ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА ГРАФІЧНОЇ ПРОГРАМИ ДЛЯ ВІДСТЕЖЕННЯ ФІНАНСІВ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ C#»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Вакуленко Станіслав Русланович
_____«____» _____202_ р.
(підпис)

Науковий керівник к.пед.н., доцент Кошова Оксана Петрівна
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2024

РЕФЕРАТ

Записка: 69 с., 8 рис., 1 таблиця, 1 додаток, 21 джерел.

НАСТІЛЬНИЙ ДОДАТОК, C#, ВІДСТЕЖЕННЯ ФІНАНСІВ, АНАЛІТИКА

Об’єкт розробки – створення настільного додатку для відстеження фінансів з використанням мови програмування C#, що включає функціонал реєстрації та авторизації користувачів, управління транзакціями, створення та ведення бюджетів, а також візуалізацію фінансової статистики.

Мета роботи – розробити настільний додаток, який забезпечує зручне управління особистими фінансами, дозволяє користувачам відстежувати доходи та витрати, планувати бюджети, та аналізувати фінансову активність за допомогою графічних засобів.

Методи дослідження – використання мови програмування C# для розробки настільного додатку з використанням технологій .NET Framework та Entity Framework для управління даними. Забезпечення інтерактивного та зручного інтерфейсу за допомогою бібліотек MaterialDesign та MahApps.Metro.IconPacks. Забезпечення безпеки даних користувачів за допомогою сучасних методів захисту інформації.

Розроблено настільний додаток, що надає можливість реєстрації та авторизації користувачів, управління транзакціями, створення та редагування бюджетів, а також відображення фінансової статистики у вигляді графіків та діаграм. Додаток використовує локальну базу даних MSSQL Server для зберігання даних користувачів та їх фінансових операцій.

Було проведено аналіз існуючих програм для фінансового менеджменту, їх методів та інструментів. Проектування та розробка настільного додатку, включаючи структуру проекту, використані бібліотеки та технології. Тестування додатку на різних операційних системах та аналіз його впливу на зручність користувачів.

Результати роботи включають розроблений настільний додаток з документацією, що містить інструкції для користувачів та рекомендації щодо

подальшого розвитку та масштабування додатку. Висвітлено переваги та недоліки використання розробленого додатку та технологій у контексті забезпечення ефективного управління особистими фінансами та безпеки даних користувачів.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕННЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
C#	Мова програмування, розроблена Microsoft, яка використовується для розробки настільних додатків
.NET	Платформа розробки від Microsoft, яка надає широкий спектр інструментів і бібліотек для створення додатків
WPF	Windows Presentation Foundation, технологія для створення інтерфейсу користувача у настільних додатках на Windows
Entity Framework	ORM (Object-Relational Mapping) для .NET, яка полегшує роботу з базами даних
ORM	Об'єктно-реляційне відображення, технологія, яка дозволяє взаємодіяти з базою даних за допомогою об'єктів
SQL Server	Реляційна база даних, розроблена Microsoft, яка використовується для зберігання даних додатків
LINQ	Language Integrated Query, технологія для виконання запитів до баз даних у C#
XAML	Extensible Application Markup Language, мова розмітки для створення графічного інтерфейсу у WPF
MVVM	Model-View-ViewModel, архітектурний шаблон для розробки додатків з розділенням логіки бізнесу та представлення
CRUD	Аббревіатура, що означає створення (Create), читання (Read), оновлення (Update), видалення (Delete) даних
UI	User Interface, інтерфейс користувача, засоби взаємодії користувача з програмою
UX	User Experience, досвід користувача, враження користувача від використання програми
JSON	JavaScript Object Notation, формат обміну даними, який є зручним для читання людиною

	та машинної обробки
ADO.NET	Технологія доступу до даних у .NET, яка дозволяє працювати з реляційними базами даних
ASP.NET	Фреймворк для розробки веб-додатків та сервісів на платформі .NET
Visual Studio	Інтегроване середовище розробки (IDE) від Microsoft для створення програмного забезпечення
MVVM Light	Бібліотека для спрощення розробки WPF додатків за архітектурним шаблоном MVVM
MaterialDesignInXAML Toolkit	Бібліотека для застосування Material Design у WPF додатках
MahApps.Metro	Бібліотека для створення сучасних та стильних інтерфейсів у WPF додатках
NuGet	Менеджер пакетів для платформи .NET, який дозволяє легко додавати бібліотеки до проекту
CSRF	Cross-Site Request Forgery, захист від підробки міжсайтових запитів, метод безпеки для захисту від атак на веб-додатки
JWT	JSON Web Token, стандарт для створення токенів доступу, що дозволяють безпечний обмін інформацією
API	Application Programming Interface, інтерфейс програмування додатків, що дозволяє взаємодію між різними програмами
SSL/TLS	Протоколи для забезпечення захищеної передачі даних через Інтернет
MVC	Model-View-Controller, архітектурний шаблон, який розділяє додаток на три взаємодіючі компоненти: модель, представлення та контролер
CLI	Command Line Interface, командний інтерфейс користувача, інтерфейс для взаємодії з додатком через командний рядок

ВСТУП

У сучасному світі, де фінансова грамотність та управління фінансами стають важливими складовими успішного життя, розробка інструментів для відстеження та планування фінансових ресурсів набуває все більшої актуальності. Настільні додатки для управління фінансами допомагають користувачам ефективно контролювати свої доходи та витрати, планувати бюджети та аналізувати фінансову активність, що є особливо важливим в умовах швидкозмінного економічного середовища.

Актуальним є розробка та вдосконалення настільних додатків для управління фінансами, які забезпечують зручність у використанні, функціональність та безпеку даних. Використання сучасних технологій, таких як C# та .NET Framework, дозволяє створювати потужні та надійні рішення, що відповідають потребам користувачів.

Об'єктом розробки є настільний додаток для відстеження фінансів, який включає функціонал реєстрації та авторизації користувачів, управління транзакціями, створення та ведення бюджетів, а також візуалізацію фінансової статистики.

Предметом розробки є система керування фінансами, реалізована на базі технологій C# та .NET Framework з використанням бібліотек MaterialDesign та MahApps.Metro.IconPacks.

Метою роботи є розробка настільного додатку, який забезпечує зручне управління фінансами, дозволяє користувачам відстежувати доходи та витрати, планувати бюджети, та аналізувати фінансову активність за допомогою графічних засобів.

Методи дослідження включають аналіз існуючих програм для фінансового менеджменту, їх методів та інструментів, розробку структури проекту, використання бібліотек та технологій, проектування та реалізацію основного функціоналу, тестування додатку на різних операційних системах, а також аналіз його впливу на зручність користувачів.

Робота складається з вступу, де визначено актуальність, мету, об'єкт і предмет дослідження; інформаційного огляду, який присвячений теоретичним основам розробки настільних додатків, інструментам та технологіям для розробки на C#, а також прикладам аналогічних додатків; теоретичної частини, що описує принципи розробки настільних додатків, особливості роботи з C# та .NET Framework та питання забезпечення безпеки даних; практичної частини, яка включає опис процесу розробки програмного забезпечення, структури проекту, використаних бібліотек та технологій, а також функціоналу програми; висновків та додатків.

Розробка настільного додатку для управління фінансами на базі C# дозволяє детально вивчити процеси організації та управління фінансовими ресурсами, забезпечення безпеки даних, а також інтеграції аналітичних інструментів для покращення фінансової грамотності користувачів. Це сприяє створенню ефективних інструментів для планування та контролю фінансів, що підвищує загальну продуктивність та організованість користувачів.

1. ПОСТАНОВКА ЗАДАЧІ

Опис проблеми

У сучасному світі ефективне управління фінансами є невід'ємною частиною особистого та професійного життя. Багато людей стикаються з проблемами в організації своїх доходів та витрат, що може призводити до фінансових труднощів та неефективного використання ресурсів. Існуючі рішення часто або занадто складні для пересічного користувача, або не забезпечують достатньої функціональності для комплексного управління фінансами.

Аналіз існуючих рішень

Існує багато програмних рішень для управління фінансами, як комерційних, так і безкоштовних. До них відносяться такі додатки, як Quicken, Mint, YNAB (You Need A Budget), які пропонують різні функціональні можливості для відстеження доходів, витрат та планування бюджетів. Однак, ці додатки мають певні недоліки, зокрема складний інтерфейс, високу вартість підписки або обмежену функціональність у безкоштовних версіях.

Постановка задачі дослідження

Метою даного дослідження є розробка настільного додатку для управління фінансами, що забезпечує зручний і зрозумілий інтерфейс, широкі функціональні можливості та високу безпеку даних. Основні завдання, які необхідно вирішити:

1. Розробити систему реєстрації та авторизації користувачів, що забезпечує безпечний доступ до даних.
2. Реалізувати функціонал для управління транзакціями, що дозволяє користувачам вводити, редагувати та видаляти записи про доходи та витрати.
3. Розробити модуль для створення та ведення бюджетів, що допомагає користувачам планувати фінансові ресурси.
4. Реалізувати візуалізацію фінансової статистики за допомогою графіків та діаграм для покращення аналізу фінансової активності.
5. Забезпечити збереження даних у локальній базі даних MSSQL Server та реалізувати безпечний доступ до них.
6. Використати сучасні бібліотеки та інструменти для створення зручного та

інтерактивного користувацького інтерфейсу.

Розробка такого додатку дозволить користувачам ефективно управляти своїми фінансами, забезпечуючи простоту використання та високу функціональність, що відповідає сучасним вимогам до програмного забезпечення для фінансового менеджменту.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Мови програмування для розробки настільних додатків

Розробка настільних додатків потребує вибору відповідної мови програмування, яка забезпечить ефективність, продуктивність та зручність у використанні. Існує декілька популярних мов програмування, які широко використовуються для створення настільних додатків, кожна з яких має свої переваги та недоліки.

C#

C# (C-Sharp) – це мова програмування, розроблена компанією Microsoft як частина платформи .NET. Вона є однією з найпопулярніших мов для розробки настільних додатків завдяки своїй продуктивності, потужним інструментам та підтримці багатьох сучасних технологій. C# забезпечує високий рівень абстракції та підтримку об'єктно-орієнтованого програмування.

Приклади настільних додатків на C#:

- **Visual Studio** – інтегроване середовище розробки (IDE), яке само по собі написане на C# та використовується для розробки програмного забезпечення.
- **Paint.NET** – графічний редактор, який також написаний на C# та є потужною альтернативою більш простим інструментам редагування зображень.

Java

Java – це об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (нині належить Oracle). Вона відома своєю портативністю завдяки принципу "пиши раз, запускай всюди" (write once, run anywhere), що забезпечується платформою Java Virtual Machine (JVM).

Приклади настільних додатків на Java:

- **Eclipse** – популярне інтегроване середовище розробки для програмістів, яке широко використовується для розробки Java-додатків.
- **NetBeans** – ще одне IDE для розробки програмного забезпечення, яке підтримує не тільки Java, але й інші мови програмування.

Python

Python – це високорівнева мова програмування, відома своєю простотою та читабельністю. Вона часто використовується для розробки різноманітних додатків, включаючи настільні, завдяки своїй гнучкості та великій кількості доступних бібліотек.

Приклади настільних додатків на Python:

- **Dropbox** – популярний сервіс для зберігання даних у хмарі, який частково написаний на Python.
- **Blender** – потужний інструмент для створення тривимірної графіки, який також використовує Python для своїх скриптів.

