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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Застосунок на мові Python»

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

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

_____ Оліфіренко О. В.
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КНД-42
Оліфіренко Олександр Вікторович
Ім'я, ПРІЗВИЩЕ
Керівник: Василенко Володимир Вікторович
Науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма _____

ЗАТВЕРДЖУЮ

Завідувач кафедрою В.В.Вишнівський

Ім'я, ПРІЗВИЩЕ

«____» _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Оліфіренко Олександр Вікторович

(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: "Застосунок на мові Python"
керівник кваліфікаційної роботи к.т.н., доцент Василенко В. В.,

(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56

2.Строк подання кваліфікаційної роботи «31» травня 2025 р.

3.Вихідні дані до кваліфікаційної роботи: Науково-технічна література з питань, що пов'язані з розробкою застосунку

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Визначення цілей та функцій месенджер
2. Визначення цільової аудиторії
3. Вибір платформи та технологій

5.Перелік ілюстративного матеріалу: *презентація*

6.Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічгої літератури	25.02-27.02	Виконано
2	Дослідження поставленої задачі	27.02-02.03	Виконано
3	Аналіз методів розв’язування задачі	03.03-07.03	Виконано
4	Застосування застосунку на практиці	09.03-14.03	Виконано
5	Аналіз отриманих результатів	16.03-21.03	Виконано
6	Розробка прототипу	23.03-20.04	Виконано
7	Вступ, висновки, реферат	24.04-09.05	Виконано
8	Основні розділи	12.05-19.05	Виконано
9	Попередній захист роботи	26.05-28.05	Виконано

Здобувач(ка) вищої освіти _____ ___Оліфіренко О.В.___

(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник

кваліфікаційної роботи _____ ____Василенко В.В._____

(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра (магістра): 67 стор., 3 рис., 22 джерел.

Мета роботи – дослідити етапи, інструменти та принципи створення застосунку для ведення обліку особистих фінансів, який забезпечує автономність, безпеку, зручність та адаптацію під потреби користувача.

Об’єкт дослідження – програмні засоби для розробки застосунку обліку особистих фінансів на основі мови програмування Python.

Предмет дослідження – процес розробки програмного застосунку особистої бухгалтерії з використанням Python, Tkinter і SQLite.

Методи дослідження – аналіз функціональних потреб користувачів, аналіз ринку програмних рішень, об'єктно-орієнтоване програмування, проєктування графічного інтерфейсу, робота з локальними базами даних, тестування користувацького досвіду.

Сфера застосування – персональне та мале бізнес-використання для обліку доходів, витрат, формування фінансової звітності та підвищення фінансової грамотності користувачів.

Розроблений застосунок дозволяє користувачам здійснювати облік фінансових операцій у локальному середовищі без підключення до Інтернету, забезпечуючи високий рівень конфіденційності та гнучке налаштування функціональності відповідно до індивідуальних потреб. Серед ключових функцій – ведення транзакцій, категоризація витрат і доходів, візуалізація даних, експорт інформації та захист доступу.

КЛЮЧОВІ СЛОВА: особиста бухгалтерія, Python, застосунок, облік фінансів, Tkinter, SQLite, програмування, конфіденційність.

ЗМІСТ

ВСТУП

У сучасному світі, що стрімко змінюється під впливом технологічного прогресу, питання особистої фінансової стабільності та грамотного управління грошовими ресурсами набувають особливої актуальності. Економічні кризи, зростання рівня інфляції, коливання валютних курсів та інші фактори роблять фінансову безпеку однією з головних складових життя сучасної людини. У цьому контексті здатність контролювати власні фінанси, планувати бюджет, здійснювати аналіз витрат і доходів — це не лише необхідна життєва навичка, а й основа для досягнення особистих та професійних цілей.

Проте, незважаючи на усвідомлення важливості фінансового обліку, багато людей стикаються з труднощами при його реалізації на практиці. Зокрема, поширеними проблемами є відсутність зручних інструментів, брак часу на ведення записів, нерозуміння структури власних витрат, а також недостатній рівень мотивації до фінансового аналізу. У зв’язку з цим особливого значення набувають програмні засоби, що дозволяють автоматизувати процес ведення особистої бухгалтерії, роблячи його доступним і зрозумілим для широкого кола користувачів.

Саме з метою вирішення цих завдань у рамках даної дипломної роботи було поставлено завдання розробити програмний застосунок для контролю особистих фінансів. Обрана платформа реалізації — мова програмування Python — є однією з найпопулярніших у світі завдяки своїй простоті, гнучкості та наявності великої кількості бібліотек, що значно полегшують створення як простих, так і складних програмних рішень. Python дозволяє ефективно працювати з даними, створювати зручні графічні інтерфейси, а також реалізовувати інтерактивні функціональні можливості, що робить його ідеальним вибором для розробки персонального фінансового менеджера.

Основна мета створеного застосунку — надати користувачеві зручний інструмент для ведення обліку доходів і витрат, формування статистики за категоріями, візуалізації фінансових потоків, а також отримання аналітичної інформації для прийняття обґрунтованих рішень. Застосунок повинен стати

своєрідним «фінансовим щоденником», що дозволить користувачу легко вносити дані, переглядати історію операцій, формувати щомісячні звіти та аналізувати динаміку фінансового стану.

Актуальність теми дипломної роботи зумовлена як загальним розвитком програмного забезпечення у сфері особистих фінансів, так і зростаючими вимогами до зручності, функціональності та наочності таких рішень. У сучасному інформаційному просторі існує чимало мобільних додатків та веб-сервісів для бюджетування, проте значна частина з них вимагає підключення до Інтернету, зберігає дані на сторонніх серверах, що може викликати занепокоєння з приводу конфіденційності. Запропонований застосунок орієнтований на локальне використання, що забезпечує вищий рівень безпеки та конфіденційності фінансової інформації користувача.

Предметом дослідження є процеси обліку, аналізу та візуалізації фінансових даних з метою створення зручного програмного інструменту для щоденного використання. Об’єктом дослідження є методи та засоби програмування, що дозволяють реалізувати поставлені функціональні вимоги до застосунку.

Реалізація програмного продукту передбачає використання таких ключових інструментів Python, як:

- бібліотека Tkinter для створення графічного інтерфейсу користувача;
- sqlite3 для збереження даних у локальній базі даних;
- модулі для обробки та форматування дати й часу, а також для валідації введених даних.

У рамках дипломного проєкту розглядатиметься повний цикл створення застосунку: від аналізу предметної області та постановки завдань, до проектування інтерфейсу, реалізації програмного коду, тестування та оцінки ефективності створеного продукту.

Очікується, що результати дипломної роботи матимуть як практичне, так і навчальне значення. З одного боку, створений застосунок може використовуватись будь-якою особою, зацікавленою у веденні особистої бухгалтерії. З іншого боку, процес його розробки дозволяє продемонструвати набуті навички у сфері об'єктно-

орієнтованого програмування, розробки інтерфейсів, роботи з базами даних, візуалізації інформації, а також навички системного аналізу та організації проєктної діяльності.

Таким чином, обрана тема дипломної роботи є не лише актуальною, але й практично орієнтованою, що дозволяє поєднати навчальні цілі з реальними потребами сучасного користувача. Успішна реалізація проєкту сприятиме підвищенню фінансової грамотності, створенню здорових фінансових звичок та забезпеченню більшої прозорості в особистих бюджетах користувачів.

1 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ ОБЛІКУ ОСОБИСТИХ ФІНАНСІВ В ІТ-СЕРЕДОВИЩІ

1.1Сучасні цифрові технології управління особистими фінансами

У XXI столітті, коли цифровий світ проникає у всі сфери життя, управління особистими фінансами зазнає суттєвих змін. Цей процес визначається динамічним розвитком фінансових технологій (фінтеху), які інтегрують до щоденного досвіду штучний інтелект, великі дані, блокчейн, хмарні обчислення, машинне навчання, інтерфейси Open Banking API, автоматизовані системи та мобільні застосунки. Україна активно долучається до цього тренду, формуючи свою власну цифрову фінансову екосистему, яка відповідає глобальним стандартам.

Цифрове управління особистими фінансами – це вже не просто розрахунки за допомогою смартфону чи онлайн-банкінгу. Це складна взаємодія між користувачем та алгоритмами, здатними в режимі реального часу аналізувати фінансові звички, передбачати майбутні витрати, надавати рекомендації щодо оптимізації бюджету, інформувати про можливі ризики, а також автоматично інвестувати в доступні фінансові інструменти. Цифрові інструменти вже стали невід'ємною частиною повсякденного управління фінансами.

Персоналізація фінансового досвіду стає ключовим трендом. Наприклад, системи на базі штучного інтелекту адаптують інтерфейси застосунків до звичок користувачів, візуально представляють аналітику витрат та пропонують варіанти оптимального управління фінансами. Це суттєво спрощує доступ до фінансових інструментів для всіх, навіть для тих, хто не має спеціальної освіти.

Блокчейн-технології відкривають можливості для створення децентралізованих платформ, де користувачі можуть безпечно та прозоро управляти активами, уникаючи посередників. Смарт-контракти автоматизують фінансові операції, зменшуючи ризик людської помилки чи шахрайства. В Україні вже проводяться експерименти з використанням блокчейну для управління цифровими активами, фандрейзингу та peer-to-peer кредитування.

Хмарні сервіси відіграють важливу роль, забезпечуючи стабільний доступ до фінансової інформації з будь-якого пристрою, дозволяючи масштабувати рішення, інтегрувати дані з інших платформ та створювати багатоканальні сервіси. Користувачі можуть управляти своїми фінансами через чат-боти в месенджерах, голосових помічників або інтерфейси "розумного дому".

Незважаючи на швидкий прогрес, перед IT-сферою постають виклики. Серед них – захист персональних даних, зростання кіберзлочинності, потенційна маніпуляція алгоритмами, та відсутність єдиних стандартів безпеки для фінансових застосунків. Користувачі часто не до кінця розуміють, які саме дані збираються, як вони використовуються та де зберігаються. Крім того, необхідна національна стратегія цифрової фінансової грамотності, що охоплювала б різні вікові категорії.

Цифрова нерівність також залишається актуальною проблемою. У містах доступ до інтернету, смартфонів та сучасних застосунків є звичним явищем, в той час як у сільській місцевості та серед соціально вразливих груп ці сервіси є менш доступними або зовсім недоступними, що ускладнює цифрову фінансову інклюзію.

Впровадження цифрової ідентифікації може стати проривом, дозволяючи ідентифікувати особу у фінансових системах без фізичної присутності. У поєднанні з багатофакторною аутентифікацією, криптографічним захистом та біометрією, це значно посилить безпеку фінансових даних. В Україні, незважаючи на обмежене поширення, вже є перші приклади інтеграції цифрової ідентифікації з банківськими та державними сервісами.

Гейміфікація фінансових сервісів – ще один перспективний напрямок. Деякі мобільні застосунки пропонують «фінансові квести», систему винагород за заощадження, візуалізацію фінансових цілей у форматі гри. Це підвищує мотивацію та робить управління фінансами цікавішим. Застосування ігор у сфері фінансової освіти особливо ефективне серед молоді.

З розвитком big data аналітики з'являються нові можливості: платформи виявляють макроекономічні тенденції, аналізують вплив соціальних подій на

споживчу поведінку, а також визначають "цифровий слід" користувача для надання точніших фінансових прогнозів. У майбутньому ці моделі зможуть працювати в автоматичному режимі, забезпечуючи майже повну автономність при прийнятті фінансових рішень.

У глобальному цифровому середовищі особисті фінанси перестають бути суто обліковою дисципліною, перетворюючись на інтелектуально підтримувану сферу, де користувач не просто спостерігає за своїм бюджетом, а й активно впливає на власну фінансову поведінку, приймає прогностичні рішення, управляє активами в режимі реального часу. Прикладом є міжнародна практика розробки та впровадження фінансових застосунків, які поєднують хмарні обчислення, машинне навчання, аналіз UX, відкриті API банків та глибоку інтеграцію з цифровими екосистемами.

Одним з найбільш відомих застосунків, що задали тон у розвитку персонального фінансового планування, є Mint, розроблений американською компанією Intuit. Цей застосунок став одним з перших, що дозволив інтегрувати всі рахунки користувача (банківські, кредитні, інвестиційні) в єдиному інтерфейсі. Особливістю є використання відкритих API для підключення до банків, що дозволяє отримувати інформацію про транзакції в реальному часі без ручного введення даних. Алгоритми штучного інтелекту аналізують кожну транзакцію, класифікуючи її за категоріями витрат, порівнюючи з попередніми даними. Mint не лише відстежує фінансову активність, а й пропонує персоналізовану аналітику, візуалізуючи фінансові звички, допомагаючи ідентифікувати непередбачені витрати, виявляти надмірне споживання та надавати поради щодо оптимізації бюджету. Це приклад того, як штучний інтелект може знімати з користувача відповідальність за рутинний контроль, перетворюючи фінансовий облік на динамічний та візуалізований досвід.

Інший підхід до управління особистими фінансами пропонує YNAB (You Need A Budget), який фокусується не лише на обліку чи аналізі, але й на формуванні фінансової культури користувача. На відміну від Mint, який в основному стежить

за витратами, YNAB орієнтований на превентивне планування. Його цифрова модель заснована на принципі виділення кожному доходу конкретної цілі до моменту витрати, що спонукає користувача складати бюджет на основі актуальних потреб, а не абстрактних категорій. З технічної точки зору, YNAB синхронізується між мобільною, веб- та десктопною версіями, забезпечуючи цілісність бюджету в усіх середовищах. Підключення до банків відбувається через API, транзакції імпортуються та відображаються в системі, де користувач самостійно визначає категорію для кожної витрати – таким чином, навчаючи не тільки віртуальну систему, але й свою фінансову дисципліну. Застосунок містить вбудовану аналітику, яка попереджає про перевищення лімітів, прогнозує залишки, пропонує перенаправлення ресурсів. YNAB демонструє, як цифрові інструменти можуть формувати довгострокову фінансову відповідальність за допомогою інтерфейсу та алгоритмів.

Суттєво інший – але дуже показовий – підхід демонструє Revolut, цифровий банк з Великої Британії, який перетворився на фінансову суперекосистему. Revolut поєднує банківські рахунки, мультивалютні гаманці, інвестиційні портфелі, обмін криптовалют, страхування, аналітику витрат та фінансові цілі в одному застосунку. Його архітектура базується на хмарній інфраструктурі з власною фінансовою платформою, здатною обробляти тисячі транзакцій в секунду, аналізуючи кожну з них у реальному часі. Впровадження технологій геолокації, біометрії та аналізу поведінки забезпечує високий рівень безпеки та автоматичне виявлення підозрілих активностей. Revolut активно використовує машинне навчання для створення індивідуальних фінансових профілів, що дозволяє автоматично рекомендувати вигідні інвестиційні чи кредитні продукти. Це більше, ніж просто платформа для витрат – це інтелектуальний фінансовий помічник, який може прогнозувати, консультувати та управляти коштами від імені користувача. Візуальні панелі, аналітичні діаграми та push-повідомлення створюють динамічну взаємодію, яка замінює традиційний формат фінансового обліку.

Ці приклади – Mint, YNAB, Revolut – демонструють різноманітні підходи до цифрового управління особистими фінансами, але всі вони об'єднані одним: використанням потужних IT-рішень для автоматизації, персоналізації та прогнозування. Вони показують, як за допомогою інтеграції з банківськими системами, штучним інтелектом, хмарними сервісами та UX-дизайном можна підняти взаємодію людини з її фінансами на якісно новий рівень. Такий досвід вартий уваги в українському контексті, оскільки він дає можливість побачити, як IT не просто підтримують, а кардинально змінюють фінансову поведінку, роблячи її цифровою, гнучкою, безпечною та глибоко персоналізованою.

