МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП’ЮТЕРНІ НАУКИ»

ЦИКЛОВА КОМІСІЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП'ЮТЕРНІ НАУКИ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи освітнього ступеня «фаховий молодший бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
на тему:

«Фінансовий калькулятор для ведення особистого бюджету»

Виконав здобувач освіти групи П-42
Демчук Олексій Юрійович

Керівник роботи:
Устименко Ярослав Іванович

Рецензент:
Устименко Людмила Миколаївна

Житомир – 2025

Зміст

Вступ _____ 6

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ _____ 8

1.1. Постановка основної задачі розробки _____ 8

1.2. Огляд та порівняння аналогічних систем. _____ 10

1.3. Вибір та обґрунтування технологій розробки _____ 16

Висновки до розділу 1 _____ 20

РОЗДІЛ 2. ПРОЕКТУВАННЯ І РОЗРОБКА _____ 22

2.1. Вибір алгоритмів та їх ефективність _____ 22

2.2. Спроєктовані класи програми _____ 25

2.3. Програмна реалізація _____ 29

Висновки до розділу 2 _____ 35

РОЗДІЛ 3. КЕРІВНИЦТВО КОРИСТУВАННЯ ПРОГРАМОЮ _____ 37

3.1. Мінімальні вимоги до системи _____ 37

3.2. Інтерфейс користувача _____ 39

Висновки до розділу 3 _____ 44

ВИСНОВКИ _____ 46

Перелік літературних джерел _____ 48

Додаток А. _____ 51

РЕФЕРАТ

Дипломна робота присвячена розробці програмного додатку "Фінансовий калькулятор для ведення особистого бюджету" з використанням технології Windows Forms. Обсяг роботи становить 60 сторінок, включає 18 рисунків, 1 таблицю, 1 додаток і 20 джерел літератури.

Метою роботи є створення зручного та функціонального настільного додатку для автоматизації обліку особистих фінансів. Основні завдання включають аналіз вимог, порівняння аналогів, вибір технологій, проєктування алгоритмів і класів, програмну реалізацію та підготовку керівництва користувача. Об’єктом дослідження є процес розробки додатку для ведення бюджету, а предметом — технологія Windows Forms як засіб його створення.

Розроблений додаток є безкоштовним, простим у використанні інструментом для ведення бюджету, що вирізняється автономністю та інтуїтивним інтерфейсом. Програма дозволяє користувачам фіксувати доходи й витрати, аналізувати фінансові дані та переглядати їх у графічному вигляді. Робота має теоретичне значення, поглиблюючи знання про створення настільних додатків, і практичне — як готовий продукт для управління фінансами.

Ключові слова: фінансовий калькулятор, особистий бюджет, Windows Forms, C#, JSON, графічна візуалізація, управління фінансами.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — Application Programming Interface (інтерфейс програмування додатків) — набір правил і засобів для взаємодії між програмними компонентами.

C# — мова програмування, розроблена компанією Microsoft, основа для реалізації логіки додатку.

DTO — Data Transfer Object (об’єкт передачі даних) — шаблон проєктування для передачі даних між компонентами програми.

GUI — Graphical User Interface (графічний інтерфейс користувача) — система візуальних елементів для взаємодії користувача з програмою.

IDE — Integrated Development Environment (інтегроване середовище розробки) — програмне забезпечення для створення додатків (у роботі — Visual Studio).

JSON — JavaScript Object Notation (нотація об’єктів JavaScript) — текстовий формат для зберігання та обміну даними, використаний для збереження транзакцій.

LINQ — Language Integrated Query (вбудована мова запитів) — технологія в C# для роботи з даними через запити, застосована для фільтрації та агрегації.

RAM — Random Access Memory (оперативна пам’ять) — апаратний компонент комп’ютера для тимчасового зберігання даних.

UI — User Interface (інтерфейс користувача) — сукупність засобів для взаємодії користувача з програмою (синонім GUI).

UWP — Universal Windows Platform (універсальна платформа Windows) — альтернативна технологія для створення додатків Windows, згадана як альтернатива Windows Forms.

WPF — Windows Presentation Foundation (платформа презентацій Windows) — сучасна технологія Microsoft для створення графічних інтерфейсів, розглянута як альтернатива.

XML — Extensible Markup Language (розширювана мова розмітки) — формат даних, що розглядалася як альтернатива JSON.

ВСТУП

Сучасний ритм життя вимагає від людини ефективного управління власними фінансами, що є ключовим фактором для забезпечення фінансової стабільності та досягнення особистих цілей. Ведення особистого бюджету дозволяє контролювати доходи та витрати, планувати заощадження та уникати непередбачуваних фінансових труднощів. У той же час, ручне ведення бюджету може бути трудомістким і схильним до помилок, що зумовлює потребу в автоматизованих інструментах. Розробка програмного забезпечення для фінансового планування є актуальним завданням, яке поєднує в собі як практичну користь для користувачів, так і можливість реалізації сучасних технологій програмування.

Технологія Windows Forms, що є частиною платформи .NET Framework, залишається популярним вибором для створення настільних додатків завдяки своїй простоті, зрозумілому інтерфейсу розробки та широким можливостям для створення графічних інтерфейсів користувача. Ця технологія дозволяє швидко створювати функціональні додатки з інтуїтивно зрозумілим дизайном, що робить її оптимальною для реалізації фінансового калькулятора, призначеного для ведення особистого бюджету.

Метою даної дипломної роботи є розробка програмного додатку "Фінансовий калькулятор для ведення особистого бюджету" на базі технології Windows Forms, який забезпечить користувачам зручний інструмент для запису, аналізу та візуалізації фінансових транзакцій. Програма має надавати можливість додавання доходів і витрат, редагування та видалення записів, фільтрації даних за різними критеріями, а також відображення фінансової статистики у вигляді графіків для кращого розуміння структури витрат і доходів.

Актуальність роботи зумовлена зростаючим інтересом до фінансової грамотності серед населення, а також необхідністю створення доступних і простих у використанні інструментів для управління особистими фінансами. У процесі виконання роботи будуть розглянуті основи проєктування

інтерфейсу користувача, методи обробки та зберігання даних, а також принципи інтеграції графічних компонентів для візуалізації фінансової інформації.

Об’єктом дослідження є процес розробки настільного додатку для ведення особистого бюджету, а предметом — технологія Windows Forms як засіб реалізації такого додатку. У ході роботи буде проаналізовано наявні аналоги, визначено їхні переваги та недоліки, а також розроблено власне рішення, яке враховує потреби сучасного користувача.

Дана дипломна робота має як теоретичне значення, що полягає у вивченні принципів створення додатків на базі Windows Forms, так і практичне — у вигляді готового програмного продукту, який може бути використаний для підвищення ефективності управління особистими фінансами.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ

1.1. Постановка основної задачі розробки

Розробка програмного забезпечення для ведення особистого бюджету є важливим завданням, яке спрямоване на створення зручного та функціонального інструменту для управління фінансами. Основна задача даної дипломної роботи полягає у проєктуванні, розробці та реалізації настільного додатку "Фінансовий калькулятор для ведення особистого бюджету" на базі технології Windows Forms, який забезпечить користувачам можливість ефективного контролю доходів і витрат, аналізу фінансового стану та планування майбутніх витрат.

Метою розробки є створення програмного продукту, який дозволить користувачам:

- фіксувати фінансові транзакції (доходи та витрати) із зазначенням опису, суми, дати та типу операції;
- редагувати та видаляти існуючі записи;
- застосовувати фільтри для аналізу транзакцій за певними критеріями (наприклад, за датою чи описом);
- відображати підсумкову фінансову статистику у вигляді балансу, загальних доходів і витрат;
- візуалізувати структуру бюджету через графічні засоби для кращого сприйняття інформації.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

1. **Аналіз вимог**: Визначити функціональні та нефункціональні вимоги до додатку, враховуючи потреби цільової аудиторії — користувачів, які прагнуть вести облік особистих фінансів.
2. **Проєктування інтерфейсу**: Розробити графічний інтерфейс користувача (GUI), який буде інтуїтивно зрозумілим, зручним і відповідатиме принципам ергономіки.

3. **Реалізація функціоналу**:
   - Створити структуру даних для зберігання інформації про транзакції (опис, сума, дата, тип).
   - Забезпечити механізми додавання, редагування та видалення транзакцій.
   - Розробити систему фільтрації даних за датою та описом.
   - Реалізувати підрахунок фінансових показників (доходів, витрат, балансу).
   - Інтегрувати графічний компонент для візуалізації структури бюджету.
4. **Збереження даних**: Розробити механізм збереження та завантаження даних у файл (наприклад, у форматі JSON) для забезпечення збереження інформації між сеансами роботи з додатком.
5. **Тестування**: Перевірити працездатність усіх функцій, коректність обчислень і стабільність роботи програми в різних сценаріях використання.
6. **Документація**: Підготувати технічну документацію та інструкцію користувача для спрощення подальшого використання та підтримки додатку.

Програмний продукт має відповідати таким основним **вимогам**:

- **Функціональність**: Забезпечення повного набору функцій для ведення бюджету, включаючи введення, обробку та аналіз даних.
- **Зручність**: Інтерфейс має бути простим і зрозумілим навіть для користувачів із мінімальним досвідом роботи з комп’ютером.
- **Надійність**: Додаток повинен коректно обробляти введені дані, уникати втрати інформації та забезпечувати стабільну роботу.
- **Переносимість**: Програма має працювати на операційних системах Windows, які підтримують .NET Framework.
- **Ефективність**: Швидка обробка даних і відображення результатів навіть при значній кількості транзакцій.

Очікувані результати

Результатом розробки стане готовий настільний додаток, побудований на технології Windows Forms, який поєднує в собі зручний інтерфейс, базовий набір функцій для ведення бюджету та графічну візуалізацію фінансових даних. Додаток буде протестовано на відповідність заявленим вимогам і забезпечить користувачам ефективний інструмент для управління особистими фінансами.

Постановка задачі розробки визначає чіткий напрямок роботи, який включає аналіз, проєктування, реалізацію та тестування програмного продукту, спрямованого на вирішення актуальної проблеми автоматизації ведення особистого бюджету.

1.2. Огляд та порівняння аналогічних систем.

