Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Бутка Владислава Олександровича*
(ПІБ)

академічної групи *122М-22-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка програми для управління особистими фінансами на платформі JavaFX з використанням аналітичних інструментів.*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
розділів кваліфікаційної роботи				
спеціальний	*Доц. Кабак Л. В.*			
Рецензент	*Доц. Шедловський І.А.*			
Нормоконтролер	*Проф. Лактіонов І.С.*			

Дніпро
2023

НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *122 Комп'ютерні науки*
студента *122м-22-3* *Бутка Владислава Олександровича*
(група) (прізвище та ініціали)
тема дипломного проекту *Розробка програми для управління особистими фінансами на платформі JavaFX з використанням аналітичних інструментів*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 09.10.2023 р. № 1227 –с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Актуальність роботи обумовлена низьким рівнем фінансової грамотності користувачів.

Об'єктом дослідження – є процес інформаційної підтримки управління особистими фінансами.

Метою роботи – є розробка програмного забезпечення на основі JavaFX для ефективного управління фінансами та аналізу фінансового стану, сприяючи підвищенню фінансової грамотності користувачів.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна полягає в реалізації вдосконалених технологій управління фінансами, що включають сучасні методи аналізу та опрацювання фінансової інформації.

Практична цінність результатів виявляється у можливості ефективного мінімізування часу на управління особистими фінансами та спрощенні процесу відбору нових інвестицій та операцій для максимізації фінансового благополуччя.

4 ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Результати досліджень мають бути подані у вигляді, що дозволяє побачити та оцінити безпосереднє використання програмного додатку управління особистими фінансами.

5 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	12.09.2023-30.10.2023
Дослідження існуючих волонтерських застосунків та бібліотек програмування подібних систем	01.11.2023-24.11.2023
Створення бекенд застосунку для автоматизації та полегшення взаємодії у сфері волонтерства	25.11.2023-14.12.2023

6 РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект від реалізації результатів роботи очікується позитивним завдяки можливості ефективно управляти фінансами ,аналізувати витрати та доходи, ставити фінансові цілі та автоматизувати багато інших фінансових операцій.

Соціальний ефект від реалізації результатів роботи очікується позитивним, завдяки підвищенню фінансової грамотності та ефективним обліком грошей.

Завдання видав _____ *Доц. Кабак Л.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Буток В.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: 12.09.2023р.

Термін подання кваліфікаційної роботи до ЕК: 16.12.2023р.

РЕФЕРАТ

Пояснювальна записка: 103 с., 27 рис., 11 табл., 2 дод., 29 джерел.

Об'єкт дослідження: процес керування власними фінансами користувача, а також відбір та впровадження нових елементів у систему обліку особистих фінансів.

Предметом дослідження є функціональність та ефективність використання розробленого програмного продукту для управління фінансами.

Актуальність дослідження полягає в тому, що фінансова грамотність користувачів є недостатньою і існує високий попит на інструменти для ефективного управління особистими фінансами.

Метою даного дослідження є розробка та дослідження системи управління особистими фінансами з метою підвищення ефективності фінансового планування та контролю всередині користувача. Мета також полягає у створенні спрощеної та зручної моделі відбору нових кадрів для впровадження у власний фінансовий план.

Для досягнення мети проекту необхідно вирішити наступні завдання:

1. Провести аналіз обраної предметної області, визначивши основні аспекти та вимоги до управління фінансами.
2. Порівняти програмні продукти-аналоги, враховуючи їхні можливості та переваги. Обрати оптимальні технології і середовище розробки для ефективної реалізації проекту.
3. Здійснити проектування програмного продукту для управління фінансами, визначивши ключові функціональності та архітектурні рішення.
4. Розробити програмний продукт, використовуючи обрані технології та середовище розробки. Забезпечити його коректну роботу та ефективність.

Практичне значенням є надання можливості для користувачів контролювати та аналізувати свої особисті фінанси.

ABSTRACT

Explanatory note: 104 pp., 27 figures, 11 tables, 2 appendices, 29 sources.

The object of the study: the process of managing the user's own finances, as well as the selection and introduction of new elements into the accounting system of personal finances.

The subject of the study is the functionality and efficiency of using the developed software product for financial management.

The relevance of the research is that the financial literacy of users is insufficient and there is a high demand for tools for effective management of personal finances.

The purpose of this study is the development and research of a personal finance management system in order to improve the effectiveness of financial planning and control within the user. The goal is also to create a simplified and convenient model for the selection of new personnel for implementation in one's own financial plan.

To achieve the goal of the project, the following tasks must be solved:

1. Conduct an analysis of the selected subject area, determining the main aspects and requirements for financial management.
2. Compare similar software products, taking into account their capabilities and advantages. Choose optimal technologies and development environment for effective implementation of the project.
3. Design a software product for financial management, defining key functionalities and architectural solutions.
4. Develop a software product using the selected technologies and development environment. To ensure its correct operation and efficiency.

The practical meaning is to enable users to monitor and analyze their personal finances.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ООП – об`єктно-орієнтоване програмування.

БД – база даних.

FXML – файловий формат для опису інтерфейсу користувача у JavaFX.

IDE – інтегроване середовище розробки.

UI (User Interface) – інтерфейс користувача.

GUI – Графічний інтерфейс користувача (Graphical User Interface).

СПИСОК КЛЮЧОВИХ СЛІВ

ТРАНЗАКЦІЯ, ПЛАГІН, ГРАФІК, PROJECT BUILDER, MAVEN, JAVA, JAVAFX, INTELLIJ IDEA, SCENE BUILDER, АЛГОРИТМ, ФОРМА, СХЕМА.

LIST OF CONDITIONAL DESIGNATIONS

OOP – is object-oriented programming.

DB – is a database.

FXML – is a file format for describing the user interface in JavaFX.

IDE – is an integrated development environment.

UI (User Interface) – user interface.

GUI – Graphical User Interface (Graphical User Interface).

LIST OF KEY WORDS

TRANSACTION, PLUGIN, CHART, PROJECT BUILDER, MAVEN, JAVA, JAVAFX, INTELLIJ IDEA, SCENE BUILDER, ALGORITHM, FORM, SCHEMA.

ВСТУП

Фінанси відіграють важливу роль в повсякденному житті кожної людини та родини. Вони безпосередньо впливають на якість життя, можливість отримання освіти, задоволення потреб, і, що найважливіше, здатність до досягнення поставлених фінансових цілей. У сучасному швидкозмінному світі, де економічні умови постійно коливаються, а можливості для інвестування та збереження грошей різноманітні, контроль над особистими фінансами стає надзвичайно важливим завданням.

Забезпечити фінансову стабільність та здатність до фінансового зростання може бути викликом, і основним інструментом для досягнення цих цілей є ефективне управління особистими фінансами. Особисті фінанси включають в себе багато аспектів, такі як витрати, доходи, інвестиції, планування бюджету, управління боргами та багато інших.

У цьому контексті, розробка програмного забезпечення для управління особистими фінансами стає важливим завданням. Сучасні технології надають унікальні можливості для створення зручних та ефективних інструментів для відстеження та аналізу особистих фінансів. Така програма може допомогти людям краще розуміти їхні фінанси, планувати витрати, робити інвестиційні рішення та досягати фінансових цілей.

У даному дипломному проекті розглядається розробка програмного забезпечення для управління особистими фінансами на платформі JavaFX, з додаванням аналітичних інструментів для забезпечення користувачів засобами для кращого контролю та планування своїх фінансів. В цьому проекті досліджується і реалізується інтерфейс користувача, можливості введення та відображення фінансових даних, а також інструменти для аналізу та прогнозування фінансових потоків.

Метою даного проекту є забезпечення користувачів потужним інструментарієм для ефективного управління особистими фінансами та підвищення фінансової грамотності. Програма буде допомагати відстежувати

витрати, розробляти бюджет, аналізувати інвестиційні можливості і приймати обґрунтовані фінансові рішення.

Завдяки розробці цієї програми, користувачі матимуть зручний і доступний інструмент для ефективного управління своїми фінансами, що сприятиме досягненню фінансових цілей та забезпеченню фінансового благополуччя.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел і трьох додатків.

У вступі розглядається аналіз та сучасний стан проблеми, наведено опис структури роботи.

У подальших розділах дипломного проекту будуть проведені аналізі щодо предметної області, конкретизується мета кваліфікаційної роботи, наведено обґрунтування актуальності теми та розроблена постановка завдання, також розглянуті деталі розробки програми, обґрунтування вибору JavaFX та аналітичних інструментів, а також методи та технічні аспекти реалізації програми.

РОЗДІЛ 1
АНАЛІЗ ТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Задача дослідження предметної області

Метою даного дипломного проекту є розробка програмного забезпечення для управління особистими фінансами на платформі JavaFX з використанням аналітичних інструментів. Головною метою цієї розробки є надання користувачам засобів для ефективного контролю над своїми фінансами, а також можливості проводити аналіз фінансових операцій та робити інформовані рішення щодо управління своїми фінансами.

У сучасному світі управління особистими фінансами є критично важливим аспектом досягнення фінансової стабільності та благополуччя для кожної особи та родини. Існує безліч факторів, що впливають на фінансову стабільність та благополуччя, такі як доходи, витрати, інвестиції, податки та борги. Кожен із нас стикається з фінансовими рішеннями щоденно, починаючи від планування бюджету і закінчуючи вибором інвестиційних можливостей.

Для кращого розуміння та оптимізації особистих фінансів необхідно мати засоби для аналізу та контролю. Загальний баланс між доходами і витратами буде однією з ключових функцій програми. Це допоможе користувачам зрозуміти, чи вони живуть в межах своїх можливостей, чи потрібно щось змінювати в своєму фінансовому плануванні.

Враховуючи цей баланс, користувачі зможуть приймати більш обґрунтовані рішення щодо свого бюджету та інвестицій.

Досягнення фінансового балансу можливе завдяки зрозумінню, скільки грошей ми заробляємо і на що їх витрачаємо.

Для цього використовуються формули для аналізу особистих фінансів, як бюджету, так і прибутку з витратами:

1. «Загальний дохід (Total Income)»:

$$\text{Загальні дохід} = \text{Зарплата} + \text{Премії} + \text{Депозит} + \text{Інші джерела доходу}, \quad (1.1)$$

де

Загальний дохід - який включає в себе зарплату, премії, доходи від депозитів та інші джерела доходу (1.1).

2. «Загальні витрати (Total Expenses)»:

$$\text{Загальні витрати} = \text{Житло} + \text{Їжа} + \text{Транспорт} + \text{Розваги} + \text{Інші витрати}, \quad (1.2)$$

де

Загальні витрати - які включають в себе витрати на житло, їжу, транспорт, розваги та інші види витрат (1.2).

3. «Загальний бюджет (Total Budget)»:

$$\text{Загальний бюджет} = \frac{\text{Загальний дохід}}{\text{Загальні витрати}}, \quad (1.3)$$

де

Загальний бюджет - який визначає різницю між загальним доходом і загальними витратами (1.3).

4. «Показник рентабельності інвестицій (ROI)»:

$$ROI = \frac{\text{Прибуток від інвестицій}}{\text{Вартість інвестицій}} \times 100\%, \quad (1.4)$$

де

ROI - показник рентабельності інвестицій.

Ця формула використовується для розрахунку покажчика рентабельності

інвестицій (1.4) і вказує на те, наскільки вигідні ваші інвестиції.

5. «Показник рентабельності капіталу (ROE)»:

$$ROE = \frac{\text{Чистий прибуток}}{\text{Власний капітал}} \times 100\%, \quad (1.5)$$

де

ROE - Показник рентабельності капіталу.

Ця формула розраховує показник рентабельності капіталу (1.5) і показує, який прибуток генерується на одиницю власного капіталу.

6. «Показник рентабельності активів (ROA)»:

$$ROA = \frac{\text{Чистий прибуток}}{\text{Загальні активи}} \times 100\%, \quad (1.6)$$

де

ROA - Показник рентабельності активів.

Ця формула розраховує показник рентабельності активів (1.6) і вказує, наскільки прибутковими є ваші активи.

7. «Показник Debt-to-Income (DTI)»:

$$DTI = \frac{\text{Загальні зобов′язання}}{\text{Загальний дохід}} \times 100\%, \quad (1.7)$$

де

DTI - Показник ідношення зобов'язань до доходу.

Ця формула розраховує показник Debt-to-Income (відношення зобов'язань до доходу) і допомагає визначити, скільки ваших доходів йде на погашення зобов'язань (1.7).

8. «Показник Current Ratio»:

$$Current\ Ratio\ = \frac{\text{Поточні активи}}{\text{Поточні зобов′язання}}, \tag{1.8}$$

де

Current Ratio - Показник поточного співвідношення.

Ця формула розраховує показник Current Ratio, який вказує на здатність виконати поточні зобов'язання за допомогою поточних активів (1.8).