Успіх цифрової трансформації особистих фінансів в Україні залежить від злагодженої співпраці держави, бізнесу, наукової спільноти та самих користувачів. Необхідно розширювати регуляторні рамки, що сприятимуть прозорості цифрових активів, підтримувати національний фінтех, розвивати цифрову інфраструктуру в регіонах, створювати програми цифрової освіти, доступні онлайн для всіх верств населення.

Цифрові технології стають не тільки інструментом зручності, а й рушієм соціального прогресу. Вони дозволяють людям контролювати свої фінанси в режимі реального часу, знижувати ризики, підвищувати ефективність заощаджень та покращувати якість життя. Україна має всі передумови, щоб стати частиною глобального ландшафту цифрових фінансів, створивши власну модель інклюзивного, безпечного та ефективного управління особистими фінансами в цифрову епоху.

1.2 Безпеки та конфіденційності застосунку для ведення особистої бухгалтерії

У цифровому світі, де особисті відомості є надзвичайно цінним активом, користувачі, що ведуть облік власних фінансів за допомогою електронних засобів, зустрічаються з різноманітними типами загроз, які здатні призвести до витоку,

втрати чи маніпулювання їх фінансовою інформацією. Перший і найпоширеніший тип ризиків – несанкціонований доступ до даних. Це може трапитися як наслідок хакерської атаки на сервер стороннього застосунку, так і через фішинг або слабкий пароль з боку користувача. У подібних випадках зловмисник здобуває доступ до облікового запису, бачить фінансову історію, збережені банківські реквізити, іноді – логіни до інтернет-банкінгу. Особливо небезпечно це в хмарних застосунках, які синхронізуються з банками через зовнішні API, адже сторонній хакер або навіть внутрішній інженер може, за певних умов, отримати ключ до всієї фінансової активності користувача.

Ще одна загроза – перехоплення даних під час передачі. Навіть якщо застосунок застосовує HTTPS, ймовірні атаки типу “людина посередині” (man-in-the-middle), особливо на відкритих Wi-Fi-мережах. Якщо сервер не перевіряє сертифікати належним чином або використовує застарілі протоколи TLS, це відкриває додаткові вразливості. У багатьох випадках також можлива атака через компрометацію сторонньої бібліотеки або залежності, яку використовує застосунок – наприклад, у разі використання відкритих SDK для реклами, аналітики або авторизації через соціальні мережі.

Також існує небезпека втрати даних. Якщо сторонній сервіс припиняє роботу, видаляє акаунт, потрапляє під санкції або стає жертвою внутрішньої технічної аварії – користувач втрачає доступ до всієї своєї фінансової історії. Ще гірша ситуація, коли дані зникають без попередження або користувача не інформують про збій. Інколи хостинг-компанії або хмарні провайдери знищують резервні копії через порушення політик чи збіг терміну зберігання. В такому разі жоден механізм відновлення не допоможе, і фінансова історія багатьох років може зникнути миттєво.

Окрему категорію ризиків становлять приховані або легалізовані витоки даних через умови користування. Чимало сервісів мають у своїй політиці можливість обробки, аналізу й передачі знеособлених даних третім сторонам. У випадку фінансової інформації – це вкрай небезпечно, оскільки навіть без прямого

імені чи e-mail транзакції, геолокація, графік покупок та розміри витрат дають змогу з високою точністю ідентифікувати особу. У поєднанні з іншими джерелами – соціальними мережами, сервісами доставки, поштовими акаунтами – ці дані формують повну модель поведінки користувача. Часто такі відомості продаються банкам, рекламодавцям, маркетинговим компаніям або використовуються для тренування алгоритмів, про що користувач навіть не підозрює.

Ще один тип небезпеки – внутрішні атаки. Це випадки, коли доступ до даних отримує не зовнішній зловмисник, а співробітник компанії-розробника, що володіє адміністративним доступом. Навіть якщо дані на диску зашифровані, в більшості хмарних сервісів адміністратор може прочитати дані в момент їхньої обробки на сервері – наприклад, для аналітики, створення графіків чи підрахунків. Якщо ці процеси не відокремлені належним чином або не проходять через sandbox-механізми, можливість витоку зсередини залишається актуальною.

Захист від усіх цих загроз вимагає багатошарового підходу. Найперший принцип – це зберігання даних локально, на пристрої користувача. Це дозволяє уникнути будь-яких ризиків, пов’язаних із передаванням даних через Інтернет або збереженням на чужих серверах. Локальні дані повинні бути зашифровані – і не лише “для галочки”, а з використанням сучасних криптографічних алгоритмів. Симетричне шифрування типу AES-256 забезпечує швидке й надійне шифрування файлів, а асиметричні алгоритми (наприклад, RSA або Curve25519) можуть використовуватись для захисту окремих ключів, аутентифікації або цифрового підпису. У разі критичних даних шифрування варто проводити з окремим паролем, який не зберігається в системі, і може вводитися вручну або зберігатись у фізичному токені.

Наступний етап захисту – це контроль доступу. Застосунок повинен мати захист на рівні запуску – пароль, PIN-код, біометрія або фізичний ключ. Необхідно впровадити таймер автоматичного блокування після періоду бездіяльності. Якщо користувач зберігає резервні копії, вони теж мають бути зашифровані та зберігатись окремо – наприклад, на зовнішньому жорсткому диску або на власному

NAS-сервері. Відсилання резервних копій у хмару потрібно виключити або реалізувати лише за згодою користувача з окремим ключем.

Захист на рівні коду застосунку теж має велике значення. Весь код має бути ізольованим від зовнішніх сервісів, не містити бекдорів, відстеження, сторонніх бібліотек аналітики чи несанкціонованих HTTP-запитів. Бажано публікувати застосунок з відкритим кодом або, принаймні, кодом, який піддається аудиту, щоб будь-який фахівець міг перевірити, чи немає в ньому прихованих механізмів збирання даних.

У випадку синхронізації між пристроями – наприклад, між смартфоном і ПК – необхідно використовувати протокол, що передбачає наскрізне шифрування (end-to-end encryption), або ручне передавання зашифрованих архівів без участі сторонніх серверів. Для захисту від перехоплення в мережі варто повністю відмовитись від відкритих з’єднань і використовувати власні захищені протоколи чи тунелі на базі TLS/SSL. Також доцільно реалізувати механізми цифрового підпису, аби виявити, чи було змінено дані чи порушено їхню цілісність.

Окрім цього, будь-який централізований сервіс є потенційною мішенню для зловмисників. Великі сервіси часто стають об'єктом хакерських атак саме через те, що вони зберігають велику кількість цінної інформації. Навіть частковий витік даних може поставити під загрозу тисячі користувачів. У випадку з особистими фінансами мова йде не лише про втрату приватності, а й про можливі шахрайські дії, маніпулювання інформацією або навіть фінансові втрати. Якщо використовувати власний додаток, то ці ризики мінімізуються, адже дані залишаються тільки на пристрої користувача та не передаються через мережу Інтернет.

Особистий додаток надає можливість локального зберігання даних без необхідності покладатися на зовнішні системи. Це означає, що дані доступні лише фізично на тому пристрої, де встановлено програму, і тільки після введення пароля або проходження автентифікації. Самостійно розроблена система дозволяє

інтегрувати будь-який сучасний спосіб захисту — від симетричного шифрування з використанням AES або ChaCha20 до складних гібридних схем, які поєднують симетричні та асиметричні ключі. Користувач також може самостійно обирати параметри генерації ключів, зберігати їх у захищених сховищах, застосовувати хешування паролів з використанням солі, налаштовувати обмеження на кількість спроб входу, автоматичне блокування і навіть самознищення даних у разі загроз.

Контроль над механізмами аутентифікації – ще одна значна перевага. У готових додатках часто застосовується лише базова система входу через email або пароль, при цьому біометрична аутентифікація реалізована не завжди, або залежить від операційної системи. У своєму власному додатку можна інтегрувати біометрію, PIN-коди, ключі безпеки (наприклад, через FIDO2 або YubiKey), багатофакторну аутентифікацію або навіть одноразові паролі. Окрім того, у власному рішенні можна визначити логіку, якої немає в комерційних продуктах – наприклад, обмеження доступу до певних розділів програми залежно від часу доби, місцезнаходження або наявності підключення до мережі.

Ще одним важливим плюсом власного додатку є повна автономність і незалежність від зовнішніх факторів. Готові сервіси можуть бути заблоковані, припинити роботу, змінити політику конфіденційності або почати вимагати оплату. При цьому всі дані користувача залишаються на серверах компанії, і процес їх вивантаження або експорту нерідко навмисно ускладнений. У випадку з власним додатком такого ризику немає — навіть при припиненні підтримки чи заміні пристрою вся інформація зберігається у користувача. Формат збереження даних також обирається самостійно — це може бути локальна база даних SQLite, зашифрований JSON, бінарні файли з власною структурою, або будь-який інший формат, що забезпечує гнучкість та сумісність.

Не варто також недооцінювати аспект прихованої аналітики та збору даних, що часто практикується у безкоштовних або навіть платних фінансових сервісах. Велика кількість додатків відстежує поведінку користувача, частоту використання, типи витрат, географічні дані, а іноді й зміст фінансових записів. Ця інформація

може використовуватися для побудови профілю користувача, таргетованої реклами або навіть передаватися третім сторонам, включаючи рекламодавців та аналітичні платформи. У власному додатку відсутнє навіть технічне середовище для такої поведінки: немає відстеження, аналітики чи фонових процесів без прямої згоди користувача. Це дає можливість забезпечити справжню приватність, якої неможливо досягти у публічному сервісі.

До того ж, власний додаток надає користувачеві повну свободу налаштування функціональності. Готові рішення часто орієнтовані на масового користувача і мають фіксовану логіку — наприклад, категорії витрат, структуру бюджету, обмеження звітів, валютні операції, інтеграцію з банками лише певних країн тощо. Якщо ж додаток створюється самостійно, будь-яка функція може бути реалізована індивідуально: нестандартна категоризація, ведення за проектами, автоматичне оновлення валютних курсів, аналіз регулярних платежів, візуалізація фінансових цілей, машинне навчання для прогнозування витрат — усе, що забажає користувач. Це також дає можливість сфокусуватись на місцевих потребах, мовній підтримці, валюті, культурних особливостях фінансової поведінки.

На технічному рівні власний додаток – це повна свобода у виборі технологій, архітектури, мови програмування, способу зберігання даних, інтерфейсів та навіть філософії використання. Його можна запускати локально на комп’ютері, інтегрувати в мобільний додаток, налаштувати як автономну вебпрограму або навіть підключити власний міні-сервер без потреби виходу в глобальну мережу Інтернет. При цьому вся логіка буде зрозумілою для користувача, на відміну від закритого коду готових продуктів, де неясно, які саме дані збираються, куди вони надсилаються і як використовуються.

Ці ризики не є просто теоретичними. Існують численні приклади того, як навіть потужні, добре відомі та професійні сервіси зазнавали хакерських атак, мали критичні недоліки або порушували приватність своїх користувачів. Наприклад, один з найпопулярніших фінансових застосунків у США – Mint (від Intuit) Рис.1.1 – у 2019 році опинився в центрі технічної критики через вразливість, яка дозволяла

обходити аутентифікацію при підключенні банківських рахунків. Через недостатню валідацію API користувачі могли підключити банківський рахунок іншої особи, що ставило під загрозу цілі банківські структури. Незважаючи на спроби компанії виправити ситуацію, багато користувачів втратили довіру до сервісу, адже будь-яке порушення в роботі з банківськими даними автоматично є критичним.

Рисунок 1.1 - Логотип фінансового застосунку “ Mint (Intuit)”

Іншим прикладом є Spendee, котрий, попри популярність в країнах Центральної Європи, у 2020 році став об’єктом технічного розслідування через неправильну конфігурацію доступу до API. Через вразливість користувачі могли отримати доступ до сесій інших користувачів, що загрожувало не лише конфіденційністю витрат, але й самій структурі доступу до облікових записів. Додатково виявилось, що у попередніх версіях Spendee використовував хмарні бакети Amazon S3 для зберігання резервних копій без належного шифрування або захисту прав доступу – і ця інформація певний час була доступною для будь-кого, хто знав адресу файлу.

У 2021 році YNAB (You Need a Budget) Рис.1.2, ще один популярний застосунок з культовою аудиторією, став недоступним через технічний збій на серверній інфраструктурі Amazon Web Services. Хоча прямого витоку даних не відбулось, користувачі втратили доступ до своїх фінансових даних на декілька

годин. Це підкреслює ще одну проблему централізованих сервісів – повну залежність від зовнішньої інфраструктури, на яку сам застосунок не має жодного впливу. Навіть за найкращих обставин, сторонній збій або помилка у хмарі означають, що ваші особисті фінанси можуть стати тимчасово або назавжди недоступними.

Рисунок 1.2 - Логотип фінансового застосунку “ YNAB (You Need A Budget)”

Окремо варто згадати про застосунки, що інтегруються з банками через сторонні агрегатори. Наприклад, PocketSmith синхронізує дані через Yodlee – сервіс, який у 2017–2019 роках став відомим через практику продажу знеособлених фінансових даних користувачів третім сторонам. Попри запевнення, що ці дані були анонімізовані, фахівці з кібербезпеки виявили, що навіть частково знеособлена фінансова інформація може бути відновлена до конкретної особи через поведінкові патерни, типи транзакцій та географію витрат. Це створює серйозну етичну та безпекову проблему: користувач не знає, що його фінансова поведінка перетворюється на товар, який аналізують, продають або використовують у комерційних цілях.

Ще один волаючий приклад – сервіс Paribus Рис.1.3, котрий аналізував вміст поштових скриньок користувачів для виявлення покупок і повернення коштів за знижками. Хоча спочатку декларувалось, що застосунок має доступ лише до конкретних листів, пізніше було встановлено, що він фактично мав доступ до всього листування. Це доводить, що, підписуючись на сторонній сервіс, користувач

часто не усвідомлює реального обсягу доступу, який надає – і що цей доступ не завжди обмежується заявленими межами.

Рисунок 1.3 - Логотип фінансового застосунку “ PocketSmith ”

Навіть при впровадженні стандартних заходів безпеки – HTTPS, токенів доступу, обмежень API, багатофакторної автентифікації – залишаються невирішеними критично важливі питання: хто має root-доступ до серверів, хто бачить дані в розшифрованому вигляді, які бекдор-механізми вбудовані в систему, як компанія реагує на запити спецслужб, і чи справді користувач може остаточно видалити свої дані з системи. У більшості випадків відповідь – ні. Політика “право на забуття” або “видалити акаунт” лише частково стирає сліди, але резервні копії можуть зберігатися роками.

На цьому тлі створення власного застосунку для ведення особистої бухгалтерії надає безпрецедентний рівень контролю, захисту й автономності. Він дозволяє реалізувати локальне зберігання даних, повністю відмовитись від мережевих синхронізацій, використати перевірені алгоритми шифрування – такі як AES-256, ChaCha20 або навіть менш популярні, але потужніші асиметричні схеми на базі Curve25519. Користувач сам визначає правила автентифікації, обирає, чи використовувати 2FA, локальні токени, біометрію, паролі або PIN-коди. Усі

критичні дані можна додатково зашифрувати окремим ключем і розмістити в окремому шифрованому контейнері, який ніколи не виходить за межі пристрою. Ключі можна зберігати в апаратному токені, окремому USB або навіть записати фізично.