Розробка програмного забезпечення для ведення особистого бюджету є популярним напрямком у сфері фінансових технологій, оскільки попит на інструменти управління фінансами стабільно зростає. На ринку існує низка аналогічних систем, які пропонують користувачам різноманітні функції для обліку доходів і витрат, аналізу фінансового стану та планування бюджету. У цьому розділі розглянуто основні аналоги додатку "Фінансовий калькулятор для ведення особистого бюджету", проаналізовано їхні можливості, переваги та недоліки, а також проведено порівняння з розробленим програмним продуктом.

1. Quicken — це один із найвідоміших настільних додатків для управління особистими фінансами, який розроблений компанією Intuit. Програма підтримує синхронізацію з банківськими рахунками, облік транзакцій, створення звітів та планування бюджету.

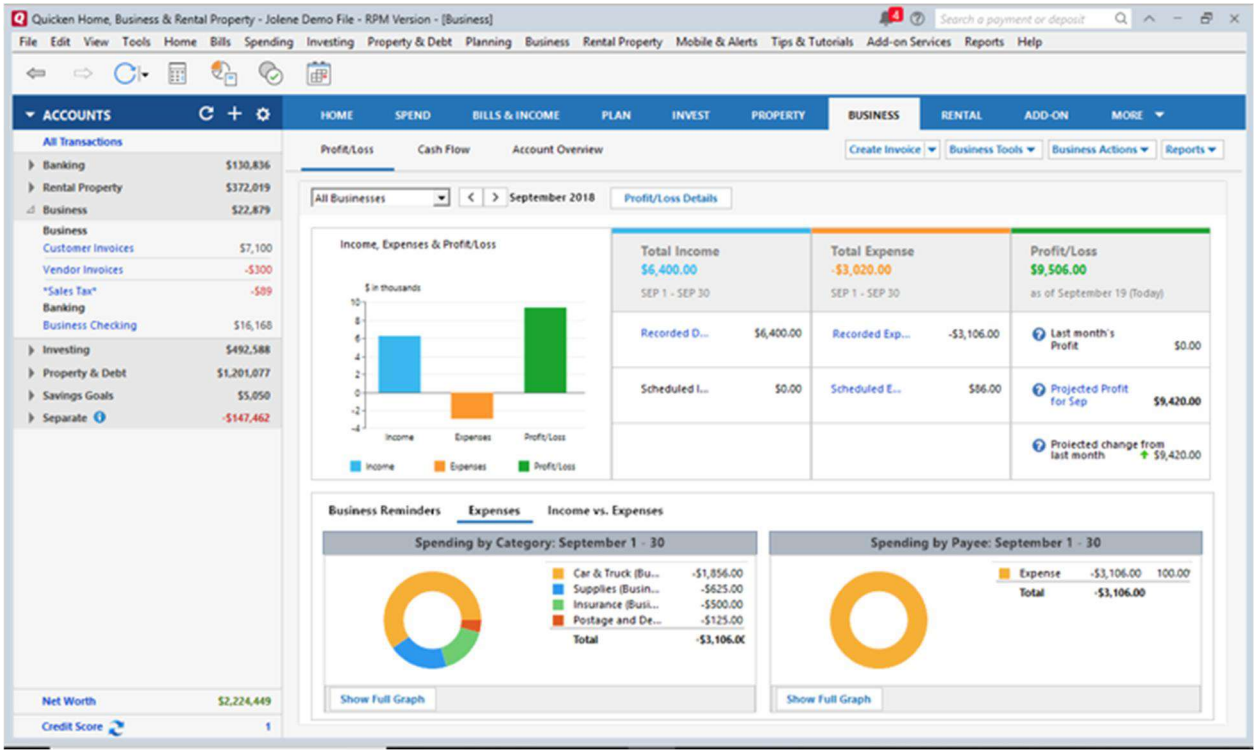

Рисунок 1.2.1 – Інтерфейс Quicken

Переваги:

- Широкий набір функцій, включаючи управління інвестиціями та податкове планування.
- Можливість автоматичної категоризації витрат.
- Підтримка понад 14 000 фінансових установ для імпорту даних.
- **Недоліки**:
- Платна підписка (від $35.99 на рік залежно від версії).
- Складний інтерфейс для початківців.
- Обмежена мобільність, оскільки основний акцент зроблено на настільній версії.

2. YNAB (You Need A Budget) — це програмне забезпечення для бюджетування, яке базується на методі "нульового бюджету", де кожен отриманий дохід розподіляється на конкретні категорії витрат. Доступне як настільна, так і веб-версія з мобільним додатком.

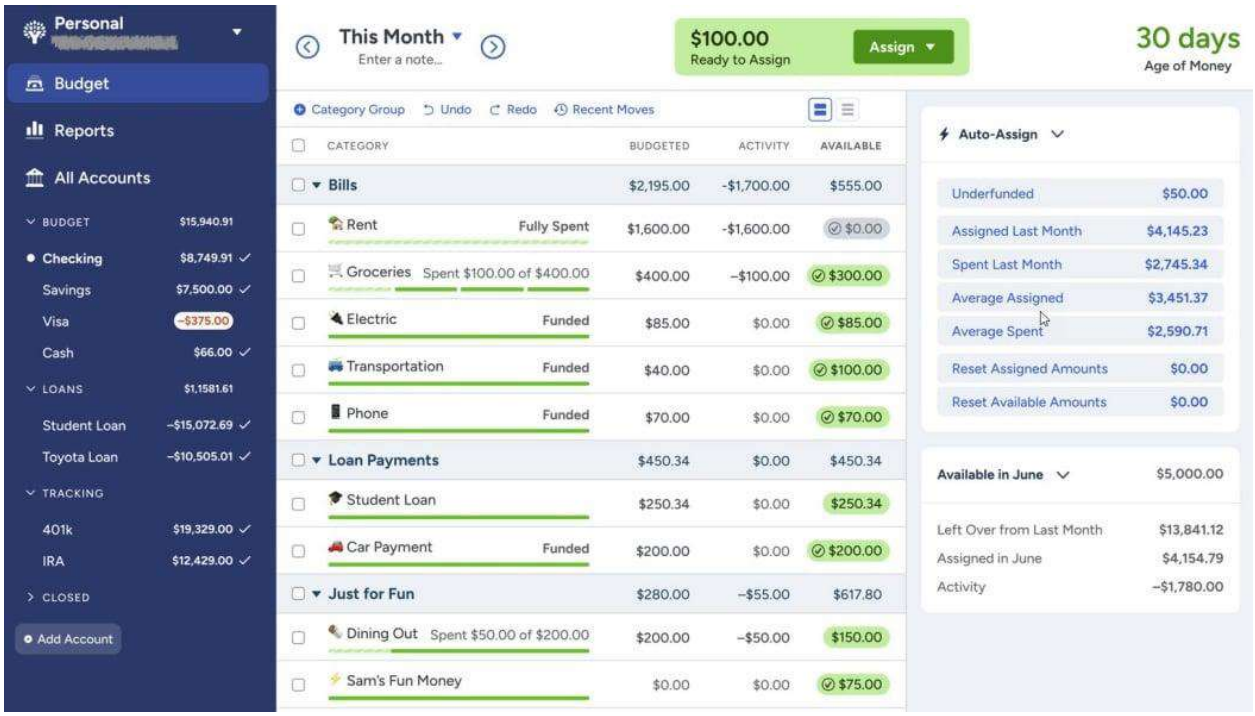

Рисунок 1.2.2 – Інтерфейс YNAB

Переваги:

- Інтуїтивний підхід до планування бюджету.
- Синхронізація з банківськими рахунками в реальному часі.
- Наявність навчальних матеріалів для підвищення фінансової грамотності.
- **Недоліки**:
- Висока вартість підписки ($99 на рік або $14.99 на місяць).
- Орієнтація на американський ринок, що може ускладнювати адаптацію до інших валют чи фінансових систем.
- Відсутність безкоштовної версії.

3. Microsoft Excel (з шаблонами бюджету) - не є спеціалізованим фінансовим додатком, але широко використовується для ведення бюджету завдяки готовим шаблонам і гнучкості налаштувань. Користувачі можуть створювати власні таблиці або завантажувати безкоштовні шаблони з офіційного сайту Microsoft.

Рисунок 1.2.3 – Інтерфейс Microsoft Excel

Переваги:

- Безкоштовність (за наявності ліцензії Office).
- Повна гнучкість у налаштуванні таблиць і формул.
- Можливість роботи офлайн.

Недоліки:

- Відсутність автоматизації (наприклад, синхронізації з банками).
- Вимагає часу та навичок для створення функціонального бюджету.
- Немає вбудованої графічної візуалізації без додаткових зусиль.

4. PocketSmith — це веб-додаток для управління фінансами з акцентом на прогнозування грошових потоків. Він підтримує підключення до банківських рахунків і дозволяє створювати бюджети на основі календарного підходу.

