисунок 2.1 – Просте вікно для введення тексту

У цьому прикладі мітка пояснює користувачеві, що саме необхідно ввести, текстове поле слугує для введення даних, а кнопка виконує дію — виводить введене ім'я у консоль.

Одним із ключових аспектів роботи з Tkinter є управління розташуванням елементів у вікні. Від цього залежить зручність використання застосунку. Tkinter надає кілька методів для розміщення елементів: pack, grid і place.

Метод pack використовується для автоматичного розташування елементів у вертикальному або горизонтальному порядку. Наприклад, якщо необхідно створити інтерфейс із кількома кнопками, розташованими одна під одною, pack буде ідеальним варіантом. Однак, цей метод має обмежені можливості для більш складного дизайну.

Більш гнучким є метод grid, який дозволяє організовувати елементи у вигляді таблиці. Кожен елемент розташовується у певному рядку та стовпці, що дозволяє точно контролювати їхнє розташування. Приклад використання методу grid зображено на рис. 2.2.

```
import tkinter as tk

root = tk.Tk()
root.title("Форма даних")

tk.Label(root, text="Ім'я:").grid(row=0, column=0, padx=10, pady=5)
tk.Label(root, text="Прізвище:").grid(row=1, column=0, padx=10, pady=5)

entry_name = tk.Entry(root)
entry_name.grid(row=0, column=1, padx=10, pady=5)

entry_surname = tk.Entry(root)
entry_surname.grid(row=1, column=1, padx=10, pady=5)

button = tk.Button(root, text="Зберегти", command=lambda: print(f"Ім'я: {entry_name.get()}
button.grid(row=2, column=0, columnspan=2, pady=10)

root.mainloop()
```

Рисунок 2.2 – Приклад використання методу grid

Цей підхід дозволяє легко розширювати інтерфейс, додаючи нові елементи без необхідності змінювати розташування існуючих.

Метод place використовується для точного розташування елементів за координатами. Він підходить для складних інтерфейсів із нерегулярним розташуванням елементів. Приклад використання методу place зображено на рис. 2.3.

```
entry = tk.Entry(root)
entry.place(x=50, y=50, width=200, height=30)

button = tk.Button(root, text="Натисніть")
button.place(x=100, y=100)
```

Рисунок 2.3 – Використання методу place

Однак цей метод вимагає більше часу на налаштування та часто менш зручний у підтримці.

Хоча Tkinter забезпечує базові можливості для створення GUI, його стандартний вигляд може здаватися застарілим. Щоб створити більш сучасний і привабливий інтерфейс, можна використовувати модуль ttk, який надає вдосконалені варіанти стандартних елементів. Наприклад, ttk.Button виглядає більш естетично, ніж звичайний Button.

Окрім того, Tkinter дозволяє налаштовувати кольори, шрифти та розміри елементів, що відкриває широкий простір для творчості. Наприклад, можна створити кнопку із власним стилем.

Однією з ключових функцій Tkinter є підтримка подій, які дозволяють реагувати на дії користувача. Наприклад, якщо користувач натискає клавішу або змінює значення в текстовому полі, програма може виконати певну дію. У Tkinter події обробляються через функції, які зв'язуються із відповідними елементами.

Ця гнучкість дозволяє створювати інтуїтивно зрозумілі інтерфейси, які реагують на дії користувача в режимі реального часу.

У складних застосунках, де інтерфейс має кілька розділів або вкладок, можна використовувати механізм створення вкладок через ttk.Notebook. Це дозволяє об'єднувати функціональність різних модулів у межах одного вікна.

Приклад створення вкладки для введення даних та перегляду результатів зображено на рис. 2.4.

```
from tkinter import ttk

notebook = ttk.Notebook(root)
tab1 = ttk.Frame(notebook)
tab2 = ttk.Frame(notebook)

notebook.add(tab1, text="Дані")
notebook.add(tab2, text="Результати")
notebook.pack(expand=True, fill="both")
```

Рисунок 2.4 – Створення вкладки для введення даних та перегляду результатів

2.3 Збереження та управління даними в базі SQLite

SQLite є однією з найбільш популярних реляційних баз даних у світі, що використовується для зберігання структурованих даних без необхідності в окремому сервері. Вона є вбудованою, легкою та повністю автономною системою, яка забезпечує швидкий доступ до даних навіть у масштабах великих проєктів. SQLite активно використовується в мобільних додатках, десктопних програмах і навіть у веб-розробці, оскільки її простота та ефективність відповідають потребам як новачків, так і досвідчених розробників.

SQLite є реалізацією реляційної бази даних, яка використовує SQL (Structured Query Language) для маніпуляції даними. База зберігається у вигляді звичайного файлу на диску, і всі операції з нею виконуються через цей файл. Цей підхід дозволяє легко переносити базу даних між різними платформами чи пристроями. SQLite не потребує налаштування серверного середовища, що суттєво спрощує її використання.

Приклад створення бази даних за допомогою бібліотеки sqlite3 зображено на рис. 2.5.

```python
import sqlite3

# Підключення до бази даних або створення нової
conn = sqlite3.connect('expenses.db')

# Створення курсора для виконання SQL-запитів
cursor = conn.cursor()

# Створення таблиці для зберігання даних про витрати
cursor.execute('''
    CREATE TABLE IF NOT EXISTS expenses (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT NOT NULL,
        category TEXT NOT NULL,
        amount REAL NOT NULL
    )
''')

# Збереження змін
conn.commit()

# Закриття з'єднання
conn.close()
```

Рисунок 2.5 – Створення бази даних за допомогою бібліотеки sqlite3

Цей код створює базу даних expenses.db із таблицею expenses, яка містить чотири колонки: ідентифікатор, дату, категорію витрат і суму. Використання CREATE TABLE IF NOT EXISTS дозволяє уникнути помилки, якщо таблиця вже існує.

Однією з головних переваг SQLite є її підтримка всіх основних конструкцій SQL, включаючи створення, оновлення, видалення і пошук даних. Важливим аспектом є також транзакційність — усі операції в SQLite виконуються у рамках транзакцій, що забезпечує цілісність даних навіть у разі збою. SQLite підтримує різні типи даних: INTEGER, REAL, TEXT, BLOB і NULL. Однак слід зазначити, що SQLite є менш строгим у типізації, ніж інші СУБД. Наприклад, у колонку

INTEGER можна записати текст, що дозволяє гнучкість, але може призвести до помилок, якщо дані не перевіряються ретельно.

При проєктуванні бази даних важливо визначити структуру таблиць, що відповідає логіці застосунку. Наприклад, для застосунку з обліку витрат можна створити таблиці expenses (витрати), categories (категорії витрат) і users (користувачі). Це дозволяє зберігати дані у вигляді, що легко аналізується.

Поряд із прямою роботою з SQLite через sqlite3 можна використовувати ORM (Object-Relational Mapping) бібліотеки, такі як SQLAlchemy. ORM забезпечує більшу абстракцію над SQL-запитами, дозволяючи працювати з базами даних через об'єкти Python. Приклад створення таблиці в SQLAlchemy зображено на рис. 2.6.

```python
from sqlalchemy import create_engine, Column, Integer, String, Float
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
engine = create_engine('sqlite:///expenses.db')

class Expense(Base):
    __tablename__ = 'expenses'
    id = Column(Integer, primary_key=True)
    date = Column(String, nullable=False)
    category = Column(String, nullable=False)
    amount = Column(Float, nullable=False)

Base.metadata.create_all(engine)
```

Рисунок 2.6 – Створення таблиці в SQLAlchemy

SQLAlchemy також підтримує складні запити та зв’язки між таблицями, що може бути корисним для масштабних проєктів. Однак для простих застосунків SQLite безпосередньо через sqlite3 є більш оптимальним вибором.

Для покращення продуктивності SQLite підтримує створення індексів. Індекси дозволяють швидше виконувати пошук і сортування даних у великих таблицях.

Незважаючи на те, що SQLite вважається “легкою” базою даних, вона цілком може забезпечити надійний облік великої кількості витрат, якщо проєкт не передбачає мільйони одночасних транзакцій. Її головні переваги — відсутність потреби в розгортанні окремого серверного середовища та простота переносу файлу бази даних з однієї машини на іншу. Це зручно не лише для локальних проєктів, але й для невеликих команд, де треба часто обмінюватися даними чи обслуговувати клієнтські копії програми.

Водночас важливо приділити увагу питанню резервного копіювання. Попри стабільність SQLite, будь-який файл бази даних ризикує бути пошкодженим у разі некоректного завершення роботи системи або збою на диску. Тому розробники зазвичай реалізують механізм автозбереження (наприклад, копіювання файлу .db у хмару) з певною періодичністю. Також бажано уникати надто частих операцій запису у таблицю, щоб не перевантажувати ресурс диска, і замість того частіше використовувати транзакції, де кілька операцій об’єднуються в одну.

2.4 Аналіз та обробка даних із використанням бібліотеки Pandas

Pandas є однією з найпотужніших бібліотек Python для роботи з табличними даними. Вона забезпечує широкий спектр інструментів для аналізу, обробки та трансформації даних, що робить її невід'ємною частиною програмного забезпечення, яке працює з фінансовою інформацією. У контексті автоматизації обліку витрат Pandas надає зручні можливості для роботи з великими обсягами даних, їхньої структурування та ефективного використання.