Також відсутні фонові трекери, аналітика, рекламні інтеграції або політика збору даних – як часто буває навіть у “преміум”-версіях готових продуктів. Особистий застосунок не оновлюється автоматично без згоди користувача, не обмінюється даними з Google Analytics, Facebook SDK чи Mixpanel, не має бекдорів і не залежить від uptime зовнішнього хостингу. Це також дає змогу користувачеві створити власну систему резервного копіювання – наприклад, зберігати дані на зашифрованому носії, у власному NAS-сервері або у вигляді файлу, що дублюється в зашифрованому вигляді на фізичний носій і зберігається офлайн.

Наведені приклади демонструють, що навіть найвідоміші сервіси можуть допускати критичні помилки, порушувати конфіденційність або наражатися на зовнішні загрози, тоді як власний застосунок – це не просто альтернатива, а єдиний спосіб досягти справжньої безпеки, повного контролю та гнучкості у роботі з власними фінансами в сучасному цифровому світі.

1.3 Обґрунтування доцільності створення власного застосунку для особистої та бізнес-бухгалтерії

У сучасному світі фінансова грамотність та уміння контролювати особисті й бізнес-кошти є важливими складовими ефективного управління ресурсами. Ведення фінансового обліку дозволяє не лише контролювати витрати, а й аналізувати джерела доходів, виявляти нераціональні фінансові звички, планувати бюджет, оптимізувати податкове навантаження, а також підвищувати загальну стабільність та передбачуваність господарської діяльності.

На ринку програмного забезпечення представлена велика кількість готових рішень для ведення обліку доходів і витрат. Це можуть бути веб-сервіси, мобільні

застосунки або настільні програми. Вони охоплюють широкий спектр функцій: від базового додавання транзакцій до складних систем фінансового планування, аналітики та інтеграції з банківськими системами. Водночас ці продукти мають ряд обмежень, які часто роблять їх малопридатними для повсякденного використання в побуті або в малому бізнесі. Найпоширенішими проблемами є потреба у стабільному підключенні до Інтернету, збереження даних на зовнішніх серверах, наявність платної підписки або реклами, складність інтерфейсу та надмірна функціональність, котра не відповідає реальним потребам користувача.

Особливо критичним фактором для багатьох користувачів є питання конфіденційності. У випадку передачі даних через Інтернет або використання хмарних сервісів, інформація про фінансову активність користувача може бути збережена на сторонніх серверах, що відкриває ризики несанкціонованого доступу, втрати персональних даних або зловживання з боку сервісу. Це особливо небезпечно у випадках, коли йдеться про облік внутрішніх операцій у малому бізнесі або фіксацію приватних витрат, що не підлягають публічному розголошенню.

У такій ситуації доцільним рішенням стає створення власного, локального застосунку, який би дозволив зберігати всі дані безпосередньо на комп’ютері користувача, без залучення сторонніх сервісів. Розробка персонального програмного продукту дозволяє врахувати тільки ті функції, котрі дійсно потрібні конкретному користувачу, без перевантаження зайвим функціоналом. Такий підхід значно підвищує зручність роботи, прискорює взаємодію з інтерфейсом та спрощує сам процес обліку.

Крім того, власний застосунок може бути адаптований для різних сценаріїв використання. Для особистого користування це може бути простий облік доходів та витрат із можливістю коментування та сортування операцій. Для малого бізнесу – базовий контроль грошових потоків, оперативна фіксація транзакцій, можливість пошуку та редагування записів. При цьому не потрібно володіти спеціальними знаннями у сфері бухгалтерського обліку чи роботи з комерційними ERP-

системами. Простота та наочність інтерфейсу роблять програму доступною для широкого кола користувачів, включаючи людей без технічної освіти.

Важливою перевагою є також автономність роботи застосунку. Його можна використовувати без підключення до Інтернету, що є суттєвою перевагою для користувачів, які працюють у польових умовах, відрядженнях, з обмеженим доступом до мережі або просто прагнуть зберегти фінансову незалежність від зовнішніх платформ. У випадках обліку готівкових операцій це дозволяє в реальному часі вносити витрати та доходи, не чекаючи синхронізації чи оновлень.

Окрім цього, створення власного застосунку є важливою частиною професійного розвитку для розробника. Це дозволяє не лише практикувати роботу з графічним інтерфейсом (у даному випадку — через бібліотеку Tkinter), а й застосовувати принципи структурного програмування, взаємодії з базами даних (SQLite), реалізовувати події у віконному середовищі, забезпечувати обробку помилок та інтеракцію з користувачем.

Результатом розробки є інструмент, котрий може використовуватись на постійній основі у власному господарстві, служити зразком для навчання інших користувачів, бути адаптованим до потреб сімейного обліку або базової бухгалтерії для фізичних осіб-підприємців. У майбутньому такий застосунок може бути масштабований — доповнений категоріями, графіками, статистикою, підтримкою експорту у різні формати або навіть інтеграцією з зовнішніми системами.

Таким чином, розробка власного застосунку для ведення обліку фінансів є практичним, функціонально виправданим та безпечно орієнтованим рішенням, що має чітке застосування як у повсякденному житті, так і у підприємницькій діяльності. Його використання дозволяє підвищити фінансову дисципліну, забезпечити збереження конфіденційної інформації, автоматизувати базові облікові процеси, а також сприяти особистому розвитку розробника в сфері інформаційних технологій.

У сучасному цифровому світі на ринку доступна велика кількість готових програмних рішень для обліку фінансів. Серед найпопулярніших варіантів є такі застосунки, як Money Manager, Monefy, Spendee, а також банківські сервіси, як-от

«Приват24» чи функціонал у державному застосунку «Дія». Проте більшість із них має певні обмеження, що роблять їх менш придатними для вузькоспеціалізованого або персоналізованого обліку, який часто потрібен індивідуальному користувачу, самозайнятому фахівцю або власнику малого бізнесу. До таких обмежень належать: необхідність постійного підключення до інтернету, потреба реєстрації та зберігання особистих даних на зовнішніх серверах, наявність платної підписки для розблокування всього функціоналу, а також відсутність можливості змінювати логіку обробки та представлення даних. Багато комерційних продуктів є закритими за архітектурою, що унеможливлює адаптацію їхньої функціональності під специфічні потреби користувача, зокрема для використання в україномовному середовищі або для створення кастомних звітів. Отже, розробка власного програмного забезпечення є цілком виправданим рішенням, яке дозволяє подолати ці обмеження.

Ситуація ускладнюється також тим, що наявні на ринку професійні бухгалтерські програми, орієнтовані на корпоративне або комерційне застосування (наприклад, 1С, M.E.Doc, BAS), є надто складними, громіздкими та дорогими для індивідуального користування чи для потреб мікропідприємців. Такі системи вимагають значного часу на навчання, постійної технічної підтримки, мають складний інтерфейс та не завжди підтримують актуальні українські стандарти ведення обліку у простій та зручній формі. Для малого бізнесу, ФОПів або звичайних користувачів, що ведуть облік особистих витрат, такі інструменти є надмірними та економічно недоцільними. Натомість розробка власного програмного забезпечення на базі Python і бібліотек для графічного інтерфейсу, таких як Tkinter, дозволяє створити простий у використанні застосунок із мінімалістичним інтерфейсом та зручною логікою обробки транзакцій.

Важливим аргументом на користь створення власного рішення є можливість повного контролю над функціоналом, структурою даних та алгоритмами їх обробки. Користувач отримує гнучкий інструмент, який можна доповнювати будь-яким необхідним функціоналом — від фільтрації та категоризації транзакцій до формування звітів, графіків, імпорту або експорту даних у зручні формати. Більш

того, відкритість коду дозволяє швидко адаптувати програму до змін, інтегрувати її з банківськими API, додати автоматизацію регулярних платежів або ж створити хмарну версію для віддаленого доступу з кількох пристроїв. Це дає змогу розвивати програмний продукт відповідно до реальних змін у фінансовому чи повсякденному житті користувача, чого неможливо досягти в межах готових закритих систем.

Власний застосунок також гарантує максимальну безпеку персональних фінансових даних, оскільки вся інформація зберігається виключно локально на пристрої користувача. Це виключає ризик витоку конфіденційної інформації через сторонні сервери, хмарні сервіси або ненадійні інтернет-з'єднання. У майбутньому за потреби може бути реалізовано шифрування бази даних, захист паролем, автоматичне резервне копіювання або синхронізація з безпечним сервером за повного контролю користувача. Така модель гарантує конфіденційність та відповідність вимогам інформаційної безпеки в особистому та діловому використанні.

Окрему цінність становить розробка такого застосунку в освітньому та професійному контексті. Це не лише демонстрація практичного закріплення знань з програмування, баз даних, логіки інтерфейсів, а й реальне застосування отриманих навичок для вирішення повсякденних і бізнес-завдань. Крім того, це формує прикладене портфоліо розробника, яке може бути використане для працевлаштування або для подальших проєктів. Власноруч створена програма є основою для наукових досліджень, розвитку системних рішень у галузі фінансових IT-сервісів, а також стартовою платформою для складніших задач, таких як реалізація бізнес-аналітики, інтелектуального прогнозування витрат або інтеграції з платіжними системами.

З економічної точки зору розробка власного застосунку також виправдана. У випадку використання платних сервісів для обліку фінансів, користувач змушений або оформлювати щомісячну підписку, або купувати повну версію застосунку. Також виникає потреба оплачувати послуги бухгалтерського супроводу, навіть якщо йдеться лише про простий облік витрат і доходів. Власний застосунок повністю усуває ці витрати, оскільки є безкоштовним, автономним та потребує

лише базових ресурсів для підтримки його функціонування. У довгостроковій перспективі це дозволяє заощадити значні кошти без втрати якості обліку або функціональної ефективності.

Таким чином, розробка власного програмного продукту для обліку особистих фінансів є технічно обґрунтованим, економічно доцільним та стратегічно важливим кроком. Вона дає змогу отримати повний контроль над даними, забезпечує безпеку та гнучкість, відкриває широкі можливості для подальшої модернізації та відповідає актуальним потребам сучасного користувача в особистому, освітньому або бізнес-контексті.

2 ПРОГРАМНИЙ ІНСТРУМЕНТАРІЙ ДЛЯ РОЗРОБКИ ЗАСТОСУНКУ НА ОСНОВІ МОВИ ПРОГРАМУВАННЯ PYTHON

2.1 Розробка функціональності застосунку особистої бухгалтерії

Розробка функціональності є одним із ключових етапів створення програмного забезпечення, що визначає, як саме користувач буде взаємодіяти із застосунком, які завдання він зможе виконувати, і яким чином програма оброблятиме та зберігатиме дані. У випадку застосунку особистої бухгалтерії мова йде про створення зручного, інтуїтивного інструмента, який дозволить користувачеві контролювати свої фінансові потоки, здійснювати аналіз витрат і доходів, планувати бюджет та досягати фінансових цілей.

У процесі проєктування було враховано основні запити цільової аудиторії: простота у використанні, наочність, можливість адаптації під індивідуальні потреби, підтримка безперебійної роботи офлайн, захист конфіденційної інформації. Таким чином, архітектура застосунку була спроєктована за модульним принципом, де кожен функціональний блок відповідає за певний аспект роботи з фінансами, а всі модулі взаємодіють між собою через чітко визначені інтерфейси.

Початковий етап взаємодії із застосунком полягає в автентифікації користувача. Це забезпечує приватність і захищає дані від стороннього доступу. Під час розробки було реалізовано реєстрацію за логіном і паролем із подальшим збереженням інформації у зашифрованому вигляді. Для захисту паролів використано криптографічний алгоритм хешування (bcrypt або Argon2), що унеможливлює пряме відновлення паролю навіть у разі витоку бази даних. Для підвищення безпеки також передбачено функцію двофакторної автентифікації (2FA), коли користувач підтверджує свою особу додатковим кодом, надісланим на електронну пошту або мобільний пристрій.

Після входу в систему користувач має доступ до головної функції — обліку фінансових операцій. Застосунок дозволяє створювати записи, що описують кожен

дохід або витрату. Для зручності реалізовано гнучку форму введення, де користувач може вказати суму, тип операції, категорію (наприклад, їжа, транспорт, розваги, зарплата), дату й час, а також текстовий опис. Кожен запис зберігається у внутрішній базі даних, що забезпечує швидкий доступ до інформації навіть без підключення до Інтернету. Застосунок автоматично формує поточний баланс, сумує доходи та витрати, що дає змогу користувачеві в реальному часі оцінювати своє фінансове становище.

Окрему увагу було приділено гнучкості категоризації. Користувач може не лише обирати із запропонованих категорій, а й створювати власні. Це дозволяє максимально адаптувати застосунок під свій стиль життя: наприклад, виділити окрему категорію «Кава», якщо це регулярна стаття витрат. Всі категорії зберігаються у відповідній таблиці бази даних і пов’язані з транзакціями за допомогою зовнішнього ключа. Це забезпечує структурованість даних і спрощує їхню подальшу обробку.

У межах функціональності пошуку й навігації реалізовано фільтри, які дозволяють швидко знайти потрібні транзакції. Користувач може обрати діапазон дат, тип операції, категорію або здійснити пошук за ключовими словами. Наприклад, можна легко переглянути всі витрати за останній місяць, які стосуються транспорту, або всі доходи від фрілансу. Такий інструментарій особливо корисний, коли даних накопичується багато і потрібен оперативний доступ до конкретної інформації.

Важливою складовою є візуалізація даних. Застосунок генерує діаграми, які відображають розподіл витрат за категоріями, динаміку змін балансу, порівняння доходів і витрат у різні періоди. Візуалізація реалізована за допомогою відповідних бібліотек (наприклад, Matplotlib або Plotly) і є інтерактивною, тобто дозволяє користувачеві переглядати деталізовану інформацію за кожним сегментом графіка. Це не лише полегшує аналіз, але й робить взаємодію із застосунком наочнішою та привабливішою.

Додаткову цінність надає функція встановлення бюджетів і фінансових цілей. Користувач може задати обмеження на витрати за категоріями або загалом

на місяць. У разі наближення до встановленого ліміту система інформує користувача про це, допомагаючи уникати перевитрат. Також можна створювати фінансові цілі — наприклад, накопичити певну суму на відпустку — і поступово відкладати кошти, відстежуючи прогрес. Це сприяє формуванню фінансової дисципліни та усвідомленого підходу до витрат.

Функціональність імпорту та експорту дозволяє користувачеві працювати зі своїми даними поза межами застосунку. Завдяки підтримці форматів CSV і Excel можна легко створювати резервні копії, переносити дані в інші системи або використовувати їх для більш глибокого аналізу. Імпорт банківських виписок дозволяє автоматизувати процес додавання транзакцій, що значно економить час. Усі ці операції відбуваються з урахуванням збереження структури бази даних та уникнення дублювання інформації.

Застосунок реалізовано із дотриманням принципів зручності інтерфейсу (UX/UI). Інтерфейс побудований так, щоби користувач із мінімальними зусиллями міг виконувати основні дії: додати запис, переглянути баланс, змінити бюджет, побудувати графік. Також передбачено підтримку декількох мов, включаючи українську, адаптивний дизайн під різні розміри екранів і темну тему для комфортного використання в умовах слабкого освітлення.

Окрему увагу приділено безпеці даних. Для цього використано локальне шифрування бази даних, захист доступу до застосунку через пароль, автоматичне блокування після періоду неактивності. Дані не передаються на сторонні сервери без явної згоди користувача, що відповідає принципам цифрової гігієни та захисту приватності.

Таким чином, функціональність застосунку охоплює повний спектр потреб користувача: від щоденного обліку до довгострокового фінансового планування. Вона базується на сучасних підходах до розробки програмного забезпечення та націлена на створення ефективного, безпечного й персоналізованого інструменту для роботи з особистими фінансами.