C++

C++ – це мова програмування, яка є розширенням мови C і підтримує об'єктно-орієнтоване програмування. Вона відома своєю продуктивністю та ефективністю, що робить її популярним вибором для розробки програмного забезпечення, яке потребує високої швидкості виконання.

Приклади настільних додатків на C++:

- **Adobe Photoshop** – один з найвідоміших графічних редакторів, частково написаний на C++.
- **Mozilla Firefox** – популярний веб-браузер, який використовує C++ для забезпечення швидкої та надійної роботи.

Swift

Swift – це мова програмування, розроблена компанією Apple для створення додатків під macOS та iOS. Вона відома своєю швидкістю, безпекою та сучасними можливостями.

Приклади настільних додатків на Swift:

- **Xcode** – інтегроване середовище розробки від Apple, яке використовується для створення додатків під macOS та iOS.
- **Slack (macOS version)** – версія популярного додатку для спільної роботи, розроблена для macOS з використанням Swift.

Kotlin

Kotlin – це сучасна мова програмування, розроблена компанією JetBrains, яка є повністю сумісною з Java та працює на JVM. Вона набирає популярності завдяки своїй лаконічності та сучасним можливостям.

Приклади настільних додатків на Kotlin:

- **TornadoFX** – фреймворк для розробки настільних додатків на Kotlin, який забезпечує зручний інтерфейс для створення GUI-додатків.
- **JetBrains Toolbox** – інструмент для управління IDE від JetBrains, який частково написаний на Kotlin.

Вибір мови програмування для розробки настільного додатку залежить від конкретних вимог проекту, наявності інструментів та бібліотек, а також від уподобань розробників. У даній роботі вибір був зроблений на користь C# завдяки його потужним можливостям, підтримці .NET Framework, та широкому використанню у розробці настільних додатків.

2.2 Огляд інструментів і технологій для розробки на C#

Розробка настільних додатків на мові програмування C# включає використання різноманітних інструментів та технологій, які допомагають ефективно реалізовувати функціональні можливості програмного забезпечення. У цьому розділі розглянемо основні інструменти та технології, які використовуються для створення настільних додатків на C#.

.NET Framework

.NET Framework – це програмна платформа, розроблена компанією Microsoft, яка надає комплексну інфраструктуру для розробки, розгортання та виконання різноманітних додатків, включаючи настільні, веб-додатки та сервіси. .NET Framework забезпечує широкий набір бібліотек та інструментів, що дозволяють ефективно розробляти додатки на мові C#.

Основні компоненти .NET Framework:

- **Common Language Runtime (CLR)** – середовище виконання, яке забезпечує управління пам'яттю, безпекою, обробкою винятків та іншими основними

послугами для виконання .NET додатків.

- **.NET Class Library** – велика бібліотека класів, що включає в себе різноманітні функції для роботи з файлами, базами даних, графічним інтерфейсом користувача, мережевими протоколами тощо.

Visual Studio

Visual Studio – це інтегроване середовище розробки (IDE), розроблене компанією Microsoft, яке надає потужні інструменти для створення, редагування, налагодження та тестування програмного забезпечення. Visual Studio підтримує різноманітні мови програмування, включаючи C#, і забезпечує зручний та інтуїтивний інтерфейс для розробників.

Основні можливості Visual Studio:

- **Редактор коду** – забезпечує синтаксичне підсвічування, автоматичне завершення коду, рефакторинг та інші функції, що полегшують написання коду.
- **Налагодження** – потужні інструменти для налагодження коду, включаючи точки зупинки, покрокове виконання, перегляд змінних та стеку викликів.
- **Інтеграція з системами контролю версій** – підтримка Git, Team Foundation Server (TFS) та інших систем контролю версій.

Entity Framework

Entity Framework – це об'єктно-реляційний відображувач (ORM), який дозволяє розробникам працювати з базами даних, використовуючи об'єктно-орієнтований підхід. Замість написання SQL-запитів для доступу до даних, розробники можуть використовувати Entity Framework для роботи з базою даних за допомогою класів та об'єктів C#.

Основні можливості Entity Framework:

- **Code First** – підхід, при якому моделі даних визначаються за допомогою класів C#, а бази даних генеруються автоматично на основі цих моделей.
- **Database First** – підхід, при якому моделі даних генеруються автоматично на основі існуючої бази даних.
- **LINQ (Language Integrated Query)** – мова запитів, яка дозволяє писати

запити до бази даних за допомогою C#.

WPF (Windows Presentation Foundation)

Windows Presentation Foundation (WPF) – це графічна підсистема для створення настільних додатків на Windows з використанням мови програмування C#. WPF забезпечує потужні можливості для створення складних графічних інтерфейсів, підтримує анімацію, двовимірну та тривимірну графіку, стилі та шаблони.

Основні можливості WPF:

- **Data Binding** – забезпечує прив'язку даних між інтерфейсом користувача та логікою програми.
- **Styles and Templates** – дозволяє створювати консистентні та стильні інтерфейси користувача.
- **Animations and Graphics** – підтримка анімацій, двовимірної та тривимірної графіки для створення інтерактивних інтерфейсів.

MaterialDesignInXAML Toolkit

MaterialDesignInXAML Toolkit – це бібліотека, яка забезпечує підтримку Material Design у WPF додатках. Вона надає широкий набір готових компонентів, стилів та контролів, що відповідають принципам Material Design.

Основні можливості MaterialDesignInXAML Toolkit:

- **Стилі та теми** – забезпечують єдиний стиль для всього додатку відповідно до принципів Material Design.
- **Контроли** – готові компоненти, такі як кнопки, текстові поля, меню та інші елементи інтерфейсу користувача.

MahApps.Metro

MahApps.Metro – це бібліотека для створення стильних та сучасних інтерфейсів користувача в WPF додатках. Вона надає різноманітні стилі та контролі, що дозволяють легко створювати привабливі та функціональні інтерфейси.

Основні можливості MahApps.Metro:

- **Metro-styled controls** – стилізовані компоненти інтерфейсу, які забезпечують

сучасний вигляд додатку.

- **Themes and customizations** – різноманітні теми та можливості налаштування зовнішнього вигляду додатку.

Використання цих інструментів та технологій забезпечує розробникам потужні можливості для створення ефективних та зручних настільних додатків на мові програмування C#. Це дозволяє зосередитися на реалізації функціональних вимог додатку та забезпеченні високої якості кінцевого продукту.

2.3 Аналіз існуючих програм для фінансового менеджменту

У сучасному світі існує безліч програм для фінансового менеджменту, які допомагають користувачам ефективно управляти своїми фінансами. Ці програми пропонують різноманітні функціональні можливості, починаючи від простого обліку доходів та витрат і закінчуючи складними аналітичними інструментами. У цьому розділі розглянемо деякі з найбільш популярних програм для фінансового менеджменту, їхні можливості, переваги та недоліки.

Quicken

Quicken – одна з найстаріших і найпопулярніших програм для управління фінансами, яка пропонує широкий набір функцій для обліку доходів і витрат, створення бюджетів та управління інвестиціями.

Основні можливості Quicken:

- Облік доходів та витрат з детальними категоріями.
- Створення та моніторинг бюджетів.
- Управління банківськими рахунками та кредитними картками.
- Інструменти для аналізу інвестиційного портфеля.
- Підтримка імпорту даних з банківських рахунків.

Переваги:

- Широкий набір функцій.
- Інтуїтивно зрозумілий інтерфейс.
- Підтримка імпорту даних з різних джерел.

Недоліки:

- Висока вартість підписки.
- Деякі функції доступні тільки в преміум-версіях.
- Може бути складним для новачків.

Mint

Mint – безкоштовний онлайн-сервіс для управління особистими фінансами, який належить компанії Intuit. Він пропонує простий у використанні інтерфейс і широкий набір функцій для обліку фінансів.

Основні можливості Mint:

- Автоматичне імпортування транзакцій з банківських рахунків та кредитних карток.
- Створення та моніторинг бюджетів.
- Категоризація доходів та витрат.
- Інструменти для аналізу фінансової активності.
- Відстеження кредитного рейтингу.

Переваги:

- Безкоштовне використання.
- Автоматизація багатьох процесів.
- Зручний та інтуїтивний інтерфейс.

Недоліки:

- Обмежена підтримка імпорту даних з деяких банків.
- Залежність від інтернет-з'єднання.
- Обмежені можливості для користувачів з складними фінансовими потребами.

YNAB (You Need A Budget)

YNAB – програма для управління фінансами, яка акцентує увагу на створенні та дотриманні бюджетів. Вона допомагає користувачам контролювати свої витрати та покращувати фінансову дисципліну.

Основні можливості YNAB:

- Створення бюджетів на основі реальних доходів.
- Відстеження доходів та витрат у режимі реального часу.

- Інструменти для аналізу фінансової активності.
- Підтримка імпорту даних з банківських рахунків.

Переваги:

- Орієнтованість на бюджетування.
- Прості та зрозумілі інструменти.
- Інтуїтивний інтерфейс.

Недоліки:

- Висока вартість підписки.
- Обмежені можливості для управління інвестиціями.
- Потреба у регулярному введенні даних вручну.

Personal Capital

Personal Capital – це фінансовий додаток, який поєднує функції управління особистими фінансами з інструментами для інвестування. Він пропонує широкий набір функцій для аналізу та управління фінансовими ресурсами.

Основні можливості Personal Capital:

- Управління доходами та витратами.
- Створення бюджетів.
- Інструменти для аналізу інвестиційного портфеля.
- Планування виходу на пенсію.
- Інтеграція з банківськими рахунками та кредитними картками.

Переваги:

- Широкий набір інструментів для управління фінансами та інвестуванням.
- Зручний інтерфейс.
- Безкоштовний базовий функціонал.

Недоліки:

- Деякі функції доступні тільки в преміум-версіях.
- Висока вартість преміум-послуг.
- Може бути складним для користувачів без досвіду в інвестуванні.

GnuCash

GnuCash – безкоштовна програма з відкритим вихідним кодом для управління

особистими та малими бізнес-фінансами. Вона пропонує потужний набір функцій для обліку доходів і витрат, створення звітів та ведення бухгалтерії.

Основні можливості GnuCash:

- Облік доходів та витрат.
- Створення та моніторинг бюджетів.
- Підтримка двійкового запису для бухгалтерського обліку.
- Генерація детальних фінансових звітів.
- Інструменти для управління інвестиціями.

Переваги:

- Безкоштовне використання.
- Потужні функції бухгалтерського обліку.
- Гнучкість та можливість налаштування.

Недоліки:

- Складний інтерфейс, який може бути важким для новачків.
- Обмежена підтримка користувачів.
- Відсутність деяких сучасних функцій, які пропонують комерційні програми.

Результати аналізу

Аналіз існуючих програм для фінансового менеджменту показує, що кожна з них має свої переваги та недоліки. Комерційні програми, такі як Quicken та YNAB, пропонують потужні інструменти для управління фінансами, але часто мають високу вартість. Безкоштовні програми, такі як Mint та GnuCash, забезпечують основні функції, але можуть бути обмежені в можливостях та підтримці користувачів. Для розробки власного додатку важливо врахувати ці особливості та забезпечити користувачам зручний, функціональний та безпечний інструмент для управління фінансами.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Принципи розробки настільних додатків

Розробка настільних додатків – це складний процес, який вимагає дотримання певних принципів та практик для забезпечення якості, ефективності та зручності використання програмного забезпечення. У цьому розділі розглянемо основні принципи, які слід враховувати під час розробки настільних додатків.