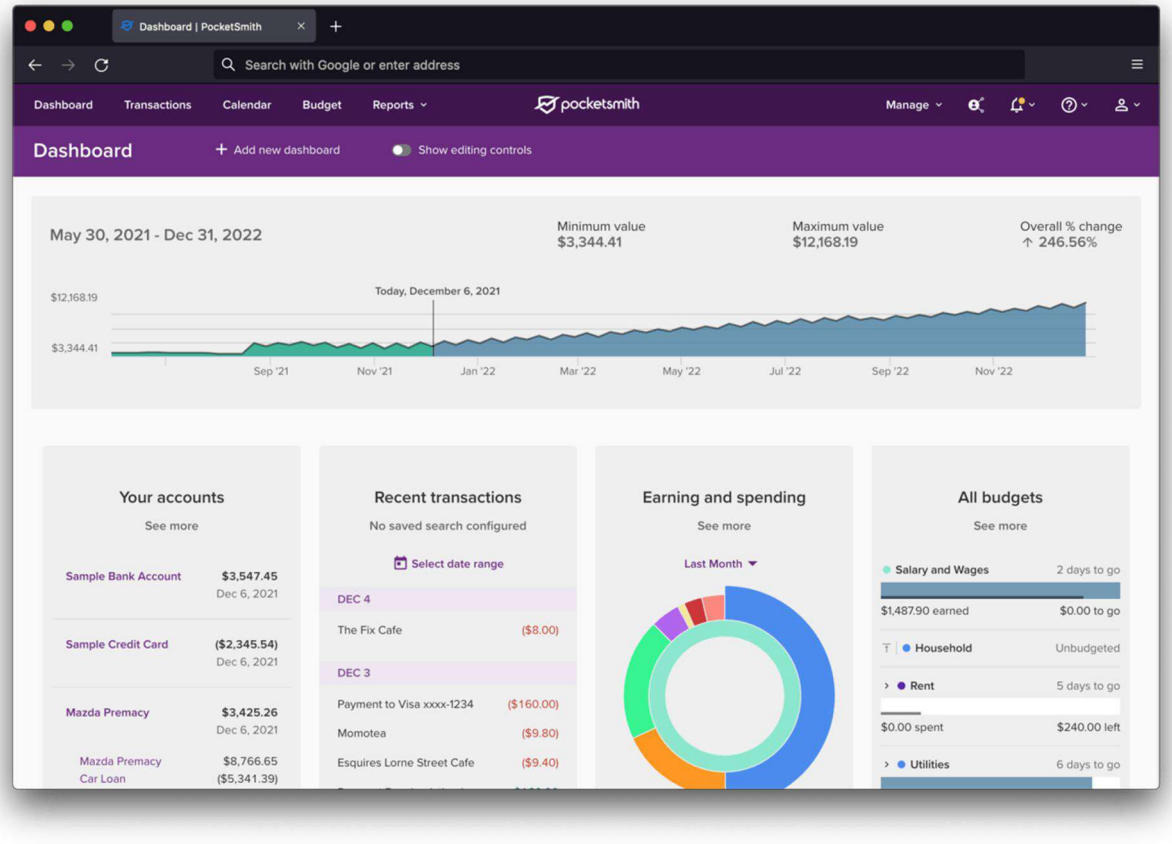

Рисунок 1.2.4 – Інтерфейс PocketSmith

Переваги:

- Прогнозування балансу на 30 років уперед.
- Підтримка кількох валют.
- Зручний інтерфейс із гнучкими налаштуваннями.
- **Недоліки**:
- Обмежений функціонал у безкоштовній версії.
- Платна підписка для повного доступу (від $9.95 на місяць).
- Веб-орієнтованість, що вимагає постійного підключення до Інтернету.

5. GnuCash — це безкоштовне програмне забезпечення з відкритим вихідним кодом, яке підходить як для особистих, так і для малого бізнесу фінансів. Воно доступне для Windows, macOS і Linux.

Рисунок 1.2.5 – Інтерфейс GnuCash

Переваги:

- Безкоштовність і відкритий код, що дозволяє модифікацію.
- Підтримка імпорту даних у форматах QIF і OFX.
- Детальні звіти про фінансові операції.

Недоліки:

- Складний інтерфейс, орієнтований на користувачів із базовими знаннями бухгалтерії.
- Відсутність автоматичної синхронізації з банками.
- Обмежена графічна візуалізація.

У таблиці 1.2.1 наведено порівняння основних характеристик.

Таблиця 1.2.1

Характеристика	Quicken	YNAB	Excel	PocketSmith	GnuCash
Технологія	Настільна	Веб/Моб	Таблиці	Веб	Настільна
Вартість	Платно	Платно	Умовно безкоштовно	Платно/Безкоштовно	Безкоштовно
Синхронізація з банками	Так	Так	Ні	Так	Ні
Графічна візуалізація	Так	Так	Ні (вимагає налаштування)	Так	Обмежено
Фільтрація даних	Так	Так	Ні	Так	Так
Збереження даних	Локально	Хмара	Локально	Хмара	Локально
Простота інтерфейсу	Середня	Висока	Низька	Висока	Низька

Огляд аналогічних систем показав, що ринок пропонує різноманітні рішення для управління особистими фінансами — від складних комерційних продуктів із розширеним функціоналом до безкоштовних інструментів із базовими можливостями.

1.3. Вибір та обґрунтування технологій розробки

Розробка програмного додатку "Фінансовий калькулятор для ведення особистого бюджету" потребує ретельного вибору технологій, які забезпечать ефективну реалізацію поставлених завдань, відповідність вимогам до функціональності, зручності та надійності, а також оптимальне використання ресурсів розробки. У цьому розділі розглянуто основні технології, обрані для створення додатку, та обґрунтовано їх вибір із урахуванням специфіки проєкту.

1. Мова програмування: C# — це сучасна об’єктно-орієнтована мова програмування, розроблена компанією Microsoft, яка є частиною платформи .NET. Вона широко використовується для створення настільних, веб- і мобільних додатків.

Обґрунтування вибору:

- **Продуктивність і зручність**: C# пропонує потужні засоби для роботи з об’єктами, колекціями та обробки даних, що спрощує реалізацію логіки

фінансового калькулятора, такої як обчислення балансу чи фільтрація транзакцій.

- **Інтеграція з Windows Forms**: Як основна мова для платформи .NET, C# ідеально підходить для роботи з Windows Forms, забезпечуючи безшовну взаємодію із графічними компонентами.
- **Широка підтримка**: Велика кількість бібліотек, документації та спільнота розробників дозволяють швидко вирішувати технічні питання, що виникають у процесі розробки.
- **Типізація та безпека**: Статична типізація та вбудована обробка винятків сприяють створенню надійного коду, що є важливим для коректної обробки фінансових даних.

2. Технологія створення інтерфейсу: Windows Forms — це бібліотека в складі .NET, призначена для створення графічних інтерфейсів настільних додатків для операційної системи Windows. Вона включає набір готових елементів управління (кнопки, текстові поля, списки тощо) та інструмент дизайнера в Visual Studio.

Обґрунтування вибору:

- **Простота розробки**: Windows Forms дозволяє швидко створювати інтерфейси за допомогою drag-and-drop у Visual Studio, що значно прискорює процес проєктування UI для фінансового калькулятора.
- **Настільна орієнтованість**: Оскільки додаток призначений для локального використання без залежності від Інтернету, Windows Forms є оптимальним вибором порівняно з веб-технологіями чи хмарними рішеннями.
- **Стабільність**: Технологія перевірена часом і підтримує стабільну роботу на всіх версіях Windows, що забезпечує широку сумісність із цільовою аудиторією.
- **Підтримка графіків**: Інтеграція з бібліотекою System.Windows.Forms.DataVisualization.Charting дозволяє легко

додавати графічну візуалізацію (наприклад, кругові діаграми), що є ключовою функцією додатку.

- **Альтернативи**: Розглядалися WPF (Windows Presentation Foundation) і UWP (Universal Windows Platform), але Windows Forms обрано через простоту освоєння, меншу ресурсоємність і достатність для базового функціоналу проєкту.

3. Середовище розробки: Visual Studio — це інтегроване середовище розробки (IDE) від Microsoft, яке підтримує створення додатків на C# із використанням Windows Forms.

Обґрунтування вибору:

- **Інструменти дизайну**: Вбудований дизайнер форм дозволяє візуально створювати інтерфейс, автоматично генеруючи код, що зменшує час розробки.
- **Дебагінг і тестування**: Visual Studio пропонує потужні засоби для налагодження коду, що сприяє виявленню та виправленню помилок на етапі розробки.
- **Інтеграція з .NET**: Повна сумісність із .NET забезпечує зручну роботу з бібліотеками та залежностями.
- **Доступність**: Безкоштовна версія Community Edition підходить для здобувачів освіти.

4. Збереження даних: JSON і Newtonsoft.Json. Для збереження даних транзакцій обрано формат JSON (JavaScript Object Notation) у комбінації з бібліотекою Newtonsoft.Json — популярним інструментом для серіалізації та десеріалізації об’єктів у C#.

Обґрунтування вибору:

- **Простота формату**: JSON є легким і читабельним форматом, що дозволяє зберігати структуровані дані (наприклад, список транзакцій) у текстовому вигляді.

- **Локальне зберігання**: Оскільки додаток не потребує підключення до бази даних чи хмарних сервісів, JSON-файл забезпечує автономність і портативність даних.
- **Швидка реалізація**: Newtonsoft.Json спрощує перетворення об’єктів C# у JSON і назад, що економить час на написання власного коду для роботи з файлами.
- **Альтернативи**: Розглядалися XML і SQLite, але JSON обрано через меншу складність і достатність для зберігання невеликого обсягу даних у проєкті.

5. Бібліотека візуалізації: System.Windows.Forms.DataVisualization.Charting – ця бібліотека є частиною .NET Framework і надає засоби для створення графіків і діаграм у Windows Forms додатках.

Обґрунтування вибору:

- **Інтеграція**: Вбудована підтримка в Windows Forms дозволяє легко додавати графіки без зовнішніх залежностей.
- **Функціональність**: Підтримка різних типів діаграм (наприклад, кругових, стовпчикових) відповідає потребі візуалізації доходів і витрат у проєкті.
- **Простота використання**: Налаштування графіка через код (наприклад, додавання точок даних) є інтуїтивним і не потребує значних зусиль.

Висновки до розділу 1

Проведений аналіз у першому розділі дипломної роботи дозволив сформувати чітке уявлення про мету, завдання та технологічні аспекти розробки програмного додатку "Фінансовий калькулятор для ведення особистого бюджету".

У підрозділі 1.1 "Постановка основної задачі розробки" визначено ключову мету проєкту — створення зручного настільного додатку для автоматизації обліку особистих фінансів, що включає функції додавання, редагування, видалення транзакцій, фільтрації даних і візуалізації фінансової статистики. Було сформульовано конкретні завдання, які охоплюють аналіз вимог, проєктування інтерфейсу, реалізацію функціоналу, збереження даних і тестування, а також встановлено вимоги до зручності, надійності та ефективності додатку. Це створило міцну основу для подальшої розробки.

У підрозділі 1.2 "Огляд та порівняння аналогічних систем" розглянуто популярні рішення для управління особистими фінансами, такі як Quicken, YNAB, Microsoft Excel, PocketSmith і GnuCash. Аналіз показав, що комерційні продукти пропонують розширений функціонал, зокрема синхронізацію з банками та прогнозування, але мають високу вартість і складність для початківців. Безкоштовні альтернативи, як-от GnuCash чи Excel, є доступними, але поступаються за зручністю та автоматизацією. Розроблений додаток займає нішу простого, безкоштовного інструменту з базовою візуалізацією, що відповідає потребам користувачів, які шукають автономне рішення без складних налаштувань.