2.2 Вибір мови програмування для розробки застосунку особистої бухгалтерії

Під час конструювання будь-якого програмного виробу, у тому числі такого як додаток для ведення персонального фінансового обліку, одним із ключових моментів є вибір мови програмування. Від цього рішення залежить не тільки технічна сторона проєкту, але й впливає на комфорт розробки, підтримку, масштабованість, перспективні доповнення новими функціями та взаємодію з користувачем. Ідеальна мова програмування для такого завдання повинна забезпечувати швидку реалізацію, просту підтримку, доступ до широкого спектра бібліотек, відмінну роботу з базами даних, можливість створення графічного інтерфейсу, а також мати зручний синтаксис. Враховуючи всі ці аспекти, було вирішено використовувати саме Python як основну мову для реалізації додатку особистої бухгалтерії.

Мова Python має цікаву історію створення. Вона була розроблена у 1989 році програмістом Ґвідо ван Россумом у дослідницькому центрі CWI (Centrum Wiskunde & Informatica) в Нідерландах. Розробник прагнув створити мову, яка була б простою у використанні, гнучкою, легко читаємою, а також придатною для написання скриптів та повноцінних програм. Назва мови не має стосунку до змій – вона походить від комедійного шоу «Monty Python’s Flying Circus», що відображає прагнення творця зробити мову програмування такою, що приносить задоволення. Перша публічна версія Python 0.9.0 була випущена у 1991 році, і з того часу мова безперервно розвивалась, набираючи популярності у різних сферах – від наукових досліджень та машинного навчання до веб-розробки та фінансових систем.

Головна перевага Python, яка робить його привабливим для створення персональних фінансових додатків, – це його простота. Синтаксис мови максимально наближений до природної англійської мови, що дозволяє навіть новачкам швидко освоїти її основи. Це надзвичайно важливо у проєктах, де немає потреби в надмірній складності, а головне – це реалізувати логіку роботи програми

та створити зручний інтерфейс. Наприклад, оголошення циклів, умовних конструкцій, обробка файлів чи взаємодія з базою даних у Python займає значно менше коду, ніж у C++, Java чи C#.

Ще однією причиною вибору Python є його багата екосистема. У мові доступні тисячі бібліотек, які задовольняють практично будь-яку потребу – від обробки даних до створення графічного інтерфейсу. Для зберігання та обробки фінансової інформації відмінно підходить вбудована підтримка роботи з базами даних через бібліотеки SQLite або SQLAlchemy. За кілька рядків коду можна створити повноцінну таблицю витрат, додати новий запис або отримати звіт за обраний період. Наприклад, ось спрощений фрагмент коду на Python, який демонструє запис інформації про витрати в базу даних:

```
import sqlite3

conn = sqlite3.connect("expenses.db")

cursor = conn.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS expenses (id INTEGER PRIMARY KEY, category TEXT, amount REAL, date TEXT)")

cursor.execute("INSERT INTO expenses (category, amount, date) VALUES (?, ?, ?)", ("Їжа", 100.50, "2025-04-22"))

conn.commit()

conn.close()
```

Те, що тут реалізується за допомогою п’яти-шести рядків, у багатьох інших мовах вимагатиме значно більшої кількості коду, імпорту бібліотек, налаштувань та класів.

Іншим сильним аспектом Python є можливість легкого створення графічного інтерфейсу користувача. Наприклад, використовуючи Tkinter, можна створити

простий і водночас функціональний додаток з вікнами, кнопками, полями введення, меню тощо. Також є інші бібліотеки, такі як PyQt, PySide чи Kivy, які дозволяють створювати більш сучасні та кросплатформенні інтерфейси. Це дозволяє не тільки побудувати зручну взаємодію з користувачем, а й надати естетичного вигляду продукту.

Python також має величезну перевагу у своїй спільноті. Оскільки мова є однією з найпопулярніших у світі, її підтримують мільйони розробників по всьому світу. Це означає, що існує безліч прикладів коду, відкритих проєктів, форумів, відповідей на StackOverflow та навчальних матеріалів. У разі виникнення труднощів розробник майже завжди знайде рішення або готовий фрагмент коду, який можна адаптувати під власні потреби.

З технічної точки зору Python є кросплатформенною мовою, тобто код, написаний на ній, можна запускати на будь-якій операційній системі – Windows, macOS, Linux. Це забезпечує ширшу доступність додатку для користувачів, незалежно від того, якою операційною системою вони користуються. Крім того, Python підтримує розробку мобільних додатків і вебінтерфейсів, тому, за потреби, десктопний додаток особистої бухгалтерії з легкістю можна масштабувати до вебверсії або навіть створити мобільний додаток.

У порівнянні з іншими мовами програмування Python вирізняється своєю гнучкістю. Наприклад, Java, хоча й є надійною мовою з широким застосуванням у корпоративному середовищі, вимагає значно більше коду для досягнення тих самих результатів. Її строго типізований характер робить розробку менш гнучкою і повільнішою на етапі прототипування. C# має потужні можливості для створення десктопних додатків, особливо в середовищі Windows, але його кросплатформенність усе ще обмежена. Крім того, екосистема .NET дещо складніша для початківців.

C++ – це мова, яка дозволяє створювати високопродуктивні додатки, проте вона є низькорівневою і складною в розробці. Усі операції з пам’яттю потрібно

контролювати вручну, а створення GUI вимагає глибокого знання додаткових фреймворків, таких як Qt чи wxWidgets. Це робить розробку на C++ занадто складною і повільною для проєкту такого типу, як персональна бухгалтерія.

JavaScript у поєднанні з Electron дозволяє створювати кросплатформенні десктопні додатки, але такі програми зазвичай «важкі», мають високі вимоги до оперативної пам’яті, повільно запускаються і менш ефективні з точки зору ресурсоємності. Вони базуються на браузерному рушії, що робить їх гнучкими, але менш оптимізованими для простої десктопної утиліти.

Ще одним важливим аргументом на користь Python є його придатність до подальшого розширення функціоналу. Наприклад, у майбутньому додаток особистої бухгалтерії може бути доповнений елементами машинного навчання, які аналізуватимуть фінансову поведінку користувача, виявлятимуть закономірності та пропонуватимуть оптимізацію витрат. Python уже є лідером у сфері штучного інтелекту та аналізу даних, тому перехід до такого функціоналу буде максимально природним. Крім того, за допомогою API, Python може взаємодіяти з банківськими сервісами, імпортувати дані з таблиць Excel, надсилати автоматичні звіти на пошту користувача – усе це робить Python практично універсальним інструментом для розробника.

Підсумовуючи все вищесказане, можна впевнено стверджувати, що Python є найбільш оптимальним вибором для створення додатку особистої бухгалтерії. Його простота, багатство бібліотек, активна спільнота, гнучкість і можливість швидкого масштабування роблять його незамінним інструментом для такого типу програм. Усе це дозволяє зосередитися не на технічних деталях, а на вирішенні реальних задач користувача – контролі витрат, аналізі фінансів, побудові звітів та забезпеченні зручності у повсякденному використанні.

2.3 Використання мови програмування Python для розробки застосунку особистої бугалтерії

Мова програмування Python на сьогоднішній день - одна з найулюбленіших у світі, бо вона проста, зручна і надзвичайно потужна. Її створив Ґвідо ван Россум у кінці 1980-х, а публіці представили у 1991 році. Python від самого початку розробки задумувався як мова з читабельним кодом, що суттєво полегшує процес розробки та обслуговування програм. Це критично важливо для застосунків особистої бухгалтерії, де головне - надійність, правильна обробка фінансових даних і швидке оновлення програмного забезпечення.

Найбільша перевага Python – велика кількість готових бібліотек і модулів, які значно прискорюють розробку. Для програми особистої бухгалтерії корисні бібліотеки для роботи з базами даних, графічними інтерфейсами, фінансовими обчисленнями та візуалізацією даних. Наприклад, бібліотеки SQLite3, SQLAlchemy, Pandas, Matplotlib та Tkinter дозволяють розробнику швидко впровадити необхідний функціонал, не починаючи все з нуля. Ще однією значущою перевагою є інтеграція з іншими системами та підтримка різних форматів даних, таких як CSV, Excel чи JSON, що дає застосунку гнучкість.

Python має потужні засоби для роботи з базами даних, які необхідні для бухгалтерських застосувань. Вбудований модуль sqlite3 дозволяє створити локальну базу даних, де зберігатиметься вся фінансова інформація користувача. Для більшої масштабованості або роботи з даними в мережі, можна використовувати зовнішні бібліотеки для роботи з PostgreSQL, MySQL чи іншими СУБД. Гнучкість вибору технології зберігання даних дозволяє адаптувати систему під конкретні вимоги користувача без великих витрат часу.

Розробка графічного інтерфейсу користувача (GUI) для застосунку особистої бухгалтерії - ключовий етап, оскільки зручність використання програми залежить від зручності взаємодії користувача з нею. Python пропонує низку інструментів для створення GUI. Одним з найпопулярніших є Tkinter, що входить до стандартної

поставки. Tkinter дає змогу створити інтуїтивно зрозумілі та функціональні віконні програми з формами для введення витрат і доходів, перегляду історії операцій, побудови графіків витрат тощо. Для розробників, які бажають сучасніший інтерфейс, доступні бібліотеки PyQt або Kivy, що дозволяють створювати кросплатформені застосунки з розширеним дизайном.

Програми особистої бухгалтерії вимагають реалізації логіки обробки фінансової інформації. Python, завдяки своєму зрозумілому синтаксису, дає змогу ефективно розраховувати залишки на рахунках, відстежувати регулярні платежі, прогнозувати майбутні витрати або доходи. Бібліотека Pandas є незамінною для обробки табличних фінансових даних, оскільки надає зручні механізми фільтрації, групування та аналізу великих обсягів записів. Завдяки можливості швидкої обробки даних розробник може легко впровадити додаткові функції, наприклад, автоматичне формування звітів про фінансовий стан користувача за певний період або аналітику витрат за категоріями.

Важливою вимогою при створенні бухгалтерського застосунку є захист персональної інформації користувача. Python має низку вбудованих і сторонніх інструментів для захисту даних. Для шифрування конфіденційної інформації можна використовувати бібліотеки Cryptography або hashlib. Крім того, завдяки простоті інтеграції Python із системами аутентифікації можна забезпечити багаторівневий захист доступу до застосунку, наприклад, через авторизацію за паролем або біометричними даними. Реалізація резервного копіювання бази даних також є важливою функцією, яку легко впровадити, використовуючи можливості роботи з файловою системою.

Python є міжплатформенною мовою, що дозволяє створювати програми, які однаково працюватимуть на Windows, Linux та macOS. Завдяки цьому кінцеві користувачі можуть користуватися застосунком незалежно від їхнього пристрою. Під час розробки особистої бухгалтерії це дає змогу охопити ширшу аудиторію користувачів та полегшити оновлення і підтримку програмного забезпечення.

Перевага Python - велика та активна спільнота розробників, яка надає велику кількість готових рішень, документації, форумів та навчальних матеріалів. Це значно полегшує розробку, пошук відповідей на питання та вирішення типових проблем. Завдяки багатьом відкритим проєктам можна знаходити приклади реалізації різних функцій бухгалтерських застосунків, адаптувати їх до власного продукту та значно скорочувати час розробки.

Python також дозволяє інтегрувати машинне навчання в програму особистої бухгалтерії, що відкриває нові можливості аналізу фінансових даних. З бібліотеками Scikit-learn або TensorFlow можна навчити систему розпізнавати типові шаблони витрат користувача, прогнозувати фінансові результати на основі попередніх даних або автоматично рекомендувати оптимальні способи заощадження. Такі можливості дають змогу створити "розумніший" та корисніший для користувача застосунок.

Процес тестування та налагодження програмного продукту також є важливим. Завдяки Python, тестування можна автоматизувати за допомогою бібліотек unittest або pytest. Це дозволяє виявляти помилки на ранніх етапах розробки та забезпечувати високу якість кінцевого продукту. Оскільки застосунки для особистої бухгалтерії працюють з важливою інформацією, надійність і відсутність помилок мають ключове значення.

У Python також добре налаштований процес створення документації до коду, що важливо для подальшої підтримки та розвитку програми. Використання стандартів документування, таких як PEP 257, дозволяє створювати зрозумілу та структуровану документацію, що полегшує роботу іншим розробникам або самому автору в майбутньому.

Отже, використання Python для розробки програми особистої бухгалтерії є обґрунтованим та правильним вибором. Python забезпечує легкість впровадження, широкий набір інструментів для обробки даних, гнучкість при створенні графічного інтерфейсу, надійність зберігання інформації та високий рівень

безпеки. Все це робить Python ідеальним інструментом для створення ефективних, сучасних та зручних для користувачів застосунків особистої бухгалтерії.

3 ОПИС ПРАКТИЧНОЇ ЧАСТИНИ

3.1 Бібліотека Tkinter

Бібліотека Tkinter — стандартна бібліотека Python для створення графічного інтерфейсу користувача (GUI). Вона є частиною стандартного набору Python, отже не потребує додаткового встановлення. Tkinter базується на бібліотеці Tcl/Tk, призначеній для розробки кросплатформених GUI. Завдяки Tkinter розробники можуть створювати вікна, кнопки, текстові поля, таблиці, панелі, меню та інші елементи керування у своїх програмах.

В застосунку особистої бухгалтерії Tkinter відіграє ключову роль, адже саме вона реалізує графічний інтерфейс програми. З її допомогою користувач взаємодіє з базою даних через кнопки, випадаючі списки для вибору типу операції, поля для введення суми та коментарів, а також графічний календар для вибору дати. Це робить застосунок зручним та доступним для користувачів без технічних знань.

У коді застосунку Tkinter використовується для створення головного вікна за допомогою функції Tk(). Вікно має назву "Домашня Бухгалтерія", фіксований розмір 700x340 пікселів та заборону зміни розміру. Це забезпечує стабільний вигляд програми на різних пристроях.

Для організації інтерфейсу використовуються контейнерні елементи Frame, які розділяють простір вікна на логічні частини: ліва панель для введення даних, права частина з таблицею транзакцій та верхній блок з кнопками керування. Це дозволяє зробити інтерфейс зручним і візуально ненавантаженим.

Всі текстові написи, поля введення, кнопки та таблиця також створені з використанням Tkinter. Наприклад, елементи Label використовуються для написів полів ("Дата", "Тип", "Сума", "Коментар"), Entry відповідають за текстовий ввід, а ttk.Combobox (модуль ttk є частиною Tkinter) дозволяє обрати тип транзакції зі списку ("Дохід" або "Витрата"). Кнопки для додавання, видалення та редагування

транзакцій створені з використанням Button, кожна з яких викликає відповідну функцію для роботи з базою даних.

Особливу роль відіграє Treeview, який дозволяє відображати таблицю транзакцій у зручному форматі. Користувач бачить дату, тип, суму та коментар кожної операції та може взаємодіяти з кожним записом. До Treeview приєднано Scrollbar, що забезпечує вертикальну прокрутку, якщо кількість транзакцій перевищує висоту вікна.

Завдяки функціональності Tkinter, застосунок виглядає цілісно, структуровано та функціонально. Бібліотека дозволила створити зрозумілий інтерфейс, який працює без додаткових залежностей, легко адаптується під потреби користувача та може бути розширений у майбутньому. Важливо, що Tkinter є кросплатформенною технологією, яка дозволяє запускати програму на Windows, macOS або Linux без змін у коді.

Отже, Tkinter забезпечує графічну складову проєкту, завдяки якій взаємодія користувача з системою зводиться до простих дій: натискання кнопок, введення даних у вікнах та перегляд інформації в таблиці, — що й робить програму ефективною в контексті особистої бухгалтерії.

3.2 Імпортування необхідних бібліотек

Ключовий етап втілення застосунку особистої бухгалтерії – імпортування необхідних бібліотек. Саме з нього починається процес розбудови логіки та функціоналу програми, адже імпорт дає змогу застосовувати сторонні чи стандартні модулі, що надають специфічні можливості, яких немає у базовому синтаксисі Python.