Об'єктно-орієнтоване програмування (ООП)

Об'єктно-орієнтоване програмування є основою для розробки настільних додатків. Основні концепції ООП включають:

- **Класи та об'єкти:** Класи визначають структуру та поведінку об'єктів, а об'єкти є конкретними екземплярами класів.
- **Інкапсуляція:** Збереження внутрішнього стану об'єкта прихованим від зовнішнього середовища і надання доступу до нього через публічні методи.
- **Наслідування:** Можливість створення нових класів на основі існуючих, що дозволяє повторно використовувати код і створювати ієрархії класів.
- **Поліморфізм:** Здатність об'єктів різних класів реагувати на одні й ті самі виклики методів по-різному.

Архітектура додатку

Правильна архітектура є ключовим аспектом розробки настільних додатків. Найбільш поширеними архітектурними підходами є:

- **MVC (Model-View-Controller):** Архітектурний шаблон, який розділяє додаток на три основні компоненти: модель (дані та бізнес-логіка), вид (інтерфейс користувача) і контролер (управління взаємодією між моделлю та видом).
- **MVVM (Model-View-ViewModel):** Розширення шаблону MVC, де додано компонент ViewModel для підтримки двостороннього зв'язування даних між моделлю та видом.

Інтерфейс користувача (UI)

Зручний та інтуїтивно зрозумілий інтерфейс користувача є критичним для

успіху настільного додатку. Основні принципи розробки інтерфейсу користувача включають:

- **Користувацький досвід (UX):** Фокус на створенні зручного та приємного досвіду для користувачів.
- **Консистентність:** Використання однакових елементів управління та стилів у всьому додатку для забезпечення послідовності.
- **Простота:** Зменшення кількості дій, необхідних для виконання завдань, та спрощення інтерфейсу.
- **Доступність:** Забезпечення можливості використання додатку людьми з обмеженими можливостями.

Управління станом

Управління станом додатку є важливим аспектом, який забезпечує правильну роботу додатку та його взаємодію з користувачем. Основні принципи управління станом включають:

- **Збереження стану:** Забезпечення збереження стану додатку між сеансами роботи користувача.
- **Реактивність:** Відповідь додатку на зміни у стані даних або дії користувача в реальному часі.
- **Однозначність джерела істини:** Використання єдиного джерела даних для управління станом додатку.

Безпека

Безпека даних та захист додатку від потенційних загроз є критичними аспектами розробки. Основні принципи безпеки включають:

- **Аутентифікація та авторизація:** Забезпечення доступу до додатку та його функцій лише для авторизованих користувачів.
- **Шифрування:** Використання шифрування для захисту конфіденційних даних під час їх зберігання та передачі.
- **Виявлення та запобігання вразливостям:** Регулярне тестування додатку на наявність вразливостей та впровадження заходів для їх усунення.

Продуктивність

Висока продуктивність є важливою для забезпечення позитивного досвіду користувача. Основні принципи забезпечення продуктивності включають:

- **Оптимізація коду:** Використання ефективних алгоритмів та структур даних, оптимізація циклів та умов.
- **Мінімізація використання ресурсів:** Зменшення використання оперативної пам'яті, процесорного часу та інших ресурсів системи.
- **Асинхронність:** Використання асинхронних операцій для запобігання блокуванню головного потоку додатку.

Тестування

Тестування є важливою частиною розробки, що забезпечує високу якість програмного забезпечення. Основні види тестування включають:

- **Юніт-тестування:** Перевірка окремих модулів та функцій додатку на відповідність очікуваним результатам.
- **Інтеграційне тестування:** Перевірка взаємодії між різними модулями додатку.
- **Функціональне тестування:** Перевірка функціональних можливостей додатку на відповідність вимогам.
- **Регресійне тестування:** Перевірка додатку після внесення змін для забезпечення того, що нові зміни не порушують існуючий функціонал.

Дотримання цих принципів дозволяє створювати високоякісні настільні додатки, які забезпечують зручність використання, високу продуктивність, безпеку та надійність.

3.2 Особливості роботи з C# та .NET

Розробка настільних додатків з використанням мови програмування C# та платформи .NET має свої особливості, які забезпечують високу продуктивність, гнучкість та зручність у розробці програмного забезпечення. У цьому розділі розглянемо основні аспекти роботи з C# та .NET.

Мова програмування C#

C# (C-Sharp) – це мова програмування, розроблена компанією Microsoft як частина платформи .NET. Вона поєднує в собі простоту та елегантність мов високого рівня з потужністю та гнучкістю мов нижчого рівня.

Основні особливості C#:

- **Синтаксис:** Простий та зрозумілий синтаксис, який робить код читабельним та легким для підтримки.
- **Об'єктно-орієнтоване програмування:** Підтримка принципів ООП, таких як інкапсуляція, наслідування, поліморфізм та абстракція.
- **Автоматичне управління пам'яттю:** Використання збору сміття (Garbage Collection) для автоматичного звільнення непотрібної пам'яті.
- **Винятки:** Механізм обробки винятків для надійного управління помилками в коді.
- **Події та делегати:** Підтримка делегатів та подій для реалізації зворотних викликів та обробки подій.
- **Асинхронність:** Підтримка асинхронного програмування за допомогою ключових слів **async** та **await**.

Платформа .NET

.NET – це програмна платформа, розроблена компанією Microsoft, яка надає інструменти та бібліотеки для розробки різноманітних типів додатків, включаючи настільні, веб-додатки, мобільні додатки та хмарні сервіси.

Основні компоненти .NET:

- **Common Language Runtime (CLR):** Середовище виконання, яке забезпечує управління пам'яттю, безпекою, обробкою винятків та іншими основними послугами для виконання .NET додатків.
- **.NET Class Library:** Велика бібліотека класів, яка надає широкий набір функцій для роботи з файлами, базами даних, графічним інтерфейсом користувача, мережевими протоколами тощо.

Інструменти для розробки на C# та .NET

Visual Studio

Visual Studio – це інтегроване середовище розробки (IDE), розроблене

компанією Microsoft, яке забезпечує потужні інструменти для створення, редагування, налагодження та тестування програмного забезпечення на C# та .NET.

Основні можливості Visual Studio:

- **Редактор коду:** Підтримка синтаксичного підсвічування, автоматичного завершення коду, рефакторингу та інших функцій, що полегшують написання коду.
- **Налагодження:** Потужні інструменти для налагодження коду, включаючи точки зупинки, покрокове виконання, перегляд змінних та стеку викликів.
- **Інтеграція з системами контролю версій:** Підтримка Git, Team Foundation Server (TFS) та інших систем контролю версій.

Entity Framework

Entity Framework – це об'єктно-реляційний відображувач (ORM), який дозволяє розробникам працювати з базами даних, використовуючи об'єктно-орієнтований підхід.

Основні можливості Entity Framework:

- **Code First:** Підхід, при якому моделі даних визначаються за допомогою класів C#, а бази даних генеруються автоматично на основі цих моделей.
- **Database First:** Підхід, при якому моделі даних генеруються автоматично на основі існуючої бази даних.
- **LINQ (Language Integrated Query):** Мова запитів, яка дозволяє писати запити до бази даних за допомогою C#.

WPF (Windows Presentation Foundation)

WPF – це графічна підсистема для створення настільних додатків на Windows з використанням C#. Вона забезпечує потужні можливості для створення складних графічних інтерфейсів, підтримує анімацію, двовимірну та тривимірну графіку, стилі та шаблони.

Основні можливості WPF:

- **Data Binding:** Забезпечує прив'язку даних між інтерфейсом користувача та логікою програми.
- **Styles and Templates:** Дозволяє створювати консистентні та стильні

інтерфейси користувача.

- **Animations and Graphics:** Підтримка анімацій, двовимірної та тривимірної графіки для створення інтерактивних інтерфейсів.

MaterialDesignInXAML Toolkit та MahApps.Metro

Ці бібліотеки забезпечують підтримку сучасного дизайну для WPF-додатків, надаючи широкий набір готових компонентів, стилів та контролів.

Основні можливості:

- **Стилі та теми:** Забезпечують єдиний стиль для всього додатку відповідно до принципів сучасного дизайну.
- **Контроли:** Готові компоненти, такі як кнопки, текстові поля, меню та інші елементи інтерфейсу користувача.

Асинхронне програмування

Асинхронне програмування є важливою частиною розробки на C# та .NET, яке дозволяє створювати високопродуктивні додатки, що не блокують користувацький інтерфейс під час виконання тривалих операцій.

Основні концепції асинхронного програмування в C#:

- **Ключові слова async та await:** Використовуються для визначення асинхронних методів та обробки асинхронних операцій.
- **Task та Task<TResult>:** Класи, які представляють асинхронні операції та результати їх виконання.
- **Асинхронні методи:** Методи, які виконують тривалі операції, такі як звернення до бази даних або зчитування файлів, без блокування основного потоку.

Безпека

Безпека є критичним аспектом розробки на C# та .NET, особливо для додатків, які працюють з конфіденційними даними. Основні аспекти безпеки включають:

- **Аутентифікація та авторизація:** Забезпечення доступу до додатку та його функцій лише для авторизованих користувачів.
- **Шифрування:** Використання шифрування для захисту конфіденційних даних

під час їх зберігання та передачі.

- **Виявлення та запобігання вразливостям:** Регулярне тестування додатку на наявність вразливостей та впровадження заходів для їх усунення.

Розробка настільних додатків на C# та .NET надає розробникам потужні інструменти та технології для створення ефективних, продуктивних та безпечних програмних рішень. Володіння особливостями цієї платформи дозволяє створювати високоякісні додатки, які відповідають сучасним вимогам та потребам користувачів.

3.3 Інтеграція з базами даних

Інтеграція з базами даних є важливим аспектом розробки настільних додатків, оскільки вона забезпечує зберігання, доступ та обробку даних. При розробці на C# та .NET існує декілька підходів та інструментів для роботи з базами даних, які забезпечують високу продуктивність та зручність використання.

Підходи до інтеграції з базами даних

1. **ADO.NET**

ADO.NET – це набір компонентів, який забезпечує доступ до даних та управління ними. ADO.NET дозволяє працювати з різноманітними джерелами даних, включаючи реляційні бази даних, XML та інші.

Основні компоненти ADO.NET:

- **Connection:** Встановлення з'єднання з базою даних.
- **Command:** Виконання команд SQL для маніпуляції даними.
- **DataReader:** Читання даних з бази даних у потоковому режимі.
- **DataSet та DataTable:** Локальне зберігання даних у пам'яті.

ADO.NET забезпечує високу продуктивність та гнучкість у роботі з базами даних, дозволяючи розробникам контролювати процес взаємодії з даними на низькому рівні.

2. **Entity Framework**

Entity Framework (EF) – це ORM, який дозволяє розробникам працювати з

базами даних, використовуючи об'єктно-орієнтований підхід. EF забезпечує зручний спосіб роботи з даними без необхідності написання великої кількості SQL-коду.

Основні можливості Entity Framework:

- **Code First:** Створення бази даних на основі моделей даних, визначених у коді.
- **Database First:** Генерація моделей даних на основі існуючої бази даних.
- **Model First:** Створення моделей даних за допомогою візуальних дизайнерів, з подальшим генеруванням бази даних.
- **LINQ to Entities:** Використання LINQ для виконання запитів до бази даних.