9. «Показник Quick Ratio»:

$$Quick\ Ratio\ = \frac{\text{Поточні активи}\ - \text{Запаси}}{\text{Поточні зобов′язання}}, \tag{1.9}$$

де

Quick Ratio - Показник швидкого співвідношення.

Ця формула розраховує показник Quick Ratio, який вказує на здатність виконати поточні зобов'язання за допомогою поточних активів, за винятком запасів (1.9).

Ці формули допомагають краще розуміти структуру фінансів та доходи, а також розходи, що є важливим для ефективного управління фінансами.

Ця інформація важлива для кожної особи та родини, оскільки управління особистими фінансами є ключовим аспектом досягнення фінансової стабільності та благополуччя. Розуміння того, на що йдуть гроші, і як оптимізувати свої фінанси, допомагає уникнути непередбачених фінансових труднощів та досягти фінансових цілей.

Створення і використання програмного забезпечення для управління фінансами також відображає сучасні тенденції в сфері технологій та використання комп'ютерів для вирішення повсякденних завдань. Ця ініціатива сприяє розвитку нових технологій у фінансовому секторі та допомагає користувачам зробити свої фінанси більш доступними і керованими.

1.2. Огляд методів аналізу фінансів

Розробка програми для управління особистими фінансами з використанням аналітичних інструментів є складним завданням, і вибір методу аналізу фінансів є ключовим аспектом успішної реалізації цього проекту. Наведемо приклад деяких методів аналізу фінансів та обґрунтуємо вибір конкретного методу:

1. «Метод аналізу фінансового балансу»: Цей метод базується на вивченні фінансового балансу, де порівнюються активи (всі ресурси та власність) та пасиви (зобов'язання та капітал). Він допомагає визначити фінансову стабільність та здатність погасити зобов'язання [1].

2. «Метод аналізу доходів та витрат»: Цей метод зосереджений на порівнянні доходів та витрат. Він допомагає визначити, чи заробляєте ви більше, ніж витрачаєте, і визначити основні джерела доходу та сфери споживання з найбільшими витратами [2].

3. «Метод аналізу інвестицій»: Якщо у вас є інвестиції, цей метод допомагає визначити прибутковість та ризики інвестицій. Ви можете використовувати різні метрики, такі як ROI (покажчик рентабельності інвестицій), для оцінки ефективності інвестицій [3].

4. «Метод аналізу кредитів та боргів»: Якщо у вас є кредити або борги, цей метод допомагає оцінити вашу здатність погасити їх. Ви можете використовувати показники, такі як Debt-to-Income Ratio, для оцінки своєї фінансової стійкості [4].

5. «Метод аналізу ризиків»: Цей метод допомагає визначити ризики, пов'язані з вашими фінансами. SWOT-аналіз (аналіз сильних і слабких сторін, можливостей та загроз) використовується для оцінки ризиків та можливостей [5].

6. «Метод аналізу портфеля»: Цей метод важливий для тих, хто має інвестиції в різні активи. Він допомагає визначити оптимальний розподіл активів у портфелі для досягнення фінансових цілей [6].

Для досягнення успішної розробки програмного продукту для управління

особистими фінансами важливим є вибір належного методу аналізу фінансів. У розгляді вище наведеного переліку методів, метод аналізу фінансового балансу визначається як оптимальний для даного проекту. Він надає можливість ефективно контролювати активи та пасиви, визначати ризики та можливості, що є критичним для досягнення фінансової стабільності та планування майбутніх інвестицій, а також допоможе визначити ризики та можливості. Це важливо для досягнення фінансових цілей та планування майбутніх інвестицій.

1.2.1. Обґрунтування вибору методу аналізу

Метод аналізу фінансового балансу обрано для програми управління особистими фінансами з кількох ключових причин:

1. Комплексний аналіз:

Метод надає можливість провести комплексний аналіз фінансового стану, включаючи всі активи та зобов'язання. Це допомагає користувачам отримати повний обзор своїх фінансів, а не обмежуватися лише доходами та витратами.

2. Визначення фінансової стабільності:

Метод дозволяє точно визначити, наскільки фінанси стабільні і здатні покрити зобов'язання. Це важливо для уникнення фінансових проблем у майбутньому.

3. Виявлення ризиків і можливостей:

Аналіз фінансового балансу допомагає виявити потенційні ризики і можливості, що дозволяє користувачам приймати обґрунтовані рішення щодо своїх фінансів.

4. Планування інвестицій:

Відомості, які можна отримати за допомогою аналізу фінансового балансу, допомагають користувачам планувати інвестиції та визначати, які частини їхнього портфелю вимагають більшої уваги.

Основні формули методу аналізу фінансового балансу:

1. «Чистий фінансовий результат (Net Financial Result)»:

$$\text{Net Financial Result} = \text{Активи} - \text{Пасиви}, \quad (1.10)$$

де

Net Financial Result - чистий фінансовий результат.

Чистий фінансовий результат (1.10) визначає різницю між активами і пасивами. Цей показник вказує на загальну фінансову позицію суб'єкта. Формула допомагає визначити, чи активи перевищують зобов'язання (позитивний результат) або навпаки (від'ємний результат).

2. «Загальний коефіцієнт покриття (Total Debt Ratio)»:

$$\text{Total Debt Ratio} = \frac{\text{Загальні зобов}'\text{язання}}{\text{Загальні активи}} \text{ x } 100\%, \quad (1.11)$$

де

Total Debt Ratio - загальний коефіцієнт покриття.

Загальний коефіцієнт покриття (1.11) вказує на відношення загальних зобов'язань до загальних активів та вимірює фінансову стійкість суб'єкта. Цей показник допомагає визначити, який відсоток активів покритий зобов'язаннями.

3. «Коефіцієнт фінансової стабільності (Financial Stability Ratio)»:

$$\text{Financial Stability Ratio} = \frac{\text{Власний капітал}}{\text{Пасиви}} \text{ x } 100\%, \quad (1.12)$$

де

Financial Stability Ratio - коефіцієнт фінансової стабільності.

Коефіцієнт фінансової стабільності (1.13) визначає співвідношення власного капіталу до пасивів та вимірює фінансову стійкість суб'єкта. Цей показник показує, який відсоток пасивів покритий власним капіталом і вказує на ступінь фінансової стійкості.

Переваги методу:

1. Прозорість фінансів:

Цей метод надає користувачам чітке уявлення про структуру їхніх фінансів, включаючи, як розподілені активи та зобов'язання. Це сприяє більшій прозорості та розумінню своєї фінансової ситуації.

2. Систематичність:

Аналіз фінансового балансу допомагає систематично відстежувати фінанси з часом. Користувачі можуть створювати фінансові звіти на різних інтервалах (щомісяця, щокварталу) і порівнювати їх для виявлення тенденцій і змін.

3. Покращення фінансового само сприйняття:

Аналіз фінансового балансу може допомогти користувачам зосередитися на своїх фінансових цілях і раціоналізувати свої витрати. Він сприяє кращому розумінню того, як їхні гроші працюють для них.

4. Зменшення фінансових ризиків:

Шляхом регулярного аналізу фінансового балансу, користувачі можуть вчасно виявляти фінансові проблеми та ризики і приймати відповідні заходи для їхнього зменшення або уникнення.

Недоліки методу:

1. Складність:

Метод вимагає ретельного збору інформації та розрахунків, що може бути важким для користувача без фінансової освіти.

2. Специфічність:

Деякі аспекти методу можуть бути важкими для зрозуміння особистими користувачами, які не мають фінансової освіти.

3. Залежність від точності даних:

Результати аналізу залежать від точності та актуальності даних, що вводяться.

1.3. Огляд літератури та існуючих програм

В ході роботи було проведено аналіз існуючих програм для управління особистими фінансами, а також ознайомимося з літературними джерелами, пов'язаними з цією темою.

Важливим етапом у розробці програми для управління особистими фінансами є вивчення літературних джерел, які містять корисну інформацію з цієї області. Для побудови теоретичної бази та отримання важливих інсайтів, я вивчив такі джерела:

1. "Основи управління фінансами" (Fundamentals of Financial Management) Автор: Еін М. Гітман (Eugene F. Brigham) і Джоел Ф. Хустон (Joel F. Houston) [7]: Ця книга є відмінним джерелом для ознайомлення з основними концепціями управління фінансами. Вона надає зрозумілі пояснення стосовно бюджетування, аналізу фінансових показників та стратегій інвестування. Важливою є інтеграція цих знань у програмному забезпеченні, оскільки вони допомагають користувачам зрозуміти і визначити свої фінансові цілі та пропонують інструменти для досягнення їх.
2. «Повна зміна грошей: перевірений план фінансової стійкості (The Total Money Makeover: A Proven Plan for Financial Fitness) авторства Дейва Рамсі [8]: Ця книга розглядає методику управління особистими фінансами з практичного погляду. Вона охоплює планування бюджету, методи керування боргами та інвестиційні стратегії для досягнення фінансового успіху. Знання та методи, наведені в цій книзі, мають безпосереднє застосування в програмному забезпеченні для управління фінансами, оскільки вони допомагають користувачам керувати своїми фінансами в повсякденному житті.
3. Наукова стаття "Personal Financial Planning and Financial Satisfaction: Self-Control as a Mediator" авторства Майкла Адусея [9] вивчає важливий аспект особистого фінансового управління та його вплив на фінансове

задоволення. Автори статті розглядають роль особистого фінансового планування та самоконтролю у взаємозв'язку між цими факторами. Стаття аналізує теоретичний фреймворк фінансового задоволення та встановлює, що особисте фінансове планування може впливати на фінансове задоволення через медіацію самоконтролю.

Автори провели емпіричне дослідження для підтвердження цього теоретичного підходу.

У дослідженні були використані статистичні методи для аналізу взаємозв'язку між фінансовим плануванням, самоконтролем та фінансовим задоволенням. Результати дослідження підтвердили гіпотезу про те, що самоконтроль виступає важливим посередником у відносинах між особистим фінансовим плануванням та фінансовим задоволенням. Ця стаття може бути корисною для тих, хто цікавиться впливом особистого фінансового планування на фінансове задоволення та роль самоконтролю в цьому процесі. Вона також надає практичні висновки для тих, хто бажає поліпшити своє фінансове задоволення через ефективне управління своїми фінансами.

Ці джерела були вибрані з урахуванням їх значущості для розуміння фінансових аспектів та стратегій, які допомагають кожній особі краще розбиратися у власних фінансах та приймати обґрунтовані фінансові рішення. Засвоєні знання з цих джерел стали основою для розробки програмного забезпечення, яке допомагає користувачам досягати фінансової стабільності та успіху.

Під час розробки програми був проведений аналіз існуючих програм для управління особистими фінансами. Цей аналіз допоміг визначити основні функції та функціональність таких програм. Серед існуючих програм можна виділити такі:

1. «Mint»:

Огляд: Mint - це веб-сервіс та додаток, що надає користувачам можливість відстежувати свої фінанси. Він автоматично імпортує фінансові дані з банківських рахунків, кредитних карток, інвестиційних облігацій та інших

джерел. Користувачі можуть створювати бюджет, відстежувати витрати, отримувати сповіщення про платежі та аналізувати фінансовий стан.

Значення: Mint надає користувачам інтуїтивний і зручний інтерфейс для керування фінансами. Однією з основних переваг є автоматичний імпорт даних, що зменшує необхідність вручну вводити фінансові операції. Ця програма надає засоби для створення і відстежування бюджету, що допомагає користувачам контролювати витрати та доходи.

2. «YNAB (You Need A Budget)»:

Огляд: YNAB спеціалізується на плануванні бюджету та контролі витрат. Вона пропонує "Правило четвертої категорії" - кожен долар має своє призначення. Користувачі визначають фінансові цілі та розподіляють кошти на категорії відповідно до своїх пріоритетів.

Значення: YNAB допомагає користувачам більш свідомо керувати своїми фінансами. Вона надає систему для визначення пріоритетів та контролю над витратами. Підкреслено важливість кожного долара, що допомагає ефективніше використовувати кошти.

3. «Quicken»:

Огляд: Quicken - це програма для керування фінансами, яка надає користувачам можливість відстежувати фінанси, включаючи інвестиції та податки. Вона дозволяє генерувати різноманітні фінансові звіти та аналізувати фінансовий стан.

Значення: Quicken є потужним інструментом для керування фінансами, особливо для тих, хто має інвестиції та інші складні фінансові активи. Вона дозволяє користувачам отримувати глибокий аналіз своїх фінансів та генерувати різноманітні звіти.

Огляд існуючих програм був корисним для зрозуміння того, які функції та можливості можуть бути включені до вашої програми для управління особистими фінансами. Вони надають важливий досвід інформаційного забезпечення для розробки програмного продукту, що задовольнить потреби користувачів у керуванні своїми фінансами.

Загальний висновок з огляду літератури та існуючих програм полягає в тому, що існують багато інструментів для управління особистими фінансами, але вони мають різні функціональність та підходи. Розробка програми для управління особистими фінансами на платформі JavaFX дозволить створити індивідуальний та придатний для користувача інструмент для досягнення фінансових цілей.