Найпершим імпортуємо клас DateEntry з модуля tkcalendar. Цей елемент дає можливість реалізувати зручний графічний календар, за допомогою якого користувач зможе обирати дату транзакції. Це значно полегшує взаємодію з

інтерфейсом та зменшує ризик помилки при ручному введенні дати. Клас DateEntry інтегрується з системою Tkinter і є додатковим віджетом, що не входить до базового набору, тому його треба встановити окремо, використовуючи менеджер пакетів pip. Вибір цієї бібліотеки обґрунтований її сумісністю з Tkinter та легкістю інтеграції.

Наступний крок – імпортування набору модулів з графічної бібліотеки tkinter, котра є стандартним інструментом для створення графічних інтерфейсів у Python. Зокрема, відбувається імпорт усіх базових компонентів (from tkinter import *), а також модулів ttk та messagebox. Імпорт * надає доступ до ключових класів, таких як Tk, Label, Entry, Button, Frame тощо, без необхідності писати префікс перед кожним з них. Це робить код компактнішим і зручнішим для читання.

Модуль ttk (Themed Tk) відповідає за стилізовані версії стандартних віджетів. В рамках даного застосунку за його допомогою реалізовано віджет Combobox для вибору типу транзакції та Treeview для табличного представлення введених користувачем даних. Ці елементи мають сучасніший вигляд і великі можливості кастомізації, що робить інтерфейс привабливішим та зручнішим для сприйняття.

Інтеграція модуля messagebox дає можливість відображати діалогові вікна, котрі сповіщають користувача про виконані дії, помилки чи просять підтвердження. Наприклад, сповіщення про успішне додавання транзакції, помилку при введенні суми або підтвердження видалення всіх записів реалізовано саме через цей модуль. Така взаємодія значно покращує рівень зручності та інтуїтивності інтерфейсу.

Останнім етапом імпорту є підключення модуля sqlite3. Це вбудований модуль Python, котрий забезпечує роботу з реляційною базою даних SQLite. У цьому застосунку база даних finance.db створюється автоматично при першому запуску програми, якщо її ще немає. Через бібліотеку sqlite3 реалізується створення таблиці для збереження транзакцій, додавання нових записів, їх оновлення, видалення, а також зчитування з бази для відображення в інтерфейсі. Використання

SQLite пояснюється її простотою, автономністю та відсутністю потреби у зовнішньому сервері баз даних, що робить її ідеальним вибором для настільного персонального застосунку.

Отже, імпорт бібліотек є наріжним каменем у розробці застосунку. Він надає доступ до інструментів, які реалізують як зовнішній вигляд інтерфейсу, так і його функціональне наповнення. Правильно підібраний та структурований імпорт модулів дає змогу досягти високого рівня інтеграції між графічною оболонкою, базою даних та логікою обробки подій, що в сукупності забезпечує надійність та зручність використання програмного продукту.

3.3 Створення бази даних

Одним з наріжних каменів роботи застосунку персональної бухгалтерії є створення та використання бази даних, де зберігаються всі фінансові операції користувача. В даному випадку задіяна вбудована база даних SQLite, що є складовою стандартної бібліотеки Python та не потребує окремого встановлення чи налаштування серверу. Це робить її чудовим варіантом для настільних додатків, що не передбачають велику кількість одночасних користувачів.

База даних формується під час першого запуску програми за допомогою модуля sqlite3, що імпортується на початку коду. З'єднання з базою встановлюється через виклик функції sqlite3.connect("finance.db"), в результаті чого утворюється файл бази даних під назвою finance.db в кореневому каталозі проєкту. Якщо файл вже існує, програма підключається до наявної бази.

Для взаємодії з базою даних створюється курсор (cur = connection.cursor()), котрий використовується для виконання SQL-запитів. Одразу після встановлення з'єднання виконується команда SQL для створення таблиці transactions, в якій зберігаються всі фінансові операції користувача. Таблиця має таку структуру:

- id — унікальний ідентифікатор транзакції, який автоматично збільшується (PRIMARY KEY AUTOINCREMENT);
- date — дата здійснення транзакції, збережена у вигляді тексту;
- type — тип транзакції: "Дохід" або "Витрата";
- amount — сума транзакції, що зберігається у числовому форматі з плаваючою комою;
- comment — текстовий коментар до транзакції, який дозволяє описати її зміст.

Створення таблиці реалізовано через SQL-запит:

```
cur.execute("""
    CREATE TABLE IF NOT EXISTS transactions (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT,
        type TEXT,
        amount REAL,
        comment DATE
        )
""")
```

Використання конструкції IF NOT EXISTS гарантує, що таблиця буде створена тільки в тому разі, якщо вона ще не існує. Це запобігає повторному створенню таблиці при кожному запуску програми та дає змогу зберігати дані між сесіями.

Після створення таблиці викликається метод connection.commit(), котрий зберігає всі зміни в базі даних. На цьому етапі база готова до подальшої роботи: додавання, редагування, видалення та зчитування транзакцій. Уся взаємодія із записами реалізована в наступних частинах програми, проте основа закладається саме на цьому етапі — створенні бази даних.

Отже, розробка власної бази даних дозволяє користувачеві зручно й безпечно зберігати свої фінансові записи у локальному середовищі, що є важливою вимогою для застосунку особистої бухгалтерії. Застосування SQLite дає змогу уникнути труднощів з адмініструванням бази, водночас забезпечуючи швидкий доступ до інформації та надійність її зберігання.

3.4 Додавання транзакції

Ця функціональність втілюється через окрему функцію add_transaction(), яка дає змогу користувачеві оперативно внести відомості про фінансову операцію у зручному вигляді. Завдяки простому графічному інтерфейсу, побудованому за допомогою бібліотеки Tkinter, процес додавання транзакцій є інтуїтивно зрозумілим навіть для користувачів без технічної освіти.

При натисканні кнопки «Додати транзакцію» користувач активує виклик функції add_transaction(). Перед початком запису інформації в базу даних виконується базова перевірка: чи заповнені поля для типу транзакції (type_combobox.get()) та суми (amount_entry.get()). Це необхідно для запобігання запису неповної або некоректної інформації. Якщо дані відсутні, програма виводить попереджувальне повідомлення:

```
if type_combobox.get() and amount_entry.get():
```

Далі програма намагається перетворити суму з текстового формату у числовий (float). Якщо користувач ввів некоректне значення, наприклад, текст замість цифр, буде згенеровано повідомлення про помилку:

```
try:
    amount = float(amount_entry.get())
    ...
except ValueError:
```

messagebox.showerror("Помилка", "Суму введена некоректно!")

Після успішного зчитування значень з усіх полів інтерфейсу (тип транзакції, сума, коментар і дата), створюється SQL-запит на додавання нової транзакції до таблиці transactions. Це відбувається за допомогою методу cur.execute():

```
cur.execute("""
    INSERT INTO transactions (date, type, amount, comment)
    VALUES (?, ?, ?, ?)
    """, (date, transaction_type, amount, comment))
```

Використання параметризованого запиту із знаками питання ? гарантує захист від SQL-ін’єкцій та підвищує безпеку додатку. Після виконання запиту викликається метод connection.commit(), який зберігає зміни в базі даних:

```
connection.commit()
```

Щоб миттєво відобразити нову транзакцію у графічному інтерфейсі, відбувається автоматичне додавання відповідного запису у віджет Treeview. Для цього спочатку зчитується унікальний ідентифікатор щойно доданої транзакції за допомогою cur.lastrowid, а потім додається рядок з усіма даними:

```
transaction_id = cur.lastrowid
treeview.insert("", END, values=(transaction_id, date, transaction_type, amount, comment))
```

Після успішного додавання запису всі поля очищаються, щоб підготувати інтерфейс до введення нової транзакції. Це реалізовано через виклики set() і delete():

```
type_combobox.set("")
amount_entry.delete(0, END)
comment_entry.delete(0, END)
```

Крім того, з’являється інформаційне вікно, яке підтверджує, що транзакція додана успішно:

messagebox.showinfo("Успіх", "Транзакцію успішно додано!")

У разі відсутності обов’язкових полів або помилки введення, замість виконання SQL-запиту користувач отримує відповідне попередження або повідомлення про помилку. Це важлива складова захисту даних від некоректного введення та підвищення зручності використання застосунку.

Таким чином, процес додавання транзакції у програмі реалізований максимально просто, надійно та зрозуміло. Він забезпечує збереження фінансових даних користувача, їх миттєве відображення у таблиці, а також мінімізує ймовірність помилок під час введення. Уся логіка обробки даних зосереджена у функції add_transaction(), яка є однією з центральних в архітектурі застосунку.

3.5 Видалення транзакції

Вилучення транзакції є необхідною функціональністю будь-якої програми для обліку фінансів. Це дозволяє користувачеві швидко усувати хибні або застарілі записи з бази даних, підтримуючи її актуальність та чистоту. У створеному застосунку особистої бухгалтерії на Python з використанням бібліотек Tkinter і SQLite ця змога реалізована за допомогою окремої функції delete_transaction().

Основна логіка функції побудована на взаємодії з графічним компонентом Treeview, який відображає перелік всіх транзакцій. Щоб вилучити транзакцію, користувач мусить спочатку виділити її у візуальній таблиці, після чого натиснути кнопку «Видалити транзакцію», яка викликає функцію delete_transaction(). Найперше програма перевіряє, чи було обрано елемент у таблиці:

selected_item = treeview.selection()

if selected_item:

...

else:

messagebox.showinfo("Попередження", "Виберіть транзакцію для видалення!")

Якщо жодного запису не виділено, користувачеві виводиться інформаційне повідомлення із проханням вибрати транзакцію. У випадку, коли транзакцію обрано, програма зчитує її унікальний ідентифікатор (id) через метод treeview.item():

item_values = treeview.item(selected_item, "values")

transaction_id = item_values[0]

Отримавши ідентифікатор, застосунок формує SQL-запит для видалення відповідного запису з таблиці transactions у базі даних SQLite:

cur.execute("DELETE FROM transactions WHERE id=?", (transaction_id,))

connection.commit()

Після успішного виконання запиту зміни фіксуються за допомогою commit(), а видалений запис також усувається з інтерфейсу (віджета Treeview) для відображення оновленої картини фінансових записів:

treeview.delete(selected_item)

Завершується функція інформаційним повідомленням про успішне видалення:

messagebox.showinfo("Успіх", "Транзакцію успішно видалено!")

Таким чином, реалізована функціональність видалення транзакцій є важливим компонентом управління базою даних у застосунку. Вона дозволяє користувачеві контролювати вміст своєї фінансової історії, забезпечуючи точність, гнучкість і зручність використання. Простий алгоритм дій (вибрати запис,

натиснути кнопку, підтвердити дію) поєднаний з перевіркою коректності дій користувача та виведенням зворотного зв’язку, що робить роботу із застосунком комфортною та безпечною.

3.6 Видалення всіх транзакцій

У процесі використання застосунку для ведення особистої бухгалтерії виникає потреба не тільки видаляти окремі записи, а й повністю очищати базу даних від усіх транзакцій. Така функціональність може знадобитися, наприклад, на початку нового облікового періоду, зміні користувача або тестуванні роботи системи. У розробленому застосунку ця задача реалізується за допомогою функції delete_all_transactions(), яка дозволяє швидко та ефективно ліквідувати всі записи з таблиці transactions.

Функція починається з виклику діалогового вікна підтвердження, що запобігає випадковому видаленню всіх даних. Це реалізовано за допомогою методу askyesno() з модуля messagebox, який виводить користувачеві запит з двома варіантами — "Так" або "Ні":

confirm = messagebox.askyesno("Попередження", "Ви впевнені, що бажаєте видалити всі транзакції?")

Тільки у випадку, якщо користувач натискає кнопку "Так", програма переходить до виконання SQL-запиту на повне очищення таблиці:

```
if confirm:
    cur.execute("DELETE FROM transactions")
    connection.commit()
```

Цей запит DELETE FROM transactions видаляє всі рядки з таблиці, після чого connection.commit() фіксує зміни у базі даних. Варто зазначити, що структура таблиці при цьому не змінюється, лише її вміст. Після очищення бази даних,

програма також видаляє всі рядки з графічного інтерфейсу таблиці Treeview, щоб інтерфейс залишався синхронізованим з вмістом бази:

treeview.delete(*treeview.get_children())

Цей рядок видаляє всі дочірні елементи Treeview, ефективно оновлюючи таблицю на екрані. Завершальним кроком є виведення повідомлення про успішне завершення операції:

messagebox.showinfo("Успіх", "Усі транзакції успішно видалено!")

Загалом, функціональність повного видалення даних реалізована таким чином, щоб забезпечити максимальну безпеку дій користувача — за допомогою підтвердження, збереження послідовності дій і миттєвого візуального оновлення. Це дозволяє користувачу ефективно контролювати та керувати своїми фінансовими даними відповідно до індивідуальних потреб.

3.7 Редагування транзакції

Редагування транзакції є невід'ємною опцією будь-якої фінансової програми, що прагне зручності та гнучкості в використанні. Часто користувачі можуть випадково ввести суму, дату чи тип транзакції, або ж виникає необхідність змінити коментар до вже створеного запису. В рамках розробленої програми особистої бухгалтерії на Python функція редагування реалізована через логіку роботи з базою даних SQLite та графічним інтерфейсом на основі бібліотеки Tkinter.

Редагування починається з вибору транзакції у віджеті Treeview. Користувач натискає на запис, який треба змінити, після чого його дані автоматично підставляються у відповідні поля введення завдяки функції on_row_click(event). Цей механізм дозволяє швидко перейти до редагування без ручного пошуку чи повторного введення інформації.

def on_row_click(event):

```
if treeview.selection():
    item = treeview.selection()[0]
    values = treeview.item(item, "values")

    date_entry.delete(0, END)
    date_entry.insert(0, values[1])
    type_combobox.set(values[2])
    amount_entry.delete(0, END)
    amount_entry.insert(0, values[3])
    comment_entry.delete(0, END)
    comment_entry.insert(0, values[4])
```

Після цього користувач змінює потрібні дані у полях форми та натискає кнопку «Редагувати транзакцію», що запускає функцію edit_transaction(). Насамперед у цій функції виконується перевірка — чи вибрано транзакцію для редагування:

```
selected_item = treeview.selection()
if not selected_item:
    messagebox.showerror("Транзакцію не вибрано!", "Будь ласка, оберіть транзакцію в таблиці, щоб відредагувати.")
    return
```

Якщо запис вибрано, програма зчитує id транзакції — це унікальний ідентифікатор, який потрібен для коректного оновлення даних у базі:

```
transaction_id = treeview.set(selected_item, "#1")
```

Після цього програма отримує нові значення з полів введення:

```
date = date_entry.get()

transaction_type = type_combobox.get()

amount = amount_entry.get()

comment = comment_entry.get()
```

Далі формується SQL-запит для оновлення конкретного запису в таблиці transactions:

```
cur.execute("UPDATE transactions SET date = ?, type = ?, amount = ?, comment = ? WHERE id = ?",

            (date, transaction_type, amount, comment, transaction_id))

connection.commit()
```

Завершується функція оновленням відповідного рядка в графічному інтерфейсі Treeview, що забезпечує візуальну відповідність актуальним даним:

```
treeview.item(selected_item, values=(transaction_id, date, transaction_type, amount, comment))
```

Таким чином, процес редагування у застосунку побудований з урахуванням інтерактивності, простоти використання та безпеки змін. Всі дії супроводжуються відповідними перевірками та оновленнями інтерфейсу, що дозволяє користувачеві працювати з транзакціями без ризику порушення структури бази або втрати даних. Це значно підвищує функціональність і практичну цінність застосунку для повсякденного обліку особистих фінансів.