Entity Framework автоматизує багато аспектів роботи з базами даних, таких як створення та управління схемами бази даних, зберігання та отримання даних, що значно спрощує розробку додатків.

3. **Dapper**

Dapper – це легковаговий ORM для .NET, який забезпечує простий та швидкий доступ до баз даних. Він поєднує в собі переваги ADO.NET та зручність ORM, забезпечуючи високу продуктивність та простоту використання.

Основні можливості Dapper:

- **Мапінг об'єктів:** Автоматичне перетворення результатів SQL-запитів у об'єкти C#.
- **Виконання SQL-запитів:** Легкість у написанні та виконанні SQL-запитів без великого накладного коду.
- **Підтримка різних типів баз даних:** Підтримка Microsoft SQL Server, MySQL, SQLite та інших баз даних.

Dapper є чудовим вибором для проєктів, де важлива висока продуктивність та мінімальний накладний код.

Вибір бази даних

Вибір бази даних для настільного додатку залежить від конкретних вимог проєкту. Основні варіанти включають:

- **Microsoft SQL Server:** Потужна реляційна база даних з широкими

можливостями для масштабування та забезпечення високої продуктивності. Підтримує великі обсяги даних та складні транзакції.

- **SQLite:** Легка реляційна база даних, яка зберігає дані у файлі та не потребує окремого серверного ПЗ, що робить її ідеальною для невеликих настільних додатків. Простота використання та висока швидкість доступу до даних.
- **MySQL:** Відкрита реляційна база даних з високою продуктивністю, яка часто використовується для веб-додатків, але також може бути застосована у настільних додатках. Підтримує великі обсяги даних та складні запити.

Приклади використання

1. **Code First підхід з Entity Framework**

```
public class ApplicationDbContext : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Transaction> Transactions { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer("YourConnectionStringHere");
    }
}

public class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Transaction> Transactions { get; set; }
}

public class Transaction
```

```
{
    public int Id { get; set; }
    public string Description { get; set; }
    public decimal Amount { get; set; }
    public int UserId { get; set; }
    public User User { get; set; }
}
```

2. **Використання Dapper для виконання запитів**

```
using (var connection = new SqlConnection("YourConnectionStringHere"))
{
    string sql = "SELECT * FROM Users WHERE Id = @Id";
    var user = connection.QueryFirstOrDefault<User>(sql, new { Id = 1 });
    Console.WriteLine(user.Name);
}
```

3. **Простий запит з використанням ADO.NET**

```
using (var connection = new SqlConnection("YourConnectionStringHere"))
{
    connection.Open();
    string sql = "SELECT * FROM Users WHERE Id = @Id";
    using (var command = new SqlCommand(sql, connection))
    {
        command.Parameters.AddWithValue("@Id", 1);
        using (var reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                Console.WriteLine(reader["Name"]);
            }
        }
    }
```

}

Інтеграція з базами даних є ключовим аспектом розробки настільних додатків на C# та .NET, оскільки вона забезпечує надійне та ефективне зберігання і доступ до даних. Використання різних підходів та інструментів дозволяє вибрати найбільш підходящий варіант для конкретного проєкту, забезпечуючи високу продуктивність та зручність роботи з даними.

3.4 Архітектура та дизайн додатків

Архітектура та дизайн додатків є ключовими аспектами розробки програмного забезпечення, які визначають його структуру, функціональні можливості та зручність використання. При розробці настільних додатків на C# та .NET важливо дотримуватися кращих практик архітектури та дизайну для забезпечення високої якості, масштабованості та підтримуваності додатку.

Архітектурні шаблони

1. **MVC (Model-View-Controller)**

MVC – це архітектурний шаблон, який розділяє додаток на три основні компоненти: модель, вид та контролер. Цей підхід допомагає розділити бізнес-логіку та інтерфейс користувача, що робить код більш організованим та легшим для підтримки.

- **Model:** Представляє дані та логіку додатку. Відповідає за управління даними, бізнес-правилами та логікою.
- **View:** Відповідає за відображення даних користувачеві. Це користувацький інтерфейс додатку.
- **Controller:** Обробляє користувацький ввід, взаємодіє з моделлю та оновлює вид. Він відповідає за керування потоком додатку.

2. **MVVM (Model-View-ViewModel)**

MVVM – це архітектурний шаблон, який є розширенням MVC, де додано компонент ViewModel для підтримки двостороннього зв'язування даних між моделлю та видом. Цей шаблон часто використовується в WPF додатках.

- **Model:** Представляє дані та логіку додатку.
- **View:** Відповідає за відображення даних користувачеві.
- **ViewModel:** Додає логіку відображення та підтримує двостороннє зв'язування даних між моделлю та видом.

3. **Clean Architecture**

Clean Architecture – це архітектурний підхід, який акцентує увагу на відокремленні бізнес-логіки від інфраструктурних та інтерфейсних шарів. Це забезпечує легку змінюваність та тестованість додатку.

Основні шари Clean Architecture:

- **Entities:** Містять бізнес-логіку та правила.
- **Use Cases:** Відповідають за конкретні дії користувача.
- **Interface Adapters:** Перетворюють дані для використання у різних середовищах.
- **Frameworks and Drivers:** Інфраструктурний шар, який включає бази даних, UI-фреймворки та інші технології.

Дизайн користувацького інтерфейсу (UI)

Зручний та інтуїтивно зрозумілий інтерфейс користувача є критичним для успіху настільного додатку. Основні принципи дизайну користувацького інтерфейсу включають:

1. **Консистентність**

Використання однакових елементів управління та стилів у всьому додатку забезпечує послідовність та знижує навчальну криву для користувачів.

2. **Простота**

Зменшення кількості дій, необхідних для виконання завдань, та спрощення інтерфейсу робить додаток більш доступним та зрозумілим.

3. **Доступність**

Забезпечення можливості використання додатку людьми з обмеженими можливостями, включаючи підтримку екранних читачів, високої контрастності та масштабування тексту.

4. **Зворотний зв'язок**

Надання зворотного зв'язку користувачам про виконання їхніх дій допомагає уникнути невизначеності та покращує загальний досвід використання додатку.

5. **Адаптивність**

Підтримка різних розмірів екранів та розширення додатку для різних платформ забезпечує зручність використання незалежно від пристрою.

Використання бібліотек та фреймворків

1. **WPF (Windows Presentation Foundation)**

WPF – це графічна підсистема для створення настільних додатків на Windows з використанням C#. Вона забезпечує потужні можливості для створення складних графічних інтерфейсів, підтримує анімацію, двовимірну та тривимірну графіку, стилі та шаблони.

2. **MaterialDesignInXAML Toolkit**

MaterialDesignInXAML Toolkit – це бібліотека, яка забезпечує підтримку Material Design у WPF додатках. Вона надає широкий набір готових компонентів, стилів та контролів, що відповідають принципам Material Design.

3. **MahApps.Metro**

MahApps.Metro – це бібліотека для створення стильних та сучасних інтерфейсів користувача в WPF додатках. Вона надає різноманітні стилі та контролі, що дозволяють легко створювати привабливі та функціональні інтерфейси.

Приклади архітектурних рішень

1. **Проектування MVC додатку на C#**

```
public class UserController
{
    private readonly IUserService _userService;

    public UserController(IUserService userService)
    {
        _userService = userService;
    }
```

```
    public IActionResult GetUser(int id)
    {
        var user = _userService.GetUserById(id);
        return View(user);
    }

    public IActionResult CreateUser(User user)
    {
        _userService.AddUser(user);
        return RedirectToAction("Index");
    }
}
```

2. **Використання MVVM у WPF додатку**

```
public class MainViewModel : INotifyPropertyChanged
{
    private string _title;
    public string Title
    {
        get { return _title; }
        set
        {
            _title = value;
            OnPropertyChanged(nameof(Title));
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
```

```
    {
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}
```

3. **Clean Architecture з використанням C#**

```
public class CreateUserUseCase : ICreateUserUseCase
{
    private readonly IUserRepository _userRepository;

    public CreateUserUseCase(IUserRepository userRepository)
    {
        _userRepository = userRepository;
    }

    public void Execute(User user)
    {
        if (string.IsNullOrEmpty(user.Name))
        {
            throw new ArgumentException("User name cannot be empty");
        }

        _userRepository.Add(user);
    }
}
```

Дотримання кращих практик архітектури та дизайну забезпечує створення високоякісних, масштабованих та підтримуваних настільних додатків на C# та .NET. Це дозволяє забезпечити зручність використання для кінцевих користувачів та легкість підтримки та розвитку для розробників.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис розробки програмного забезпечення

Розробка настільного додатку для відстеження фінансів Finance Manager App включає кілька основних етапів: аналіз вимог, проектування, реалізація, тестування та розгортання. У цьому розділі описуються кожен з цих етапів, а також використовувані технології та інструменти.

Аналіз вимог

Першим етапом розробки було зібрання вимог до додатку. Основні функціональні вимоги включали:

- Реєстрація та авторизація користувачів.
- Управління транзакціями (додавання, редагування та видалення).
- Створення та ведення бюджетів.
- Відображення фінансової статистики.
- Повідомлення користувачів про успішні операції та помилки.

Нефункціональні вимоги включали:

- Забезпечення безпеки даних користувачів.
- Інтуїтивно зрозумілий інтерфейс користувача.
- Висока продуктивність та надійність додатку.

Проектування

На етапі проектування була розроблена архітектура додатку, яка включає наступні основні компоненти:

- **Модель даних:** Визначає структуру даних та взаємодію з базою даних.
- **Бізнес-логіка:** Відповідає за обробку даних та реалізацію функціональних можливостей.
- **Інтерфейс користувача:** Відповідає за взаємодію з користувачем та відображення даних.

Архітектура додатку базується на шаблоні MVVM (Model-View-ViewModel), що забезпечує чітке розділення між логікою представлення та бізнес-логікою.

Реалізація

Етап реалізації включав написання коду для кожного з компонентів додатку. Основні технології та інструменти, що використовувалися під час розробки, включають:

- **Мова програмування C#:** Основна мова програмування для розробки додатку.
- **.NET Framework:** Платформа для створення настільних додатків.
- **Entity Framework:** ORM для роботи з базою даних.
- **WPF (Windows Presentation Foundation):** Технологія для створення інтерфейсу користувача.
- **MaterialDesignInXAML та MahApps.Metro:** Бібліотеки для стилізації інтерфейсу користувача.