Підрозділ 1.3 "Вибір та обґрунтування технологій розробки" обґрунтував використання C# як основної мови програмування, Windows Forms для створення інтерфейсу, Visual Studio як середовища розробки, JSON із бібліотекою Newtonsoft.Json для збереження даних і System.Windows.Forms. DataVisualization.Charting для графічної візуалізації. Вибір цих технологій зумовлений їхньою простотою, сумісністю, достатньою функціональністю для реалізації проєкту та можливістю швидкого створення стабільного

настільного додатку. Таке поєднання забезпечує баланс між легкістю розробки та відповідністю поставленим вимогам.

Перший розділ заклав теоретичний фундамент для розробки програмного продукту, визначивши його цілі, порівнявши з аналогами та обґрунтувавши технологічний стек. Отримані результати слугуватимуть основою для практичної реалізації, описаної в наступних розділах роботи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ І РОЗРОБКА

2.1. Вибір алгоритмів та їх ефективність

На етапі проєктування програмного додатку "Фінансовий калькулятор для ведення особистого бюджету" ключову роль відіграє вибір алгоритмів, які забезпечують реалізацію основних функцій: додавання, редагування, видалення транзакцій, фільтрації даних, обчислення фінансових показників і візуалізації результатів. Ефективність цих алгоритмів впливає на швидкодію програми, зручність її використання та коректність обробки даних. У цьому підрозділі розглянуто обрані алгоритми, їхню логіку та аналіз ефективності з точки зору часової та просторової складності.

1. Алгоритм додавання та редагування транзакцій

Для додавання нової транзакції використовується простий алгоритм вставки елемента в кінець списку (List<Transaction>), а для редагування — заміна елемента за індексом.

Логіка:

- Перевірка введених даних (опис, сума, дата, тип).
- Створення об’єкта Transaction із введених значень.
- Якщо це додавання — виклик методу Add для списку; якщо редагування — заміна елемента за індексом (transactions[index] = newTransaction).
- Оновлення інтерфейсу та збереження даних у файл.

Ефективність:

- **Часова складність**: O(1) для додавання в кінець списку та редагування за індексом, оскільки List<T> у C# реалізує динамічний масив із константним доступом за індексом.
- **Просторова складність**: O(n), де n — кількість транзакцій у списку, оскільки кожна транзакція займає фіксований обсяг пам’яті.
- **Обґрунтування**: Простота операцій додавання та редагування відповідає потребам додатку, де кількість транзакцій зазвичай невелика (десятки чи сотні записів), що робить цей алгоритм оптимальним за швидкодією.

2. Алгоритм видалення транзакцій

Видалення транзакції реалізовано через видалення елемента зі списку за індексом, отриманим із Tag вибраного елемента в ListView.

Логіка:

- Отримання індексу вибраної транзакції.
- Виклик методу RemoveAt(index) для списку transactions.
- Оновлення інтерфейсу та збереження змін.

Ефективність:

- **Часова складність**: O(n) у середньому, оскільки RemoveAt у List<T> зсуває всі наступні елементи на одну позицію вліво.
- **Просторова складність**: O(n) для зберігання списку транзакцій.
- **Обґрунтування**: Незважаючи на лінійну складність, операція видалення виконується рідко і на невеликих обсягах даних, тому вплив на продуктивність мінімальний. Альтернатива (наприклад, використання LinkedList) ускладнила б доступ за індексом, що є неприйнятним для даного проєкту.

3. Алгоритм фільтрації транзакцій

Фільтрація реалізована за допомогою LINQ-запитів для відбору транзакцій за датою та описом.

Логіка:

- Якщо активовано фільтр за датою, відбираються транзакції в заданому діапазоні (t.Date >= startDate && t.Date <= endDate).
- Якщо введено текст опису, перевіряється наявність підрядка в описі транзакції (t.Description.Contains(filterText)).
- Результат конвертується в список і передається для відображення.

Ефективність:

- **Часова складність**: O(n) для проходження по всіх елементах списку, де n — кількість транзакцій. LINQ оптимізує фільтрацію, але не змінює лінійну природу алгоритму.

- **Просторова складність**: O(m), де m — кількість відфільтрованих транзакцій (у гіршому випадку m=n).
- **Обґрунтування**: Лінійна складність є прийнятною для невеликих наборів даних, а використання LINQ забезпечує читабельність коду та легкість додавання нових умов фільтрації. Для великих обсягів даних можна було б розглянути індексацію, але це надмірно для даного проєкту.

4. Алгоритм обчислення фінансових показників

Опис: Обчислення доходів, витрат і балансу реалізовано через агрегацію даних із списку транзакцій за допомогою LINQ (Sum).

Логіка:

- Доходи: transactions.Where(t => t.IsIncome).Sum(t => t.Amount).
- Витрати: transactions.Where(t => !t.IsIncome).Sum(t => t.Amount).
- Баланс: різниця між доходами та витратами.

Ефективність:

- **Часова складність**: O(n) для проходження по списку транзакцій.
- **Просторова складність**: O(1), оскільки зберігаються лише підсумкові значення.
- **Обґрунтування**: Алгоритм простий і ефективний для невеликих наборів даних. Для великих обсягів можна було б кешувати результати, але в контексті проєкту це не потрібно.

5. Алгоритм візуалізації даних

Опис: Візуалізація реалізується через заповнення кругової діаграми (Chart) даними про доходи та витрати.

Логіка:

- Очищення серії даних.
- Додавання двох точок (Points.AddXY) для доходів і витрат із відповідними значеннями.
- Налаштування стилю відображення (підписи, формат).

Ефективність:

- **Часова складність**: O(1), оскільки додається фіксована кількість точок (2). Обчислення сум доходів і витрат уже враховано в попередньому алгоритмі.
- **Просторова складність**: O(1), оскільки використовується фіксована структура графіка.
- **Обґрунтування**: Алгоритм максимально простий і відповідає потребі базової візуалізації без надмірних обчислень.

Обрані алгоритми для додатку "Фінансовий калькулятор для ведення особистого бюджету" є простими, але ефективними з огляду на невеликий обсяг даних, з яким працює програма. Більшість операцій мають часову складність O(n) або O(1), що забезпечує швидкодію для типового використання (десятки чи сотні транзакцій). Використання LINQ підвищує читабельність і гнучкість коду, а структура List<T> є оптимальною для зберігання та обробки даних у пам’яті. Таким чином, обрані алгоритми відповідають вимогам проєкту за продуктивністю, простотою реалізації та підтримкою базового функціоналу, що робить їх доцільними для подальшої розробки.

2.2. Спроєктовані класи програми

На етапі розробки програмного додатку "Фінансовий калькулятор для ведення особистого бюджету" було створено набір класів, які забезпечують реалізацію основного функціоналу, обробку даних і взаємодію з інтерфейсом користувача. Класи спроєктовано з урахуванням принципів об’єктно-орієнтованого програмування, таких як інкапсуляція та модульність, що сприяє зрозумілості коду, легкості його підтримки та розширюваності. У цьому підрозділі розглянуто розроблені класи, їхню структуру, призначення та роль у структурі програми.

1. Клас Transaction – є базовою моделлю даних, яка представляє окрему фінансову транзакцію. Він використовується для зберігання інформації про доходи та витрати.

```
namespace FinancialCalc
{
    public class Transaction
    {
        public string? Description { get; set; }
        public decimal Amount { get; set; }
        public DateTime Date { get; set; }
        public bool IsIncome { get; set; }
    }
}
```

Рисунок 2.2.1 – Структура класу Transaction

Поля та властивості:

- Description (тип string?) — опис транзакції, може бути необов'язковим (nullable).
- Amount (тип decimal) — сума транзакції, обрано decimal для точних фінансових обчислень.
- Date (тип DateTime) — дата транзакції.
- IsIncome (тип bool) — логічний прапорець, що визначає тип транзакції (дохід чи витрата).

Призначення:

- Зберігання даних про кожну транзакцію в уніфікованому форматі.
- Використовується як елемент списку List<Transaction> для обробки в основному класі програми.

Особливості:

- Клас є простим DTO (Data Transfer Object) без методів, що відповідає його ролі як контейнера даних.
- Усі властивості мають публічні get і set, що дозволяє легко модифікувати об'єкти ззовні.

2. Клас Form1 – є основним класом програми, який успадковується від System.Windows.Forms.Form і реалізує головне вікно додатку з усією логікою управління транзакціями та інтерфейсом.

```
namespace FinancialCalc
{
    public partial class Form1 : Form
    {
        private List<Transaction> transactions = new List<Transaction>();
        private string filePath = "budget_data.json";
        private bool isEditing = false;

        // Конструктор
        public Form1()
        {
            InitializeComponent();
        }

        // Методи для обробки подій, оновлення UI, збереження/завантаження даних тощо
        private void Form1_Load(object sender, EventArgs e) { /* ... */ }
        private void btnAdd_Click(object sender, EventArgs e) { /* ... */ }
        private void AddOrEditTransaction(int editIndex = -1) { /* ... */ }
        private void UpdateTransactionList(List<Transaction> filteredTransactions = null) { /* ... */ }
        private void UpdateBalance() { /* ... */ }
        private void UpdateChart(List<Transaction>? filteredTransactions = null) { /* ... */ }
        private void SaveDataToFile() { /* ... */ }
        private void LoadDataFromFile() { /* ... */ }
        // Інші методи
    }
}
```

Рисунок 2.2.2 – Структура класу Form1

Поля:

- transactions (тип List<Transaction>) — список усіх транзакцій, основна структура даних програми.
- filePath (тип string) — шлях до файлу для збереження даних у форматі JSON.
- isEditing (тип bool) — прапорець для визначення стану редагування транзакції.

Методи:

- Form1_Load — ініціалізація компонентів і завантаження даних при старті програми.
- AddOrEditTransaction — додавання або редагування транзакції з валідацією введених даних.
- UpdateTransactionList — оновлення списку транзакцій у ListView.
- UpdateBalance — підрахунок і відображення фінансових показників (доходи, витрати, баланс).
- UpdateChart — оновлення кругової діаграми для візуалізації.

- SaveDataToFile і LoadDataFromFile — збереження та завантаження даних у JSON-файл.

Призначення:

- Управління логікою програми та взаємодією з користувачем.
- Координація роботи всіх компонентів інтерфейсу (кнопок, текстових полів, графіка тощо).