1.4. Постановка задачі

Для досягнення поставленої мети були сформульовані наступні завдання та розглянуті їх значення:

1. Розробити інтерфейс користувача:
– Завдання: Створити інтерактивний та дружній інтерфейс для користувачів, який дозволить легко вводити фінансові дані.
– Значення: Зручний інтерфейс є ключовим для залучення користувачів і спрощує процес введення фінансових даних.
2. Збереження фінансових даних:
– Завдання: Розробити систему для ефективного збереження фінансових записів користувача в базі даних.
– Значення: Надійне збереження даних забезпечує конфіденційність та легкий доступ до фінансової інформації.
3. Аналіз фінансових операцій:
– Завдання: Розробити функціонал для обчислення загального доходу та витрат.
– Значення: Ця функція надає користувачам можливість легко аналізувати свій фінансовий стан та розуміти, як розподілені їхні гроші.
4. Візуалізація фінансових даних:
– Завдання: Розробити засоби для побудови графіків та діаграм, які візуалізують фінансову інформацію.
– Значення: Графіки та діаграми допомагають користувачам більш

наочно розуміти рух своїх фінансів та виявляти тенденції.

5. Забезпечити можливість визначення та відстеження інших параметрів транзакції:

– Завдання: Надати користувачам можливість аналізувати додаткові параметри транзакцій.
– Значення: Ця можливість допомагає користувачам здійснювати більш глибокий аналіз своїх фінансів та приймати обґрунтовані рішення.

Програма, що розробляється, спрямована на поліпшення фінансової грамотності користувачів і надає їм інструменти для досягнення фінансових цілей. Вона допомагає краще контролювати особисті фінанси та приймати обґрунтовані фінансові рішення, що є важливим для досягнення фінансової стабільності та благополуччя.

1.5. Висновки

Актуальність цієї роботи необхідно розглядати в контексті сучасного світу, який зазнає постійних змін та вимагає від нас управління фінансами на все більш професійному рівні.

Зараз, більше ніж будь-коли раніше, ми стикаємося з потребою управляти складними фінансами. Це може бути пов'язано з управлінням особистими бюджетами, розподілом доходів і витрат, вкладаннями та інвестиціями, оподаткуванням і податками, а також з купівлею, продажею і обліком різних активів.

Зручний інструмент управління фінансами, створений на платформі JavaFX, відкриває доступ до широкого спектру можливостей. Користувачі отримують засіб для аналізу своїх фінансових операцій, що дозволяє їм легко визначити, куди йде гроші і як вони ростуть. Планування бюджету стає ефективним і простим завдяки програмі, яка допомагає визначити оптимальний розподіл коштів та прогнозувати доходи та витрати.

Програмне забезпечення для управління фінансами на платформі JavaFX

стає надійним помічником у підтримці фінансової стабільності та досягненні фінансових цілей. Кожен користувач отримує інструмент, який допомагає раціонально управляти своїми фінансами, а це в свою чергу сприяє покращенню фінансової грамотності та сприяє здійсненню інформованих рішень щодо фінансів.

РОЗДІЛ 2
МЕТОДИ ТА ТЕХНОЛОГІЇ ВИРІШЕННЯ ЗАДАЧІ

2.1. Вибір мови програмування

Однією з найважливіших галузей є банківська справа. Стабільність, довгострокова підтримка та безпека - все це враховується при виборі правильного технологічного стеку. Організації повинні розвиватися і модернізувати свої системи, щоб йти в ногу з постійним технологічним прогресом.

Завдяки своїй надійності, безпеці та ефективності, банківська та фінансова індустрія охопила мову програмування Java, а компанії, що займаються веб-розробкою на Java, використовують її як потужний інструмент для створення цілого ряду банківських додатків, від систем онлайн-банкінгу до платформ мобільних платежів.

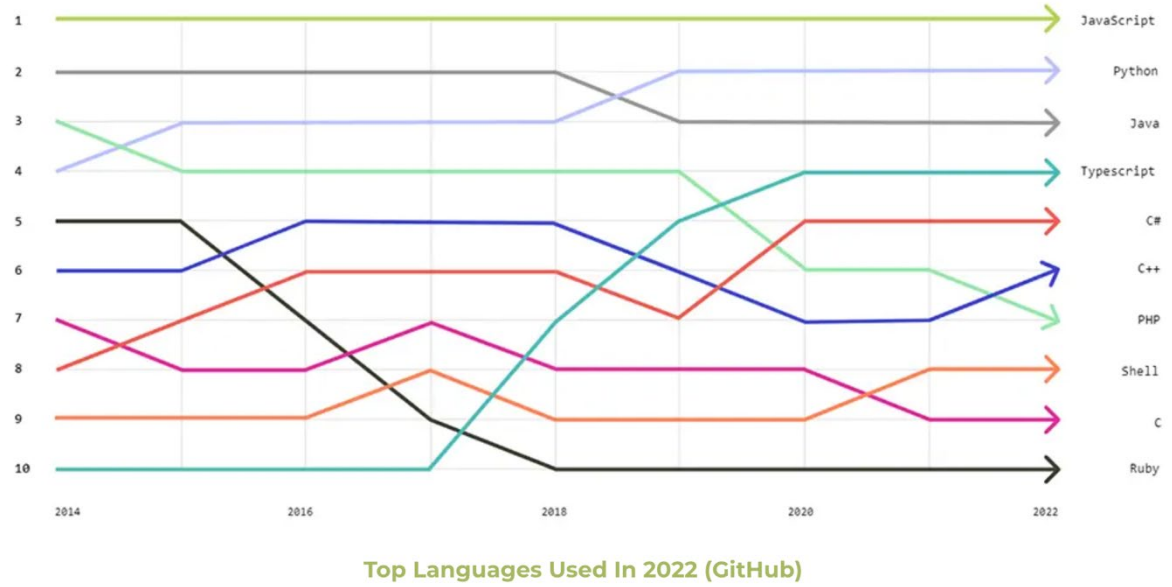

Рис. 2.1. Популярність мов програмування за часовою шкалою

Не дивно, що згідно з дослідженням GitHub, ця мова програмування тісно пов'язана з технологією, яка найчастіше використовувались розробниками у 2022 році [10].

Рис. 2.2. Логотип мови програмування Java

Java - це потужна мова програмування, яка може обробляти великі обсяги транзакцій і може бути легко розширена в міру зростання компанії. Java не залежить від платформи і може працювати без змін на широкому спектрі операційних систем, що робить фінансові системи простими в обслуговуванні та модернізації. Легкість в обслуговуванні та модернізації. Фінансові системи легко обслуговувати та модернізувати. Крім того, мова програмування Java [11] підтримується і оновлюється великою і динамічною спільнотою розробників, тому програмісти мають доступ до набору інструментів, які допомагають їм розробляти фінансові системи.

Переваги розробки додатків на Java для банківського сектору:

– Надійність:

Java визнана своєю надійністю у секторі фінансових технологій. Завдяки специфічним функціям обробки помилок, винятків та автоматичного управління пам'яттю, мова програмування стає ідеальною для побудови стійких фінансових систем, які мають високі вимоги до надійності та доступності.

– Безпека:

Java Consulting [12] надає різноманітні функції безпеки, які відповідають вимогам фінансових додатків. Вбудована підтримка шифрування даних та

безпечних мережевих з'єднань допомагає захистити конфіденційну інформацію від хакерів та інших загроз.

– Ефективність:

Розробка програмного забезпечення на Java оптимізована для ефективної обробки великих обсягів даних і транзакцій, що особливо важливо для банківського сектору з великим потоком інформації. Висока масштабованість та підтримка паралельної обробки роблять Java ідеальним вибором для швидких банківських систем.

– Кросплатформенність:

Java є платформо-нейтральною, що дозволяє створювати додатки, які працюють на різних операційних системах, спрощуючи підтримку та розгортання.

– Ринкова гнучкість:

Java є оптимальним вибором для компаній-розробників, які створюють банківські системи для різних гаджетів і операційних систем. Код, написаний на Java, може працювати на різних платформах без змін.

– Активна спільнота розробників:

Активна спільнота розробників постійно вдосконалює мову та надає інструменти для швидкого розвитку банківських систем.

– Оновлення та підтримка від Oracle:

Корпорація Oracle регулярно випускає оновлення та патчі безпеки для підтримки актуальності та безпеки Java.

Ці характеристики роблять Java важливим інструментом для розробки фінансових додатків, забезпечуючи надійність, безпеку та ефективність у банківському секторі.

2.2. Технології графічного інтерфейсу

2.2.1. UI та UX

UI та UX - це концепції, які працюють паралельно при створенні веб-сайту, сервісу або будь-якого програмного продукту.

UI (користувацький інтерфейс) стосується візуальних характеристик продукту і включає в себе зовнішній вигляд, відчуття, дизайн і структуру всіх елементів, з якими необхідно взаємодіяти.

UX (користувацький досвід), з іншого боку, стосується поведінкових характеристик інтерфейсу і, більш конкретно, визначає, як користувачі можуть взаємодіяти з вашим продуктом і як вони почуватимуться під час його використання [13].

Однак вони взаємозалежні. Якщо одна з них відсутня, інша стає марною.

Інтерфейс користувача - це точка доступу, через яку користувач взаємодіє з дизайном. Існує три типи користувацьких інтерфейсів:

Рис. 2.3. Типи інтерфейсів

– Графічний інтерфейс користувача (GUI) - користувач взаємодіє з візуальним представленням на цифровій панелі управління. Робочий стіл комп'ютера - це графічний інтерфейс.

– Голосовий інтерфейс (VUI) - користувачі взаємодіють за допомогою голосу; більшість розумних помічників, таких як Siri на iPhone і Alexa на пристроях Amazon, є VUI.
– Інтерфейси на основі жестів - користувачі взаємодіють з 3D-просторами дизайну за допомогою рухів тіла.

2.2.2. JavaFX GUI

Згідно зі статистикою Google Trends за 2021-2022 роки, найвпливовішим інструментарієм або програмним забезпеченням в Java Swing проти JavaFx є JavaFx [14]. У грудні 2021 року використання JavaFX було дуже високим у порівнянні з Swing. З тих пір JavaFX досить широко використовується у світі в цілому.

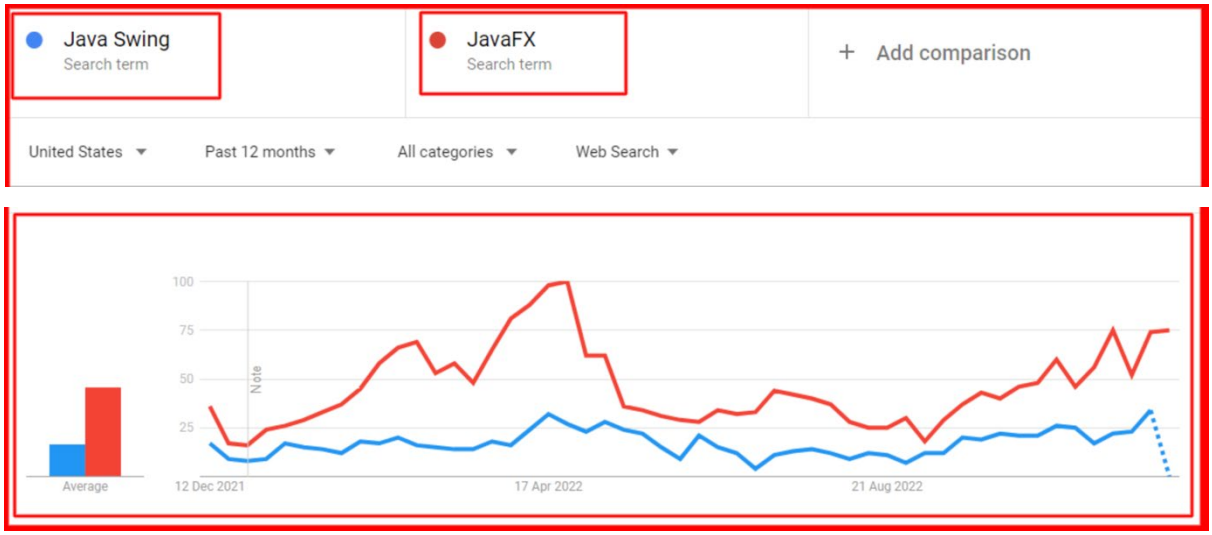

Рис. 2.4. Популярність технологій графічного інтерфейсу мови Java

Таблиця 2.1

Таблиця відмінностей JavaFX від Java Swing.