Pandas базується на двох ключових структурах даних: DataFrame та Series. DataFrame є двовимірною структурою, схожою на таблицю в базах даних або електронних таблицях, де рядки представляють записи, а стовпці — атрибути цих записів. Series, у свою чергу, є одновимірною структурою, яка часто використовується як стовпець або рядок даних у DataFrame.

У рамках роботи з фінансовими даними DataFrame дозволяє зберігати інформацію про кожну транзакцію, включаючи дату, категорію витрат, суму та інші метадані. Ці дані можна сортувати, фільтрувати, групувати, обчислювати підсумки тощо.

Pandas дозволяє імпортувати дані з різних джерел, таких як файли CSV, Excel, бази даних або навіть API. Наприклад, якщо користувач завантажив звіт про витрати у форматі CSV, бібліотека Pandas забезпечує легке завантаження цих даних у DataFrame за допомогою функції read_csv().

Практично всі сучасні системи обліку витрат включають засоби аналізу, оскільки “сирі” таблиці транзакцій не завжди дають чітку картину фінансової поведінки користувача. Використання Pandas відкриває можливість швидко формувати звіти за категоріями (продукти, проїзд, розваги та ін.), групувати витрати за днями/тижнями, порівнювати витрати за різні місяці або виявляти аномальні витрати, що значно виходять за середні показники. Простий виклик методів groupby() чи pivot_table() вже дозволяє отримувати консолідовані дані, які у вигляді звичайної SQL-вибірки формувалися б значно складніше. Приклад використання бібліотеки Pandas зображено на рис. 2.7.

```python
import pandas as pd

# Приклад даних у вигляді словника
data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35],
    "City": ["New York", "Paris", "Kyiv"]
}

# Створюємо DataFrame
df = pd.DataFrame(data)

# Відображаємо датафрейм
print(df)

# Припустимо, хочемо відфільтрувати рядки, де вік понад 28
filtered_df = df[df["Age"] > 28]
print("\nФільтровані дані, де вік понад 28:\n", filtered_df)
```

Рисунок 2.7 – Приклад використання Pandas

Для інтеграції з базою даних SQLite можна використовувати методи Pandas разом із бібліотекою SQLAlchemy. Це дає змогу зчитувати дані безпосередньо з бази даних і перетворювати їх у зручний для аналізу формат.

Аналіз фінансових даних за допомогою Pandas передбачає виконання різних операцій:

- Групування дозволяє агрегувати витрати за категоріями або за часом. Наприклад, за допомогою методу groupby() можна підрахувати суму витрат для кожної категорії або місяця.
- Pandas надає потужні інструменти для вибірки певних даних із великого масиву. Наприклад, можна вибрати всі витрати, які перевищують певну суму, або всі транзакції, пов’язані з конкретною категорією.
- Pandas підтримує роботу з датами через свій модуль datetime. Це дозволяє виконувати операції, пов’язані з датами, такі як сортування транзакцій за часом, обчислення проміжків між подіями або аналіз сезонних трендів витрат.
- Для обробки даних Pandas пропонує широкий спектр функцій для трансформації та підготовки даних.
- Під час аналізу витрат може виникнути необхідність створити нові стовпці, наприклад, «Тип витрати» або «Частка в загальних витратах». Це можна зробити простим присвоєнням значення за допомогою виразів.
- У випадках, коли в даних є пропуски (наприклад, відсутність інформації про певну транзакцію), методи dropna() або fillna() дозволяють очистити або заповнити пропуски.
- Для аналізу витрат може бути корисним сортувати дані за сумою витрат, датою або іншими параметрами. Метод sort_values() дозволяє виконати цю операцію ефективно.

Pandas добре масштабується для роботи з великими обсягами даних, що є важливим у контексті обліку витрат. Наприклад, якщо система зберігає всі транзакції за кілька років, DataFrame дозволяє швидко виконувати операції над сотнями тисяч рядків.

Крім того, для роботи з дуже великими наборами даних можна використовувати функції оптимізації, такі як завантаження даних частинами або зменшення розміру стовпців шляхом зміни їх типу (наприклад, перетворення float64 на float32).

Наприклад, припустимо, що користувач хоче визначити найбільш витратну категорію за останній місяць. За допомогою Pandas можна швидко виконати групування даних за категоріями, відфільтрувати записи за датами та обчислити загальну суму для кожної категорії. Це дозволяє отримати результат у вигляді табличних даних або передати їх для візуалізації через бібліотеку Matplotlib.

Для інтеграції Pandas із графічним інтерфейсом користувача, створеним у Tkinter, можна організувати завантаження даних із бази даних SQLite безпосередньо у DataFrame. Потім ці дані можна обробляти й показувати у вигляді таблиці або графіка.

Наприклад, після обробки фінансових даних користувачеві може бути наданий звіт із ключовими показниками: загальна сума витрат за період, найбільш витратна категорія, середня сума транзакцій тощо. Pandas забезпечує не лише аналіз, а й підготовку цих даних у зручній для відображення формі.

2.5 Візуалізація витрат за допомогою Matplotlib

Matplotlib є однією з найбільш поширених бібліотек Python для роботи з візуалізацією даних. Вона дозволяє створювати як базові графіки, так і складніші багатовимірні візуалізації. У контексті обліку витрат Matplotlib є важливим інструментом для представлення фінансових даних у вигляді графіків, діаграм або інших форм, які спрощують аналіз і прийняття рішень.

Основна перевага Matplotlib полягає в її універсальності. З її допомогою можна створювати лінійні графіки, гістограми, кругові діаграми, стовпчикові графіки та багато іншого. Крім того, бібліотека дозволяє повністю налаштовувати вигляд графіків: змінювати кольори, шрифти, підписи, стиль ліній, додавати сітки, легенди тощо.

Наприклад, у випадку автоматизації обліку витрат, Matplotlib може використовуватися для створення кругової діаграми, яка відображає розподіл витрат за категоріями, або лінійного графіка, що показує динаміку витрат за певний період. Це допомагає користувачеві швидко зрозуміти, які категорії є найбільш затратними або як змінювалися витрати з часом.

Важливим аспектом використання Matplotlib є підготовка даних. Для цього часто використовуються інші бібліотеки, наприклад Pandas, які дозволяють зручно обробляти дані та перетворювати їх у формат, готовий для візуалізації. Наприклад, перед створенням графіка витрат за категоріями необхідно згрупувати дані за цими категоріями та підрахувати загальну суму витрат для кожної з них.

Для побудови графіків Matplotlib пропонує основний об’єкт Figure, який може містити одну або кілька областей побудови (Axes). Така структура дозволяє створювати як прості, так і складні багатопанельні графіки.

Одним із найзручніших способів показати розподіл витрат за категоріями є використання кругової діаграми. Вона дозволяє інтуїтивно зрозуміти, яка частина витрат припадає на кожну категорію. Наприклад, якщо в системі зберігаються дані про витрати на їжу, транспорт та комунальні послуги, кругова діаграма покаже, яка частка бюджету використовується для кожної категорії.

У Matplotlib кругова діаграма налаштовується гнучко. Користувач може змінювати кольори, додавати підписи або навіть виділяти окремі сегменти, щоб зробити акцент на певних даних.

Аналіз змін витрат із часом є ще однією ключовою задачею. Лінійний графік, створений за допомогою Matplotlib, дозволяє наочно показати, як змінювалися витрати протягом місяця, року або іншого періоду. Користувач може

побачити піки витрат у певні дні або періоди, що може бути корисним для оптимізації бюджету.

Особливо корисною може бути функціональність “composite” графіків: наприклад, на одному полотні (figure) можна вивести лінійний графік із загальною сумою витрат по місяцях і поруч — стовпчик для порівняння з аналогічним періодом минулого року. Така наочність допомагає швидко оцінити тенденції: чи витрати “зростають”, чи “стабілізуються”. Згодом, коли розробники хочуть підвищити інтерактивність, можна перейти до бібліотек Seaborn чи Plotly, але Matplotlib залишається стабільним вибором для більш традиційних фінансових звітів і діаграм.

Окрім лінійних графіків, динаміку витрат можна представити за допомогою стовпчикових графіків. Такий підхід зручний, коли необхідно порівняти витрати в різні дні, місяці або квартали.

Matplotlib дозволяє гнучко налаштовувати всі аспекти графіка, починаючи з кольору та форми ліній і закінчуючи розташуванням підписів і легенд. Наприклад, у графіку динаміки витрат можна виділити окремі точки, що відповідають найбільшим витратам, за допомогою маркерів або підписів.

Також бібліотека дозволяє створювати складні багатопанельні графіки, які одночасно показують динаміку витрат, розподіл за категоріями та порівняння між різними періодами.

Одним із важливих моментів є інтеграція Matplotlib із графічним інтерфейсом, створеним за допомогою Tkinter. Це дозволяє виводити графіки прямо в інтерфейс програми. Наприклад, користувач може натиснути кнопку «Показати графік» і побачити відповідну діаграму у вікні програми.

Для інтеграції Matplotlib у Tkinter зазвичай використовується модуль FigureCanvasTkAgg, який дозволяє додавати графіки у вигляді віджетів. Це значно покращує користувацький досвід, оскільки надає можливість переглядати графіки безпосередньо у програмі, не звертаючись до зовнішніх вікон.

Візуалізація даних є незамінним інструментом для аналізу фінансової інформації. Завдяки Matplotlib користувач може швидко отримати уявлення про

те, як витрачаються кошти, і знайти способи оптимізації бюджету. Відображення даних у зрозумілому графічному вигляді спрощує процес ухвалення рішень, дозволяючи зосередитися на найважливіших аспектах.