Особливості:

- Клас є partial, тобто частина коду (ініціалізація UI) згенерована дизайнером Windows Forms і розміщена в окремому файлі (Form1.Designer.cs).
- Використовує бібліотеки Newtonsoft.Json для роботи з JSON і System.Windows.Forms.DataVisualization.Charting для графіка.

3. Клас About – є допоміжним класом, який представляє діалогове вікно з інформацією про програму.

Структура:

```
namespace FinancialCalc
{
    public partial class About: Form
    {
        public About()
        {
            InitializeComponent();
        }
    }
}
```

Рисунок 2.2.3 – Структура класу About

Призначення:

- Відображення інформації про додаток

Особливості:

- Реалізовано як окреме вікно, що викликається через ShowDialog() із Form1.
- Не містить складної логіки, виконує лише інформаційну функцію.

Взаємодія класів

- **Клас Transaction** слугує моделлю даних, об’єкти якого створюються та зберігаються в списку transactions у Form1.
- **Клас Form1** є центральним елементом програми, який:
  - Створює, редагує та видаляє об’єкти Transaction.
  - Оновлює інтерфейс на основі даних із transactions.
  - Зберігає список у JSON-файл і завантажує його при старті.
- **Клас About** викликається з Form1 для відображення додаткової інформації та не впливає на основну логіку роботи з транзакціями.

Розроблені класи для програми "Фінансовий калькулятор для ведення особистого бюджету" забезпечують чіткий розподіл функціональності: Transaction відповідає за структуру даних, Form1 — за логіку та інтерфейс, а About — за інформаційну підтримку. Така організація відповідає принципам модульності та дозволяє легко розширювати функціонал. Використання стандартних засобів C# і Windows Forms сприяє простоті реалізації та стабільності роботи програми.

2.3. Програмна реалізація

Програмна реалізація додатку "Фінансовий калькулятор для ведення особистого бюджету" є завершальним етапом розробки, на якому спроєктовані алгоритми та класи втілено у вигляді робочого програмного продукту. Реалізація виконана з використанням мови програмування C# і технології Windows Forms у середовищі Visual Studio. У цьому підрозділі описано основні аспекти програмної реалізації, включаючи структуру коду, ключові функції, обробку подій, збереження даних і візуалізацію, а також наведено приклади коду для ілюстрації.

1. Структура програми

Програма складається з двох основних класів: Transaction (модель даних) і Form1 (головна форма з логікою та інтерфейсом). Додатково передбачено клас About для інформаційного вікна.

Основна логіка зосереджена в класі Form1, який:

- Ініціалізує інтерфейс у конструкторі через виклик InitializeComponent().
- Містить список List<Transaction> transactions для зберігання даних у пам’яті.
- Використовує файл budget_data.json для збереження даних між сеансами.

2. Ініціалізація та завантаження даних

Ініціалізація програми відбувається в обробнику події Form1_Load, який викликається при запуску форми:

```
private void Form1_Load(object sender, EventArgs e)
{
    listViewTransactions.Columns.Add("Дата", 100);
    listViewTransactions.Columns.Add("Опис", 200);
    listViewTransactions.Columns.Add("Сума", 100);
    listViewTransactions.Columns.Add("Тип", 80);

    comboBoxType.Items.Add("Дохід");
    comboBoxType.Items.Add("Витрата");
    comboBoxType.SelectedIndex = 0;

    LoadDataFromFile();
    UpdateTransactionList();
    UpdateBalance();
    UpdateChart();
}
```

Рисунок 2.3.1 – Обробник події Form1_Load

- Налаштовуються колонки для ListView і варіанти вибору в ComboBox.
- Викликається метод LoadDataFromFile() для завантаження даних із JSON-файлу:

```
private void LoadDataFromFile()
{
    try
    {
        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath);
            transactions = JsonConvert.DeserializeObject<List<Transaction>>(json)
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка при завантаженні даних: {ex.Message}");
        transactions = new List<Transaction>();
    }
}
```

Рисунок 2.3.2 – Обробник події LoadDataFromFile

Використовується бібліотека Newtonsoft.Json для десеріалізації, з обробкою помилок через try-catch.

3. Додавання та редагування транзакцій

Основна функція додавання та редагування реалізована в методі AddOrEditTransaction:

```
private void AddOrEditTransaction(int editIndex = -1)
{
    try
    {
        if (string.IsNullOrEmpty(txtDescription.Text))
        {
            MessageBox.Show("Введіть опис транзакції!");
            return;
        }
        if (!decimal.TryParse(txtAmount.Text, out decimal amount) || amount <= 0)
        {
            MessageBox.Show("Введіть коректну суму!");
            return;
        }

        Transaction transaction = new Transaction
        {
            Description = txtDescription.Text,
            Amount = amount,
            Date = dateTimePicker1.Value.Date,
            IsIncome = comboBoxType.SelectedIndex == 0
        };

        if (editIndex == -1)
            transactions.Add(transaction);
        else
            transactions[editIndex] = transaction;

        UpdateTransactionList();
        UpdateBalance();
        UpdateChart();
        SaveDataToFile();
        ClearInputs();
    }
```

Рисунок 2.3.3 – Обробник події AddOrEditTransaction

- Перевіряється коректність введених даних (опис і сума).
- Створюється об’єкт Transaction, який додається до списку або замінює існуючий запис залежно від параметра editIndex.
- Оновлюються інтерфейс, баланс, графік і зберігаються дані.

4. Оновлення інтерфейсу

Метод UpdateTransactionList відповідає за відображення транзакцій у ListView:

```
private void UpdateTransactionList(List<Transaction> filteredTransactions = null)
{
    listViewTransactions.Items.Clear();
    var displayList = filteredTransactions ?? transactions;

    foreach (var trans in displayList)
    {
        ListViewItem item = new ListViewItem(trans.Date.ToShortDateString());
        item.SubItems.Add(trans.Description);
        item.SubItems.Add(trans.Amount.ToString("C"));
        item.SubItems.Add(trans.IsIncome ? "Дохід" : "Витрата");
        item.Tag = transactions.IndexOf(trans);
        listViewTransactions.Items.Add(item);
    }
}
```

Рисунок 2.3.4 – Обробник події UpdateTransactionList

- Використовується параметр filteredTransactions для відображення відфільтрованих даних або всіх транзакцій за замовчуванням.
- Кожен елемент зберігає індекс транзакції в Tag для подальшого редагування чи видалення.

5. Обчислення фінансових показників

Метод UpdateBalance обчислює доходи, витрати та баланс:

```
private void UpdateBalance()
{
    decimal income = transactions.Where(t => t.IsIncome).Sum(t => t.Amount);
    decimal expenses = transactions.Where(t => !t.IsIncome).Sum(t => t.Amount);

    lblIncome.Text = $"Доходи: {income:C}";
    lblExpenses.Text = $"Витрати: {expenses:C}";
    lblBalance.Text = $"Баланс: {(income - expenses):C}";
}
```

Рисунок 2.3.5 – Обробник події UpdateBalance

Використовуються LINQ-запити для агрегації даних, а результат відображається у мітках із форматуванням валюти.

6. Візуалізація даних

Метод UpdateChart створює кругову діаграму:

```
private void UpdateChart(List<Transaction>? filteredTransactions = null)
{
    var displayList = filteredTransactions ?? transactions;

    chart.Series.Clear();
    chart.Series.Add("Budget");
    chart.Series["Budget"].ChartType = SeriesChartType.Pie;

    decimal income = displayList.Where(t => t.IsIncome).Sum(t => t.Amount);
    decimal expenses = displayList.Where(t => !t.IsIncome).Sum(t => t.Amount);

    if (income > 0)
        chart.Series["Budget"].Points.AddXY("Доходи", income);
    if (expenses > 0)
        chart.Series["Budget"].Points.AddXY("Витрати", expenses);

    chart.Series["Budget"]["PieLabelStyle"] = "Outside";
    chart.Series["Budget"].IsValueShownAsLabel = true;
    chart.Series["Budget"].LabelFormat = "C";
}
```

Рисунок 2.3.6 – Обробник події UpdateChart

Використовується бібліотека System.Windows.Forms.DataVisualization.Charting для побудови діаграми з доходами та витратами.

7. Збереження даних

Метод SaveDataToFile зберігає список транзакцій у JSON-файл:

```
private void SaveDataToFile()
{
    try
    {
        string json = JsonConvert.SerializeObject(transactions, Formatting.Indented);
        File.WriteAllText(filePath, json);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка при збереженні даних: {ex.Message}");
    }
}
```

Рисунок 2.3.7 – Обробник події SaveDataToFile

Викликається після кожної зміни даних і при закритті форми (Form1_FormClosing).

Програмна реалізація додатку "Фінансовий калькулятор для ведення особистого бюджету" забезпечує повний цикл роботи з фінансовими транзакціями: від введення даних до їхньої обробки, збереження та візуалізації. Код структуровано з використанням подієво-орієнтованого підходу Windows Forms, із чітким розподілом функцій між методами. Використання LINQ, JSON і графічних компонентів дозволило створити зручний і функціональний додаток, який відповідає поставленим вимогам. Реалізація є стабільною для невеликих обсягів даних і може бути розширена в майбутньому.

Висновки до розділу 2

Другий розділ дипломної роботи присвячений проєктуванню та розробці програмного додатку "Фінансовий калькулятор для ведення особистого бюджету", що дозволило перейти від теоретичних основ до практичної реалізації поставлених завдань. У процесі роботи було виконано детальний аналіз і втілення ключових етапів створення програми, що забезпечило її відповідність визначеним вимогам до функціональності, зручності та надійності.

У підрозділі 2.1 "Вибір алгоритмів та їх ефективність" проаналізовано та обґрунтовано алгоритми для основних функцій додатку: додавання, редагування, видалення транзакцій, фільтрації даних, обчислення фінансових показників і візуалізації. Обрані алгоритми, переважно з часовою складністю O(n)O(n)O(n) або O(1)O(1)O(1), є оптимальними для невеликих обсягів даних, з якими працює програма. Використання LINQ підвищило читабельність і гнучкість коду, а простота алгоритмів забезпечила швидкодію та легкість реалізації, що відповідає потребам проєкту.