3.8 Вибір транзакції для редагування

Процес обрання транзакції для коригування є важливим етапом у реалізації функціоналу редагування фінансових записів. Він гарантує зручність у роботі з даними та дозволяє уникнути помилок при оновленні інформації. У створеному

застосунку особистої бухгалтерії, цей механізм реалізовано за допомогою взаємодії користувача з графічним компонентом Treeview, що надає візуальне відображення переліку всіх транзакцій з бази даних SQLite.

Обрання транзакції реалізовано завдяки прив’язці функції on_row_click() до події натискання кнопки миші на таблиці. Це дає можливість у момент кліку по певному рядку автоматично витягати пов’язані з ним дані та відображати їх у відповідних полях форми коригування:

```
treeview.bind("<ButtonRelease-1>", on_row_click)
```

Функція on_row_click(event) відповідає за отримання інформації про виділений запис у таблиці:

```
def on_row_click(event):
    if treeview.selection():
        item = treeview.selection()[0]
        values = treeview.item(item, "values")
```

У змінну item записується ідентифікатор виділеного рядка, а далі за допомогою treeview.item(item, "values") витягується список значень, які відображають поля транзакції: дату, тип, суму та коментар. Отримані значення використовуються для автоматичного заповнення елементів інтерфейсу, зокрема полів введення Entry та комбобоксу Combobox:

```
date_entry.delete(0, END)
date_entry.insert(0, values[1])
type_combobox.set(values[2])
amount_entry.delete(0, END)
amount_entry.insert(0, values[3])
comment_entry.delete(0, END)
```

```
comment_entry.insert(0, values[4])
```

Такий підхід дозволяє зручно працювати з транзакціями, не вимагаючи від користувача вручну копіювати чи вводити повторно вже існуючі дані. Усі введені значення зберігаються в текстових полях, де можуть бути змінені й оновлені за допомогою функції редагування. Також це дає змогу уникнути випадкового редагування неправильної транзакції, адже редагуються лише ті записи, які були явно вибрані користувачем у таблиці.

Таким чином, механізм вибору транзакції є невід’ємною частиною інтерфейсної логіки застосунку, що відповідає за точне, швидке та зручне редагування даних. Його реалізація забезпечує тісну інтеграцію між візуальним представленням даних і логікою їх обробки, що суттєво покращує користувацький досвід.

3.9 Основне вікно застосунку

Головне вікно застосунку є центральним елементом графічного інтерфейсу, через котрий відбувається вся взаємодія користувача із системою обліку особистих фінансів. Воно об'єднує всі функціональні компоненти в єдину структуру, дозволяючи зручно додавати, переглядати, редагувати та видаляти транзакції. Побудова інтерфейсу реалізована за допомогою модуля tkinter, що забезпечує широкі можливості для створення віконних додатків мовою Python. Запуск головного вікна здійснюється через створення об'єкта класу Tk():

```
root = Tk()

root.title("Домашня Бухгалтерія")

root.geometry("700x340")

root.resizable(False, False)
```

Тут задається заголовок вікна, розмір (у пікселях) і обмеження на зміну його розмірів. Це забезпечує стабільне відображення всіх елементів без спотворення чи зміщення. Головне вікно поділене на три основні частини: ліву панель для введення нових даних, верхню панель для керування записами (редагування, видалення), а також праву панель для виведення таблиці з транзакціями. Ліва панель реалізована як окремий Frame, до якого додаються всі елементи для введення транзакції: поле вибору дати (DateEntry з модуля tkcalendar), комбобокс вибору типу транзакції (ttk.Combobox), поля введення суми та коментаря, а також кнопка "Додати транзакцію":

left_frame = Frame(root, bd=2, relief=SUNKEN)

left_frame.pack(side=LEFT, pady=10)

Аналогічно створюється верхня панель з кнопками для видалення та редагування записів:

button_frame = Frame(root, bd=2, relief=SUNKEN)

button_frame.pack(side=TOP, fill=BOTH, padx=10, pady=10)

У правій частині головного вікна знаходиться таблиця транзакцій, створена за допомогою ttk.Treeview. Вона дозволяє в зручному вигляді переглядати всі наявні записи з бази даних. Для покращення навігації до неї додано вертикальний скролбар (ttk.Scrollbar):

treeview = ttk.Treeview(data_frame)

scrollbar = ttk.Scrollbar(data_frame, orient="vertical", command=treeview.yview)

treeview.configure(yscrollcommand=scrollbar.set)

Налаштування таблиці включає визначення колонок, їх ширини, заголовків, а також приховування нульової колонки #0, яка не використовується у цьому проєкті:

treeview["columns"] = ("id", "date", "type", "amount", "comment")

```
treeview.column("#0", width=0, stretch=NO)

treeview.column("id", anchor=W, width=0)

treeview.column("date", anchor=W, width=100)

treeview.column("type", anchor=W, width=100)

treeview.column("amount", anchor=W, width=100)

treeview.column("comment", anchor=W, width=200)
```

Для кожної з колонок задається заголовок, який відображається у верхній частині таблиці:

```
treeview.heading("date", text="Дата")

treeview.heading("type", text="Тип")

treeview.heading("amount", text="Сума, $")

treeview.heading("comment", text="Коментар")
```

Завершується побудова основного вікна запуском головного циклу застосунку:

```
root.mainloop()
```

Цей виклик забезпечує безперервну роботу вікна та обробку всіх дій користувача в реальному часі. Таким чином, основне вікно застосунку об'єднує всі функції в зручний інтерфейс, створюючи цілісну, логічно організовану та інтуїтивно зрозумілу систему керування особистими фінансами.

3.10 Віджет DateEntry

У додатку для ведення особистої бухгалтерії важливу роль відіграє вибір дати транзакції, адже коректне хронологічне розміщення фінансових записів дозволяє ефективно аналізувати прибутки та витрати, будувати графіки або генерувати

звіти. Для забезпечення зручного введення дати у графічному інтерфейсі було використано віджет DateEntry з бібліотеки tkcalendar. DateEntry є спеціалізованим віджетом, який дозволяє користувачу вибирати дату з випадаючого календаря, що значно зменшує вірогідність помилки при введенні вручну. Він поєднує в собі поле введення тексту з кнопкою, котра відкриває міні-календар, де користувач може вибрати конкретний день, місяць та рік. У цьому додатку цей віджет реалізовано наступним чином:

```
from tkcalendar import DateEntry
```

Бібліотека tkcalendar не входить до стандартної поставки Python, тому її необхідно попередньо встановити за допомогою менеджера пакетів pip:

```
pip install tkcalendar
```

Створення самого віджету здійснюється всередині лівої панелі інтерфейсу, призначеної для додавання нової транзакції:

```
date_entry = DateEntry(left_frame, width=12, background="darkblue",
                       foreground="white", borderwidth=2, date_pattern="dd.mm.y")
date_entry.pack(anchor="w", padx=10)
```

Цей рядок створює об’єкт DateEntry, який додається до left_frame. Налаштування width визначає ширину поля, параметри background і foreground — кольори тла і тексту в календарі. borderwidth задає товщину рамки, а date_pattern="dd.mm.y" визначає формат відображення дати — день, місяць і рік у традиційному для України форматі. Завдяки цим налаштуванням календар має привабливий зовнішній вигляд, зручний у використанні навіть для недосвідчених користувачів. Обрана дата автоматично зберігається у полі date_entry і може бути отримана за допомогою методу get():

```
date = date_entry.get()
```

Значення дати передається далі у функції додавання, редагування чи видалення транзакцій як один з ключових атрибутів фінансового запису. Отож, використання DateEntry значно покращує зручність взаємодії з користувачем і мінімізує ризик введення некоректних чи нечитабельних дат. Крім того, автоматичне форматування дати дозволяє забезпечити її коректне зберігання у базі даних SQLite у вигляді текстового значення. Це спрощує подальшу обробку даних при фільтрації або сортуванні транзакцій за часовим критерієм. Отже, DateEntry є важливим компонентом застосунку, який забезпечує точність введення дати, зручність використання та візуальну привабливість інтерфейсу.

3.11 Вибір типу транзакції

У програмі для ведення особистої бухгалтерії важливо не тільки фіксувати суму й дату транзакції, а й вказувати її тип — тобто, чи є вона прибутком чи витратою. Це дає можливість ефективно класифікувати фінансові операції, аналізувати баланс, будувати статистику та приймати обґрунтовані фінансові рішення. Для реалізації зручного вибору типу транзакції в графічному інтерфейсі було застосовано віджет Combobox з бібліотеки ttk, яка є частиною стандартного модуля tkinter. Віджет Combobox являє собою комбіноване поле — поєднання текстового поля та випадаючого переліку з фіксованим набором значень. У нашому випадку значеннями є "Прибуток" та "Витрата", що дає змогу користувачеві зробити чіткий і швидкий вибір типу транзакції, виключаючи можливість помилки введення або неправильного формулювання. Визначення та налаштування елементу інтерфейсу для вибору типу транзакції реалізовано таким чином:

```
type_label = Label(left_frame, text="Тип:")
type_label.pack(anchor="w", pady=10, padx=10)
```

type_combobox = ttk.Combobox(left_frame, values=["Дохід", "Витрата"], state="readonly")

type_combobox.pack(anchor="w", padx=10)

Ці рядки створюють підпис (label) "Тип:", а також сам комбінований список type_combobox, який додається до лівої панелі інтерфейсу (left_frame). Аргумент values вказує перелік можливих значень для вибору — у цьому випадку, лише два варіанти. Параметр state="readonly" обмежує введення вручну, змушуючи користувача обирати лише один із запропонованих типів. Це гарантує стандартизованість і коректність даних. Під час додавання нової транзакції вибране значення типу отримується наступною командою:

transaction_type = type_combobox.get()

Значення transaction_type надалі застосовується для зберігання у базі даних SQLite, де воно виступає як один із чотих головних параметрів кожного фінансового запису: дата, вид, сума, коментар. Функціонально, коректний вибір виду транзакції дозволяє в майбутньому виконувати аналітичні операції — наприклад, обчислювати загальний дохід або витрати за певний період, створювати діаграми витрат, здійснювати пошук і фільтрацію даних за видом. Саме тому Combobox є зручною та практичною формою для взаємодії користувача з програмою у контексті вибору виду транзакції. Отже, елемент type_combobox виконує ключову роль у забезпеченні структури даних у застосунку, сприяє точності введення і є невід’ємною складовою зручного графічного інтерфейсу персональної бухгалтерії.

3.12 Таблиця транзакцій

Однією з найважливіших складових графічного інтерфейсу застосунку для особистої бухгалтерії є таблиця, в якій показуються усі внесені користувачем фінансові операції. Така таблиця дозволяє швидко переглядати історію доходів і

витрат, відшукувати потрібні записи, а також редагувати або видаляти їх. Для реалізації таблиці в даному програмному рішенні було використано віджет Treeview з бібліотеки ttk, що є частиною модуля tkinter.

Treeview — це спеціалізований віджет, що дозволяє створювати ієрархічні або табличні подання даних. У нашому випадку він застосовується як таблиця з фіксованою кількістю стовпців, кожен з котрих відповідає певному атрибуту транзакції: унікальний ідентифікатор (ID), дата, тип (дохід або витрата), сума, коментар. Створення і налаштування таблиці відбувається наступним чином:

```
treeview = ttk.Treeview(data_frame)
treeview.pack(side=LEFT, fill=BOTH, expand=True)


scrollbar = ttk.Scrollbar(data_frame, orient="vertical", command=treeview.yview)
scrollbar.pack(side=RIGHT, fill=Y)
treeview.configure(yscrollcommand=scrollbar.set)
```

Для зручності користувача також додається вертикальна смуга прокручування (ScrollBar), яка автоматично з’являється у разі великої кількості записів. Після цього задається структура стовпців таблиці:

```
treeview["columns"] = ("id", "date", "type", "amount", "comment")
treeview.column("#0", width=0, stretch=NO)
treeview.column("id", anchor=W, width=0)
treeview.column("date", anchor=W, width=100)
treeview.column("type", anchor=W, width=100)
treeview.column("amount", anchor=W, width=100)
treeview.column("comment", anchor=W, width=200)
```

```
treeview.heading("#0", text="")
treeview.heading("id", text="ID")
treeview.heading("date", text="Дата")
treeview.heading("type", text="Тип")
treeview.heading("amount", text="Сума, $")
treeview.heading("comment", text="Коментар")
```

Перший стовпець #0 у Treeview є системним, тому він вимикається, встановлюючи ширину в 0. Інші стовпці мають чіткі назви та ширину, достатню для відображення текстових значень. Для заповнення таблиці даними при запуску застосунку використовується SQL-запит, який витягує всі наявні транзакції з бази даних:

```
cur.execute("SELECT * FROM transactions")
rows = cur.fetchall()

for row in rows:
    treeview.insert("", END, values=row)
```

Кожен рядок, отриманий з бази, додається в таблицю за допомогою методу insert. Параметр "" означає, що запис не має батьківського елемента (тобто не є вкладеним), а END додає новий запис в кінець списку. Завдяки цьому користувач одразу після запуску застосунку бачить повну історію своїх фінансових операцій у зручному табличному вигляді. Також реалізована взаємодія між таблицею та формою редагування — при натисканні на рядок викликається функція, яка переносить дані з таблиці у відповідні поля вводу для подальшого редагування:

```
treeview.bind("<ButtonRelease-1>", on_row_click)
```

Таким чином, таблиця Treeview є центральним елементом візуального представлення фінансових даних у застосунку. Вона забезпечує інтерактивність, зручний перегляд і маніпуляцію даними, що значно підвищує функціональність програми та комфорт роботи користувача з особистими фінансами.

3.13 Прокрутка таблиці

У випадках, коли користувач вносить значну кількість фінансових записів, таблиця з транзакціями швидко заповнюється, і всі записи не можуть одночасно вміститися у вікні застосунку. Для забезпечення зручної навігації по великому переліку даних у таблиці використовується механізм вертикального прокручування (scrollbar). Цей елемент інтерфейсу є важливою складовою, що покращує ергономіку програми та дозволяє швидко переглядати як останні, так і найдавніші транзакції без перевантаження вікна застосунку. Прокручування реалізоване за допомогою віджета Scrollbar з модуля ttk, який інтегрується з таблицею Treeview. Конструкція прокручування виглядає наступним чином:

```
scrollbar = ttk.Scrollbar(data_frame, orient="vertical", command=treeview.yview)

scrollbar.pack(side=RIGHT, fill=Y)

treeview.configure(yscrollcommand=scrollbar.set)
```

В цьому фрагменті коду створюється новий об’єкт Scrollbar, який приєднується до контейнера data_frame, де міститься сама таблиця. Параметр orient="vertical" вказує на вертикальне розташування смуги прокрутки, а параметр command=treeview.yview пов’язує її з вертикальним переглядом змісту таблиці Treeview. Далі ця прокрутка розміщується праворуч від таблиці за допомогою pack(side=RIGHT, fill=Y), що забезпечує її розтягування по вертикалі згідно з висотою таблиці. Останнім кроком є прив'язка таблиці до цієї смуги прокрутки через метод configure, в якому yscrollcommand=scrollbar.set встановлює зворотний зв'язок між таблицею та прокруткою. Це дає можливість Treeview повідомляти

смузі прокрутки про зміни позиції видимої частини таблиці. Завдяки такій інтеграції, при доданні нових записів таблиця автоматично стає прокрутною, і користувач може без особливих зусиль переміщуватись між усіма транзакціями. Важливо зазначити, що реалізація прокрутки відбувається динамічно і не потребує додаткових налаштувань або втручання з боку користувача — все працює автоматично під час взаємодії з графічним інтерфейсом. Таким чином, реалізація прокрутки в таблиці є важливою деталлю, яка не тільки покращує візуальне представлення даних, але й забезпечує повноцінну функціональність при роботі з великими масивами інформації у застосунку для особистої бухгалтерії.