Основні характеристики	Java Swing	JavaFX
Користувацький досвід	Пропонує традиційний досвід інтерфейсу користувача, що може здаватися застарілим для сучасних додатків.	Забезпечує сучасний інтерфейс користувача з підтримкою анімації та інших сучасних елементів.
Архітектура	Орієнтований на компоненти з подійною моделлю програмування.	Використовує граф сцени для опису структури інтерфейсу користувача.
Мова програмування	Розроблено на основі Swing API та написано мовою Java.	Використовує JavaFX API та мову програмування Java.
Підтримка CSS	Не підтримує стилізацію за допомогою CSS.	Дозволяє стилізувати інтерфейс користувача за допомогою CSS, спрощуючи керування виглядом.
Гнучкість і розширюваність	Може викликати проблеми з гнучкістю та розширюваністю через обмежену архітектуру	Має гнучку архітектуру, яка полегшує розширення та налагодження. |
Підтримка вбудованих діаграм	Не має вбудованих інструментів для створення діаграм.	Має вбудовані компоненти для створення графіків та діаграм.
Мультимедійна підтримка	Має обмежену підтримку мультимедіа	Забезпечує вбудовану підтримку аудіо, відео та 2D/3D графіки.
Взаємодія з веб-контентом	Важко взаємодіяти з веб-контентом в межах додатка.	Має компонент WebView для вбудованої взаємодії з веб-контентом.
Відкритість	Є частиною стандартної бібліотеки Java.	Незалежний фреймворк з відкритим вихідним кодом.

Java Swing - це бібліотека для створення графічного інтерфейсу користувача (GUI) для програм на мові Java. Вона надає набір компонентів інтерфейсу, таких як кнопки, тексти, таблиці та інші, що дозволяє розробникам легко створювати інтерактивні додатки. Java Swing була випущена як частина Java Foundation Classes (JFC) і стала популярною завдяки своїй переносимості між різними платформами.

JavaFX - це набір графічних мультимедійних пакетів, які дозволяють розробникам проектувати, створювати, тестувати, налагоджувати та розгортати багатофункціональні клієнтські програми. JavaFX був представлений як альтернатива Swing, бібліотеці графічного інтерфейсу Java, і фокусується на наданні більш сучасного та візуально привабливого інтерфейсу користувача.

Тепер щодо порівняння між Java Swing і JavaFX. Наведена вище табл. 2.1, представляє основні відмінності між цими двома бібліотеками. Java Swing добре підходить для створення традиційних десктопних додатків і має велику кількість компонентів інтерфейсу користувача, але його продуктивність може бути нижчою порівняно з JavaFX.

В JavaFX існують різні контейнери [15] (класи-контейнери), які визначають макет і розміщення елементів інтерфейсу користувача. Ось кілька з них:

– HBox (Горизонтальна скринька):

  Розташовує дочірні елементи горизонтально (один за одним зліва направо).

– VBox (Вертикальна скринька):

  Розташовує дочірні елементи вертикально (один під одним зверху вниз).

– Group (Група):

  Групує дочірні елементи, але не накладає на них жодного макету.

– FlowPane (Потокова панель):

Розташовує дочірні елементи в потоці зліва направо та зверху вниз, автоматично переносячи їх, якщо не вистачає місця.

– BorderPane (Панель з обрамленням):

Розташовує дочірні елементи в панелі з п'ятьма областями: верхньою,

нижньою, лівою, правою та центральною.

– Region (Область):

Базовий клас для всіх елементів у JavaFX, які можна відображати та мають свої властивості розміщення та розміру.

– Pane (Панель):

Базовий клас для всіх контейнерів, які розташовують свої дочірні елементи в координатній системі.

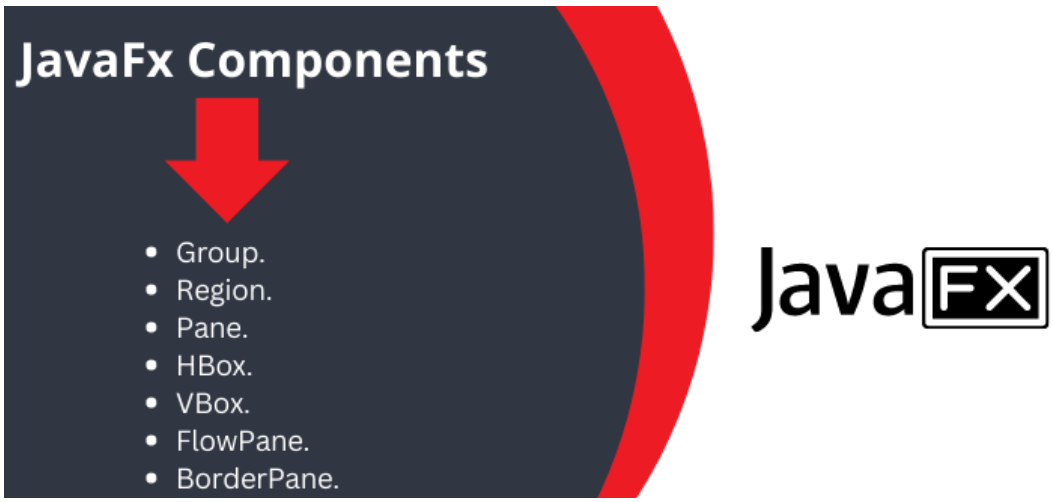

Рис. 2.5. Компоненти JavaFX

Контейнери надають різні можливості розміщення та розташування елементів у вашому інтерфейсі користувача в залежності від потреб і дизайну.

Основними особливостями JavaFX є

– Покращений користувацький інтерфейс:

Надає широкий спектр елементів керування, макетів та стилів інтерфейсу користувача для створення візуально привабливих та інтерактивних інтерфейсів. Підтримуються сучасні принципи дизайну, включаючи анімацію, переходи та власні стилі.

– Сценічна графіка:

Використовує систему візуалізації на основі сценічної графіки, яка дозволяє розробникам створювати складні користувацькі інтерфейси, збираючи ієрархію вузлів. Такий підхід спрощує розробку користувацького інтерфейсу та заохочує використання модульних та багаторазових компонентів.

– Стилі CSS:

Стилізація компонентів інтерфейсу користувача за допомогою JavaFX може бути легко здійснена за допомогою каскадних таблиць стилів (CSS). Це дозволяє легко налаштовувати зовнішній вигляд додатків, не змінюючи код Java.

– Підтримка мультимедіа:

Має вбудовану підтримку мультимедійних елементів, таких як аудіо, відео та 2D/3D графіка. Медіафайли можна легко інтегрувати в додатки.

– WebView:

Компонент WebView дозволяє вбудовувати веб-вміст у ваш JavaFX-додаток, відображати веб-сторінки та взаємодіяти з веб-сервісами.

– Вбудована графіка:

Має компоненти для побудови діаграм, такі як гістограми, кругові та лінійні діаграми, які дозволяють легко візуалізувати дані.

– 3D-графіка:

Підтримує побудову 3D-діаграм, що дозволяє створювати захоплюючі 3D-візуалізації та симуляції.

– Кросплатформеність:

Є крос-платформним і може використовуватися на різних операційних системах, включаючи Windows, macOS і Linux. Це робить його ідеальним вибором для розробки додатків, які повинні працювати на різних пристроях і платформах.

– Інтеграція з Java:

JavaFX повністю інтегрований з мовою програмування Java, що дозволяє розробникам повною мірою використовувати можливості мови Java і використовувати існуючий код Java.

Ці можливості роблять JavaFX потужним інструментом для розробки сучасних, багатофункціональних і привабливих користувацьких інтерфейсів для багатьох різних типів додатків.

2.3. Збирачі проектів

2.3.1. Project Builder

Project Builder - це інструмент для автоматизації процесу створення, тестування та розгортання програмних продуктів і проектів. Основні функції Project Builder, який також називають інструментом побудови проектів або системою побудови, включають управління залежностями, компіляцію коду, тестування, пакування та розгортання.

Основні можливості Project Builder:

– Керування залежностями:

Визначає та керує залежностями між різними частинами проекту, бібліотеками та іншими компонентами. Це автоматизує процес завантаження та оновлення необхідних бібліотек та інших залежностей.

– Компіляція та генерація:

Автоматично компілює вихідний код проекту у виконуваний код. Сюди входить обробка всіх вихідних файлів, компіляція і генерація виконуваних або проміжних файлів.

– Тестування:

Запуск автоматизованих тестів для перевірки правильності та стабільності коду. Тестування можна зробити частиною процесу збірки, щоб переконатися, що внесені зміни не зруйнують існуючу функціональність.

– Пакування:

Створення пакунків або архівів, що містять виконуваний код, бібліотеки, конфігураційні файли та інші ресурси, готові до розгортання в реальному середовищі.

– Розгортання:

Автоматичне розгортання готового продукту на сервері або в іншому середовищі для використання.

Різні мови програмування використовують різні інструменти для створення проектів, детальніше в табл. 2.2. Вони дозволяють визначати

структуру проекту, залежності, завдання збірки та інші аспекти розробки.

Таблиця 2.2

Таблиця порівняння існуючих Project Builder`s

Особливості	Maven	Gradle	Ant	Apache Buildr	Bazel
Мова конфігурації	XML	Groovy (можливо Kotlin та інші)	XML	Ruby (також підтримується Scala)	Starlark (власна DSL, схожа на Python)
Залежності	Централізовані, описані в pom.xml	Централізовані, описані в build.gradle	Централізовані, описані в build.xml	Централізовані, описані в Buildfile	Розподілені, описані в BUILD-файлі
Розширюваність	Плагіни	Плагіни	Задачі та розширення Ant	Плагіни	Плагіни, додаткові мови програмування
Декларативність	Так	Так	Ні	Так	Так
Робочі файли	pom.xml	build.gradle	build.xml	Buildfile	BUILD-файл
Спрощена конфігурація	Так	Так	Залежить від конфігурації, може бути XML	Так	Так
Спільнота	Велика	Велика	Середня	Середня	Зростаюча

2.3.1. Apache Maven

Apache Maven [16] - це потужний інструмент для автоматизації процесу створення, тестування та розгортання програмного забезпечення; це один з найпопулярніших інструментів, що використовується у світі Java-розробки для управління проектами будь-якої складності.

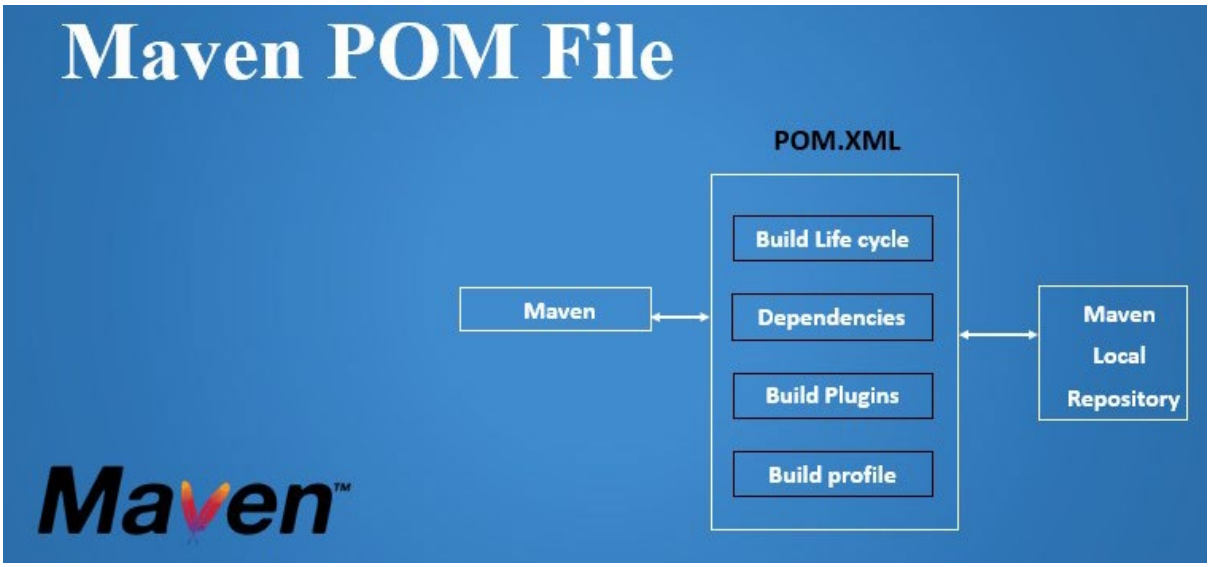

Рис. 2.6. Цикл роботи Maven

Робочий процес Maven при створенні проекту:

– Вирішення залежностей:

Завантажує всі необхідні бібліотеки (залежності), вказані у файлі pom.xml.

Зазвичай вони завантажуються з центрального сховища Maven або інших вказаних сховищ.

– Збірка:

Компілює вихідний код проекту в директорії src/main/java і розміщує скомпільовані класи в директорії target/classes.

– Тестування:

Запускає тести в каталозі src/test/java. Якщо тести пройшли успішно, Maven продовжує виконання.

– Пакування:

Пакує скомпільовані класи та інші ресурси у JAR, WAR або інший архів,

залежно від типу проекту. Запаковані результати зберігаються у цільовому каталозі.

– Інтеграційні тести:

Якщо проект має інтеграційні тести, їх можна запустити після пакування.

– Вивантаження:

Запаковані результати завантажуються до локального репозиторію Maven. Це корисно, якщо інші проекти, що розробляються на тому ж комп'ютері, хочуть використовувати артефакт як залежність.