Таким чином, Matplotlib є потужним і гнучким інструментом, який допомагає перетворювати числові дані на зрозумілі та інформативні візуалізації. Його можливості роблять його незамінною частиною програмного стеку для створення систем обліку витрат.

Хороша візуалізація здатна “розповісти історію” даних значно швидше, ніж довгий перелік чисел. У контексті витрат це означає, що користувач за лічені секунди може побачити, де найбільше грошей “згорає” щомісяця. Крім побудови статичних графіків, Matplotlib дозволяє створювати інтерактивні діаграми, коли при наведенні миші можна дізнатися додаткову інформацію про точку на графіку або сегмент кола.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1 Архітектура застосунку: структура та взаємодія модулів

Архітектура побудована на модульному підході, який забезпечує зрозумілий розподіл функцій між компонентами, підвищує зручність підтримки коду та спрощує його масштабування. Основним принципом модульного дизайну є розподіл функціональності між окремими файлами, кожен з яких відповідає за виконання специфічних задач. Це дозволяє легко вносити зміни в одну частину застосунку без ризику порушити роботу інших компонентів. Усі модулі працюють у тісній взаємодії, передаючи один одному дані та функції для досягнення загальної мети — забезпечення роботи системи обліку витрат.

Головним модулем застосунку є main.py, який слугує точкою входу в програму. Він виконує ініціалізацію бази даних через модуль db_manager.py, створює об'єкт графічного інтерфейсу через ui_manager.py і запускає програму, викликаючи метод run. Основна відповідальність цього модуля — забезпечити старт усіх компонентів і передати управління користувачу. Цей файл є найменш завантаженим функціональністю, оскільки його завдання полягає лише в запуску основних процесів.

Модуль ui_manager.py займає центральне місце в архітектурі застосунку, оскільки відповідає за створення графічного інтерфейсу, навігацію між вкладками та інтеграцію всіх інших модулів. Він створює основне вікно програми, задає його розміри, встановлює тему оформлення та додає меню, яке дозволяє користувачу обирати між вкладками "Витрати", "Бюджет" і "Графіки". Для кожної вкладки створюється окремий менеджер, відповідальний за її функціонал: модуль expense_manager.py керує вкладкою "Витрати", модуль budget_manager.py — вкладкою "Бюджет", а visualization.py забезпечує візуалізацію даних у вкладці "Графіки". Взаємодія між ui_manager.py та іншими модулями є двосторонньою: графічний інтерфейс передає користувацькі дії у вкладки, а ті, у свою чергу,

оновлюють відображення на основі змін у базі даних або інших джерел даних. Графічне представлення залежностей зображено на рис. 3.1.

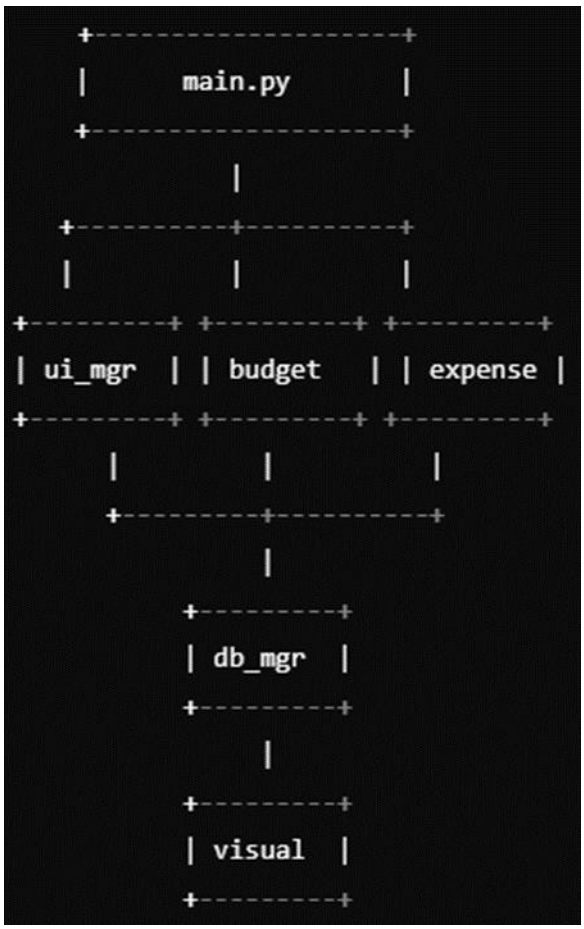


Рисунок 3.1 – Графічне представлення залежностей

Одним із ключових компонентів застосунку є модуль db_manager.py, який відповідає за роботу з базою даних SQLite. Його функціональність включає створення таблиць, додавання нових записів, видалення існуючих та виконання вибірок для отримання необхідної інформації. Цей модуль працює безпосередньо з файлами бази даних і забезпечує інтеграцію застосунку з SQL-запитами. Наприклад, при додаванні витрат через вкладку "Витрати", db_manager.py отримує виклик від модуля expense_manager.py і зберігає нові дані у таблиці expenses. Аналогічно, при отриманні інформації для побудови графіків модуль візуалізації звертається до db_manager.py для виконання запиту про розподіл витрат за категоріями або за періодами. Завдяки цьому всі дані зберігаються у

централізованому місці, що гарантує їхню узгодженість і доступність для всіх частин застосунку.

Модулі expense_manager.py та budget_manager.py відповідають за управління вкладками "Витрати" та "Бюджет" відповідно. Вони мають схожу структуру, оскільки обидва використовують таблиці для відображення даних, а також надають функціонал для додавання, редагування та видалення записів. Наприклад, у вкладці "Витрати" користувач може натиснути кнопку "Додати витрату", після чого модуль expense_manager.py відкриває вікно для введення нової витрати, викликаючи метод із модуля add_expense.py. Після введення даних ці записи передаються в db_manager.py для збереження у базі, а таблиця на екрані оновлюється для відображення нових даних. Вкладка "Бюджет" працює за аналогічним принципом, але додатково забезпечує функціонал порівняння витрат із встановленими бюджетами та виведення повідомлення, якщо витрати перевищують ліміт.

Особливу роль у застосунку відіграє модуль visualization.py, який забезпечує створення графіків для візуалізації даних. Використовуючи бібліотеку Matplotlib, він будує кругові діаграми, що відображають розподіл витрат за категоріями, або лінійні графіки, які показують динаміку витрат за певний період. Для отримання даних цей модуль звертається до db_manager.py, використовуючи SQL-запити для агрегування інформації. Наприклад, при побудові кругової діаграми витрат за категоріями модуль отримує назви категорій та відповідні їм суми, а потім відображає ці дані у вигляді графіка, який дозволяє користувачеві швидко оцінити, на що витрачається найбільше грошей. Код модуля main.py зображен на рис. 3.2.

```python
# main.py
from ui_manager import AppUI
from db_manager import DBManager

if __name__ == "__main__":
    db_manager = DBManager("expenses.db")
    app_ui = AppUI(db_manager)
    app_ui.run()
```

Рисунок 3.2 – Інтеграція модулів у main.py

3.2 Реалізація модуля введення та валідації даних витрат

Модуль введення та валідації даних витрат, що реалізований у файлі add_expense.py, є ключовим елементом у роботі системи обліку витрат. Він забезпечує користувачу інтуїтивно зрозумілий інтерфейс для введення інформації про витрати, перевіряє коректність цієї інформації та зберігає її у базі даних. Його функціональність спрямована на забезпечення максимальної зручності для користувача та запобігання помилкам, які можуть виникнути під час введення даних.

Робота модуля починається з відкриття окремого вікна, яке надає користувачу можливість ввести дату, категорію, опис і суму витрати. Це вікно створюється за допомогою бібліотеки CustomTkinter, яка дозволяє створювати сучасні графічні елементи з кастомізованими стилями. Усі поля для введення даних мають чітке призначення. Поле дати дозволяє вводити значення у форматі YYYY-MM-DD, що є загальноприйнятим форматом для зберігання дат у базах даних. Поле категорії дає змогу класифікувати витрату, наприклад, як "Їжа", "Транспорт", "Розваги" чи будь-яку іншу категорію, визначену користувачем. Поле опису надає можливість ввести додаткову інформацію про витрату, яка може допомогти краще зрозуміти її призначення. Поле суми призначене для введення числового значення, що представляє розмір витрати. Код для перевірки коректності введення даних зображено на рис. 3.3.

```
def validate_expense_data(date, category, amount):
    if not date:
        raise ValueError("Дата не може бути порожньою")
    if not category:
        raise ValueError("Категорія не може бути порожньою")
    try:
        amount = float(amount)
        if amount <= 0:
            raise ValueError("Сума має бути більше нуля")
    except ValueError:
        raise ValueError("Сума має бути числом")
    return True
```