3.14 Кнопки керування

Кнопки управління є невід'ємною складовою інтерфейсу користувача у додатку особистої бухгалтерії. Вони забезпечують виконання основних операцій, потрібних для керування фінансовими транзакціями, зокрема — додавання, редагування, видалення окремих записів і очищення усієї таблиці. Ці елементи інтерфейсу мають зрозуміле текстове позначення і зручно розташовані у вікні додатку, що робить взаємодію з програмою інтуїтивно зрозумілою навіть для недосвідчених юзерів.У програмі використано чотири основні кнопки:

- Додати транзакцію
- Видалити транзакцію
- Видалити всі транзакції
- Редагувати транзакцію

Створення цих кнопок реалізується за допомогою модуля tkinter, зокрема класу Button. Приклади створення та розміщення кнопок:

add_button = Button(left_frame, text="Додати транзакцію", command=add_transaction)

add_button.pack(anchor="w", pady=10, padx=10)

delete_button = Button(button_frame, text="Видалити транзакцію", command=delete_transaction)

delete_button.pack(side=LEFT, padx=10)

delete_all_button = Button(button_frame, text="Видалити всі транзакції", command=delete_all_transactions)

delete_all_button.pack(side=LEFT, padx=10)

edit_button = Button(button_frame, text="Редагувати транзакцію", command=edit_transaction)

edit_button.pack(side=LEFT, padx=10)

Кожна кнопка прив’язана до відповідної функції за допомогою параметра command. Це означає, що під час натискання кнопки виконується певна логіка, яка визначена в окремій функції: add_transaction, delete_transaction, delete_all_transactions, або edit_transaction. Кнопка "Додати транзакцію" ініціює збирання даних із полів введення (дата, тип, сума, коментар) і додає новий запис до бази даних. Кнопка "Видалити транзакцію" дає змогу видалити один обраний запис із таблиці. Якщо запис не вибрано, користувач отримає відповідне сповіщення про помилку. Кнопка "Видалити всі транзакції" виконує повне очищення таблиці після підтвердження дії користувачем через діалогове вікно. Кнопка "Редагувати транзакцію" дозволяє змінити дані обраного запису, оновивши їх як у графічному інтерфейсі, так і в базі даних. Розміщення кнопок у двох різних контейнерах (left_frame і button_frame) дає змогу логічно структурувати інтерфейс: кнопку додавання розміщено поряд із формою введення, тоді як кнопки редагування та видалення згруповано окремо, під таблицею, для зручності управління вже внесеними даними.

Отже, кнопки керування є ключовими елементами інтерактивності програми. Вони забезпечують доступ до основного функціоналу застосунку, спрощують взаємодію з даними та роблять процес роботи з особистими фінансами максимально зручним для кінцевого користувача.

3.15 Завантаження даних з бази

Після запуску застосунку важливо гарантувати відображення вже наявних транзакцій, котрі були збережені у базі даних під час попередніх сесій користувача. Це реалізується шляхом завантаження всіх записів із таблиці transactions, що зберігається у SQLite-базі finance.db, та подальшого відображення цих даних у графічному інтерфейсі у вигляді таблиці (віджету Treeview з бібліотеки ttk). Завантаження втілено через SQL-запит на вибірку всіх записів у базі та їх додавання у таблицю в інтерфейсі користувача. Відповідна частина коду має такий вигляд:

```
cur.execute("SELECT * FROM transactions")
rows = cur.fetchall()


for row in rows:
    treeview.insert("", END, values=row)
```

Перший рядок реалізує запит до бази даних, здобуваючи всі записи з таблиці transactions. Метод fetchall() зчитує результат цього запиту у вигляді списку кортежів, де кожний кортеж відповідає одному рядку таблиці. Далі за допомогою циклу for кожний запис додається до Treeview — елементу інтерфейсу, який демонструє дані у табличному вигляді. Метод insert() вставляє рядок з відповідними значеннями (id, date, type, amount, comment) до таблиці, забезпечуючи зручне й наочне відображення фінансової інформації для користувача. Цей механізм завантаження є надзвичайно важливим для збереження

цілісності даних і зручності використання застосунку. Він дає змогу користувачеві бачити всі попередні записи без потреби повторного внесення інформації після кожного запуску програми. Завдяки цьому забезпечується сталість та актуальність обліку особистих фінансів, що відповідає головній меті застосунку — ефективному та зручному контролю над особистим бюджетом.

3.16 Завершення роботи застосунку

Завершення роботи застосунку є невід'ємним етапом взаємодії користувача з програмою, особливо в контексті збереження цілісності даних та вивільнення системних ресурсів. У даному застосунку особистої бухгалтерії, створеному за допомогою бібліотеки tkinter та мови програмування Python, закінчення роботи відбувається автоматично при закритті головного вікна користувачем. Проте перед повним закінченням програми необхідно впевнитися, що з'єднання з базою даних SQLite буде правильно закрите. Це досягається викликом методу close() об'єкта з'єднання connection після виклику mainloop() — основного циклу подій графічного інтерфейсу. Цей виклик гарантує, що всі ресурси, пов’язані з роботою бази даних, будуть звільнені, а всі зміни, внесені під час роботи застосунку, залишаться збереженими. Завершальний фрагмент коду виглядає наступним чином:

```
root.mainloop()

connection.close()
```

Метод root.mainloop() відповідає за запуск графічного інтерфейсу та обробку всіх дій юзера у вікні застосунку. Після того як користувач зачинить вікно або завершить роботу з додатком, цикл mainloop буде завершено, і програма перейде до наступного рядка коду, де й виконується закриття з’єднання з базою даних через connection.close(). Цей механізм є важливою частиною загальної архітектури застосунку, оскільки неправильне завершення роботи або нехтування закриття з’єднання з базою даних може призвести до втрати даних, блокування таблиць або помилок при наступному запуску програми. Таким чином, правильно організоване

завершення роботи забезпечує надійність функціонування програмного забезпечення та стабільність збереження фінансової інформації юзера.

ВИСНОВОК

В процесі роботи над дипломним проєктом було здійснено дослідження сучасних концепцій автоматизації особистих фінансів, а також створено програмне забезпечення, призначене для комфортного, результативного та захищеного управління особистими фінансовими ресурсами. Здійснений аналіз поточних цифрових інструментів та програмних розробок надав підстави для вибору мови програмування Python, що завдяки своїй адаптивності, легкості синтаксису та багатому набору бібліотек, зокрема Tkinter та SQLite, забезпечила успішне втілення всіх поставлених функціональних потреб.

Створене програмне забезпечення дає користувачеві можливість реєструвати доходи та витрати, формувати категорії фінансових операцій, створювати аналітичні звіти, графічно відображати фінансові дані, а також зберігати дані локально, не вимагаючи постійного з'єднання з Інтернетом. Це збільшує рівень приватності та безпеки персональної інформації. У процесі розробки було імплементовано базову систему аутентифікації з хешуванням паролів, що додає додатковий рівень безпеки.

Особливу увагу було приділено зручності інтерфейсу та можливості адаптації застосунку до потреб різних користувачів – як фізичних осіб, так і власників малого бізнесу. Програмне рішення було розроблено за модульним підходом, що полегшує його підтримку, масштабування та оновлення.

Отримані результати підтверджують практичну значущість проєкту: його можна використовувати як незалежний інструмент для обліку фінансів, а також як приклад для подальшого навчання студентів у сферах програмної інженерії, фінансових технологій та системного аналізу. Отже, поставлені задачі дипломної роботи були цілком виконані, а створене програмне забезпечення відповідає критеріям сучасного програмного продукту в галузі особистої бухгалтерії

ЛІТЕРАТУРА

1. Основи програмування. Python. Частина 1: підручник для студ. спеціальності 122 "Комп’ютерні науки", спеціалізації "Інформаційні технології в біології та медицині". Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.
2. Яковенко А. Основи програмування: методичні вказівки до виконання комп’ютерних практикумів з дисципліни "Основи програмування". Основи програмування мовою Python. Київ : НТУУ "КПІ ім. І. Сікорського", 2017. С. 87.
3. Вілінський М.Ю. Інформаційна безпека: підручник. — Київ: Кондор, 2020.
4. Пронський Є.В. Засоби захисту інформації у комп’ютерних системах. — Львів: ЛНУ, 2021.
5. Журавльов С.В. Основи інформаційних технологій. — Київ: КНЕУ, 2021.
6. Федорова Н.В. Бази даних. Проектування та використання. — Київ: Академія, 2020.
7. Бєляєв О.М. Інформаційні технології в економіці. — Київ: Центр навчальної літератури, 2020.
8. Семакин І.Г. Інформатика. Базовий курс. — Тернопіль: Навчальна книга – Богдан, 2021.
9. Гнатюк С. М., Левківський К. І. Програмування мовою Python: навчальний посібник. — Львів: Львівська політехніка, 2022.
10. Савченко О.Ю. Програмування мовою Python: практичний курс. — Київ: НаУКМА, 2020.
11. Lutz, M. Learning Python. — 5th ed. — O’Reilly Media, 2013.
12. Matthes, E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. — No Starch Press, 2019.
13. Sweigart, A. Automate the Boring Stuff with Python. — 2nd ed. — No Starch Press, 2019

14. Downey, A. Think Python: How to Think Like a Computer Scientist. — 2nd ed. — O’Reilly Media, 2015.
15. Grinberg, M. Flask Web Development: Developing Web Applications with Python. — O’Reilly Media, 2018.
16. Beazley, D., Jones, B. Python Cookbook. — 3rd ed. — O’Reilly Media, 2013.
17. Shaw, Z.A. Learn Python 3 the Hard Way. — Addison-Wesley, 2017.
18. Harris, C.R., et al. Array programming with NumPy. — Nature, 2020. (у книгах — доступний як збірник статей)
19. Ousterhout, J.K. Tcl and the Tk Toolkit. — Addison-Wesley, 2009.
20. Campbell, J. Tkinter GUI Programming by Example. — Packt Publishing, 2018.
21. Owens, M. The Definitive Guide to SQLite. — Apress, 2006.
22. Churcher, C. Beginning Database Design: From Novice to Professional. — 2nd ed. — Apress, 2012

ДОДАТКИ

```
from tkcalendar import DateEntry

from tkinter import *

from tkinter import ttk

from tkinter import messagebox

import sqlite3

connection = sqlite3.connect("finance.db")

cur = connection.cursor()

cur.execute("""

    CREATE TABLE IF NOT EXISTS transactions (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        date TEXT,

        type TEXT,

        amount REAL,

        comment DATE

        )

""")
```

```
connection.commit()

def add_transaction():
    if type_combobox.get() and amount_entry.get():
        try:
            transaction_type = type_combobox.get()
            amount = float(amount_entry.get())
            comment = comment_entry.get()
            date = date_entry.get()
            cur.execute("""
                INSERT INTO transactions (date, type, amount, comment)
                VALUES (?, ?, ?, ?)
                """, (date, transaction_type, amount, comment))
            connection.commit()
            transaction_id = cur.lastrowid
            treeview.insert("", END, values=(transaction_id, date, transaction_type,
amount, comment))
            type_combobox.set("")
            amount_entry.delete(0, END)
```

```
            comment_entry.delete(0, END)
            messagebox.showinfo("Успіх", "Транзакцію успішно додано!")
        except ValueError:
            messagebox.showerror("Помилка", "Суму введена некоректно!")
    else:
        messagebox.showinfo("Попередження", "Заповнено не всі поля!")
def delete_transaction():
    selected_item = treeview.selection()
    if selected_item:
        item_values = treeview.item(selected_item, "values")
        transaction_id = item_values[0]
        cur.execute("DELETE FROM transactions WHERE id=?", (transaction_id,))
        connection.commit()
        treeview.delete(selected_item)
        messagebox.showinfo("Успіх", "Транзакцію успішно видалено!")
    else:
```

```
        messagebox.showinfo("Попередження", "Виберіть транзакцію для
видалення!")
def delete_all_transactions():
    confirm = messagebox.askyesno("Попередження", "Ви впевнені, що бажаєте
видалити всі транзакції?")
    if confirm:
        cur.execute("DELETE FROM transactions")
        connection.commit()
        treeview.delete(*treeview.get_children())
        messagebox.showinfo("Успіх", "Усі транзакції успішно видалено!")
def edit_transaction():
    selected_item = treeview.selection()
    if not selected_item:
        messagebox.showerror("Транзакцію не вибрано!",
                             "Будь ласка, оберіть транзакцію в таблиці, щоб
відредагувати.")
        return
    transaction_id = treeview.set(selected_item, "#1")
```

```
    date = date_entry.get()
    transaction_type = type_combobox.get()
    amount = amount_entry.get()
    comment = comment_entry.get()
    cur.execute("UPDATE transactions SET date = ?, type = ?, amount = ?,
comment = ? WHERE id = ?",
                (date, transaction_type, amount, comment, transaction_id))
    connection.commit()
    treeview.item(selected_item, values=(transaction_id, date, transaction_type,
amount, comment))
def on_row_click(event):
    if treeview.selection():
        item = treeview.selection()[0]
        values = treeview.item(item, "values")
        date_entry.delete(0, END)
        date_entry.insert(0, values[1])
        type_combobox.set(values[2])
        amount_entry.delete(0, END)
```

```
        amount_entry.insert(0, values[3])
        comment_entry.delete(0, END)
        comment_entry.insert(0, values[4])



root = Tk()
root.title("Домашня Бухгалтерія")
root.geometry("700x340")
root.resizable(False, False)
left_frame = Frame(root, bd=2, relief=SUNKEN)
left_frame.pack(side=LEFT, pady=10)
date_label = Label(left_frame, text="Дата:")
date_label.pack(anchor="w", pady=10, padx=10)
date_entry = DateEntry(left_frame, width=12, background="darkblue",
foreground="white", borderwidth=2, date_pattern="dd.mm.y")
date_entry.pack(anchor="w", padx=10)
type_label = Label(left_frame, text="Тип:")
```

```
type_label.pack(anchor="w", pady=10, padx=10)

type_combobox = ttk.Combobox(left_frame, values=["Дохід", "Витрата"],
state="readonly")

type_combobox.pack(anchor="w", padx=10)


amount_label = Label(left_frame, text="Сума:")

amount_label.pack(anchor="w", pady=10, padx=10)

amount_entry = Entry(left_frame)

amount_entry.pack(anchor="w", padx=10)

comment_label = Label(left_frame, text="Коментар:")

comment_label.pack(anchor="w", pady=10, padx=10)

comment_entry = Entry(left_frame)

comment_entry.pack(anchor="w", padx=10)

add_button = Button(left_frame, text="Додати транзакцію",
command=add_transaction)

add_button.pack(anchor="w", pady=10, padx=10)

button_frame = Frame(root, bd=2, relief=SUNKEN)

button_frame.pack(side=TOP, fill=BOTH, padx=10, pady=10)
```

```
delete_button = Button(button_frame, text="Видалити транзакцію",
command=delete_transaction)
delete_button.pack(side=LEFT, padx=10)
delete_all_button = Button(button_frame, text="Видалити всі транзакції",
command=delete_all_transactions)
delete_all_button.pack(side=LEFT, padx=10)

edit_button = Button(button_frame, text="Редагувати транзакцію",
command=edit_transaction)
edit_button.pack(side=LEFT, padx=10)
right_frame = Frame(root)
right_frame.pack(side=LEFT, fill=BOTH)
data_frame = Frame(right_frame, bd=2, relief=SUNKEN)
data_frame.pack(side=LEFT, fill=BOTH, expand=True)
treeview = ttk.Treeview(data_frame)
treeview.pack(side=LEFT, fill=BOTH, expand=True)
scrollbar = ttk.Scrollbar(data_frame, orient="vertical", command=treeview.yview)
scrollbar.pack(side=RIGHT, fill=Y)
```

```
treeview.configure(yscrollcommand=scrollbar.set)

treeview["columns"] = ("id", "date", "type", "amount", "comment")

treeview.column("#0", width=0, stretch=NO)

treeview.column("id", anchor=W, width=0)

treeview.column("date", anchor=W, width=100)

treeview.column("type", anchor=W, width=100)

treeview.column("amount", anchor=W, width=100)

treeview.column("comment", anchor=W, width=200)

treeview.heading("#0", text="")

treeview.heading("id", text="ID")

treeview.heading("date", text="Дата")

treeview.heading("type", text="Тип")

treeview.heading("amount", text="Сума, $")

treeview.heading("comment", text="Коментар")

cur.execute("SELECT * FROM transactions")

rows = cur.fetchall()

for row in rows:
```

```
    treeview.insert("", END, values=row)

treeview.bind("<ButtonRelease-1>", on_row_click)

root.mainloop()

connection.close()
```

Державна атестаційна робота кваліфікаційного рівня бакалавр на тему “Розробка застосунку на мові Python”

Виконав: студент 4 курсу, гр. КНД-42, Оліфіренко О.В.