– Розгортка:

Запаковані артефакти можна розгорнути у віддаленому сховищі і надати доступ до них іншим розробникам та системам збірки.

Основними можливостями Maven є:

– Асистент створення проекту:

Використовує файл POM [17] (Project Object Model) для визначення проекту, його залежностей, конфігурації та інших параметрів. Це дозволяє чітко визначити структуру та конфігурацію проекту.

– Управління залежностями:

Автоматично завантажує необхідні бібліотеки та інші залежності з центрального сховища, такого як Maven Central. Це спрощує процес додавання, оновлення та видалення залежностей у проекті.

– Життєвий цикл проекту:

Визначає стандартні цикли життєвого циклу для різних типів робіт, таких як збірка, тестування, пакування та розгортання. Це полегшує виконання рутинних завдань.

– Плагіни:

Підтримує ряд плагінів, які розширюють його функціональність. Це дозволяє розробникам використовувати різноманітні інструменти для виконання конкретних завдань.

– Управління конфігурацією:

Спрощує управління конфігурацією, дозволяючи зберігати параметри

проекту в POM-файлах. Це допомагає забезпечити узгодженість і повторюваність у розробці.

– Центральне сховище:

Використовує центральне сховище для зберігання та спільного використання бібліотек, що полегшує використання стандартів та обмін кодом між проектами.

Apache Maven є високоефективним інструментом для автоматизації рутинних завдань розробки, спрощуючи процес створення, тестування та розгортання програмного забезпечення.

2.4. Середовище розробки

2.4.1. Вибір IDE

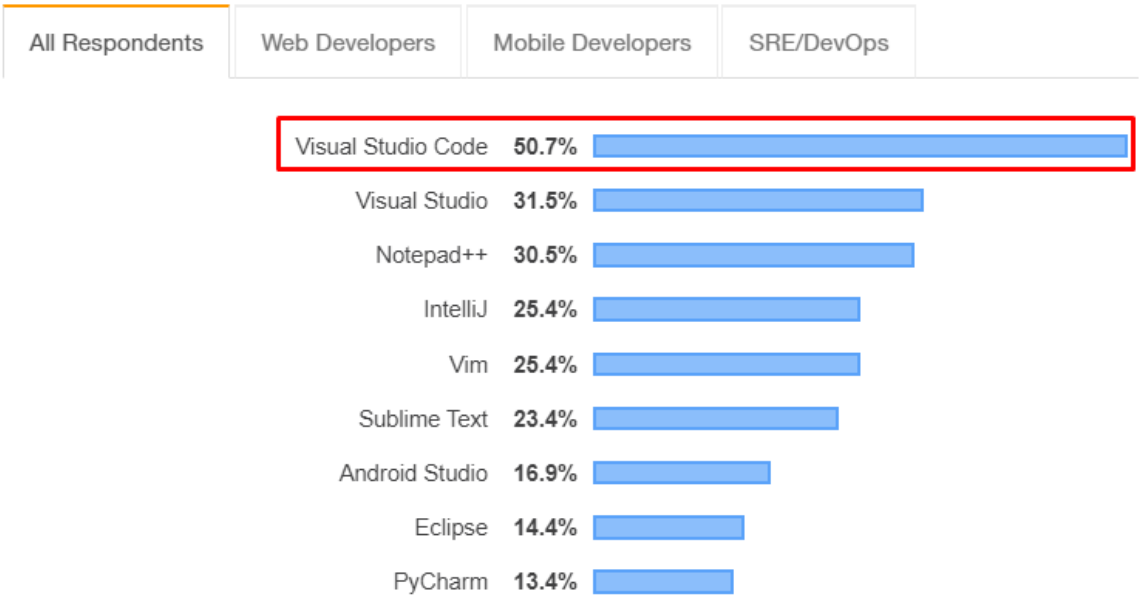

Рис. 2.7. Загальна популярність вибору IDE

Загалом, вибір IDE [18] є важливим етапом у процесі створення програмного продукту. Для оптимальної продуктивності та комфорту розробника важливо враховувати не тільки популярність середовища розробки, як це показано на рис. 2.7, а також й на декілька інших аспектів при виборі.

По-перше, потрібно враховувати мову програмування, яку ви плануєте

використовувати: обирати IDE, яка підтримує не тільки Java, але й інші мови, такі як Kotlin, Python та JavaScript.

Другий аспект - це можливості та інструменти IDE. Важливо перевірити, чи пропонує обрана IDE інтегровану систему контролю версій, рефакторинг коду, автодоповнення, аналіз коду, підтримку фреймворків та інші інструменти.

Зручність використання - третій елемент: Важливо, щоб інтерфейс IDE був зручним і комфортним. Деякі розробники віддають перевагу простим IDE, в той час як іншим потрібні розширені можливості потужного середовища розробки.

Діяльність спільноти та підтримка розробників - це четвертий елемент. Якщо виникає проблема, активна спільнота значно полегшує отримання відповідей та рішень.

Інтеграція з іншими інструментами, платформами підтримки та особистий досвід також відіграють важливу роль у виборі найкращої для вас IDE.

Вибір IDE - це особиста справа кожного, і найкраще рішення залежить від особистих уподобань, вимог проекту та вашого власного досвіду. Багато розробників тестують кілька IDE, перш ніж вибрати ту, яка найкраще відповідає їхнім потребам.

2.4.2. IntelliJ IDEA

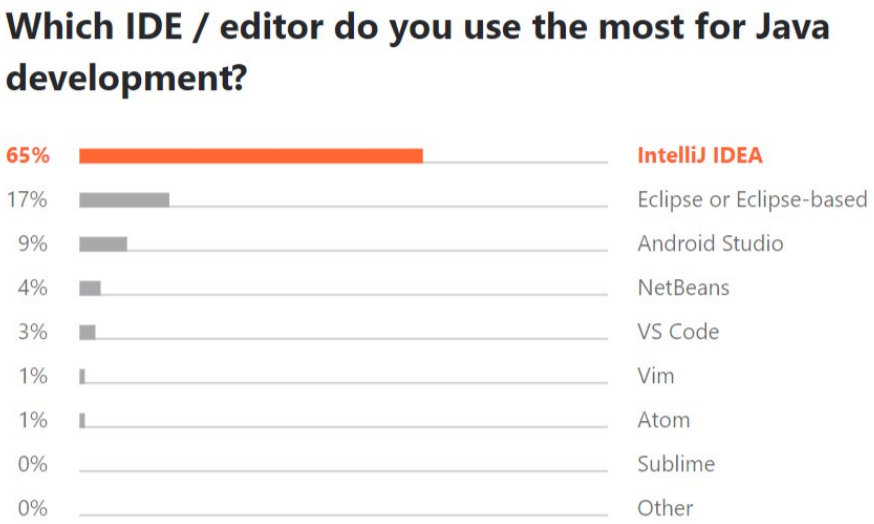

Рис. 2.8. Популярність IDE на мові програмування JAVA

IntelliJ IDEA - це дуже популярне інтегроване середовище розробки (IDE), розроблене компанією JetBrains. Воно в першу чергу призначене для Java, але також підтримує широкий спектр мов програмування, включаючи Kotlin, Groovy та Scala. Цей інструмент розробки програмного забезпечення є дуже корисним для програмістів, оскільки дозволяє їм ефективно створювати, редагувати, налагоджувати та керувати кодом для широкого спектру додатків.

Можливості IntelliJ

– Інтелектуальна підтримка коду

IntelliJ IDEA пропонує інтелектуальне завершення коду, аналіз і рекомендації, щоб зробити кодування швидшим і безпомилковим. Він також може розуміти контекст вашого коду і надавати відповідні рекомендації.

– Розширений рефакторинг.

Потужні інструменти для перейменування, вилучення методів та оптимізації імпорту дозволяють розробникам легко та впевнено рефакторити код.

– Перевірка коду

IDE постійно перевіряє код на наявність потенційних проблем і надає швидкі виправлення та рекомендації для підтримки якості коду та зменшення кількості помилок.

– Інтеграція з системами контролю версій

Повна інтеграція з популярними системами контролю версій, такими як Git, SVN і Mercurial, полегшує спільну роботу над кодом і відстеження змін.

– Інструменти для роботи з базами даних

Розробники можуть керувати базами даних та робити запити до них в IDE з підтримкою SQL, діаграм баз даних та управління джерелами даних.

– Тестування та покриття

Потужні інструменти для виконання тестів та покриття коду дозволяють проводити всебічне тестування додатків.

– Екосистема плагінів

Велика бібліотека плагінів розширює функціональність завдяки підтримці

додаткових мов, фреймворків та інструментів.

– Крос-платформна розробка

IntelliJ IDEA дозволяє розробникам працювати над проектами для різних платформ, включаючи веб, десктопні, мобільні та хмарні додатки.

– Інтелектуальна налагодження

Широкі можливості налагодження, такі як вбудована оцінка значень змінних і виразів, дозволяють легко виявляти і виправляти проблеми в коді.

– Живі шаблони та генерація коду

Розробники можуть створювати власні шаблони коду та використовувати генерацію коду для прискорення роботи над проектом.

– Підтримка Kotlin

IntelliJ IDEA підтримує Kotlin, мову програмування JetBrains, забезпечуючи розширену допомогу в кодуванні та безшовну інтеграцію.

– Зручний інтерфейс

Зручний інтерфейс з темами, що налаштовуються, і комбінаціями клавіш покращує загальний досвід розробки.

– Постійні оновлення

Однією з найважливіших особливостей є те, що JetBrains регулярно випускає оновлення з новими функціями та покращеною продуктивністю.

– Спільнота та повні версії

IntelliJ IDEA доступна у двох редакціях: Community Edition є безкоштовною і має відкритий вихідний код, в той час як Ultimate Edition пропонує розширені можливості і підтримку різних технологій.

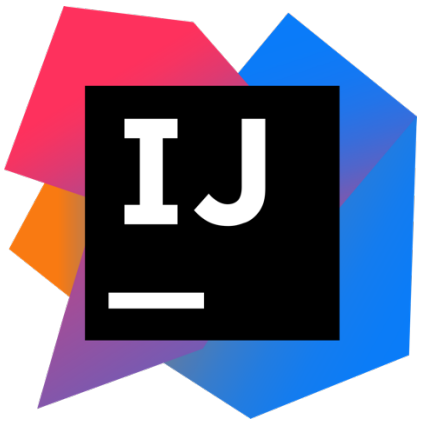

Рис. 2.9. Лого IntelliJ IDEA

IntelliJ IDEA - це потужне та універсальне інтегроване середовище розробки, яке значно підвищує продуктивність та досвід розробників у різних галузях програмування. Інтелектуальна підтримка коду, надійні інструменти рефакторингу, безшовна інтеграція з системою контролю версій та підтримка декількох систем збірки дозволяють розробникам ефективно писати високоякісний код. Розгалужена екосистема плагінів та постійно оновлюваний зручний інтерфейс роблять його найкращим вибором для розробки програмного забезпечення. Незалежно від того, чи є ви початківцем або досвідченим розробником, IntelliJ IDEA забезпечує міцну основу для створення великих програмних проектів [19].

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЙОГО ЗАСТОСУВАННЯ

3.1. Діаграми процесу обліку особистих фінансів у нотаціях IDEF0

Перед розробкою програмного продукту необхідно визначити структуру та функціональність програми [20] для фінансового обліку та створити функціональну модель.

Функціональна модель - це представлення системи або процесу в термінах її функцій, операцій і взаємодій між ними. Модель дозволяє уявити, як працює система або процес, як взаємодіють її компоненти і як вони виконують певні завдання.

Застосування функціональної моделі:

1. аналіз бізнес-процесів: допомагає визначити послідовність функцій і взаємодій бізнес-процесів.
2. визначення вимог: використовується для формалізації та документування функціональних вимог до системи.
3. моделювання системи: використовується для абстрагування та представлення функцій та їх взаємодій у складних системах.
4. оптимізація процесів: Допомагає знайти шляхи підвищення ефективності та продуктивності.

Функціональні моделі можуть бути представлені у вигляді діаграм, графіків або таблиць, залежно від методології та потреб проекту.

Окремі процеси фінансового обліку представлені за допомогою контекстних діаграм у нотації IDEF0.

IDEF0 (Integration Definition for Function Modelling) - це методологія функціонального моделювання, яка надає інструмент для створення функціональної моделі системи [21].

– IDEF0 стандартизує і впорядковує процес моделювання функціональних

аспектів системи або процесу.

– IDEF0 використовується для візуалізації та аналізу взаємодії різних функцій в системі або процесі.

Основна мета - описати і формалізувати функції, що виконуються системою, та їх взаємодію.

Основними елементами IDEF0 є функціональні блоки, стрілки, керуючий текст, контекстні блоки та інші елементи, які допомагають створювати діаграми та моделі функціональних взаємозв'язків всередині системи [22].