Підрозділ 2.2 "Спроєктовані класи програми " описав структуру класів, які складають основу додатку. Клас Transaction слугує моделлю даних для зберігання транзакцій, тоді як клас Form1 об’єднує логіку програми та взаємодію з інтерфейсом. Допоміжний клас About забезпечує інформаційну

функцію. Такий розподіл функціональності відповідає принципам модульності та інкапсуляції, що полегшує підтримку та можливе розширення програми в майбутньому.

У підрозділі 2.3 "Програмна реалізація" детально описано втілення проєкту в коді. Реалізація охоплює ініціалізацію інтерфейсу, обробку подій, валідацію даних, збереження у форматі JSON і візуалізацію через кругову діаграму. Код структуровано з використанням подієво-орієнтованого підходу Windows Forms, із застосуванням бібліотек Newtonsoft.Json і System.Windows.Forms.DataVisualization.Charting. Отриманий програмний продукт є функціональним і стабільним рішенням для ведення особистого бюджету.

Розділ 2 продемонстрував комплексний підхід до створення додатку, від вибору алгоритмів і проєктування класів до їхньої практичної реалізації. Розроблений "Фінансовий калькулятор для ведення особистого бюджету" відповідає поставленим цілям, забезпечуючи користувачам зручний інструмент для управління фінансами, і може слугувати основою для подальших удосконалень.

РОЗДІЛ 3. КЕРІВНИЦТВО КОРИСТУВАННЯ ПРОГРАМОЮ

3.1. Мінімальні вимоги до системи

Для коректної роботи програмного додатку "Фінансовий калькулятор для ведення особистого бюджету" необхідно, щоб комп’ютер користувача відповідав певним технічним вимогам. Ці вимоги визначають мінімальні апаратні та програмні характеристики, які забезпечують стабільне функціонування програми, її швидкодію та можливість виконання всіх заявлених функцій, таких як обробка транзакцій, збереження даних і відображення графіків. У цьому підрозділі наведено перелік мінімальних системних вимог, а також рекомендації для оптимального використання додатку.

Апаратні вимоги

1. **Процесор**:
   - Мінімум: одноядерний процесор із частотою 1 ГГц або вище.
   - Рекомендується: двоядерний процесор із частотою 2 ГГц для забезпечення швидкої обробки даних при роботі з великою кількістю транзакцій.
   - Обґрунтування: додаток не є ресурсоємним, але сучасні процесори забезпечують комфортну швидкість роботи інтерфейсу та обчислень.
2. **Оперативна пам’ять (RAM)**:
   - Мінімум: 512 МБ.
   - Рекомендується: 2 ГБ або більше.
   - Обґрунтування: Windows Forms і .NET Framework потребують певний обсяг пам’яті для базового функціонування, а більший обсяг RAM покращує продуктивність при одночасному запуску кількох програм.
3. **Вільне місце на диску**:
   - Мінімум: 50 МБ для встановлення програми та зберігання JSON-файлу з даними.

- Рекомендується: 100 МБ для забезпечення місця під зростання обсягу даних у файлі budget_data.json.
- Обґрунтування: розмір програми невеликий, але обсяг файлу даних залежить від кількості збережених транзакцій.

4. **Роздільна здатність екрана**:
   - Мінімум: 800x600 пікселів.
   - Рекомендується: 1280x720 пікселів або вище.
   - Обґрунтування: інтерфейс розроблено з фіксованим розміром вікна (884x461 пікселів), тому вища роздільна здатність забезпечує комфортне розміщення всіх елементів.

Програмні вимоги

1. **Операційна система**:
   - Мінімум: Windows 7 із встановленим пакетом Service Pack 1.
   - Рекомендується: Windows 10 або Windows 11.
   - Обґрунтування: додаток побудовано на .NET Framework, який підтримується на всіх версіях Windows, починаючи з Windows XP, але Windows 7 є мінімальною стабільною версією для сучасного використання.
2. **Платформа .NET Framework**:
   - Мінімум: .NET Framework 4.0.
   - Рекомендується: .NET Framework 4.8 (остання версія на момент розробки).
   - Обґрунтування: Windows Forms і використані бібліотеки (Newtonsoft.Json, System.Windows.Forms.DataVisualization.Charting) потребують .NET Framework 4.0 або новішу версію. Оновлення до >The latest version of the Framework покращує сумісність і безпеку.
3. **Додаткове програмне забезпечення**:
   - Не потрібне, оскільки програма є автономною і не залежить від зовнішніх серверів чи баз даних.

- Рекомендується: наявність антивірусного програмного забезпечення для захисту файлу budget_data.json від несанкціонованого доступу чи пошкодження.

Додаткові зауваження

- **Права доступу**: Користувач повинен мати права на запис і читання в директорії, де розташовано програму, для коректного збереження файлу budget_data.json.
- **Локалізація**: Програма використовує формат валюти, що залежить від налаштувань операційної системи. Рекомендується перевірити регіональні налаштування для коректного відображення символів валюти (наприклад, гривні).
- **Сумісність**: Додаток не потребує підключення до Інтернету, що робить його придатним для використання в офлайн-режимі.

Мінімальні системні вимоги до додатку "Фінансовий калькулятор для ведення особистого бюджету" є досить низькими, що забезпечує його доступність для широкого кола користувачів із базовими комп'ютерами під управлінням Windows. Апаратні характеристики, такі як 512 МБ RAM і процесор 1 ГГц, достатні для запуску програми, тоді як .NET Framework 4.0 є єдиною обов'язковою програмною залежністю. Рекомендовані параметри (Windows 10/11, 2 ГБ RAM) підвищують комфорт роботи, особливо при обробці великих обсягів транзакцій. Таким чином, додаток є легким і універсальним рішенням для ведення особистого бюджету.

3.2. Інтерфейс користувача

Інтерфейс користувача (UI) додатку "Фінансовий калькулятор для ведення особистого бюджету" розроблено з урахуванням принципів зручності, інтуїтивності та простоти, щоб забезпечити ефективну взаємодію з програмою навіть для користувачів із мінімальним досвідом роботи з комп'ютером. Інтерфейс побудовано на базі технології Windows Forms і складається з кількох ключових зон, кожна з яких відповідає за певний аспект роботи з

фінансовими даними. У цьому підрозділі описано структуру інтерфейсу та алгоритми взаємодії користувача з його елементами.

Головне вікно програми має розмір 884x461 пікселів і поділено на три основні зони (Рисунок 3.2.1).

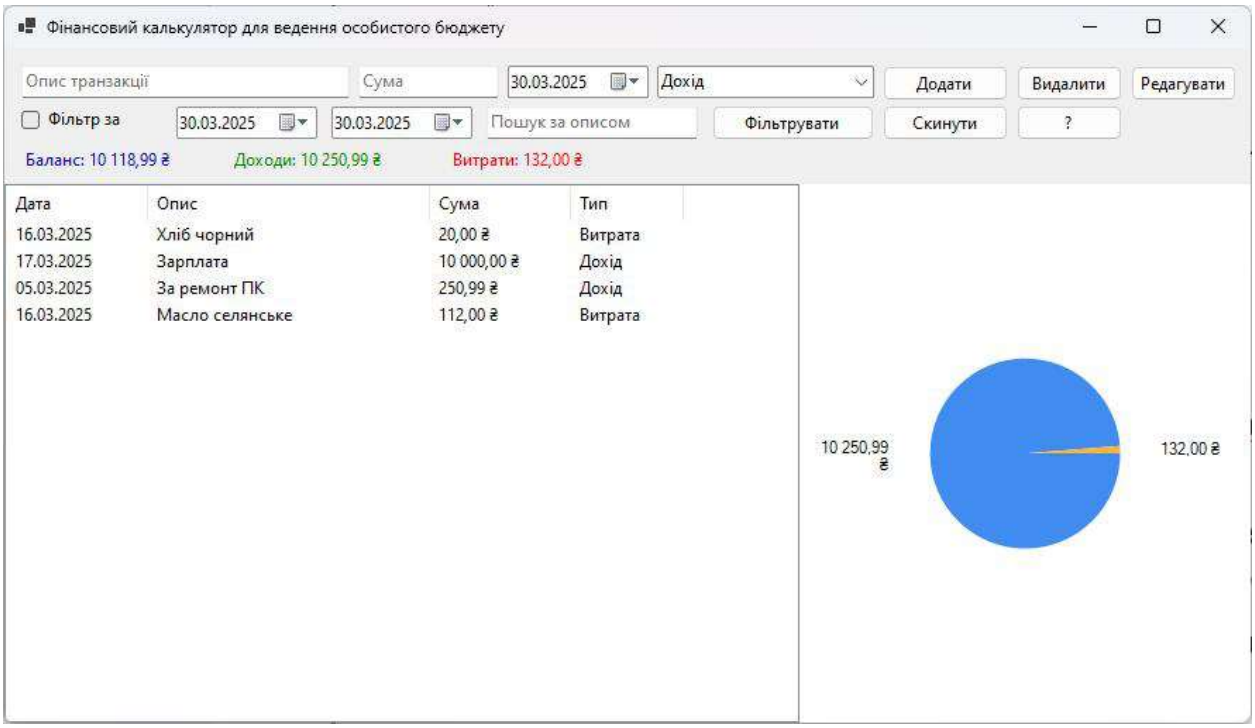

Рисунок 3.2.1 – Головне вікно програми

1. **Панель управління (верхня частина)**:

Розташована у верхній частині вікна (Dock = Top), розмір: 884x96 пікселів. Елементи:

- txtDescription (текстове поле): введення опису транзакції з плейсхолдером "Опис транзакції".
- txtAmount (текстове поле): введення суми з плейсхолдером "Сума".
- dateTimePicker1 (вибір дати): календар для вибору дати транзакції у короткому форматі (наприклад, "30.03.2025").
- comboBoxType (список, що розкривається): вибір типу транзакції ("Дохід" або "Витрата").
- btnAdd (кнопка): додавання або збереження транзакції (текст змінюється на "Зберегти зміни" при редагуванні).
- btnEdit (кнопка): редагування вибраної транзакції.

- btnDelete (кнопка): видалення вибраної транзакції.
- lblBalance, lblIncome, lblExpenses (мітки): відображення балансу (синій), доходів (зелений) і витрат (червоний).
- Елементи фільтрації: checkBoxDate (прапорець "Фільтр за датою"), dateTimePickerFilterStart і dateTimePickerFilterEnd (діапазон дат), txtFilterDescription (пошук за описом), btnFilter (кнопка "Фільтрувати"), btnResetFilter (кнопка "Скинути").
- buttonAbout (кнопка "?"): виклик інформаційного вікна.