Моделі даних

Для зберігання даних користувачів, транзакцій та бюджетів були створені відповідні моделі даних:

```
namespace FinanceManagerApp.Models
{
    using System;
    using System.ComponentModel.DataAnnotations;

    public partial class User
    {
        [Key]
        public int Id { get; set; }

        [StringLength(150)]
        public string FirstName { get; set; }

        [StringLength(150)]
        public string LastName { get; set; }
```

```
        [StringLength(150)]
        public string Email { get; set; }

        [StringLength(150)]
        public string Password { get; set; }

        public int? Age { get; set; }
        public string BudgetAmount { get; set; }
        public DateTime? DateRegister { get; set; }
    }
}

namespace FinanceManagerApp.Models
{
    using System;
    using System.ComponentModel.DataAnnotations;

    public partial class Transaction
    {
        [Key]
        public int Id { get; set; }

        [StringLength(50)]
        public string TranType { get; set; }

        [StringLength(50)]
        public string TranAmount { get; set; }

        [StringLength(50)]
```

```
        public string TranCategory { get; set; }

        public DateTime? TranDate { get; set; }

        public string TranDescription { get; set; }

        public int? UserId { get; set; }
    }
}

namespace FinanceManagerApp.Models
{
    using System;
    using System.ComponentModel.DataAnnotations;

    public partial class Budget
    {
        [Key]
        public int Id { get; set; }

        [StringLength(50)]
        public string BudgCategory { get; set; }

        [StringLength(50)]
        public string BudgMaxAmount { get; set; }

        public DateTime? BudgPeriodDate { get; set; }

        public int? UserId { get; set; }
    }
```

```
}
```

Взаємодія з базою даних

Для взаємодії з базою даних використовувався Entity Framework, який дозволяє працювати з даними через об'єктно-реляційний підхід. Був створений клас **DataContext**, який успадковується від **DbContext**:

```
namespace FinanceManagerApp.EF
{
    using Models;
    using System.Data.Entity;

    public partial class DataContext : DbContext
    {
        public DataContext(string connect) : base(connect)
        {
        }

        public virtual DbSet<BudgetExceedanceWarning> BudgetExceedanceWarnings { get; set; }
        public virtual DbSet<Transaction> Transactions { get; set; }
        public virtual DbSet<Budget> Budgets { get; set; }
        public virtual DbSet<User> Users { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
        }
    }
}
```

Бізнес-логіка

Бізнес-логіка реалізована у класі **FMLogic**, який включає методи для обробки транзакцій, управління бюджетами та іншими функціями додатку. Наприклад,

метод для оновлення бюджету користувача:

```
using FinanceManagerApp.Security;
using FinanceManagerApp.Windows;
using System.Data.SqlClient;
using System;

namespace FinanceManagerApp
{
    public class FMLogic
    {
        public static bool UpdateUserBudgetADO(int userId, string newBudget, bool isCost = false)
        {
            try
            {
                if (!isCost)
                {
                    if (int.Parse(MainWindow.CurrentUser.BudgetAmount) - int.Parse(newBudget) < 0)
                        new ErrorBox($"Вказана сумма є більшою за основний бюджет..", "Перевищення значення сумми").ShowDialog();
                    else
                        MainWindow.ChangedBudget =
                        (int.Parse(MainWindow.CurrentUser.BudgetAmount) - int.Parse(newBudget)).ToString();
                }
                else // Якщо витрати
                {
                    MainWindow.ChangedBudget =
                        (int.Parse(MainWindow.CurrentUser.BudgetAmount)
```

```
                    + int.Parse(newBudget)).ToString();
                }
                using (var connection = new SqlConnection(Credential.ConnStr))
                {
                    connection.Open();
                    string query = $"UPDATE Users SET BudgetAmount = @BudgetAmount WHERE Id = @Id";
                    using (var command = new SqlCommand(query, connection))
                    {
                        command.Parameters.AddWithValue("@BudgetAmount", MainWindow.ChangedBudget);
                        command.Parameters.AddWithValue("@Id", userId);
                        int rowsAffected = command.ExecuteNonQuery();
                        return rowsAffected > 0;
                    }
                }
            }
            catch (Exception ex)
            {
                new ErrorBox($"An error occurred: {ex.Message}", "UpdateUserBudgetADO issues..");
                return false;
            }
        }
    }
}
```

Інтерфейс користувача

Інтерфейс користувача розроблено з використанням WPF та XAML. Основні вікна додатку включають вікна авторизації, реєстрації, головне вікно, вікна для управління транзакціями та бюджетами, а також вікна для відображення

статистики.

```
<Application x:Class="FinanceManagerApp.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:FinanceManagerApp"

    StartupUri="Login.xaml">

    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="ModernButtonTheme.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

Тестування

На етапі тестування проводилось юніт-тестування, інтеграційне тестування та функціональне тестування додатку. Основною метою було забезпечення правильності роботи всіх функціональних можливостей та виявлення можливих помилок.

Розгортання

Після успішного завершення тестування додаток був підготовлений до розгортання. Для розгортання використовувались стандартні засоби .NET, які дозволяють створити інсталяційний пакет для Windows.

Настільний додаток Finance Manager App розроблений з використанням сучасних технологій та інструментів, що забезпечують високу продуктивність, надійність та зручність у використанні. Додаток дозволяє користувачам ефективно управляти своїми фінансами, планувати бюджети та аналізувати фінансову активність.

4.2. Реалізація функціональності для користувача

Настільний додаток Finance Manager App пропонує широкий спектр функціональних можливостей для користувачів, забезпечуючи зручність і ефективність управління фінансами. У цьому розділі детально розглянуто основні функції додатку, включаючи реєстрацію та авторизацію користувачів, управління транзакціями, ведення бюджетів та відображення статистики.

Реєстрація та авторизація користувачів

Процес реєстрації та авторизації користувачів реалізовано з використанням моделі **User**. Користувачі можуть зареєструвати новий акаунт, надаючи необхідні дані, такі як ім'я, прізвище, електронну пошту та пароль. Після реєстрації вони можуть увійти в систему за допомогою своїх облікових даних.

- **Реєстрація нового користувача:** Користувач заповнює форму реєстрації, після чого дані зберігаються у базі даних. Система перевіряє унікальність електронної пошти.
- **Авторизація:** Користувач вводить свої облікові дані, які перевіряються на відповідність записам у базі даних. У разі успішної авторизації користувач перенаправляється до головного вікна додатку.

Управління транзакціями

Додаток дозволяє користувачам додавати, редагувати та видаляти фінансові транзакції. Транзакції можуть бути доходами або витратами, що допомагає користувачам відстежувати свої фінансові операції.

- **Додавання транзакцій:** Користувач обирає тип транзакції (дохід або витрата), заповнює поля, такі як сума, категорія, дата та опис, і зберігає транзакцію.
- **Редагування транзакцій:** Користувач може змінити будь-які дані існуючої транзакції, що зберігається у базі даних.
- **Видалення транзакцій:** Користувач може видалити непотрібну транзакцію, що також оновлює дані у базі.

public class TransactionController

```
{
    private readonly DataContext _context;

    public TransactionController(DataContext context)
    {
        _context = context;
    }

    public void AddTransaction(Transaction transaction)
    {
        _context.Transactions.Add(transaction);
        _context.SaveChanges();
    }

    public void EditTransaction(Transaction transaction)
    {
        _context.Entry(transaction).State = EntityState.Modified;
        _context.SaveChanges();
    }

    public void DeleteTransaction(int id)
    {
        var transaction = _context.Transactions.Find(id);
        if (transaction != null)
        {
            _context.Transactions.Remove(transaction);
            _context.SaveChanges();
        }
    }
}
```

Ведення бюджетів

Користувачі можуть створювати та керувати бюджетами для різних категорій витрат. Це допомагає їм контролювати свої витрати та уникати перевищення встановлених лімітів.

- **Створення бюджету:** Користувач заповнює форму, вказуючи категорію бюджету, максимальну суму та період дії. Бюджет зберігається у базі даних.
- **Перегляд бюджетів:** Усі створені бюджети відображаються у відповідному вікні, де користувач може бачити актуальний стан кожного бюджету.
- **Редагування та видалення бюджетів:** Користувач може оновлювати або видаляти бюджети за потреби.

```
public class BudgetController
{
    private readonly DataContext _context;

    public BudgetController(DataContext context)
    {
        _context = context;
    }

    public void AddBudget(Budget budget)
    {
        _context.Budgets.Add(budget);
        _context.SaveChanges();
    }

    public void EditBudget(Budget budget)
    {
        _context.Entry(budget).State = EntityState.Modified;
        _context.SaveChanges();
    }
```

```
    public void DeleteBudget(int id)
    {
        var budget = _context.Budgets.Find(id);
        if (budget != null)
        {
            _context.Budgets.Remove(budget);
            _context.SaveChanges();
        }
    }
}
```

Відображення статистики

Finance Manager App надає користувачам можливість переглядати детальну фінансову статистику. Це включає графіки та діаграми, що показують доходи, витрати та залишки бюджету за певні періоди.

- **Графіки доходів та витрат:** Візуалізація даних допомагає користувачам зрозуміти свої фінансові звички та зробити відповідні корективи.
- **Звіти про бюджет:** Користувачі можуть переглядати, скільки вони витратили у порівнянні з встановленими бюджетами, та отримувати попередження про перевищення лімітів.

```
public class StatisticsController
{
    private readonly DataContext _context;

    public StatisticsController(DataContext context)
    {
        _context = context;
    }

    public IEnumerable<Statistic> GetStatistics(int userId)
```

```
        {
            var transactions = _context.Transactions.Where(t => t.UserId == userId).ToList();
            var statistics = transactions.Select(t => new Statistic
            {
                Type = t.TranType,
                Amount = t.TranAmount,
                Category = t.TranCategory,
                Date = t.TranDate.ToString(),
                Description = t.TranDescription
            });

            return statistics;
        }
    }
```

Повідомлення користувачів

Додаток інформує користувачів про успішні операції та можливі помилки через спеціальні вікна повідомлень.

- **Успішні операції:** Повідомлення про успішне додавання, редагування або видалення транзакцій та бюджетів.
- **Помилки:** Повідомлення про помилки, що виникають під час роботи з додатком, з поясненням причини та рекомендаціями щодо виправлення.

```
public class NotificationService
{
    public void ShowSuccessMessage(string message)
    {
        new Congratulate { Message = message }.ShowDialog();
    }