Для побудови структурно-функціональної моделі було виділено такі елементи: вхідні дані, функції та операції, вихідні дані:

– Вхідні дані.

1. Вибір користувача:
    a. Вхід за ім'ям та прізвищем користувача.
2. Завантаження підготовлених даних:
    a. Завантаження збережених даних користувача, якщо такі є.
3. Додавання операції:
    a. Тип операції (дохід або витрата).
    b. Сума.
    c. Дата і час операції.
    d. Категорії та підкатегорії.

– Функції та операції:

1. Зберігання та обробка транзакцій:
    a. Додавання нових транзакцій до бази даних.
    b. Розрахунок залишків відповідно до введених транзакцій.
2. Створення та відображення балансів.
    a. Створення таблиці, що відображає поточний баланс і розподіл місячних коштів за категоріями.
3. Створення історії транзакцій:
    a. Додавання нових транзакцій до загальної історії.
4. Створення діаграм і графіків:

a. Побудова діаграми, що показує структуру витрат і доходів.

b. Побудова графіків, що показують фінансові зміни з часом.

– Вихідні дані.

1. Баланс:

a. Показує щомісячні залишки та розподіл грошей за категоріями.

2. Історія транзакцій:

a. Інтерактивний звіт про всі транзакції.

3. Діаграми та графіки:

– Візуалізація даних у вигляді діаграм і графіків для кращого розуміння фінансової ситуації.

Контекстну діаграму зображено на рис. 3.1.

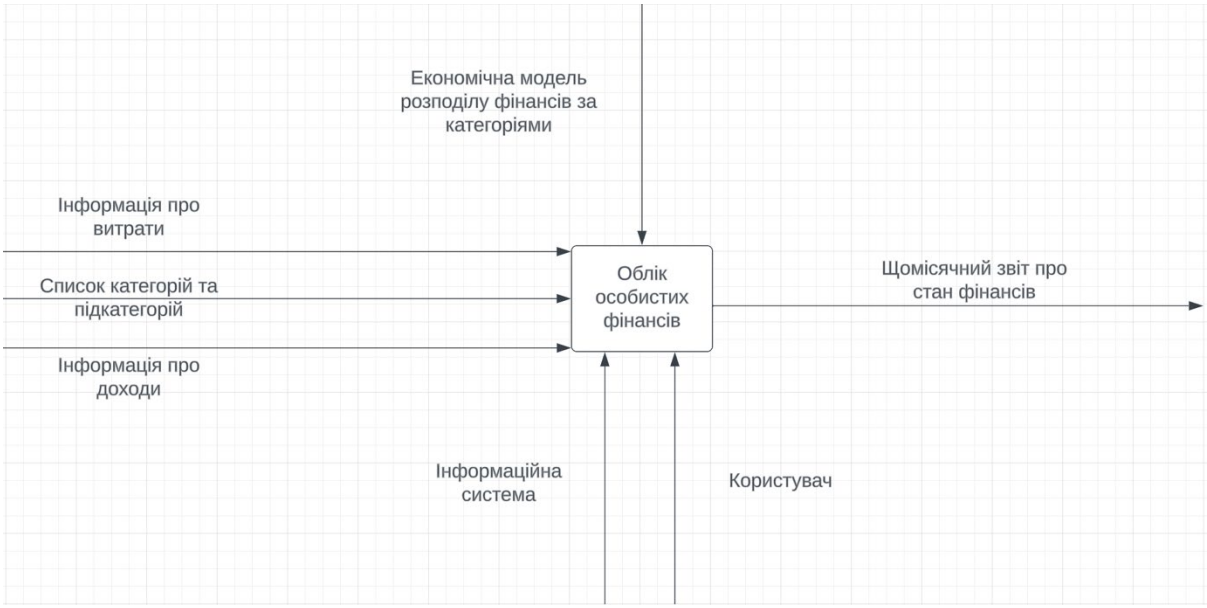

Рис. 3.1. Контекстна діаграма обліку особистих фінансів

Після створення контекстної діаграми необхідно виконати функціональну декомпозицію.

Функціональна декомпозиція - це процес розбиття системи на менші, більш керовані компоненти або підсистеми з метою полегшення декомпозиції та розвитку кожної частини системи окремо [23]. Такий підхід дозволяє зосередитися на конкретних завданнях кожної підсистеми і краще зрозуміти їхні функції та взаємодію.

Після декомпозиції процесу було виділено такі блоки:

1. Авторизація користувача в системі.
2. Обробка даних про доходи.
3. Обробка даних про витрати.
4. Генерація звітів та візуалізація.

Діаграму декомпозиції процесу зображено на рис. 3.2.

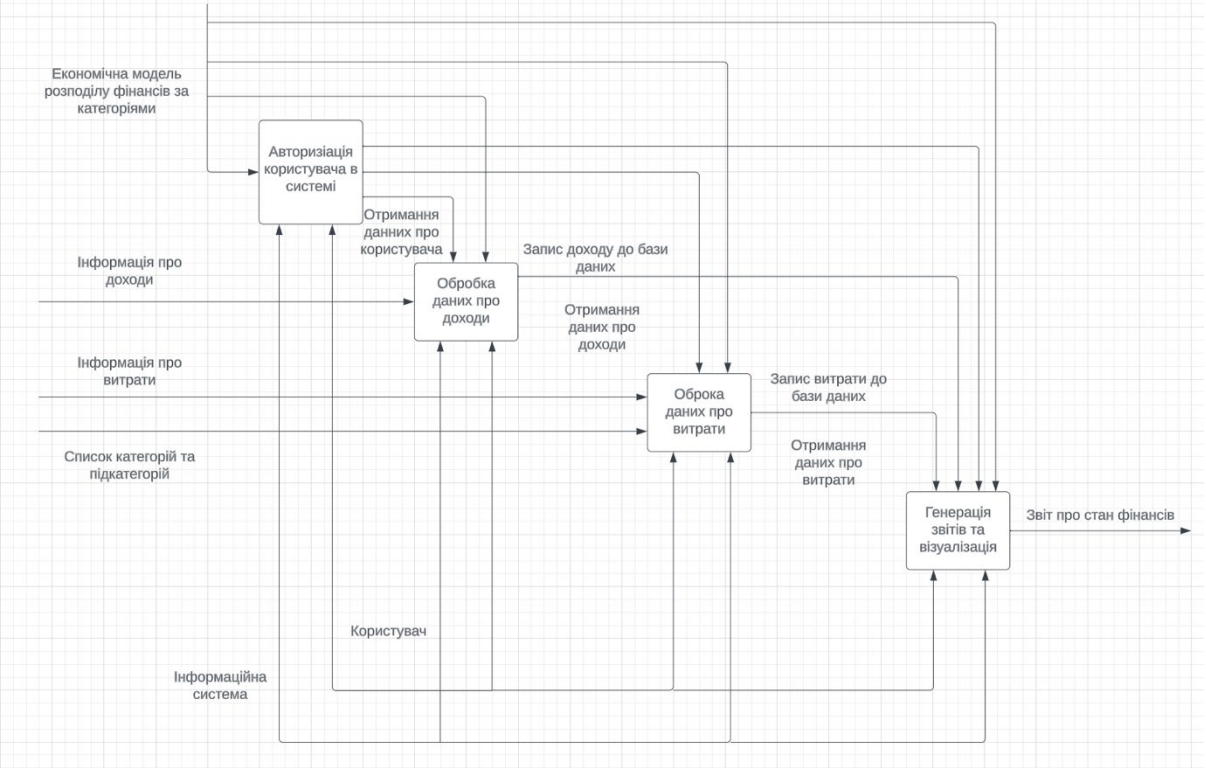

Рис. 3.2. Діаграма декомпозиції процесу обліку особистих фінансів

Потім був виконаний другий рівень декомпозиції.

Було декомпозовано процес додавання нової транзакції на такі блоки:

1. Вибір категорії.
2. Вибір підкатегорії.
3. Вибір дати та часу.
4. Вибір суми.
5. Обробка введених даних системою.

Діаграму декомпозиції другого рівня можна оглянути на рис. 3.3.

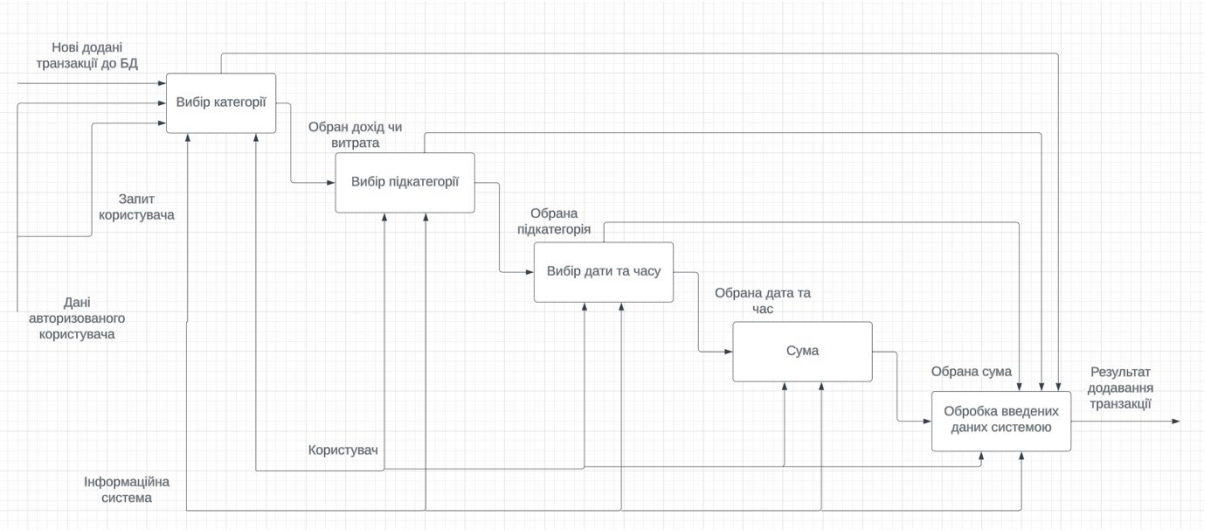

Рис. 3.3. Діаграма декомпозиції другого рівня системи обліку фінансів

3.2. Моделювання варіантів використання

3.2.1. Діаграма варіантів використання

На наступному етапі було розроблено схему варіантів використання програмного додатку обліку особистих фінансів.

Діаграма варіантів використання (Use Case Diagram) - це графічний інструмент, який використовується для моделювання функціональності системи з точки зору користувачів і зовнішніх систем. Цей тип діаграм допомагає описати, як система взаємодіє з різними акторами (користувачами та іншими системами) в різних варіантах використання.

Основні елементи діаграми варіантів використання

- Актори (Actors): представляють ролі або сутності, які взаємодіють з системою. Прикладами можуть бути користувачі, зовнішні системи та інші сервіси.
- Варіанти використання (Use Cases): описує конкретну дію або функціональність, яку система надає акторові.
- Взаємозв'язки (Relationships): показують взаємодію між акторами та варіантами використання.

Діаграма варіантів використання допомагає прояснити і візуалізувати, як

система буде використовуватися в реальному сценарії [24].

Для користувача у системі доступні наступні дії:

– управління витратами (внесення нової витрати, видалення витрати)
– управління прибутком (внесення нового прибутку, видалення прибутку)
– перегляд історії транзакцій.
– візуалізація графіків та діаграм (аналітичні інструменти)

Діаграму варіантів використання зображено на рис. 3.4.