Рисунок 3.3 – Перевірка коректності введення даних

Однією з важливих складових роботи модуля є перевірка правильності введених даних, яка здійснюється у кілька етапів. На першому етапі перевіряється, чи заповнені всі обов’язкові поля. Якщо якесь із них залишилося порожнім, користувач отримує відповідне повідомлення із проханням заповнити всі поля. Далі перевіряється формат дати, щоб він відповідав заданому шаблону. Для цього використовується функція з модуля datetime, яка дозволяє виявляти помилки у форматуванні. Якщо формат дати неправильний, система виводить повідомлення про помилку, вказуючи, що дата повинна бути введена у форматі YYYY-MM-DD. Перевірка суми полягає у визначенні того, чи введене значення є числом і чи воно більше нуля. Якщо сума не відповідає цим критеріям, користувачеві повідомляється про необхідність виправлення. Код для реалізації додавання витрат зображено на рис. 3.4.

```python
# expense_manager.py
def add_expense(self, date, category, description, amount):
    if not date or not category or not amount:
        raise ValueError("Всі поля, крім опису, є обов'язковими")
    try:
        amount = float(amount)
    except ValueError:
        raise ValueError("Сума має бути числом")

    query = "INSERT INTO expenses (date, category, description, amount) VALUES (?, ?, ?, ?)"
    self.db_manager.execute_query(query, (date, category, description, amount))
    print("Витрати успішно додано")
```

Рисунок 3.4 – Додавання витрат

Після успішного проходження всіх перевірок введені дані передаються у базу даних. Для цього використовується метод add_expense з модуля db_manager.py, який забезпечує запис даних у таблицю expenses. У цьому процесі формується SQL-запит INSERT INTO, який додає новий рядок із вказаними значеннями до бази. Наприклад, якщо користувач вводить витрату на суму 500 гривень, з датою 2024-12-20, категорією "Їжа" та описом "Ресторан", ці дані додаються у таблицю, ідентифікатором для нового запису стає унікальне значення, що генерується автоматично.

Головною перевагою цього модуля є те, що він інтегрує кілька важливих процесів в одну послідовність дій, зрозумілу для користувача. Після введення даних і натискання кнопки "Зберегти" користувач отримує повідомлення про успішне додавання витрати, а вікно закривається, повертаючи його до основного інтерфейсу застосунку. Це значно покращує взаємодію користувача із системою, забезпечуючи швидке введення інформації без зайвих дій.

Окрім базових функцій введення даних, модуль також обробляє можливі помилки, які можуть виникнути під час роботи з базою. Наприклад, якщо база даних тимчасово недоступна або не може обробити запит, користувачеві буде виведено відповідне повідомлення із пропозицією повторити спробу. Це дозволяє зберегти стабільність системи навіть у разі технічних збоїв, а також мінімізує ризик втрати введених даних.

Таким чином, модуль введення та валідації даних витрат є важливим компонентом системи, який забезпечує користувачу зручний спосіб взаємодії з базою даних. Він поєднує функціональність введення, перевірки, обробки та збереження інформації, а також інтегрується з іншими модулями, створюючи основу для подальшого аналізу даних та їхньої візуалізації.

Модуль введення та валідації даних витрат, реалізований у файлі add_expense.py, відіграє важливу роль у функціонуванні всього застосунку, оскільки забезпечує користувачеві інтерфейс для додавання нових витрат, а також здійснює перевірку правильності введених даних перед їхнім збереженням у базі. Він тісно взаємодіє з кількома іншими модулями, зокрема з expense_manager.py, db_manager.py, ui_manager.py та, опосередковано, з visualization.py. Кожна взаємодія має свою чітко визначену мету та послідовність.

Спочатку користувач взаємодіє з модулем ui_manager.py, який є основним координатором усього застосунку. Після відкриття вкладки "Витрати" за допомогою відповідного методу в ui_manager.py, відображається інтерфейс вкладки, що реалізований у модулі expense_manager.py. У цій вкладці користувач має доступ до кількох функцій, таких як перегляд таблиці витрат, редагування наявних записів і додавання нових. При натисканні на кнопку "Додати витрату" у

вкладці викликається метод open_add_expense_window модуля expense_manager.py, який ініціалізує клас AddExpense, розташований у add_expense.py. Це відкриває нове вікно, у якому користувач може ввести дані про витрати.

Під час роботи вікна додавання витрат модуль add_expense.py забезпечує перевірку введених даних. Наприклад, здійснюється перевірка формату дати, який має відповідати шаблону YYYY-MM-DD, а також перевірка, чи сума витрат є числовим значенням, більшим за нуль. Якщо будь-яке з полів заповнено некоректно, користувачу виводиться відповідне повідомлення про помилку. У разі успішного проходження валідації дані передаються у модуль db_manager.py для збереження. db_manager.py виконує функцію управління базою даних, зокрема її оновлення. Коли викликається метод add_expense, дані додаються до таблиці expenses у базі SQLite за допомогою SQL-запиту INSERT INTO. Успішне збереження даних у базі дозволяє модулю add_expense.py завершити свою роботу, закрити вікно додавання витрат та передати управління назад до expense_manager.py.

Модуль expense_manager.py після додавання нового запису до бази даних оновлює таблицю витрат, яка відображається у вкладці. Для цього він звертається до функції get_expenses у db_manager.py, яка виконує SQL-запит SELECT * FROM expenses і повертає актуальний список витрат. Отримані дані форматуються і відображаються у вигляді таблиці, що дає користувачеві змогу бачити всі введені витрати у зручному форматі.

Хоча модуль add_expense.py не взаємодіє напряму з модулем visualization.py, його робота впливає на візуалізацію даних у графіках. Усі витрати, які додаються через add_expense.py, стають доступними для модуля visualization.py, що використовує їх для побудови кругових діаграм, стовпчастих графіків або інших форм візуалізації. Для отримання даних графіки використовують методи модуля db_manager.py, такі як get_expenses_by_category, що групує витрати за категоріями і повертає агреговані суми.

Таким чином, модуль add_expense.py інтегрується в загальну архітектуру застосунку, забезпечуючи користувачу зручний інтерфейс для введення витрат, а також гарантує, що всі введені дані коректно обробляються і зберігаються. Його взаємодія з expense_manager.py дозволяє оновлювати дані у вкладці "Витрати", зв'язок із db_manager.py забезпечує роботу з базою, а вплив на visualization.py дозволяє створювати актуальні графіки, які відображають нові витрати. Ця модульна взаємодія гарантує гнучкість і масштабованість застосунку, забезпечуючи чіткий розподіл відповідальності між компонентами.

3.3 Алгоритми обробки та аналізу фінансових даних

Алгоритми обробки та аналізу фінансових даних є невід’ємною частиною функціональності системи обліку витрат. Основною метою цих алгоритмів є забезпечення точного опрацювання введених даних, їхнє структурування, а також підготовка для подальшого аналізу та візуалізації. Ці процеси охоплюють перевірку даних, їхнє сортування, агрегування, розрахунок залишків бюджету та підготовку звітів за визначеними параметрами.

Обробка даних починається з моменту, коли вони потрапляють до бази даних через модуль введення витрат або бюджетів. У таблиці expenses дані зберігаються у форматі, що включає ідентифікатор запису, дату витрати, категорію, опис і суму. Подібним чином, таблиця budget містить ідентифікатор, період і суму бюджету. Проте дані в такому вигляді є лише "сирим матеріалом", який потребує додаткової обробки, щоб стати корисним для користувача. Наприклад, суми витрат за різні дати або категорії потрібно агрегувати, щоб отримати загальну картину витрат за певний період або за визначеною категорією.

Для реалізації цих задач у застосунку використовуються алгоритми, написані за допомогою бібліотеки Pandas. Ця бібліотека дозволяє обробляти дані у вигляді таблиць, що значно полегшує виконання таких операцій, як сортування,

фільтрація, групування та підсумовування. Наприклад, для отримання загальної суми витрат за категоріями використовується функція groupby, яка групує записи за значеннями у певній колонці та підсумовує їх. Код для реалізації аналізу витрат за категоріями зображено на рис. 3.5.

```python
import pandas as pd

def analyze_expenses_by_category(db_manager):
    # Отримання даних з бази
    expenses = db_manager.get_expenses()

    # Створення DataFrame
    df = pd.DataFrame(expenses, columns=["ID", "Дата", "Категорія", "Опис", "Сума"])

    # Групування та підсумовування
    category_summary = df.groupby("Категорія")["Сума"].sum().reset_index()

    # Сортування за сумою витрат
    category_summary = category_summary.sort_values(by="Сума", ascending=False)
    return category_summary
```

Рисунок 3.5 – Аналіз витрат за категоріями

3.4 Генерація графіків і звітів для візуалізації результатів

Генерація графіків і звітів є важливим компонентом, оскільки вона надає користувачеві змогу аналізувати свої фінанси у зручному і зрозумілому вигляді. Графіки забезпечують наочність, яка значно перевищує текстові або табличні дані, а звіти дозволяють отримувати структуровану інформацію, що може бути використана для прийняття рішень. У цьому модулі реалізовано кілька ключових функцій, які використовують бібліотеку Matplotlib для створення графіків і формування візуального представлення фінансових даних.

Перш за все, для побудови графіків система використовує дані, збережені у базі даних. Ці дані включають інформацію про витрати (дата, категорія, сума) та бюджети (період і сума). Однак перед візуалізацією дані проходять попередню

обробку, яка включає їхнє групування, сортування та агрегацію. Наприклад, якщо користувач бажає побачити, на які категорії витрат він витрачає найбільше грошей, дані спочатку групуються за категоріями, а потім підраховується загальна сума витрат для кожної категорії.