    public void ShowErrorMessage(string message)
```

```
    {
        new ErrorBox { Message = message }.ShowDialog();
    }
}
```

Настільний додаток Finance Manager App надає користувачам повний набір інструментів для ефективного управління фінансами. Користувачі можуть легко реєструватися та авторизовуватися, додавати та керувати транзакціями, створювати та вести бюджети, а також переглядати детальну статистику своєї фінансової діяльності. Усі ці функції реалізовані з урахуванням сучасних вимог до безпеки та зручності використання.

4.3. Інструкція для користувача

Finance Manager App – це настільний додаток, розроблений для управління особистими фінансами, який надає користувачам можливість відстежувати свої доходи, витрати, планувати бюджети та аналізувати фінансову активність. У цьому розділі буде розглянуто вибір технологій, використаних для розробки додатку, а також обґрунтування їхнього вибору.

Вибір технологій

Мова програмування C#

C# була обрана як основна мова програмування для розробки Finance Manager App з кількох причин:

1. **Синтаксис і структура:** C# має зручний і зрозумілий синтаксис, що спрощує розробку і підтримку коду.
2. **Інтеграція з .NET:** C# тісно інтегрована з .NET Framework, що забезпечує доступ до широкого спектру бібліотек і інструментів.
3. **Популярність і підтримка:** C# широко використовується в індустрії, має велику спільноту розробників і хорошу підтримку від Microsoft.

Платформа .NET

.NET Framework була обрана як платформа для розробки додатку через її

численні переваги:

1. **Широкі можливості:** .NET надає великий набір бібліотек і компонентів для розробки додатків різних типів, включаючи настільні, веб-додатки, сервіси та мобільні додатки.
2. **Зручність розробки:** Інструменти .NET, такі як Visual Studio, забезпечують зручність розробки, тестування і налагодження коду.
3. **Підтримка багатьох мов:** .NET підтримує різні мови програмування, що робить платформу гнучкою і зручною для розробників з різним досвідом.

Інструменти та бібліотеки

1. **WPF (Windows Presentation Foundation):**

WPF була обрана для створення інтерфейсу користувача завдяки своїм потужним можливостям:

- **Розширені можливості для створення інтерфейсу:** WPF підтримує двовимірну і тривимірну графіку, анімацію, стилі і шаблони.
- **Прив'язка даних:** WPF забезпечує зручні засоби для прив'язки даних між елементами інтерфейсу та моделями даних.
- **Гнучкість:** Можливість створення адаптивного та інтерактивного інтерфейсу.

2. **Entity Framework:**

Entity Framework (EF) була обрана як ORM (Object-Relational Mapping) для роботи з базою даних з кількох причин:

- **Простота використання:** EF спрощує взаємодію з базою даних, дозволяючи працювати з даними через об'єктно-орієнтований підхід.
- **Автоматизація:** EF автоматично генерує SQL-запити на основі C# моделей, що зменшує кількість коду і мінімізує помилки.
- **Підтримка LINQ:** EF дозволяє використовувати LINQ (Language Integrated Query) для виконання запитів до бази даних, що робить код більш читабельним і підтримуваним.

3. **MaterialDesignInXAML та MahApps.Metro:**

Ці бібліотеки були обрані для стилізації інтерфейсу користувача,

забезпечуючи сучасний і привабливий дизайн:

- **Material Design:** MaterialDesignInXAML забезпечує підтримку принципів Material Design, надаючи готові компоненти і стилі для створення привабливого інтерфейсу.
- **Metro UI:** MahApps.Metro надає стилі та контролі, що відповідають дизайну Metro UI, створюючи сучасний і зручний інтерфейс.

Обґрунтування вибору технологій

Забезпечення продуктивності та масштабованості

Використання C# та .NET забезпечує високу продуктивність додатку, завдяки оптимізації компілятора і ефективному управлінню пам'яттю. Ці технології також дозволяють легко масштабувати додаток для підтримки більшої кількості користувачів і збільшення обсягу даних.

Забезпечення безпеки

Безпека є ключовим аспектом розробки Finance Manager App. Використання .NET Framework забезпечує вбудовані механізми безпеки, такі як захист від SQL-ін'єкцій, шифрування даних і безпечне управління автентифікацією та авторизацією користувачів.

Зручність розробки та підтримки

Використання Visual Studio як основного інструменту для розробки забезпечує зручність написання коду, налагодження та тестування додатку. Інтеграція з Git дозволяє легко управляти версіями коду і співпрацювати з іншими розробниками.

Гнучкість та адаптивність інтерфейсу

WPF у поєднанні з бібліотеками MaterialDesignInXAML та MahApps.Metro дозволяє створювати інтерфейс, який легко адаптується під різні розміри екранів і забезпечує зручний користувацький досвід. Прив'язка даних у WPF спрощує управління даними в інтерфейсі, зменшуючи кількість коду та покращуючи підтримку.

Вибір технологій для розробки Finance Manager App був зроблений з урахуванням їхніх переваг у продуктивності, безпеці, зручності розробки та

підтримки, а також можливостей створення сучасного і привабливого інтерфейсу користувача. Використання C#, .NET Framework, WPF, Entity Framework та бібліотек MaterialDesignInXAML та MahApps.Metro забезпечує створення високоякісного додатку, який відповідає сучасним вимогам та потребам користувачів.

4.4. Інструкція для користувача

Цей розділ містить детальні інструкції для користувачів щодо встановлення, налаштування та використання настільного додатку Finance Manager App для управління фінансами. Інструкція охоплює всі основні функції програми, включаючи реєстрацію та авторизацію, управління транзакціями, ведення бюджетів та перегляд фінансової статистики.

Встановлення та налаштування додатку

1. **Встановлення додатку:**
    - **Скачайте та розпакуйте проект:** Завантажте інсталяційний пакет додатку з офіційного сайту або з надійного джерела та розпакуйте його у зручному місці на вашому комп'ютері.
    - **Запуск інсталяційного файлу:** Запустіть файл установки (наприклад, **setup.exe**) і слідуйте інструкціям інсталятора для завершення процесу установки.
2. **Налаштування бази даних:**
    - **Конфігурація підключення до бази даних:** Під час першого запуску додатку вам буде запропоновано вказати параметри підключення до бази даних (наприклад, ім'я сервера, назва бази даних, облікові дані для доступу).
    - **Ініціалізація бази даних:** Додаток автоматично створить необхідні таблиці та початкові дані у базі даних при першому підключенні.

Використання додатку

1. **Реєстрація та авторизація:**

- **Реєстрація нового користувача:**
  - Відкрийте вікно реєстрації, натиснувши кнопку "Реєстрація" на екрані входу.
  - Заповніть форму реєстрації, вказавши ім'я, прізвище, електронну пошту та пароль. (див. рис. 4.1)

Рисунок 4.1 – Форма реєстрації

  - Натисніть кнопку "Зареєструватися". Після успішної реєстрації ви будете перенаправлені на сторінку входу.
- **Вхід у систему:**
  - Відкрийте вікно входу, натиснувши кнопку "Увійти" на

головному екрані.

- Введіть свою електронну пошту та пароль для входу в систему. (див. рис. 4.2)

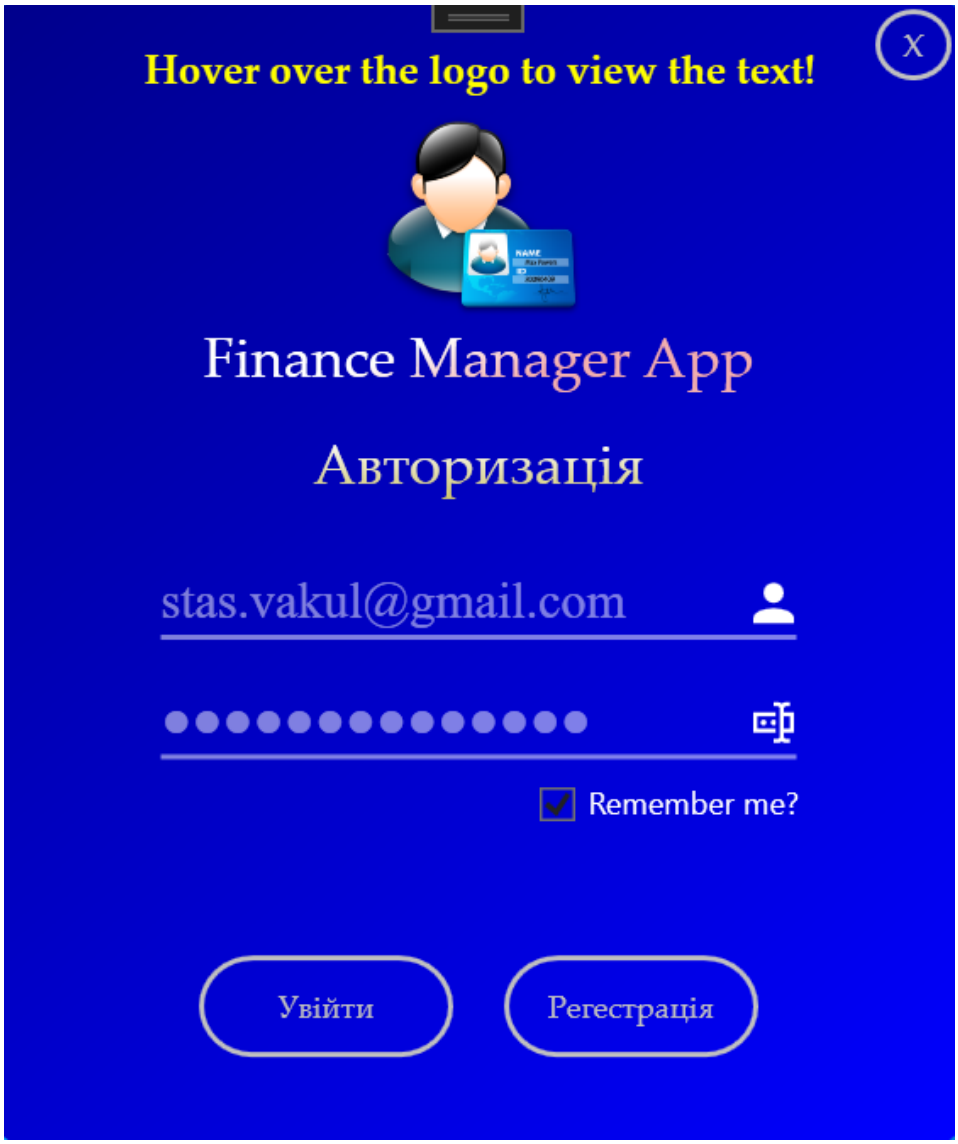

Рисунок 4.2 – Форма логіну

- Натисніть кнопку "Увійти". Після успішного входу ви будете перенаправлені на головну сторінку додатку. (див. рис. 4.3)