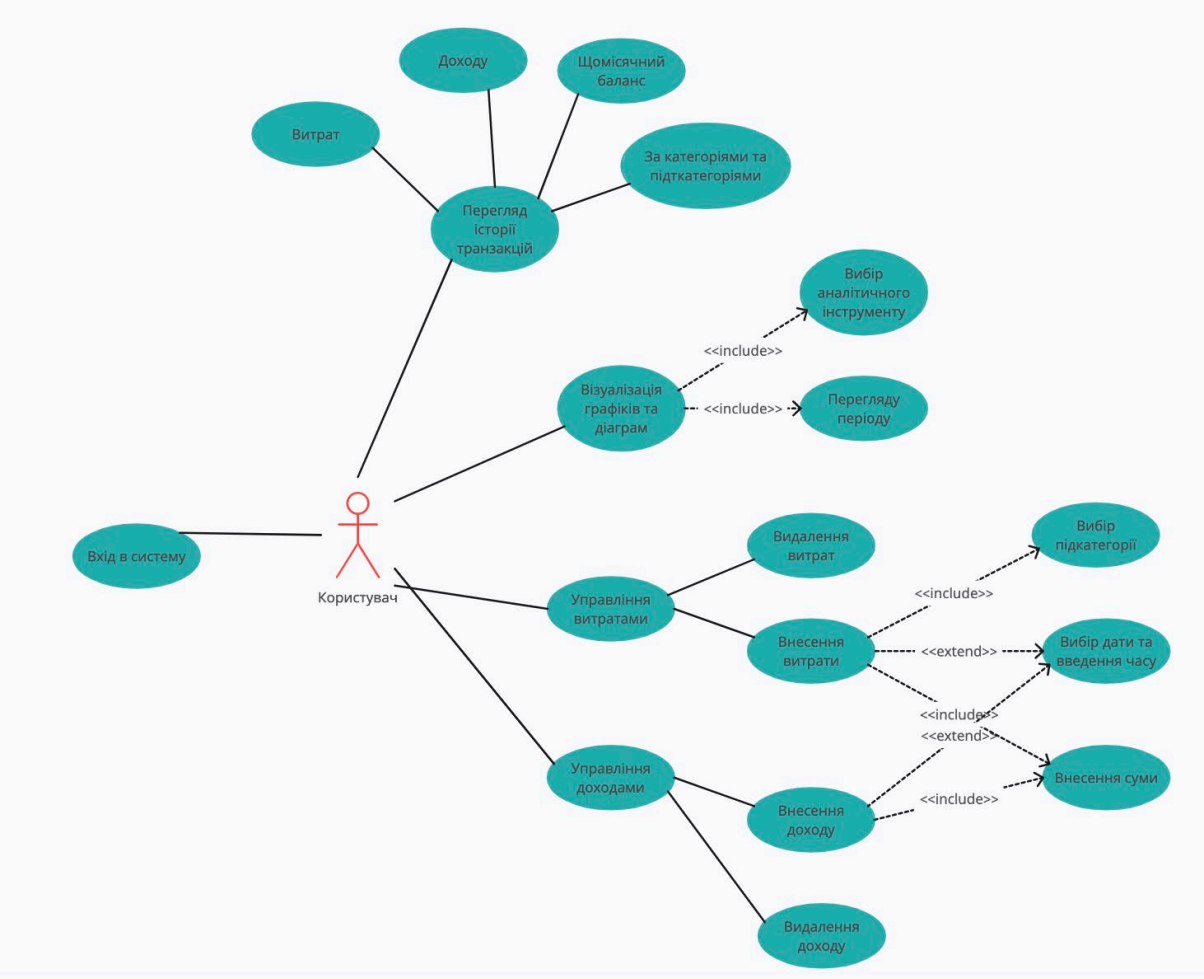

Рис. 3.4. UML діаграма варіантів використання

Опис дій:

1. Дії «Внесення даних про прибуток» та «Внесення даних про витрати» включають у себе додавання нової транзакції (В залежності від вибіру це «Прибуток» чи «Витрата» користувач повинен обрати підкатегорію,

ввести суму та на вибір вибір дати та часу) а далі натиснути на кнопку «Додати запис»

2. Дія «Перегляд історії транзакцій» включає в себе перегляд загальної історії, де містяться транзакції за категоріями, підкатегоріями, сумами, датою та часом.
3. Дія «Візуалізація графіків та діаграм » включає у себе вибір варіанту відображення даних, будь це табличний варіант, графік чи діаграма.

3.2.2. Специфікація варіантів використання

Специфікацію варіантів використання наведено у таблицях 3.1 – 3.9.

Таблиця 3.1

Таблиця варіанту використання «Внесення прибутку»

Контекст використання	Внесення прибутку
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Отриманий новий прибуток
Сценарій	1. Скористатися формой додавання нової транзакції. 2. Обрати підкатегорію. 3. Ввести суму. 4. Обрати дату прибутку (опціонально) 5. Обрати час. 6. Натиснути на кнопку «Додати запис» 7. Ввести опис транзакції.
Постумова	Оновлюється історія транзакцій, таблиця балансу, прибутку та діаграми з графіками.

Таблиця 3.2

Таблиця варіанту використання «Внесення витрати»

Контекст використання	Внесення прибутку
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Отримана нова витрата
Сценарій	1. Скористатися формой додавання нової транзакції. 2. Обрати підкатегорію. 3. Ввести суму. 4. Обрати дату витрати (опціонально) 5. Обрати час. 6. Натиснути на кнопку «Додати запис» 7. Ввести опис транзакції.
Постумова	Оновлюється історія транзакцій, таблиця балансу, витрат та діаграми з графіками.

Таблиця 3.3

Таблиця варіану використання «Перегляд історії транзакцій»

Контекст використання	Внесення прибутку
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Необхідність перевірити історію транзакцій
Сценарій	1. Скористатись таблицею «Історія транзакцій» 2. При необхідності скролити до потрібної дати.
Постумова	Відображається вся інформація транзакцій в одній таблиці.

Таблиця 3.4

Таблиця варіанту використання «Розподіл прибутку та витрат за категоріями»

Контекст використання	Перегляд діаграми, яка відображає співвідношення прибутку до витрат.
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Необхідність переглянути співвідношення витрат з прибутком
Сценарій	1. Скористатися кнопкою «Аналітичні інструменти» 2. Обрати кругову діаграму (Pie Chart)
Постумова	Відкривається вікно аналітичних інструментів, а при виборі кругової діаграми для перегляду стає доступним розподіл витрат та прибутку за категоріями.

Таблиця 3.5

Таблиця варіанту використання «Баланс за часом»

Контекст використання	Перегляд лінійного графіку зміни балансу за часом.
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Необхідність переглянути зміну балансу за часом
Сценарій	1. Скористатися кнопкою «Аналітичні інструменти» 2. Обрати лінійний графік (Line Chart)
Постумова	Відкривається вікно аналітичних інструментів, а при виборі лінійного графіку для перегляду стає доступною зміна балансу за часом (датою)

Таблиця 3.6

Таблиця варіанту використання «Баланс за часом»

Контекст використання	Перегляд лінійного графіку зміни балансу за часом.
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Необхідність переглянути зміну балансу за часом
Сценарій	1. Скористатися кнопкою «Аналітичні інструменти» 2. Обрати лінійний графік (Line Chart)
Постумова	Відкривається вікно аналітичних інструментів, а при виборі лінійного графіку для перегляду стає доступною зміна балансу за часом (датою)

Таблиця 3.7

Таблиця варіанту використання «Суми витрат та прибутку за часом»

Контекст використання	Перегляд гістограми сум витрат, прибутку за часом.
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Необхідність переглянути співвідношення прибутку та витрат за часом на гістограмі.
Сценарій	1. Скористатися кнопкою «Аналітичні інструменти» 2. Обрати гістограму (Bar Chart)
Постумова	Відкривається вікно аналітичних інструментів, а при виборі гістограми для перегляду стає доступною історія витрат та прибутку за часом (датою)

Таблиця 3.8

Таблиця варіанту використання «Загальна сума прибутку та витрат за категорією»

Контекст використання	Перегляд точкової діаграми загальних сум.
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Необхідність переглянути на що та скільки витрачається коштів.
Сценарій	1. Скористатися кнопкою «Аналітичні інструменти» 2. Точкову діаграму (Scatter Chart)
Постумова	Відкривається вікно аналітичних інструментів, а при виборі точковою діаграми для перегляду стає доступною статистика загальних витрат, доходу за всіма підкатегоріями.

Таблиця 3.9

Таблиця варіанту використання «Прибуток, витрати та баланс»

Контекст використання	Перегляд всіх транзакцій за категоріями та перевірка балансу (щомісячна)
Діюча особа	Користувач
Передумова	Додаток працює
Тригер	Необхідність переглянути усі транзакції лише прибутку чи витрати, або переглянути щомісячний залишок балансу (скільки було витрачено за місяць та зароблено)
Сценарій	– Перегляд прибутку: 1. Скористатися кнопкою «Аналітичні інструменти» 2. Таблиця прибутку (Income Table) – Перегляд витрат: 1. Скористатися кнопкою «Аналітичні інструменти» 2. Таблиця витрат (Expense Table) – Перегляд балансу: 1. Скористатися кнопкою «Аналітичні інструменти» 2. Таблиця балансу (Balance Table)

Продовження таблиці 3.9.

Постумова	Відкривається вікно аналітичних інструментів, при потребності користувача він вибирає, що йому треба переглянути, та відкривається відповідне вікно (з повною інформаціє про: прибуток, витрати чи баланс)

3.3. Проектування моделі бази даних

Для ефективного фінансового обліку та взаємодії з користувачами було вирішено використовувати текстові файли для зберігання інформації про транзакції [25]. Цей метод має кілька обґрунтованих переваг:

– Простота інтеграції:

Текстові файли прості у використанні і легко інтегруються. Їх легко створювати, змінювати та передавати, що робить їх корисними для зберігання невеликих обсягів даних.

– Простота використання:

Текстові файли можна переглядати та редагувати, що корисно для користувачів, які віддають перевагу простоті та прозорості.

– Незалежність від платформи:

Текстові файли є незалежним від платформи носієм інформації, тому їх можна без проблем використовувати в широкому спектрі операційних систем.

У майбутньому планується перехід на більш потужні та досконалі системи зберігання даних, наприклад, за рахунок використання реляційних баз даних (БД). Цей крок може стати необхідним у разі збільшення обсягу даних або розширення функціональності програмного продукту. Для подальшого розвитку продукту наперед було створено ER - діаграму.

Діаграма "сутність-зв'язок" (ER-діаграма) - це графічний інструмент, який використовується для моделювання структури бази даних. ER-діаграми [26] використовуються для опису сутностей в базі даних, їх атрибутів і зв'язків між ними. ER-діаграми представляють модель "сутність-зв'язок" (ER-модель, яка відображає концепції ER-моделі).

Основні елементи ER-діаграми:

– Сутність (Entities):

Сутність представляє об'єкт або концепцію, яка має значення і може зберігатися в базі даних.

– Атрибути (Attributes):

Атрибути є властивостями сутності і описують характеристики сутності.

– Зв'язки (Relationships):

Зв'язки вказують на зв'язки між сутностями.

– Ключі (Keys):

Ключі визначають унікальність сутності або зв'язку. Кожна сутність має первинний ключ, який ідентифікує конкретний запис.

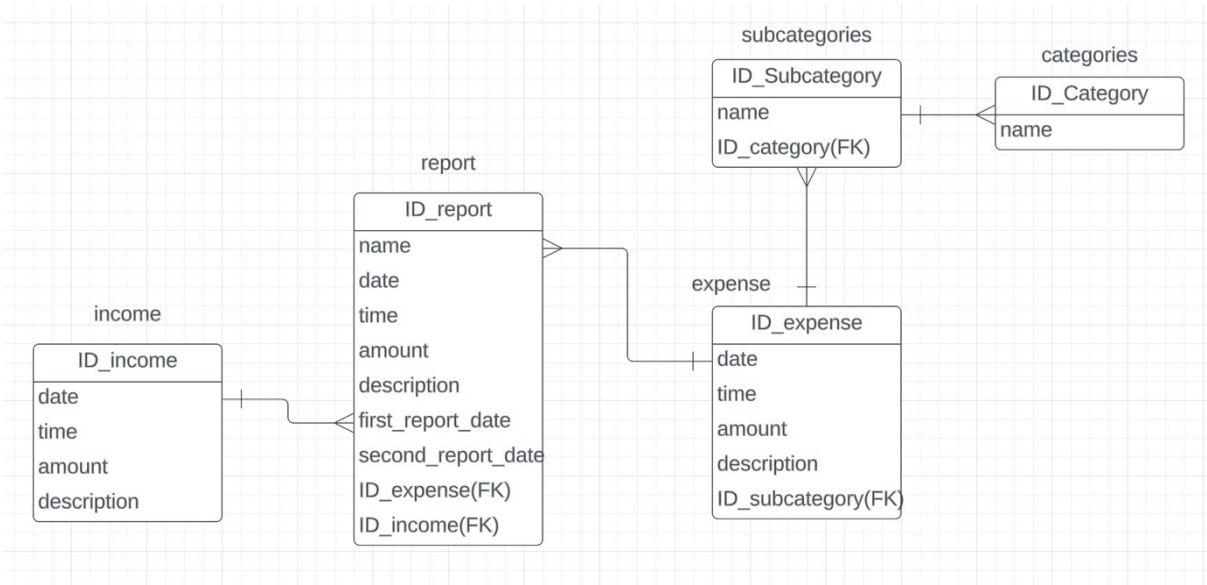

Рис. 3.5. Логічна модель даних

Дана база даних буде містити 5 таблиць, що мають певні атрибути:

1. Таблиця "income" (Дохід):

– ID_income (Ідентифікатор доходу): Унікальний ідентифікатор кожного доходу.

– date (Дата): Дата отримання доходу.

– time (Час): Час отримання доходу.

– amount (Сума): Кількість грошей, які надійшли від користувача.

– description (Опис): Опис доходу.

2. Таблиця "expense" (Витрати):

– ID_expense (Ідентифікатор витрати): Унікальний ідентифікатор кожної витрати.
– date (Дата): Дата здійснення витрати.
– time (Час): Час здійснення витрати.
– amount (Сума): Кількість грошей, яку користувач витратив.
– description (Опис): Опис витрати.
– ID_subcategory (Ідентифікатор підкатегорії): Зовнішній ключ (FK), який посилається на поле "subcategory_id" таблиці "subcategories".