Одним із найпоширеніших типів графіків, що використовується у системі, є кругова діаграма. Цей тип графіка чудово підходить для відображення розподілу витрат за категоріями. Наприклад, якщо витрати користувача за місяць розподіляються на категорії "Їжа", "Транспорт", "Розваги" та "Інше", система створює кругову діаграму, де кожен сектор відповідає певній категорії, а його розмір пропорційний сумі витрат у цій категорії. Користувач може одразу побачити, яка категорія є найбільш витратною, і прийняти рішення про необхідність оптимізації витрат. Код для створення кругової діаграми зображено на рис. 3.6.

```
import matplotlib.pyplot as plt

def plot_expense_distribution(expenses):
    categories = [expense["Категорія"] for expense in expenses]
    amounts = [expense["Сума"] for expense in expenses]

    plt.figure(figsize=(8, 8))
    plt.pie(amounts, labels=categories, autopct='%1.1f%%', startangle=140)
    plt.title("Розподіл витрат за категоріями")
    plt.show()
```

Рисунок 3.6 – Створення кругової діаграми

Окрім графіків, система дозволяє створювати текстові звіти, які можуть бути збережені у форматі CSV або PDF для подальшого використання. Наприклад, звіт може містити таблицю з усіма витратами за місяць, розділеними за категоріями, разом із підсумковими значеннями для кожної категорії. Це дозволяє користувачу легко переглядати свої витрати, навіть якщо він не має доступу до застосунку.

Звіти створюються за допомогою модуля Pandas, який дозволяє експортувати дані у формат CSV. Наприклад, щоб створити звіт про витрати за категоріями, система використовує метод to_csv для запису даних у файл. Реалізація створення звітів у форматі CSV зображено на рис. 3.7.

```python
import pandas as pd

def generate_expense_report(expenses, file_name="expense_report.csv"):
    df = pd.DataFrame(expenses)
    df.to_csv(file_name, index=False)
    print(f"Звіт збережено у файл {file_name}")
```

Рисунок 3.7 – Створення звітів у форматі CSV

Інтеграція функцій візуалізації та звітності дозволяє створити потужний інструмент для аналізу фінансових даних. Наприклад, після додавання нових витрат користувач може одразу побачити, як вони впливають на його загальний бюджет або розподіл витрат за категоріями. Крім того, система підтримує інтерактивну взаємодію: користувач може вибирати періоди або категорії для аналізу, і графіки та звіти оновлюються відповідно до вибраних параметрів.

Завдяки використанню Matplotlib для побудови графіків і Pandas для роботи з табличними даними, система забезпечує високу точність і швидкість виконання обчислень. Це робить застосунок зручним як для швидкого перегляду витрат, так і для глибокого аналізу фінансової поведінки користувача. Генерація графіків і звітів є завершальним етапом обробки даних у системі, який забезпечує користувачу доступ до зручної, зрозумілої і наочної інформації.

3.5 Інтерфейс користувача: інтеграція функціоналу з Tkinter

Інтерфейс користувача є ключовою складовою будь-якого програмного застосунку, оскільки саме через нього користувач взаємодіє з системою. Для створення інтерфейсу в застосунку обліку витрат було використано бібліотеку

Tkinter, яка входить до стандартної бібліотеки Python. Tkinter забезпечує розробнику набір інструментів для створення графічних елементів, таких як кнопки, поля вводу, таблиці та меню. Ці елементи об'єднуються в логічну структуру, яка забезпечує зручний доступ до функціоналу програми.

Інтерфейс користувача побудований у вигляді вкладок, кожна з яких відповідає за певний аспект роботи з фінансами. Наприклад, вкладка "Витрати" дозволяє додавати, редагувати та видаляти записи про витрати, а вкладка "Бюджет" надає можливість керувати бюджетом і порівнювати витрати із встановленими лімітами. Основне вікно застосунку було реалізовано за допомогою класу CTk, який є розширенням стандартного класу Tk із бібліотеки CustomTkinter. Цей клас дозволяє використовувати сучасні стилі оформлення, такі як світлі та темні теми, і підтримує налаштування розміру вікна, що робить застосунок адаптивним до потреб користувача.

Після створення основного вікна програма додає до нього меню навігації, яке забезпечує доступ до вкладок. Це меню реалізовано у вигляді кнопок, які змінюють активну вкладку, викликаючи відповідні методи для відображення їхнього функціоналу. Наприклад, натискання кнопки "Витрати" активує вкладку "Витрати", викликаючи метод show_expenses у модулі ui_manager.py. Цей метод видаляє з екрану всі попередні елементи та додає нові, що відповідають вкладці "Витрати".

Для інтеграції функціоналу вкладок з основним інтерфейсом використовуються менеджери геометрії Tkinter, такі як pack, grid та place. Вони дозволяють розміщувати елементи інтерфейсу у вікні у певному порядку. Наприклад, у вкладці "Витрати" використовується менеджер grid, який дозволяє створювати таблиці, де дані про витрати впорядковуються у стовпці "Дата", "Категорія", "Опис" і "Сума". Кожен рядок у цій таблиці представляє окрему витрату, а користувач може взаємодіяти з ними за допомогою кнопок "Додати запис", "Редагувати запис" і "Видалити запис". Створення таблиці витрат зображено на рис. 3.8.

```
def render_expenses_table(self):
    """Створення таблиці для відображення витрат."""
    self.tree = ttk.Treeview(self.main_frame, columns=("Дата", "Категорія", "Опис", "Сума"
    self.tree.heading("Дата", text="Дата")
    self.tree.heading("Категорія", text="Категорія")
    self.tree.heading("Опис", text="Опис")
    self.tree.heading("Сума", text="Сума")
    self.tree.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

    # Додавання скролбарів
    scrollbar = ttk.Scrollbar(self.main_frame, orient="vertical", command=self.tree.yview)
    self.tree.configure(yscrollcommand=scrollbar.set)
    scrollbar.grid(row=0, column=1, sticky="ns")
```

Рисунок 3.8 – Реалізація таблиці витрат

Однією з особливостей інтерфейсу є його інтерактивність. Наприклад, коли користувач натискає на кнопку "Додати запис", відкривається нове вікно, у якому він може ввести дані про витрату. Це вікно створюється як дочірнє вікно основного інтерфейсу за допомогою класу CTkToplevel. Усі елементи цього вікна, такі як поля вводу та кнопка "Зберегти", організовані таким чином, щоб забезпечити інтуїтивно зрозумілий досвід роботи. Створення вікна для додавання запису зображено на рис. 3.9.

```
def open_add_expense_window(self):
    """Відкрити вікно для додавання нової витрати."""
    self.add_window = ctk.CTkToplevel()
    self.add_window.title("Додати витрату")
    self.add_window.geometry("400x300")

    ctk.CTkLabel(self.add_window, text="Дата:").pack(pady=5)
    self.date_entry = ctk.CTkEntry(self.add_window)
    self.date_entry.pack(pady=5)

    ctk.CTkLabel(self.add_window, text="Категорія:").pack(pady=5)
    self.category_entry = ctk.CTkEntry(self.add_window)
    self.category_entry.pack(pady=5)

    ctk.CTkLabel(self.add_window, text="Опис:").pack(pady=5)
    self.description_entry = ctk.CTkEntry(self.add_window)
    self.description_entry.pack(pady=5)

    ctk.CTkLabel(self.add_window, text="Сума:").pack(pady=5)
    self.amount_entry = ctk.CTkEntry(self.add_window)
    self.amount_entry.pack(pady=5)
```

Рисунок 3.9 – Створення вікна для додавання запису

У вкладці "Бюджет" інтеграція функціоналу реалізована аналогічним чином. Користувач може додавати нові записи про бюджет, редагувати існуючі або видаляти їх, використовуючи зручний інтерфейс. Наприклад, при редагуванні бюджету відкривається окреме вікно, де можна змінити період і суму, після чого нові дані автоматично оновлюються у таблиці.

Завдяки використанню Tkinter і CustomTkinter вдалося створити інтерфейс, який поєднує простоту використання з сучасним дизайном. Усі елементи розташовані так, щоб забезпечити логічну послідовність дій, а інтерфейс адаптується до різних розмірів екранів. Це дозволяє користувачу легко орієнтуватися у програмі та максимально ефективно використовувати її функціонал.

3.6 Розробка та структура бази даних SQLite

База даних є ключовим компонентом застосунку для обліку витрат, оскільки вона забезпечує зберігання, управління та доступ до даних про витрати, бюджети та інші фінансові операції. Для реалізації бази даних у застосунку використовується SQLite — легка реляційна система управління базами даних (СУБД), яка не вимагає налаштування серверного середовища. SQLite відрізняється простотою інтеграції, високою швидкістю роботи та низькими вимогами до ресурсів, що робить її ідеальним вибором для невеликих застосунків.

У структурі бази даних застосунку визначено дві основні таблиці: expenses та budget. Таблиця expenses використовується для зберігання даних про витрати, а таблиця budget — для збереження інформації про бюджети. Кожна таблиця має свою чітко визначену структуру, яка відповідає вимогам застосунку. Наприклад, таблиця expenses містить поля id, date, category, description та amount, які забезпечують зберігання ідентифікатора витрати, дати її здійснення, категорії, опису та суми відповідно. Таблиця budget, у свою чергу, містить поля id, period та

amount, які відповідають ідентифікатору, періоду бюджету та його сумі. Створення структури таблиць зображено на рис. 3.10.

```
import sqlite3

def initialize_database():
    conn = sqlite3.connect("expenses.db")
    cursor = conn.cursor()

    # Створення таблиці витрат
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS expenses (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT NOT NULL,
        category TEXT NOT NULL,
        description TEXT,
        amount REAL NOT NULL
    )
    ''')

    # Створення таблиці бюджету
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS budget (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        period TEXT NOT NULL,
        amount REAL NOT NULL
    )
    ''')

    conn.commit()
    conn.close()
```

Рисунок 3.10 – Створення структури таблиць

Ця структура забезпечує базові вимоги до зберігання даних, але також може бути легко розширена у разі необхідності додавання нових функцій або полів.