2. **Список транзакцій (ліва частина)**:

- Реалізовано через listViewTransactions (Dock = Fill), розмір: 564x365 пікселів.
- Відображає таблицю з колонками: "Дата", "Опис", "Сума", "Тип".
- Підтримує вибір рядків (FullRowSelect = true) для редагування чи видалення.

3. **Графічна зона (права частина)**:

- Реалізовано через chart (Dock = Right), розмір: 320x365 пікселів.
- Відображає кругову діаграму з розподілом доходів і витрат, з підписами у форматі валюти.

Заголовок вікна: "Фінансовий калькулятор для ведення особистого бюджету". Вікно запускається по центру екрана (StartPosition = CenterScreen).

Алгоритми взаємодії з інтерфейсом користувача

Взаємодія користувача з програмою відбувається через подієво-орієнтований підхід, де дії (натискання кнопок, введення даних) викликають відповідні обробники подій. Нижче описано основні алгоритми взаємодії:

1. **Додавання транзакції**:

Щоб додати нову транзакцію, користувачу необхідно заповнити відповідні поля: ввести опис у текстове поле, вказати суму, обрати дату за допомогою календаря та вибрати тип транзакції зі списку, що розкривається. Після цього слід натиснути кнопку "Додати". Нові дані автоматично

додаються до таблиці транзакцій і зберігаються у файлі, оновлюючи відображення у списку.

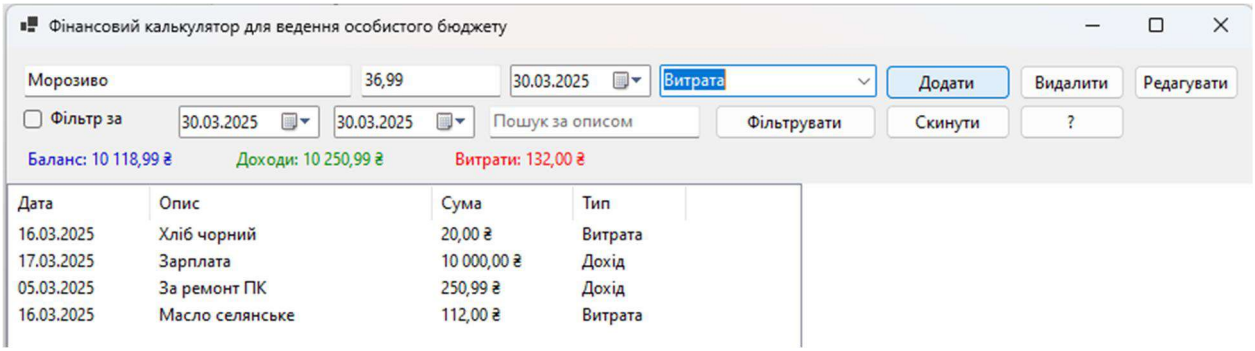

Рисунок 3.2.2 – Додавання транзакції

Введені дані перевіряються на коректність, потім додаються до списку і зберігаються у файл. Поля очищаються після додавання нової транзакції

2. **Редагування транзакції**:

Щоб відредагувати транзакцію, користувачу необхідно спочатку виділити її у списку таблиці, клацнувши мишкою, а потім натиснути кнопку "Редагувати". Після цього дані автоматично переносяться у відповідні поля для редагування. Користувач може внести потрібні зміни та підтвердити їх, натиснувши кнопку "Зберегти". Оновлені дані миттєво відображаються в таблиці та зберігаються у файлі.

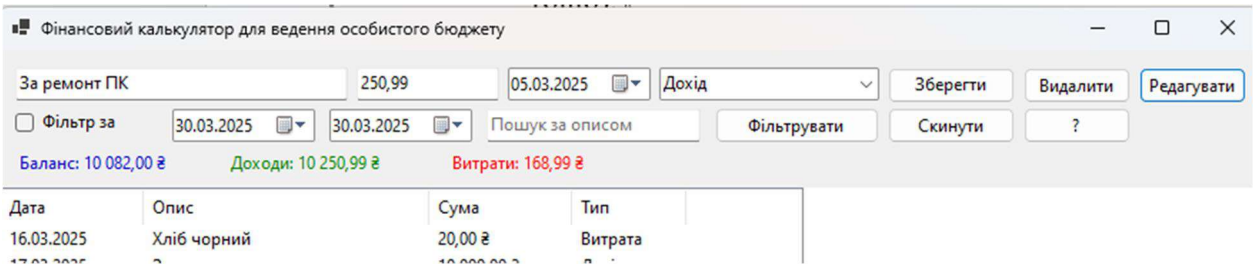

Рисунок 3.2.3 – Додавання транзакції

3. **Видалення транзакції**:

Для видалення транзакції користувачу необхідно спочатку виділити її у списку таблиці, клацнувши мишкою, а потім натиснути кнопку "Видалити". З’явиться діалогове вікно з проханням підтвердити дію. У разі згоди користувача транзакція буде видалена зі списку, а оновлені дані негайно відобразяться в таблиці та збережуться у файлі.

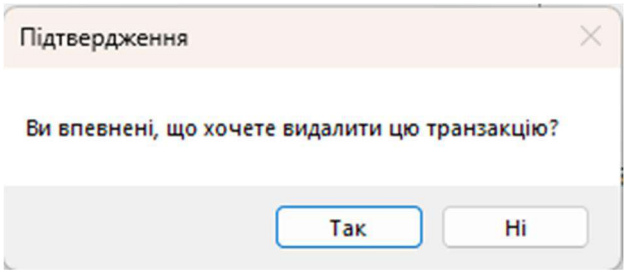

Рисунок 3.2.4 – Підтвердити дію

4. **Фільтрація транзакцій**:

Щоб відфільтрувати транзакції, користувачу необхідно активувати потрібні параметри: увімкнути прапорець "Фільтр за датою" та обрати діапазон дат у відповідних календарях або ввести ключові слова в поле пошуку за описом. Після цього слід натиснути кнопку "Фільтрувати". Відфільтрований список транзакцій автоматично відобразиться в таблиці, дозволяючи користувачу переглядати лише вибрані дані.

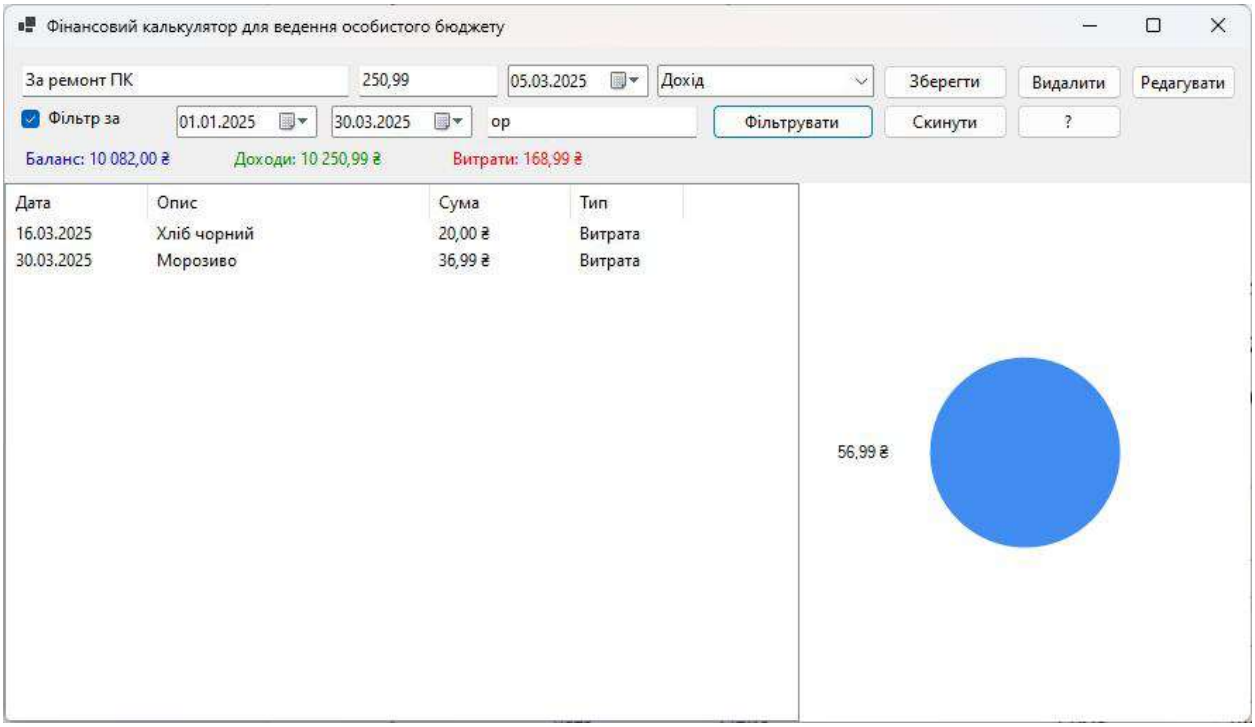

Рисунок 3.2.5 – Фільтрація

5. **Перегляд інформації про автора**:

Щоб переглянути інформацію про автора програми, користувачу достатньо натиснути кнопку зі знаком "?" на панелі управління. У відповідь відкриється модальне вікно, де відобразяться дані про розробника. Після ознайомлення користувач може закрити вікно, натиснувши відповідну кнопку.