Рисунок 4.3 – Головна сторінка програми

- **Вихід з системи:**
  - Для виходу з системи натисніть кнопку "Вийти" у верхньому правому куті головного вікна. Після виходу ви будете перенаправлені на сторінку входу.

2. **Управління транзакціями:**
   - **Додавання транзакції:**
     - Натисніть кнопку "Додати транзакцію" у головному вікні або в меню навігації.
     - Заповніть форму додавання транзакції, вказавши тип (дохід або витрата), суму, категорію, дату та опис. (див. рис. 4.4)

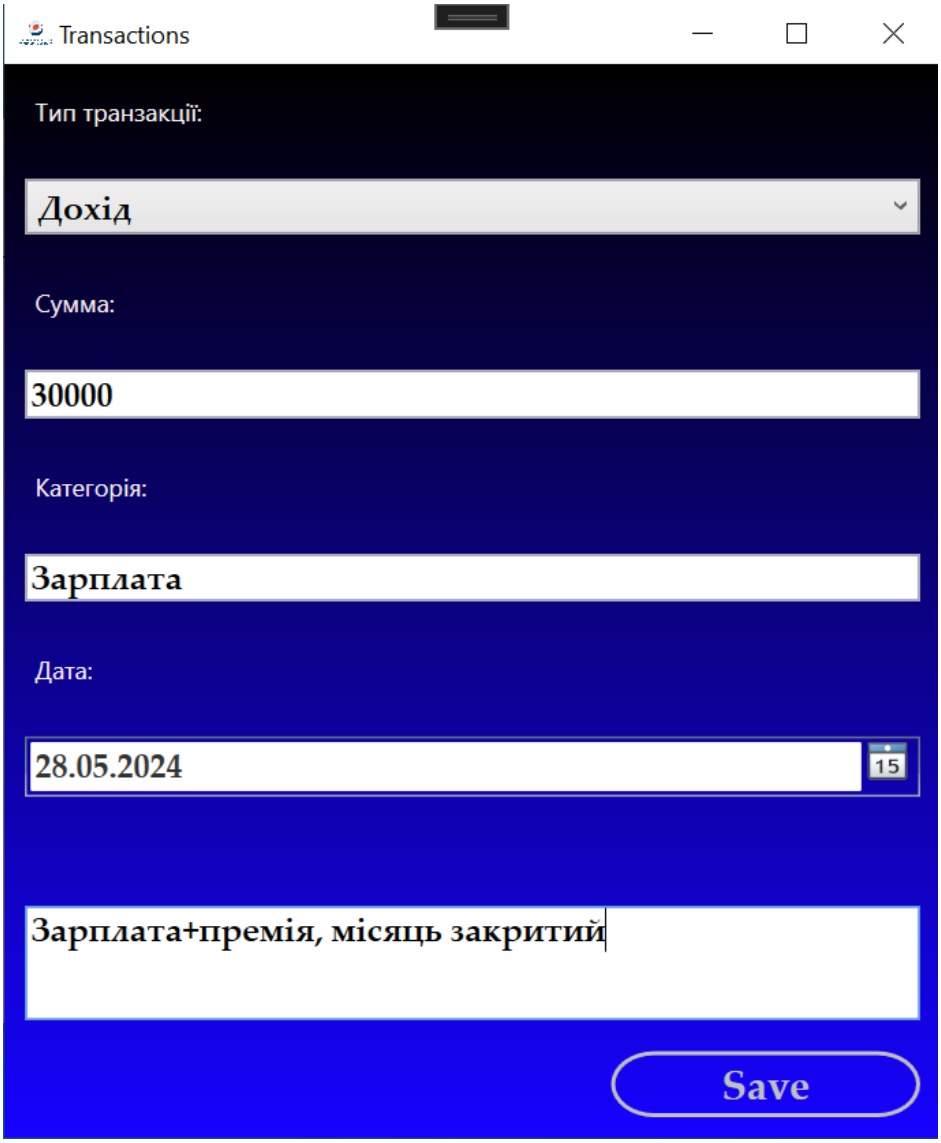

Рисунок 4.4 – Форма додавання транзакцій

- Натисніть кнопку "Додати". Після успішного додавання транзакції дані будуть збережені у базі даних.

- **Редагування транзакції:**
  - На сторінці статистики оберіть транзакцію, яку ви хочете редагувати.
  - Натисніть на місце, яке ви хочете змінити, та почніть змінювання. (див. рис. 4.5)

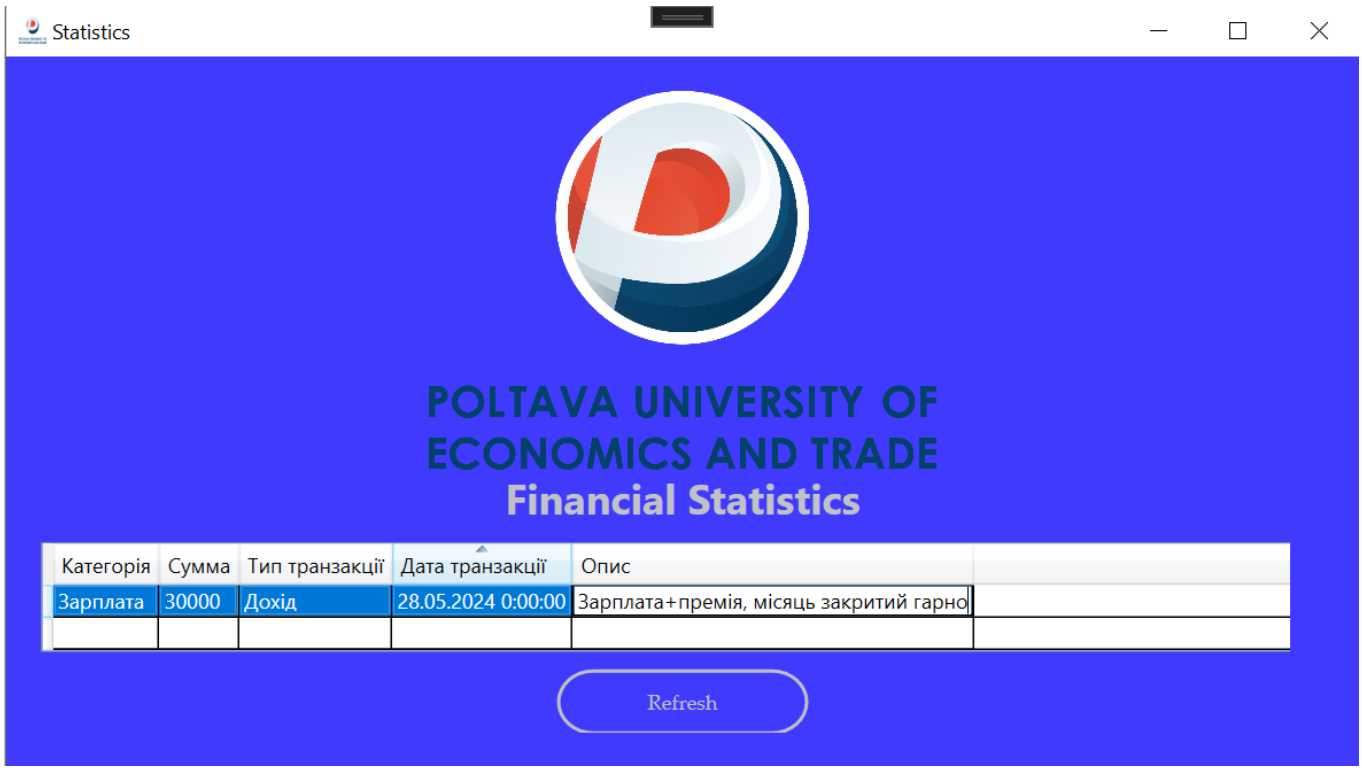

Рисунок 4.5 – Редагування транзакцій

3. **Ведення бюджетів:**
    - **Створення нового бюджету:**
        - Натисніть кнопку "Додати бюджет" у меню навігації або на сторінці управління бюджетами.
        - Заповніть форму створення бюджету, вказавши категорію, максимальну суму та період дії бюджету. (див. рис. 4.6)

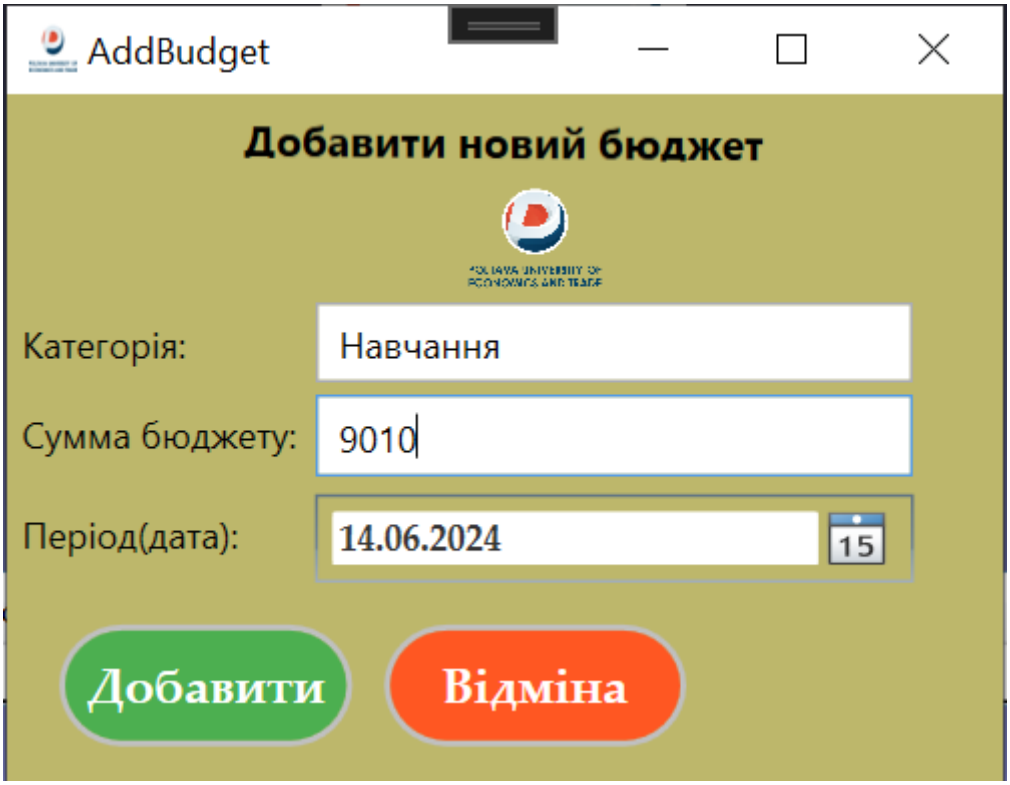

Рисунок 4.6 – Додавання бюджету

- Натисніть кнопку "Зберегти". Після успішного створення бюджет буде додано до бази даних.

- **Редагування бюджету:**
  - На сторінці управління бюджетами оберіть бюджет, який ви хочете редагувати.
  - Натисніть на місце, яке ви хочете змінити, та почніть змінювання

4. **Повідомлення користувачів:**
   - **Успішні операції:**
     - Після успішного виконання операції (наприклад, додавання або редагування транзакції) додаток відобразить повідомлення про успішне виконання дії. (див. рис. 4.7)

4.7 – Приклад повідомлення про успішну дію

- **Повідомлення про помилки:**
  - У разі виникнення помилки під час роботи з додатком відобразиться відповідне повідомлення з описом проблеми та можливими способами її вирішення. (див. рис. 4.8)

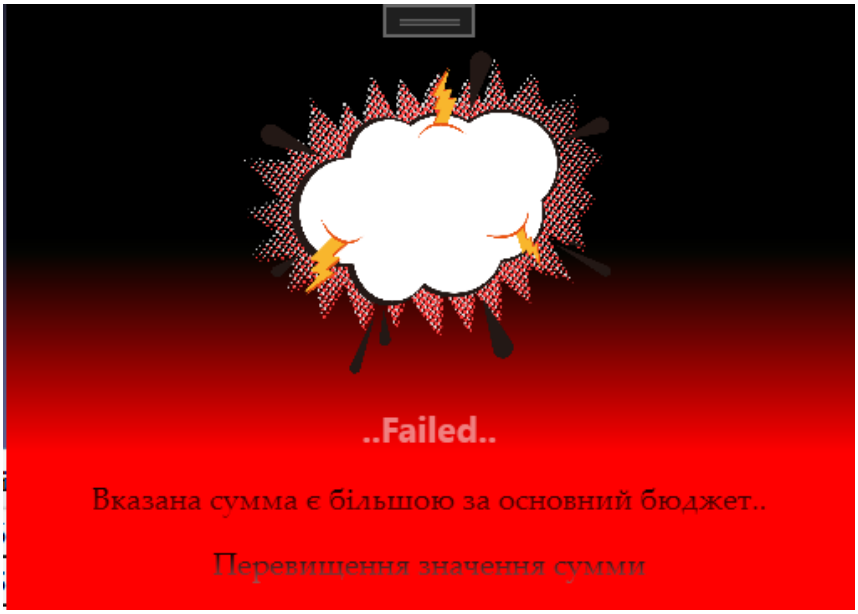

Рисунок 4.8 – Приклад повідомлення про помилку

Ця інструкція надає користувачам необхідні знання для встановлення, налаштування та використання настільного додатку Finance Manager App.

Дотримуючись вказаних кроків та рекомендацій, користувачі можуть ефективно керувати своїми фінансами, планувати бюджети та аналізувати фінансову активність.

ВИСНОВКИ

Під час виконання дипломної роботи був детально розглянутий теоретичний матеріал з теми «Розробка графічної програми для відстеження фінансів з використанням мови програмування C#». Було проведено огляд сучасних методів та інструментів розробки настільних додатків, а також розроблено настільний додаток Finance Manager App, який включає широкий спектр функцій для ефективного управління фінансами користувачів.

Результати роботи:

1. Проведено інформаційний огляд літературних джерел та робіт з розробки настільних додатків на платформі .NET та інших аналогічних платформах.
2. Виділено позитивні та негативні сторони різних методів розробки настільних додатків та управління фінансами.
3. Обґрунтовано вибір технологій для розробки додатку, включаючи мову програмування C#, платформу .NET, технології WPF для створення інтерфейсу користувача, а також Entity Framework для роботи з базою даних.
4. Розроблено настільний додаток для управління фінансами, який включає функції реєстрації та авторизації користувачів, додавання, редагування та видалення транзакцій, створення та управління бюджетами, а також відображення фінансової статистики.
5. Реалізовано та оптимізовано модулі для керування фінансами, що забезпечують ефективне зберігання та обробку даних у базі даних MSSQL Server.
6. Створено інтерфейс користувача, який забезпечує зручний спосіб взаємодії з додатком, включаючи можливість додавання, редагування, видалення транзакцій та управління бюджетами.
7. Проведено відладку та виправлення помилок у додатку, використовуючи інструменти налагодження та тестування Visual Studio.
8. Проведено тестування настільного додатку для перевірки коректності його роботи, зручності використання та стійкості проти потенційних загроз.

9. Розроблено інструктаж для користувачів щодо встановлення, налаштування та використання настільного додатку для управління фінансами.

Робота дозволила глибше зрозуміти принципи розробки настільних додатків на базі C# та .NET, а також забезпечення безпеки даних у таких додатках. Було розроблено практичний інструмент, який може бути використаний для ефективного управління фінансами користувачів, а також для навчальних та дослідницьких цілей у сфері розробки настільних додатків.

Finance Manager App надає користувачам можливість зручно та ефективно керувати своїми фінансами, що сприяє покращенню фінансової дисципліни та плануванню. Додаток може бути подальше розширений та адаптований відповідно до вимог конкретних користувачів або організацій, забезпечуючи додаткові можливості та інтеграції.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Michael Stoecklein. C# Programming Guide. Microsoft Docs. Доступно онлайн за адресою: https://docs.microsoft.com/en-us/dotnet/csharp/
2. Michael Crump. Introduction to WPF. Microsoft Docs. Доступно онлайн за адресою: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/
3. Julie Lerman. Programming Entity Framework: Code First. O'Reilly Media, 2011. 192 с.
4. Chris Sells, Ian Griffiths. Programming WPF. O'Reilly Media, 2007. 928 с.
5. Mark Michaelis. Essential C# 8.0. Addison-Wesley Professional, 2020. 1040 с.
6. Stephen Cleary. Concurrency in C# Cookbook. O'Reilly Media, 2019. 320 с.
7. Jesse Liberty, Donald Xie. Programming C# 5.0: Building Windows 8, Web, and Desktop Applications for the .NET 4.5 Framework. O'Reilly Media, 2012. 812 с.
8. Adam Freeman. Pro WPF in C# 2010: Windows Presentation Foundation in .NET 4. Apress, 2010. 1184 с.
9. Dino Esposito. Programming Microsoft ASP.NET MVC. Microsoft Press, 2014. 544 с.
10. Mahesh Chand. ADO.NET Programming. Apress, 2002. 816 с.
11. Andy Wigley, Daniel Moth, Peter Foot. Mobile Development with C#. Microsoft Press, 2012. 624 с.
12. Jon Skeet. C# in Depth. Manning Publications, 2019. 528 с.
13. Rod Stephens. Visual Basic Programming Guide. John Wiley & Sons, 2020. 960 с.
14. Eric Freeman, Elisabeth Robson. Head First Design Patterns: A Brain-Friendly Guide. O'Reilly Media, 2020. 694 с.
15. Ian Griffiths. Programming C# 8.0: Build Cloud, Web, and Desktop Applications. O'Reilly Media, 2019. 800 с.
16. Microsoft. Entity Framework Documentation. Доступно онлайн за адресою: https://docs.microsoft.com/en-us/ef/
17. Brian Lagunas. WPF MVVM In Depth. Pluralsight. Доступно онлайн за адресою: https://www.pluralsight.com/courses/wpf-mvvm-in-depth
18. Dmitri Nesteruk. Design Patterns in C#. Packt Publishing, 2018. 405 с.

19. Bill Wagner. Effective C#: 50 Specific Ways to Improve Your C#. Addison-Wesley Professional, 2017. 320 с.
20. John Sharp. Microsoft Visual C# Step by Step. Microsoft Press, 2018. 800 с.
21. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2022. – 67 с. – 1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

1. DataContext.cs