Для роботи з базою даних у застосунку використовується окремий модуль db_manager.py, який реалізує функціонал додавання, видалення, оновлення та вибірки даних. Наприклад, метод add_expense додає новий запис до таблиці expenses, а метод get_expenses повертає всі записи у вигляді списку. Це забезпечує гнучкість у роботі з базою даних і дозволяє легко інтегрувати її з іншими модулями.

Важливою частиною роботи з базою даних є забезпечення її цілісності та коректності. Для цього застосунок виконує перевірку даних перед їхнім додаванням у базу. Наприклад, усі дати перевіряються на відповідність формату YYYY-MM-DD, а суми витрат мають бути числовими і більше нуля. Крім того, застосунок підтримує механізм оновлення даних. Наприклад, користувач може редагувати вже існуючі записи про витрати, змінюючи їхню категорію, суму чи опис.

Для зручності аналізу даних база підтримує SQL-запити, які дозволяють групувати, фільтрувати та агрегувати дані. Наприклад, для обчислення загальної суми витрат за категоріями застосовується SQL-запит із функцією GROUP BY, який повертає підсумкові значення для кожної категорії.

Безпека бази даних також є важливим аспектом. Усі запити до бази виконуються через параметризовані SQL-запити, що дозволяє уникнути SQL-ін'єкцій і гарантує безпечну роботу з даними. Крім того, у системі передбачено механізм створення резервних копій бази, який дозволяє зберігати копії файлу expenses.db у безпечному місці для уникнення втрати даних у разі збою.

Розробка бази даних і її інтеграція з іншими компонентами системи забезпечує її стабільність і ефективність. Завдяки використанню SQLite застосунок може обробляти дані швидко та надійно, що є важливим для забезпечення позитивного досвіду користувача. Така архітектура дозволяє не лише зберігати дані, а й ефективно їх використовувати для побудови звітів, аналізу витрат і планування бюджету.

3.7 Огляд кінцевого продукту

Відкривши застосунок на екрані з’явиться головна таблиця (рис. 3.11), яка містить у собі інформацію надану користувачем.

Id	Date	Category	Description	Amount
4	2024-12-22	Розваги	-	99999.0
5	2024-12-30	Продукти		88888.0
6	2024-12-02	Інше		77777.0

Рисунок 3.11 – Головна таблиця

Таблиця містить колонки: ID, Date, Category, Description, Amount, які користувач самостійне заповнює інформацією. Щоб додати запис до таблиці, зверху знаходяться поля (рис. 3.12), у які потрібно внести інформацію. Коли користувач увів вхідні дані, треба натиснути “Додати”, після цього запис з’явиться у таблиці.

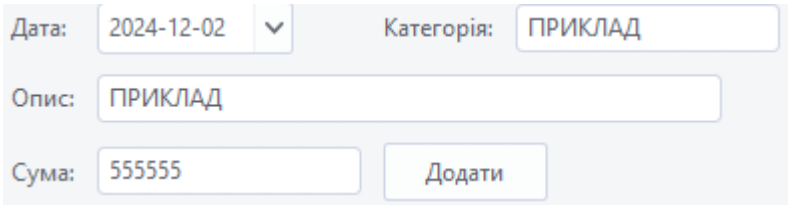


Рисунок 3.12 – Поля для заповнення

Результат додавання запису у таблицю зображено на рис. 3.13.

Id	Date	Category	Description	Amount
5	2024-12-30	Продукти		88888.0
6	2024-12-02	Інше		77777.0
7	2024-12-02	ПРИКЛАД	ПРИКЛАД	99999.0

Рисунок 3.13 – Результат додавання запису

За потреби відредагувати таблицю, знизу знаходяться 3 функіональні кнопки (рис. 3.14), які дають можливість видалити та відредагувати запис або оновити таблицю.

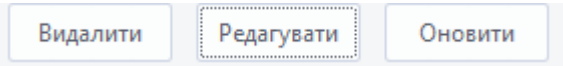


Рисунок 3.14 – Взаємодія з таблицею

Результат видалення запису “Приклад” з таблиці зображено на рис. 3.15

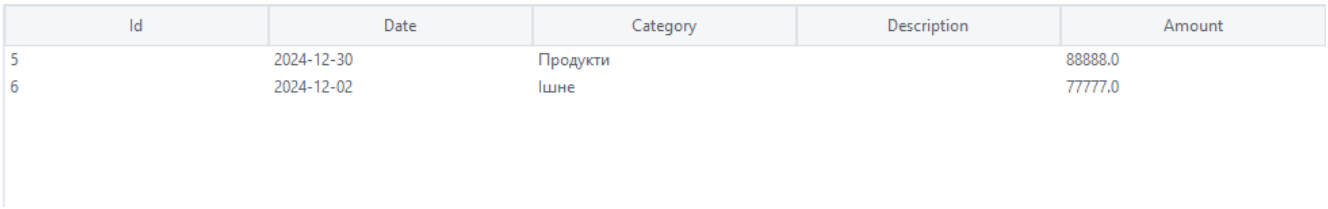


Рисунок 3.15 – Видалення запису з таблиці

У випадку, якщо потрібно відсортувати записи у таблиці за категоріями або порядковим номер, треба звернутись до блоку, який містить цей функціонал (рис. 3.16).

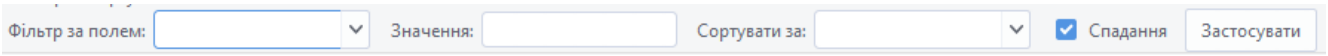


Рисунок 3.16 – Сортування записів

Результат сортування за витратами зображено на рис. 3.17.

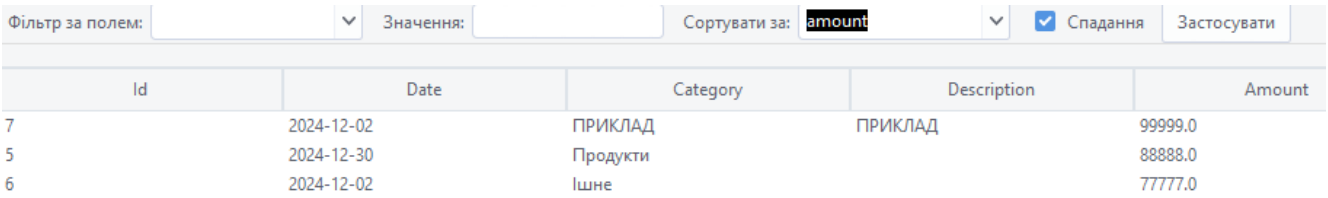


Рисунок 3.17 – Сортування за витратами

Додатково є можливість встановити бюджет (рис. 3.18) на конкретний період часу, що дозволить раціональніше використовувати фінанси та відстежити, скільки грошей залишилось.

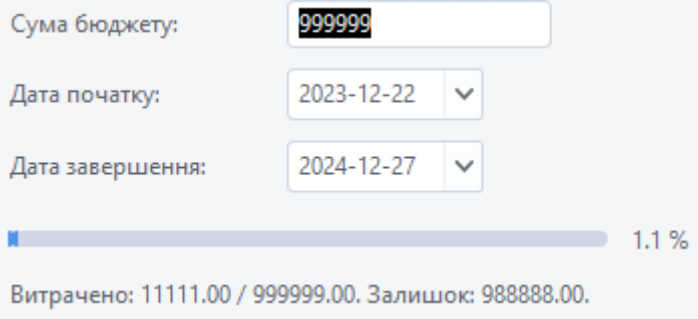


Рисунок 3.18 – Відстеження бюджету

Для більш зручного відстеження витрат можна відтворити їх зображення у вигляді даіграми (рис. 3.19). Це дозволить більш точніше оглянути та проаналізувати, на які потреби і скільки грошей було витрачено.

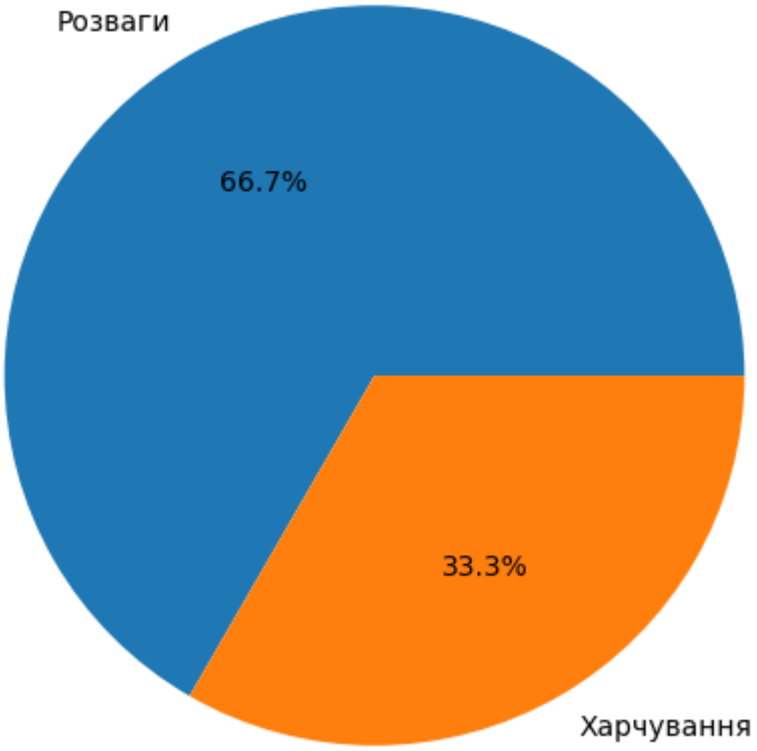


Рисунок 3.19 – Діаграма витрат

Додатково є можливість експортувати дані у CSV та Excel (рис. 3.20).