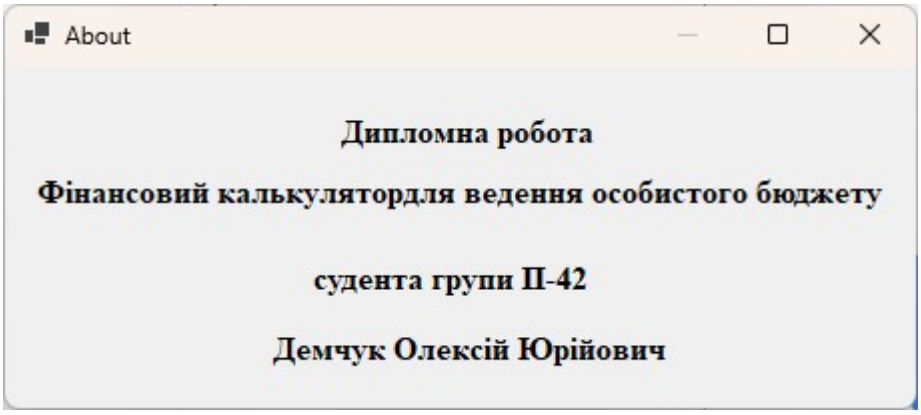

Рисунок 3.2.6 – Про автора програми

Особливості взаємодії

- **Валідація**: Перед додаванням чи редагуванням перевіряється наявність опису та коректність суми, з відображенням повідомлень про помилки (MessageBox).
- **Оновлення**: Кожна дія (додавання, редагування, видалення, фільтрація) супроводжується оновленням списку, міток і графіка для відображення актуального стану.
- **Збереження**: Зміни автоматично зберігаються у файл budget_data.json після кожної операції та при закритті програми.

Інтерфейс програми "Фінансовий калькулятор для ведення особистого бюджету" є логічно структурованим і зручним для користувача, з чітким розподілом зон для введення, перегляду та аналізу даних. Алгоритми взаємодії забезпечують інтуїтивну роботу з додатком: від простого додавання транзакцій до фільтрації та графічної візуалізації. Реалізація базується на подієво-орієнтованій моделі Windows Forms, що гарантує швидкий відгук на дії користувача та стабільність роботи.

Висновки до розділу 3

Третій розділ дипломної роботи присвячений опису практичного використання програмного додатку "Фінансовий калькулятор для ведення особистого бюджету", що забезпечує користувачів необхідною інформацією для ефективної роботи з програмою. У процесі аналізу було детально

розглянуто системні вимоги та особливості інтерфейсу, а також алгоритми взаємодії, що гарантують зручність і простоту експлуатації.

Підрозділ 3.1 "Мінімальні вимоги до системи" визначив апаратні та програмні характеристики, необхідні для запуску додатку. Завдяки низьким вимогам, таким як процесор із частотою 1 ГГц, 512 МБ оперативної пам’яті та .NET Framework 4.0, програма є доступною для широкого кола користувачів із базовими комп’ютерами під управлінням Windows. Рекомендовані параметри, як-от Windows 10/11 і 2 ГБ RAM, підвищують комфорт роботи, забезпечуючи стабільність і швидкодію навіть при обробці значної кількості транзакцій.

У підрозділі 3.2 "Інтерфейс програми та алгоритми взаємодії з інтерфейсом користувача" описано структуру головного вікна програми, що складається з панелі управління, списку транзакцій і графічної зони. Інтерфейс розроблено з акцентом на інтуїтивність: користувач може легко додавати, редагувати, видаляти транзакції, застосовувати фільтри та переглядати інформацію про автора завдяки чітко структурованим елементам управління. Алгоритми взаємодії, реалізовані через подієво-орієнтований підхід Windows Forms, забезпечують швидкий відгук на дії користувача, автоматичне оновлення даних і їхнє збереження, що робить роботу з програмою простою та ефективною.

Цей розділ підтвердив, що додаток "Фінансовий калькулятор для ведення особистого бюджету" є зручним і доступним інструментом для управління особистими фінансами. Поєднання низьких системних вимог із продуманим інтерфейсом і логічними алгоритмами взаємодії дозволяє користувачам легко освоїти програму та використовувати її для повсякденного обліку доходів і витрат.

ВИСНОВКИ

Дипломна робота присвячена розробці програмного додатку "Фінансовий калькулятор для ведення особистого бюджету" на базі технології Windows Forms, що дозволило створити зручний і функціональний інструмент для автоматизації обліку особистих фінансів. У процесі виконання роботи було проведено комплексний аналіз, проєктування та реалізацію програмного продукту, що відповідає сучасним потребам користувачів у сфері фінансового планування.

У першому розділі сформульовано мету та завдання розробки, проведено огляд аналогічних систем (Quicken, YNAB, Microsoft Excel, PocketSmith, GnuCash), що дозволило визначити конкурентні переваги розробленого додатку — безкоштовність, простоту та автономність. Обґрунтовано вибір технологій (C#, Windows Forms, Visual Studio, JSON, System.Windows.Forms.DataVisualization.Charting), які забезпечили ефективну реалізацію проєкту завдяки їхній сумісності, простоті та достатній функціональності.

Другий розділ охопив етапи проєктування та розробки. Обрані алгоритми з часовою складністю O(1) або O(n) виявилися оптимальними для обробки невеликих обсягів даних, а структура класів (Transaction, Form1) забезпечила модульність і легкість підтримки коду. Програмна реалізація підтвердила працездатність додатку, який підтримує додавання, редагування, видалення транзакцій, фільтрацію, обчислення фінансових показників і графічну візуалізацію у вигляді кругової діаграми.

Третій розділ продемонстрував практичну спрямованість роботи. Визначено мінімальні системні вимоги, які роблять додаток доступним для широкого кола користувачів із базовими комп’ютерами. Опис інтерфейсу та алгоритмів взаємодії підкреслив його інтуїтивність і зручність: користувач може легко вводити дані, аналізувати їх і переглядати статистику, що сприяє ефективному управлінню бюджетом.

Розроблений "Фінансовий калькулятор для ведення особистого бюджету" має як теоретичне, так і практичне значення. Теоретично робота поглибила розуміння принципів створення настільних додатків на Windows Forms, а практично — створила готовий продукт, який може бути використаний для підвищення фінансової грамотності та дисципліни. Програма є простою альтернативою комерційним рішенням, пропонуючи базовий набір функцій без складних налаштувань.

Перспективи подальшого розвитку включають додавання функцій експорту даних, підтримки кількох валют, інтеграції з банківськими API чи створення мобільної версії.

Дипломна робота досягла поставленої мети, створивши доступний і ефективний інструмент для ведення особистого бюджету, який може бути вдосконалений у майбутньому.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Альбано А. C# для чайників / А. Альбано; пер. з англ. О. В. Соловйов. — К.: Діалектика, 2020. — 432 с.
2. Буч Г. Об’єктно-орієнтований аналіз і проєктування з прикладами додатків / Г. Буч, Р. Максимчук, М. Енгл; пер. з англ. — 3-тє вид. — К.: Видавнича група BHV, 2018. — 720 с.
3. Гаврилов О. В. Програмування на C# для початківців / О. В. Гаврилов. — Х.: Фактор, 2019. — 256 с.
4. Еванс Е. Предметно-орієнтоване проєктування (DDD): структурування складних програмних систем / Е. Еванс; пер. з англ. — К.: Фабула, 2021. — 448 с.
5. Кнут Д. Мистецтво програмування. Том 1. Основні алгоритми / Д. Кнут; пер. з англ. — 3-тє вид. — К.: Видавництво "Вільямс", 2017. — 672 с.
6. Ліберти Дж. Програмування на C# для професіоналів / Дж. Ліберти, Дж. Мак-Дональд; пер. з англ. — К.: Діалектика, 2019. — 864 с.
7. Макконнелл С. Досконалий код / С. Макконнелл; пер. з англ. — 2-ге вид. — К.: Видавнича група BHV, 2020. — 896 с.
8. Нагін П. Оволодіння Windows Forms у C# / П. Нагін; пер. з англ. — СПб.: Пітер, 2016. — 512 с.
9. Седжвік Р. Алгоритми на C#. Частини 1–4: Основи, структури даних, сортування, пошук / Р. Седжвік; пер. з англ. — К.: Видавництво "Вільямс", 2018. — 960 с.
10. Троелсен Е. C# 10 і .NET 6 для програмістів / Е. Троелсен, Ф. Джепікс; пер. з англ. — К.: Діалектика, 2022. — 1248 с.
11. Фаулер М. Рефакторинг: удосконалення існуючого коду / М. Фаулер; пер. з англ. — К.: Фабула, 2019. — 432 с.
12. Шилдт Г. C#: повний довідник / Г. Шилдт; пер. з англ. — К.: Видавництво "Вільямс", 2020. — 1056 с.
13. Бондаренко О. М. Основи алгоритмізації та програмування: навчальний посібник / О. М. Бондаренко. — К.: КНЕУ, 2017. — 320 с.
14. Коваленко В. І. Програмування графічних інтерфейсів у Windows Forms / В. І. Коваленко // Вісник НТУУ "КПІ". Інформатика, управління та обчислювальна техніка. — 2019. — № 56. — С. 45–52.
15. Петров О. С. Використання JSON для збереження даних у настільних додатках / О. С. Петров // Наукові записки ХНУРЕ. — 2020. — № 3. — С. 78–85.
16. Microsoft Docs: Огляд Windows Forms [Електронний ресурс] // Microsoft. — Режим доступу: https://docs.microsoft.com/uk-ua/dotnet/desktop/winforms/overview/. — Дата звернення: 25.03.2025.
17. Newtonsoft.Json Documentation [Електронний ресурс] // Newtonsoft. — Режим доступу: https://www.newtonsoft.com/json/help/html/Introduction.htm. — Дата звернення: 26.03.2025.

18. Chart Controls in Windows Forms [Електронний ресурс] // Microsoft Docs. — Режим доступу: https://docs.microsoft.com/uk-ua/dotnet/desktop/winforms/controls/chart-control. — Дата звернення: 27.03.2025.
19. Stack Overflow: C# Programming Questions [Електронний ресурс] // Stack Overflow. — Режим доступу: https://stackoverflow.com/questions/tagged/c%23. — Дата звернення: 28.03.2025.
20. GitHub: Приклади коду для Windows Forms [Електронний ресурс] // GitHub. — Режим доступу: https://github.com/topics/windows-forms. — Дата звернення: 29.03.2025.

ДОДАТКИ

ДОДАТОК А.

Програмний код модулів

```
using System.Windows.Forms.DataVisualization.Charting;

namespace FinancialCalc
{
   partial class Form1
   {
      /// <summary>
      ///  Required designer variable.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      ///  Clean up any resources being used.
      /// </summary>
      /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }

      #region Windows Form Designer generated code

      /// <summary>
      ///  Required method for Designer support - do not modify
      ///  the contents of this method with the code editor.
      /// </summary>
      private void InitializeComponent()
```