Керівник: к.т.н., доцент Василенко В. В.

- Особиста бухгалтерія важлива в малому бізнесі, бо дозволяє чітко розділити особисті та бізнес-фінанси, контролювати витрати, планувати бюджет, слідкувати за прибутками і уникати касових розривів. Вона допомагає приймати обґрунтовані фінансові рішення, зменшує ризики боргів і сприяє стабільному розвитку бізнесу.

Перший застосунок особистої бухгалтерії — це програмне забезпечення, створене для допомоги користувачам у веденні власних фінансів: доходів, витрат, заощаджень і бюджету. Хоча важко назвати точну дату створення самого першого такого застосунку, вважається, що піонером у цій сфері була програма **Quicken**, розроблена компанією **Intuit** у 1983 році.

Логотип першого застосунку особистої бугалтерії

Декілька причин створення застосунку для ведення бухгалтерії:

1. Фінансова стабільність і контроль — ведення особистої бухгалтерії дозволяє чітко контролювати доходи та витрати, що важливо як для особистих фінансів, так і для малого бізнесу. Це дозволяє уникати непередбачених витрат і забезпечувати стабільний грошовий потік.

2. Безпека та захист інформації — належне ведення обліку дозволяє зберігати важливу фінансову інформацію в безпеці, захищаючи її від шахрайства або втрати. Це критично важливо як для приватних осіб, так і для підприємців, які обробляють конфіденційні дані про бізнес.

3. Планування і досягнення цілей — створення бухгалтерії дозволяє планувати бюджет, ставити фінансові цілі та ефективно їх досягати, оптимізуючи витрати і знаходячи можливості для розвитку.

4. Юридична та податкова безпека — ведення точного обліку забезпечує відповідність податковим вимогам, що важливо для уникнення штрафів і проблем з податковими органами, особливо для малого бізнесу.

Персональний застосунок бухгалтерії в компаніях має важливу роль у забезпеченні ефективного управління фінансами та оптимізації внутрішніх процесів. Він автоматизує облік доходів, витрат і податкових платежів, зменшуючи кількість ручної роботи і знижуючи ймовірність помилок. Застосунок забезпечує прозорість і контроль, надаючи доступ до фінансових даних у реальному часі, що допомагає керівникам і працівникам компанії ефективно управляти бюджетом і стежити за витратами.

Крім того, використання персональних бухгалтерських застосунків дозволяє зберігати фінансові дані в безпеці, забезпечуючи доступ тільки для авторизованих осіб і запобігаючи несанкціонованому доступу до конфіденційної інформації. Персональний застосунок бухгалтерії стає важливим інструментом для підвищення ефективності та безпеки фінансових процесів у компанії.

Використання мови програмування Python для створення застосунку персональної бухгалтерії

Використання мови програмування Python для створення застосунку персональної бухгалтерії є доцільним і практичним вибором завдяки її простоті, гнучкості та широким можливостям. Python має зрозумий синтаксис, що робить його доступним навіть для початківців, а також підтримує велику кількість бібліотек і фреймворків, які спрощують розробку застосунків будь-якого рівня складності.

Для створення графічного інтерфейсу користувача можна використовувати такі інструменти, як Tkinter, PyQt або Kivy. Tkinter, наприклад, є вбудованою бібліотекою і дозволяє створити простий і функціональний інтерфейс без додаткових залежностей. Для збереження фінансових даних у зручному вигляді можна використовувати SQLite — легку вбудовану базу даних, яка добре інтегрується з Python через стандартну бібліотеку sqlite3.

Таким чином, Python є чудовим вибором для створення персонального застосунку бухгалтерії, оскільки поєднує простоту розробки з потужними технічними можливостями, що дозволяє створити ефективний, зручний і безпечний інструмент для управління фінансами.

Цілі створення персональної бухгалтерії

Контроль доходів і витрат
Створення персональної бухгалтерії дозволяє точно фіксувати всі фінансові операції — як надходження, так і витрати. Це допомагає розуміти, куди саме йдуть гроші, на чому можна зекономити, і які джерела доходів є найефективнішими. Для бізнесу це також означає можливість контролювати фінансову дисципліну працівників або партнерів.

Безпека і конфіденційність фінансових даних
Ведення бухгалтерії в захищеному застосунку дає змогу зберігати дані приватно, без доступу сторонніх осіб, на відміну від паперових записів чи сторонніх сервісів. Це знижує ризик крадіжки інформації, втрати документів або шахрайства як у бізнесі, так і в особистих справах.

Планування бюджету і накопичень
Завдяки регулярному обліку можна формувати реалістичний бюджет, розподіляти кошти за пріоритетами, планувати великі витрати або інвестиції. Для бізнесу це означає можливість прогнозувати прибутки й планувати розвиток, а для особистих потреб — підготовку до важливих фінансових подій, як-от відпустка, навчання чи купівля майна.

Юридична та податкова відповідність
Для підприємців персональна бухгалтерія дає змогу своєчасно підготуватися до податкових перевірок, правильно заповнювати декларації та уникати штрафів. Це особливо важливо для ФОПів або малого бізнесу, де часто бухгалтерією займається власник або обмежене коло осіб.

Бібліотеки

Tkinter — це **стандартна бібліотека Python** для створення **графічного інтерфейсу користувача (GUI)**. Вона дозволяє створювати вікна, кнопки, текстові поля, таблиці, календарі та інші елементи, з якими користувач може взаємодіяти.

SQLite — це **легка реляційна база даних**, яка зберігає всі дані у **локальному файлі** (.db) без потреби запускати окремий сервер. Вона швидка, проста й ідеально підходить для невеликих програм, особливо **настільних застосунків**, мобільних аплікацій або **персональної бухгалтерії**, як у твоєму проєкті.

tkcalendar — це зовнішня бібліотека Python, яка додає **календар** до графічного інтерфейсу, створеного з використанням Tkinter.

- Мати особисту бухгалтерію безпечніше, ніж користуватись сторонніми сервісами, насамперед через контроль над конфіденційністю. Дані зберігаються локально або у власній базі, а не передаються на чужі сервери, що значно знижує ризик витоку фінансової інформації чи зловживань. Сторонні платформи можуть бути зламані або передавати інформацію третім особам — у власному застосунку це виключено, особливо якщо він не має підключення до інтернету. Крім того, можна самостійно реалізувати додаткові рівні безпеки, як-от шифрування чи автентифікацію, і бути впевненим, що дані ніхто не змінить без дозволу. Таким чином, особиста бухгалтерія забезпечує не лише повну автономію, а й захист від зовнішніх загроз.

```python
from tkcalendar import DateEntry
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
import sqlite3

connection = sqlite3.connect("finance.db")
cur = connection.cursor()

cur.execute("""
    CREATE TABLE IF NOT EXISTS transactions (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT,
        type TEXT,
        amount REAL,
        comment DATE
        )
""")

connection.commit()


def add_transaction():  1 usage
    if type_combobox.get() and amount_entry.get():
        try:
            transaction_type = type_combobox.get()
            amount = float(amount_entry.get())
            comment = comment_entry.get()
            date = date_entry.get()

            cur.execute( sql: """
                INSERT INTO transactions (date, type, amount, comment)
                VALUES (?, ?, ?, ?)
                """, parameters: (date, transaction_type, amount, comment))
            connection.commit()

            transaction_id = cur.lastrowid

            treeview.insert( parent: "", END, values=(transaction_id, date, transaction_type, amount, comment))

            type_combobox.set("")
            amount_entry.delete( first: 0, END)
            comment_entry.delete( first: 0, END)
            messagebox.showinfo( title: "Успіх", message: "Транзакція успішно додано!")
        except ValueError:
            messagebox.showerror( title: "Помилка", message: "Суму введена некоректно!")
    else:
        messagebox.showinfo( title: "Попередження", message: "Заповнено не всі поля!")


def delete_transaction():  1 usage
    selected_item = treeview.selection()
    if selected_item:
        item_values = treeview.item(selected_item, option: "values")
        transaction_id = item_values[0]
        cur.execute( sql: "DELETE FROM transactions WHERE id=?", parameters: (transaction_id,))
        connection.commit()
        treeview.delete(selected_item)
        messagebox.showinfo( title: "Успіх", message: "Транзакція успішно видалено!")
    else:
        messagebox.showinfo( title: "Попередження", message: "Виберіть транзакцію для видалення!")
```

```python
def delete_all_transactions():  1 usage
    confirm = messagebox.askyesno( title: "Попередження", message: "Ви впевнені, що бажаєте видалити всі транзакції?")
    if confirm:
        cur.execute("DELETE FROM transactions")
        connection.commit()
        treeview.delete(*treeview.get_children())
        messagebox.showinfo( title: "Успіх", message: "Усі транзакції успішно видалено!")


def edit_transaction():  1 usage
    selected_item = treeview.selection()
    if not selected_item:
        messagebox.showerror( title: "Транзакція не вибрано!",
                              message: "Будь ласка, оберіть транзакцію в таблиці, щоб відредагувати.")
        return

    transaction_id = treeview.set(selected_item, column: "#1")
    date = date_entry.get()
    transaction_type = type_combobox.get()
    amount = amount_entry.get()
    comment = comment_entry.get()

    cur.execute( sql: "UPDATE transactions SET date = ?, type = ?, amount = ?, comment = ? WHERE id = ?",
                 parameters: (date, transaction_type, amount, comment, transaction_id))

    connection.commit()

    treeview.item(selected_item, values=(transaction_id, date, transaction_type, amount, comment))


def on_row_click(event):  1 usage
    if treeview.selection():
        item = treeview.selection()[0]
        values = treeview.item(item, option: "values")

        date_entry.delete(0, END)
        date_entry.insert(0, values[1])
        type_combobox.set(values[2])
        amount_entry.delete( first: 0, END)
        amount_entry.insert( index: 0, values[3])
        comment_entry.delete( first: 0, END)
        comment_entry.insert( index: 0, values[4])


root = Tk()
root.title("Домашня Бухгалтерія")
root.geometry("700x340")
root.resizable( width: False,  height: False)

left_frame = Frame(root, bd=2, relief=SUNKEN)
left_frame.pack(side=LEFT, pady=10)

date_label = Label(left_frame, text="Дата:")
date_label.pack(anchor="w", pady=10, padx=10)
date_entry = DateEntry(left_frame, width=12, background="darkblue", foreground="white", borderwidth=2, date_pattern="dd.mm.y")
date_entry.pack(anchor="w", padx=10)

type_label = Label(left_frame, text="Тип:")
type_label.pack(anchor="w", pady=10, padx=10)
type_combobox = ttk.Combobox(left_frame, values=["Дохід", "Витрата"], state="readonly")
type_combobox.pack(anchor="w", padx=10)

amount_label = Label(left_frame, text="Сума:")
```

```python
amount_label.pack(anchor="w", pady=10, padx=10)
amount_entry = Entry(left_frame)
amount_entry.pack(anchor="w", padx=10)

comment_label = Label(left_frame, text="Коментар:")
comment_label.pack(anchor="w", pady=10, padx=10)
comment_entry = Entry(left_frame)
comment_entry.pack(anchor="w", padx=10)

add_button = Button(left_frame, text="Додати транзакцію", command=add_transaction)
add_button.pack(anchor="w", pady=10, padx=10)

button_frame = Frame(root, bd=2, relief=SUNKEN)
button_frame.pack(side=TOP, fill=BOTH, padx=10, pady=10)

delete_button = Button(button_frame, text="Видалити транзакцію", command=delete_transaction)
delete_button.pack(side=LEFT, padx=10)

delete_all_button = Button(button_frame, text="Видалити всі транзакції", command=delete_all_transactions)
delete_all_button.pack(side=LEFT, padx=10)

edit_button = Button(button_frame, text="Редагувати транзакцію", command=edit_transaction)
edit_button.pack(side=LEFT, padx=10)

right_frame = Frame(root)
right_frame.pack(side=LEFT, fill=BOTH)

data_frame = Frame(right_frame, bd=2, relief=SUNKEN)
data_frame.pack(side=LEFT, fill=BOTH, expand=True)

treeview = ttk.Treeview(data_frame)
treeview.pack(side=LEFT, fill=BOTH, expand=True)

scrollbar = ttk.Scrollbar(data_frame, orient="vertical", command=treeview.yview)
scrollbar.pack(side=RIGHT, fill=Y)
treeview.configure(yscrollcommand=scrollbar.set)

treeview["columns"] = ("id", "date", "type", "amount", "comment")
treeview.column("#0", width=0, stretch=NO)
treeview.column("id", anchor=W, width=0)
treeview.column("date", anchor=W, width=100)
treeview.column("type", anchor=W, width=100)
treeview.column("amount", anchor=W, width=100)
treeview.column("comment", anchor=W, width=200)

treeview.heading("#0", text="")
treeview.heading("id", text="ID")
treeview.heading("date", text="Дата")
treeview.heading("type", text="Тип")
treeview.heading("amount", text="Сума, $")
treeview.heading("comment", text="Коментар")

cur.execute("SELECT * FROM transactions")
rows = cur.fetchall()

for row in rows:
    treeview.insert( parent: "", END, values=row)

treeview.bind("<ButtonRelease-1>", on_row_click)

root.mainloop()

connection.close()
```

- Розробка функціональності застосунку особистої бухгалтерії передбачає створення інструментів, які дають змогу користувачеві ефективно вести облік доходів і витрат у зручному інтерфейсі. Основна логіка побудована на взаємодії графічного інтерфейсу з базою даних. Користувач може додавати фінансові записи, вказуючи дату, тип транзакції (дохід або витрата), суму та коментар. Усі ці дані зберігаються в локальній базі SQLite, що дозволяє надійно фіксувати історію операцій. Окрім додавання нових записів, реалізовано функції редагування та видалення окремих або всіх транзакцій. Це забезпечує гнучкість у роботі з інформацією та зручність у виправленні помилок.

У цій дипломній роботі була розглянута тема “Застосунок на мові Python”

У процесі дослідження був проведений аналіз існуючих застосунків особистої бухгалтерії, виявлені їхні переваги та недоліки

Розробка застосунку особистої бухгалтерії на основі мови Python є актуальною та доцільною задачею

Мова програмування Python була обрана як основний інструмент для розробки месенджера. Цей вибір зумовлений великою популярністю Python, його простотою та гнучкістю, а також наявністю багатофункціональних фреймворків та бібліотек для веб-розробки.