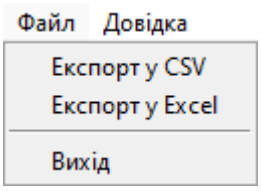


Рисунок 3.20 – Експорт даних

Результат експортування даних зображено на рис. 3.21.

	A	B	C	D
1	id,date,category,description,amount			
2	1,2024-12-09,Example1,,11111.0			
3	2,2024-12-17,Example2,,22222.0			
4	3,2024-12-22,Example3,,222222.0			

Рисунок 3.21 – Експортовані дані

ВИСНОВКИ

У даній дипломній роботі було розглянуто тему розробки застосунку для обліку витрат та управління бюджетом із використанням мови програмування Python та бібліотек Tkinter, SQLite, Pandas і Matplotlib. Під час дослідження проведено аналіз сучасних підходів до автоматизації фінансового обліку та вивчено особливості залучених технологій.

Було проведено огляд аналогів. Незважаючи на широкий вибір сучасних застосунків, більшість із них мають низку недоліків. Основні проблеми включають складність інтерфейсу для нових користувачів, обмежені можливості для персоналізації та прив’язаність до інтернет-з’єднання для роботи. Крім того, багато аналогічних програм є платними або містять рекламу, що знижує їхню привабливість для користувачів

Розробка застосунку включала створення графічного інтерфейсу для взаємодії користувача, локального збереження даних у базі SQLite, обробки фінансової інформації за допомогою Pandas і її візуалізації через Matplotlib. Було реалізовано ключові функції: додавання, редагування та видалення записів, контроль бюджету із попередженнями про перевищення ліміту, а також створення графіків і діаграм для детального аналізу витрат.

Результатом роботи став повноцінний, функціональний і зручний у використанні застосунок, який забезпечує ефективне управління фінансами. Завдяки модульній архітектурі програма є легкою для розширення та адаптації до нових вимог.

Отримані результати свідчать про те, що використання сучасних технологій дозволяє створювати продуктивні, динамічні та інтуїтивно зрозумілі програмні продукти. Застосунок для обліку витрат може бути адаптований для використання в інших фінансових задачах або інтегрований із хмарними платформами для розширення його функціоналу. Ця робота також закладає основу для подальших досліджень і вдосконалення методів автоматизації фінансового обліку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lewis J. Introduction to Data Visualization: Matplotlib for Beginners. – New York: Data Insights Press, 2021.
2. Петров А.В. Програмування графічних інтерфейсів на Python: приклади використання Tkinter. – К.: Техніка, 2020.
3. Іванов С.О., Коваленко Т.М. Аналіз сучасних інструментів для візуалізації даних у Python. // Вісник університету, 2022.
4. Tkinter User Interface Programming [Електронний ресурс] // Режим доступу до ресурсу: https://realpython.com/python-gui-tkinter/
5. SQLite Documentation [Електронний ресурс] // Режим доступу до ресурсу: https://www.sqlite.org/docs.html
6. Pandas User Guide [Електронний ресурс] // Режим доступу до ресурсу: https://pandas.pydata.org/docs/user_guide/index.html
7. Matplotlib Overview [Електронний ресурс] // Режим доступу до ресурсу: https://matplotlib.org/stable/users/overview.html
8. Chen J. Data Analysis with Python: Examples with Pandas and Matplotlib. – Sydney: Tech Press, 2021.
9. Building Expense Tracker App with Python [Електронний ресурс] // Режим доступу до ресурсу: https://towardsdatascience.com/building-expense-tracker-app-with-python-abc123456
10. Васильєв І.О. Впровадження автоматизованих систем обліку витрат: огляд існуючих рішень. // Збірник доповідей науково-технічної конференції, 2021.
11. Schneider C. Tkinter Advanced Concepts: Building Modern UI. – Berlin: Springer, 2020.
12. Green D. Financial Data Analysis with Python. – Kyiv: Educational Technologies, 2022.

ДОДАТОК А

Код головного модуля

```
# main.py
import tkinter as tk
from ttkthemes import ThemedTk
from database import create_tables, create_user, check_user_credentials
from ui import create_ui

def main():
    create_tables()
    show_login_window()

def show_login_window():
    login_window = tk.Tk()
    login_window.title("Авторизація")

    tk.Label(login_window, text="Логін:").grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)
    username_entry = tk.Entry(login_window)
    username_entry.grid(row=0, column=1, padx=5, pady=5)

    tk.Label(login_window, text="Пароль:").grid(row=1, column=0, padx=5, pady=5, sticky=tk.W)
    password_entry = tk.Entry(login_window, show="*")
    password_entry.grid(row=1, column=1, padx=5, pady=5)

    info_label = tk.Label(login_window, text="", fg="red")
    info_label.grid(row=2, column=0, columnspan=2, pady=5)

    def try_login():
        username = username_entry.get().strip()
        password = password_entry.get().strip()
        if not username or not password:
            info_label.config(text="Введіть логін і пароль!")
            return
        user_id = check_user_credentials(username, password)
        if user_id:
            login_window.destroy()
            show_main_ui(user_id, username)
        else:
            info_label.config(text="Невірний логін або пароль!")

    def try_register():
        username = username_entry.get().strip()
        password = password_entry.get().strip()
        if not username or not password:
            info_label.config(text="Заповніть поля!")
            return
        new_id = create_user(username, password)
        if new_id is None:
            info_label.config(text="Користувач із таким логіном уже існує!")
        else:
```

```
            info_label.config(text=f"Реєстрацію завершено (user_id={new_id})!")

    tk.Button(login_window, text="Увійти", command=try_login).grid(row=3, column=0, padx=5,
pady=5)
    tk.Button(login_window, text="Реєстрація", command=try_register).grid(row=3, column=1,
padx=5, pady=5)

    login_window.mainloop()

def show_main_ui(user_id, username):
    root = ThemedTk(theme="arc")
    root.title(f"Expense Tracker - {username}")

    def logout_callback():
        # Закриваємо головне вікно і повертаємося до форми логіну
        root.destroy()
        show_login_window()

    create_ui(root, user_id, logout_callback, style_theme="arc")
    root.mainloop()

if __name__ == "__main__":
    main()
```

ДОДАТОК Б

Демонстраційні матеріали

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

1

ДИПЛОМНА РОБОТА
на ступінь вищої освіти магістр
із спеціальності 122 Комп'ютерні науки

Система автоматизації процесу обліку грошових витрат за допомогою програмного застосунку на основі Python

Виконав: студент 6 курсу, групи КНДМ -61
Маламен Євген Миколайович
Керівник: к.т.н., доцент Сєрих С.О.

Київ - 2024

2

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ДИПЛОМНОЇ РОБОТИ

Мета дослідження	аналіз сучасних методів автоматизації фінансових процесів, огляд сучасних підходів до створення застосунків для обліку витрат, розробка програмного застосунку для обліку фінансових витрат із використанням Python та його бібліотек .
Об'єкт дослідження	автоматизовані системи обліку фінансових витрат
Предмет дослідження	розробка програмного застосунку для обліку витрат
Методи дослідження	мова програмування Python, бібліотеки Tkinter, SQLite, Pandas, Matplotlib

3

АКТУАЛЬНІСТЬ ТЕМИ

Сьогодні автоматизація фінансових процесів є важливою складовою ефективного управління коштами. У багатьох людей виникає потреба в інструментах для точного обліку витрат і контролю бюджету, які дозволяють уникнути помилок і раціоналізувати витрати.

Фінансові програми допомагають:

- **Вести облік витрат і доходів.**
- **Аналізувати фінансові звички.**
- **Візуалізувати дані для кращого розуміння.**

4

ОСНОВНІ ВИМОГИ ДО ЗАСТОСУНКУ

Зручність використання:

•Інтуїтивно зрозумілий інтерфейс для користувачів без технічних знань.

•Простота навігації між функціональними розділами (витрати, бюджет, аналіз).

Надійність:

•Стабільна робота програми без помилок і збоїв.

Функціональність:

•Додавання, редагування та видалення записів про витрати.

•Ведення щомісячного бюджету з можливістю попередження про перевищення ліміту.

•Побудова графіків і звітів для аналізу фінансових даних.

Адаптивність:

•Робота на різних операційних системах (Windows, macOS).

•Можливість масштабування для мобільних пристроїв у майбутньому.

Легкість у впровадженні:

•Відсутність потреби в складних налаштуваннях.

•Можливість швидкого розгортання та інтеграції в робочі процеси.