```
using System.Data.Entity;
using FinanceManagerApp.Models;

namespace FinanceManagerApp.EF
{
    public partial class DataContext : DbContext
    {
        public DataContext(string connectionString) : base(connectionString) { }

        public virtual DbSet<User> Users { get; set; }
        public virtual DbSet<Transaction> Transactions { get; set; }
        public virtual DbSet<Budget> Budgets { get; set; }
        public virtual DbSet<BudgetExceedanceWarning> BudgetExceedanceWarnings { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            // Add any model configurations here
        }
    }
}
```

2. User.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace FinanceManagerApp.Models
{
    public partial class User
    {
        [Key]
        public int Id { get; set; }

        [StringLength(150)]
        public string FirstName { get; set; }

        [StringLength(150)]
        public string LastName { get; set; }

        [StringLength(150)]
        public string Email { get; set; }

        [StringLength(150)]
        public string Password { get; set; }

        public int? Age { get; set; }
```

```
        public string BudgetAmount { get; set; }

        public DateTime? DateRegister { get; set; }
    }
}
```

3. Transaction.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace FinanceManagerApp.Models
{
    public partial class Transaction
    {
        [Key]
        public int Id { get; set; }

        [StringLength(50)]
        public string TranType { get; set; }

        [StringLength(50)]
        public string TranAmount { get; set; }

        [StringLength(50)]
        public string TranCategory { get; set; }

        public DateTime? TranDate { get; set; }

        public string TranDescription { get; set; }

        public int? UserId { get; set; }
    }
}
```

4. Budget.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace FinanceManagerApp.Models
{
    public partial class Budget
    {
        [Key]
        public int Id { get; set; }

        [StringLength(50)]
        public string BudgCategory { get; set; }

        [StringLength(50)]
        public string BudgMaxAmount { get; set; }
```

```
        public DateTime? BudgPeriodDate { get; set; }

        public int? UserId { get; set; }
    }
}
```

5. BudgetExceedanceWarning.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace FinanceManagerApp.Models
{
    public partial class BudgetExceedanceWarning
    {
        [Key]
        public int Id { get; set; }

        public int? BudgetId { get; set; }

        [StringLength(50)]
        public string AmountExcess { get; set; }

        public DateTime? WarningDate { get; set; }
    }
}
```

6. FMLogic.cs

```
using FinanceManagerApp.Security;
using FinanceManagerApp.Windows;
using System.Data.SqlClient;
using System;

namespace FinanceManagerApp
{
    public class FMLogic
    {
        public static bool UpdateUserBudgetADO(int userId, string newBudget, bool isCost = false)
        {
            try
            {
                if (!isCost)
                {
                    if (int.Parse(MainWindow.CurrentUser.BudgetAmount) - int.Parse(newBudget) < 0)
                        new ErrorBox($"Вказана сума є більшою за основний бюджет..", "Перевищення значення
суми").ShowDialog();
                    else
                        MainWindow.ChangedBudget =
                        (int.Parse(MainWindow.CurrentUser.BudgetAmount) - int.Parse(newBudget)).ToString();
                }
                else // Якщо витрати
                {
```

```
                MainWindow.ChangedBudget =
                    (int.Parse(MainWindow.CurrentUser.BudgetAmount)
                    + int.Parse(newBudget)).ToString();
            }
            using (var connection = new SqlConnection(Credential.ConnStr))
            {
                connection.Open();
                string query = $"UPDATE Users SET BudgetAmount = @BudgetAmount WHERE Id = @Id";
                using (var command = new SqlCommand(query, connection))
                {
                    command.Parameters.AddWithValue("@BudgetAmount", MainWindow.ChangedBudget);
                    command.Parameters.AddWithValue("@Id", userId);
                    int rowsAffected = command.ExecuteNonQuery();
                    return rowsAffected > 0;
                }
            }
        }
        catch (Exception ex)
        {
            new ErrorBox($"An error occurred: {ex.Message}", "UpdateUserBudgetADO issues..");
            return false;
        }
    }
}
```

7. MainWindow.xaml

```
<Window x:Class="FinanceManagerApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Finance Manager" Height="450" Width="800">
    <Grid>
        <TextBlock Text="Вітаємо у Finance Manager App" FontSize="24" HorizontalAlignment="Center"
VerticalAlignment="Top" Margin="0,20,0,0"/>
        <Button Content="Додати транзакцію" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="150" Height="50" Margin="10,80,0,0" Click="AddTransaction_Click"/>
        <Button Content="Переглянути бюджети" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="150" Height="50" Margin="10,140,0,0" Click="ViewBudgets_Click"/>
        <Button Content="Статистика" HorizontalAlignment="Left" VerticalAlignment="Top" Width="150"
Height="50" Margin="10,200,0,0" Click="ViewStatistics_Click"/>
    </Grid>
</Window>
```

8. MainWindow.xaml.cs

```
using System.Windows;

namespace FinanceManagerApp
{
    public partial class MainWindow : Window
    {
        public static User CurrentUser { get; set; }
```

```
        public static string ChangedBudget { get; set; }

        public MainWindow()
        {
            InitializeComponent();
        }

        private void AddTransaction_Click(object sender, RoutedEventArgs e)
        {
            // Відкрити вікно для додавання транзакції
        }

        private void ViewBudgets_Click(object sender, RoutedEventArgs e)
        {
            // Відкрити вікно для перегляду бюджетів
        }

        private void ViewStatistics_Click(object sender, RoutedEventArgs e)
        {
            // Відкрити вікно для перегляду статистики
        }
    }
}
```

9. Congratulate.xaml

```
<Window x:Class="FinanceManagerApp.Windows.Congratulate"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Успішна операція" Height="200" Width="400">
    <Grid>
        <TextBlock Name="Message" Text="Операція виконана успішно!" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="18"/>
    </Grid>
</Window>
```

10. Congratulate.xaml.cs

```
using System.Windows;

namespace FinanceManagerApp.Windows
{
    public partial class Congratulate : Window
    {
        public Congratulate()
        {
            InitializeComponent();
        }

        public string Message
        {
            get => Message.Text;
            set => Message.Text = value;
```

```
        }
    }
}
```

11. ErrorBox.xaml

```
<Window x:Class="FinanceManagerApp.Windows.ErrorBox"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Помилка" Height="200" Width="400">
    <Grid>
        <TextBlock Name="ErrorMessage" Text="Сталася помилка!" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="18"/>
    </Grid>
</Window>
```

12. ErrorBox.xaml.cs

```
using System.Windows;

namespace FinanceManagerApp.Windows
{
    public partial class ErrorBox : Window
    {
        public ErrorBox()
        {
            InitializeComponent();
        }

        public string Message
        {
            get => ErrorMessage.Text;
            set => ErrorMessage.Text = value;
        }
    }
}
```