ЕТАПИ СТВОРЕННЯ

5

Аналіз вимог:

•Визначення цільової аудиторії та її потреб.

•Формулювання основних функціональних вимог до застосунку.

Планування та проектування:

•Розробка архітектури програми: поділ на модулі (інтерфейс, база даних, аналітика).

•Проектування бази даних для зберігання витрат і бюджету.

Розробка:

•Створення інтерфейсу користувача за допомогою Tkinter.

•Реалізація бізнес-логіки: облік витрат, ведення бюджету, аналітика.

•Інтеграція з базою даних SQLite та бібліотеками Pandas і Matplotlib.

Тестування:

•Перевірка функціональності кожного модуля.

•Виправлення помилок і оптимізація роботи програми.

•Оцінка зручності інтерфейсу користувачами.

Реліз та впровадження:

•Розгортання застосунку та підготовка інструкцій для користувачів.

•Моніторинг роботи програми після запуску.

АНАЛІЗ ПРОГРАМНОГО СТЕКУ

6

Мова програмування Python займає провідні позиції серед інструментів для створення програмного забезпечення. Вона відома своєю універсальністю, легкістю у використанні та можливістю застосування в різних галузях — від веб-розробки до автоматизації бізнес-процесів.

6.1 Результат опитування користувачів Reddit за 2024 рік

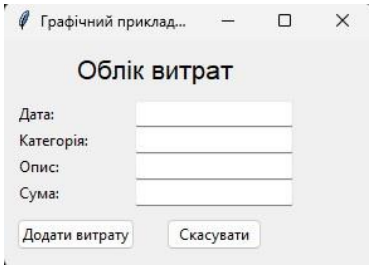
АНАЛІЗ ПРОГРАМНОГО СТЕКУ

7

Tkinter — це стандартна бібліотека Python для створення графічного інтерфейсу користувача (GUI). Вона дозволяє розробляти вікна, кнопки, текстові поля, меню та інші елементи інтерфейсу. Використовується для забезпечення інтуїтивної взаємодії користувача із застосунком.

Основні переваги:

- **Простота використання**
- **Кросплатформеність**
- **Гнучкість**
- **Широкий набір віджетів**
- **Підтримка графіки**
- **Інтеграція з іншими бібліотеками**



7.1 Створення простого вікна

АНАЛІЗ ПРОГРАМНОГО СТЕКУ

8

Pandas — це потужна бібліотека Python для обробки та аналізу даних. Вона дозволяє працювати з табличними даними (DataFrame), виконувати сортування, фільтрацію, групування, обчислення та багато іншого. Pandas широко використовується в науці про дані, фінансах та розробці програмного забезпечення.

Основні переваги

- **Інтуїтивно зрозумілий інтерфейс — робота з таблицями, подібна до електронних таблиць (Excel).**
- **Потужні можливості обробки даних — злиття, фільтрація, групування, агрегація.**
- **Підтримка великих обсягів даних — ефективна робота навіть із великими наборами даних.**
- **Інтеграція з іншими бібліотеками — легко поєднується з Matplotlib, NumPy та іншими бібліотеками.**
- **Обробка часо́вих рядів — аналіз і модифікація даних, пов'язаних із часом.**
- **Автоматизація рутинних завдань — спрощує повторювані процеси аналізу.**

АНАЛІЗ ПРОГРАМНОГО СТЕКУ

9

SQLite — це легка реляційна база даних, яка не потребує серверної інфраструктури. Вона є вбудованою в більшість мов програмування, включаючи Python, і дозволяє працювати з даними швидко та ефективно. SQLite використовується для локального зберігання інформації в невеликих проєктах.

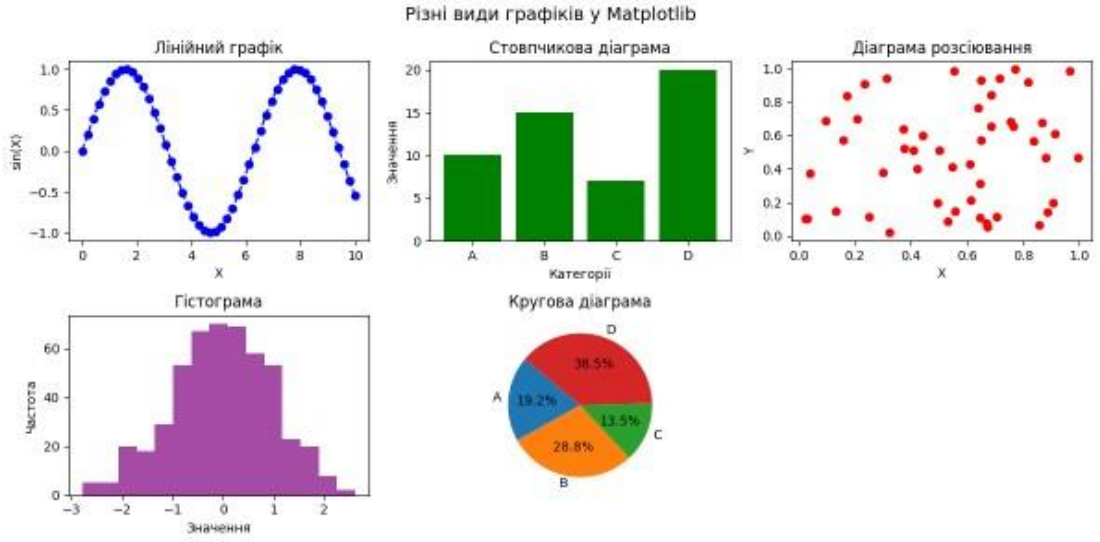
СУБД можна застосовувати в різних областях і для різних типів проектів завдяки легкості, простоті та вбудованості. Варіанти використання SQLite:

- **база даних є популярним вибором для розробки мобільних застосунків на платформах Android та iOS. СУБД підходить для зберігання локальних даних (налаштувань, списків контактів, журналів подій тощо);**
- **SQLite можна застосовувати для розробки невеликих десктопних застосунків, які не вимагають значних серверних баз даних. Наприклад, це може бути програма для управління списком завдань або простий редактор;**
- **SQLite ідеально підходить для вбудованих систем і пристроїв: планшетів, телевізорів, роутерів, GPS-трекерів тощо. Він дозволяє легко зберігати та обробляти дані, не навантажуючи систему;**
- **СУБД використовують для зберігання даних у веб-браузерах. Наприклад Google Chrome застосовує SQLite для зберігання історії перегляду та закладок;**
- **SQLite підходить для створення систем управління даними та кешування даних для веб-додатків;**

АНАЛІЗ ПРОГРАМНОГО СТЕКУ

10

Matplotlib — це одна з найпотужніших бібліотек Python для візуалізації даних, яка забезпечує широкі можливості для створення статичних, інтерактивних і навіть анімаційних графіків. Вона використовується для аналізу великих обсягів даних у фінансових, наукових і бізнесових проєктах, надаючи розробникам інструменти для перетворення числових даних у зрозумілі графічні представлення.



9.1 Створення графіків за допомогою Matplotlib

РОЗРОБКА ЗАСТОСУНКУ

11

Застосунок побудований на модульній архітектурі . Основні модулі:

- **Інтерфейс (Tkinter): забезпечує взаємодію користувача із системою через вкладки для управління витратами , бюджетом і графіками.**
- **База даних (SQLite): використовується для збереження витрат і бюджетів.**
- **Обробка даних (Pandas): для аналізу витрат за категоріями та періодами.**
- **Візуалізація (Matplotlib): створює графіки для аналізу фінансів.**

Ключові функції: додавання, редагування та видалення записів, контроль бюджету, побудова графіків витрат .
Архітектура забезпечує простоту, інтерактивність і гнучкість у роботі з фінансами.

12

ВИСНОВКИ

У даній дипломній роботі було розглянуто тему розробки застосунку для обліку витрат та управління бюджетом із використанням мови програмування Python та бібліотек Tkinter, SQLite, Pandas і Matplotlib. Під час дослідження проведено аналіз сучасних підходів до автоматизації фінансового обліку та вивчено особливостізалучених технологій.

Було проведено огляд аналогів. Незважаючи на широкийвибір сучасних застосунків, більшістьіз них мають низку недоліків. Основні проблеми включають складність інтерфейсу для нових користувачів, обмежені можливості для персоналізації та прив'язаність до інтернет-з'єднання для роботи. Крім того, багато аналогічних програм є платними або містять рекламу, що знижує їхню привабливість для користувачів

Розробка застосунку включала створення графічного інтерфейсу для взаємодії користувача, локального збереження даних у базі SQLite, обробки фінансової інформації за допомогою Pandas і її візуалізації через Matplotlib. Було реалізовано ключові функції: додавання, редагування та видалення записів, контроль бюджету із попередженнями про перевищення ліміту, а також створення графіків і діаграм для детального аналізу витрат.

Результатом роботи став повноцінний, функціональний і зручний у використанні застосунок, який забезпечує ефективне управління фінансами. Завдяки модульній архітектурі програма є легкою для розширення та адаптації до нових вимог.

Отримані результати свідчать про те, що використання сучасних технологій дозволяє створювати продуктивні, динамічні та інтуїтивно зрозумілі програмні продукти. Застосунок для обліку витрат може бути адаптований для використання в інших фінансових задачах або інтегрований із хмарними платформами для розширення його функціоналу. Ця робота також закладає основу для подальших досліджень і вдосконалення методів автоматизації фінансового обліку.