3. Таблиця "report" (Звіт):

– ID_report (Ідентифікатор звіту): Унікальний ідентифікатор кожного звіту.
– name (Назва): Назва звіту.
– date (Дата): Дата створення звіту.
– time (Час): Час створення звіту.
– amount (Сума): Загальна сума доходів та витрат у звіті.
– description (Опис): Опис звіту.
– first_report_date, second_report_date: Додаткові дати для деталізації періоду звіту.
– ID_expense (Ідентифікатор витрати): Зовнішній ключ (FK), який посилається на поле "ID_expense" таблиці "expense".
– ID_income (Ідентифікатор доходу): Зовнішній ключ (FK), який посилається на поле "ID_income" таблиці "income".

4. Таблиця "categories" (Категорії):

– ID_Category (Ідентифікатор категорії): Унікальний ідентифікатор кожної категорії.
– name (Назва): Назва категорії.

5. Таблиця "subcategories" (Підкатегорії):

- ID_Subcategory (Ідентифікатор підкатегорії): Унікальний ідентифікатор кожної підкатегорії.
- name (Назва): Назва підкатегорії.
- ID_category (Ідентифікатор категорії): Зовнішній ключ (FK), який посилається на поле "ID_Category" таблиці "categories".

3.4. Архітектура додатку обліку особистих фінансів

Архітектура програмного додатку - це високорівневе планування та організація, що визначає структуру та взаємодію елементів, які складають систему [27]. Архітектура визначає, як взаємодіють компоненти програми і як вони організовані в системі.

Монолітна архітектура обрана з метою спрощення розробки та управління проектом, а також для швидкого впровадження та виправлення невеликих змін [28].

Основні характеристики використання монолітної архітектури включають:

- Однокомпонентність:

Усі функції та модулі системи обліку фінансів знаходяться в одному кодовому базисі.

- Внутрішня взаємодія:

Компоненти програми взаємодіють між собою без необхідності мережевого обміну даними.

- Одномовна розгортання:

Всі частини програмного продукту розгортаються одночасно на вибраному сервері або платформі.

- Загальний кодовий базис:

Весь код системи обліку фінансів зосереджений в єдиному репозиторії та компілюється в єдиний виконуваний файл.

– Масштабування:

Розширення функціональності можливе за рахунок збільшення обсягу ресурсів апаратного забезпечення.

– Зручність для розробки та тестування:

Один кодовий базис полегшує процес розробки та тестування, сприяючи зручності управління.

– Одномоментний реліз:

Оновлення та виправлення випускаються в одномоментно, що спрощує управління версіями програми.

До компонентів архітектури додатку обліку особистих фінансів відносяться:

1. Користувач: Основна вхідна точка взаємодії з системою. Вводить команди, отримує результати.
2. Система обліку фінансів: Логічний блок, що об'єднує в собі логіку обробки команд та взаємодії з базою даних у вигляді текстового файлу.
3. Обробник команд: Складова частина системи обліку фінансів, що приймає команди від користувача та взаємодіє з текстовим файлом бази даних для виконання операцій.
4. База даних (Текстовий файл): Місце зберігання та організації даних про фінанси. Використовується системою обліку фінансів для отримання та зберігання інформації.

Архітектура додатку зображена на рис. 3.6.

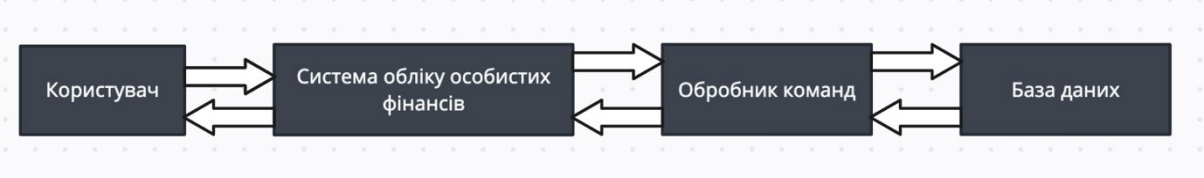

Рис. 3.6. Архітектура додадку обліку особистих фінансів

3.5. Візуалізація та функціональність програмного забезпечення

Програмне забезпечення для обліку особистих фінансів, розроблене з використанням JavaFX, створене з урахуванням простоти та зручності використання для різних користувачів. Завдяки відмінній візуалізації та розширеній функціональності, програма надає користувачам можливість ефективно вести облік своїх фінансів та аналізувати їх стан.

3.5.1. Головна Форма

Після запуску програми відкривається головне вікно, яке включає:

– Меню файл:

В меню "Файл" доступні опції "Зберегти", "Завантажити" та "Вийти", які дозволяють користувачеві зберігати та завантажувати дані, а також виходити з програми.

– Блок додавання операції:

В цьому блоку користувач може швидко внести нову транзакцію, обравши категорію, вказавши суму, дату та опис операції.

– Таблиця історії транзакцій:

Під блоком додавання транзакції розташована таблиця, де відображається історія всіх здійснених фінансових операцій. Кожен рядок містить дані про дату, суму, категорію та опис транзакції.

– Кнопка аналітичні інструменти:

Надає доступ до графіків, діаграм та таблиць, що надають користувачеві визначення структури та динаміки його фінансових операцій.

На рис. 3.7 можна побачити головний екран програми.

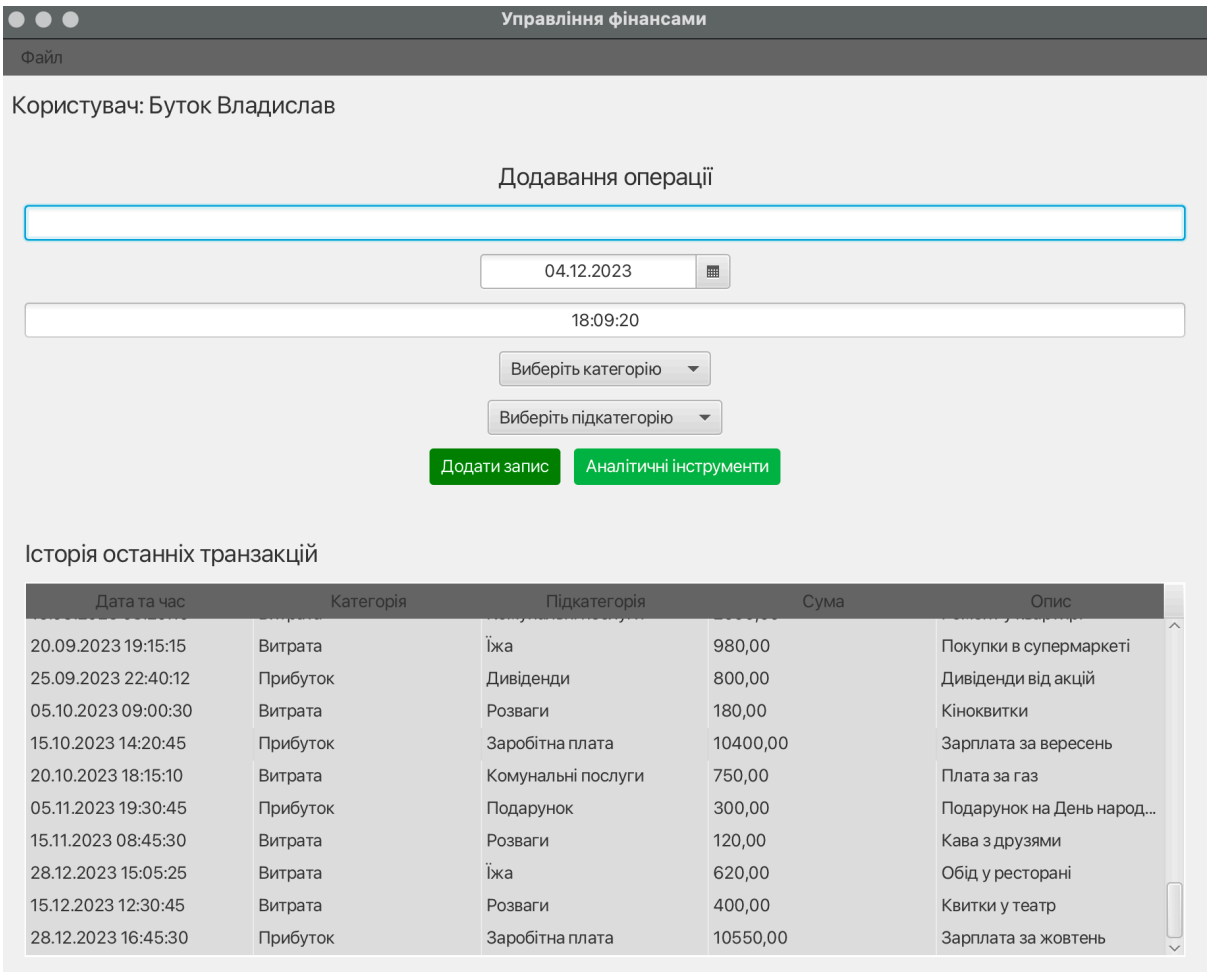

Рис. 3.7. Головна форма екрану програми

Меню «Файл» (рис. 3.8) містить три основні опції: «Зберегти», «Завантажити» та «Вийти». Ці опції дозволяють користувачеві зберігати та завантажувати дані, а також виходити з програми з хот-кеями для швидкого використання.

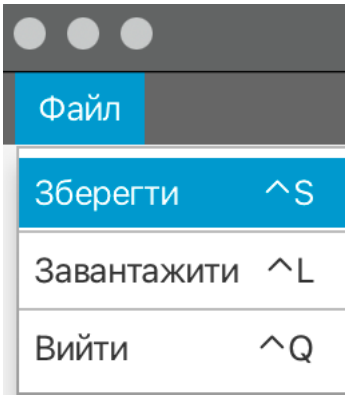

Рис. 3.8. Меню файл

3.5.2. Створення нової операції та їх історія транзакцій

Блок "Додати Транзакцію" (рис. 3.9) дозволяє користувачеві вводити нові фінансові операції. Таблиця "Історія Транзакцій" (рис. 3.10) відображає історію всіх здійснених транзакцій.

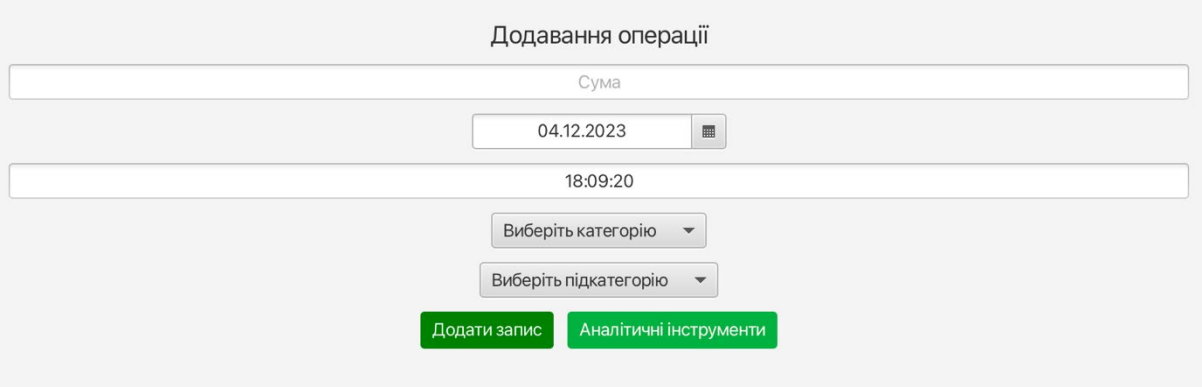

Рис. 3.9. Блок створення нової транзакції

Історія останніх транзакцій

Дата та час	Категорія	Підкатегорія	Сума	Опис
20.09.2023 19:15:15	Витрата	Їжа	980,00	Покупки в супермаркеті
25.09.2023 22:40:12	Прибуток	Дивіденди	800,00	Дивіденди від акцій
05.10.2023 09:00:30	Витрата	Розваги	180,00	Кіноквитки
15.10.2023 14:20:45	Прибуток	Заробітна плата	10400,00	Зарплата за вересень
20.10.2023 18:15:10	Витрата	Комунальні послуги	750,00	Плата за газ
05.11.2023 19:30:45	Прибуток	Подарунок	300,00	Подарунок на День народ...
15.11.2023 08:45:30	Витрата	Розваги	120,00	Кава з друзями
28.12.2023 15:05:25	Витрата	Їжа	620,00	Обід у ресторані
15.12.2023 12:30:45	Витрата	Розваги	400,00	Квитки у театр
28.12.2023 16:45:30	Прибуток	Заробітна плата	10550,00	Зарплата за жовтень

Рис. 3.10. Таблиця «Історія останніх транзакцій»

3.5.3. Аналітичні Інструменти

Кнопка "Аналітичні Інструменти" (рис. 3.11) відкриває розділ з графіками, таблицями та діаграмами, які надають користувачеві можливість аналізувати свої фінансові дані.

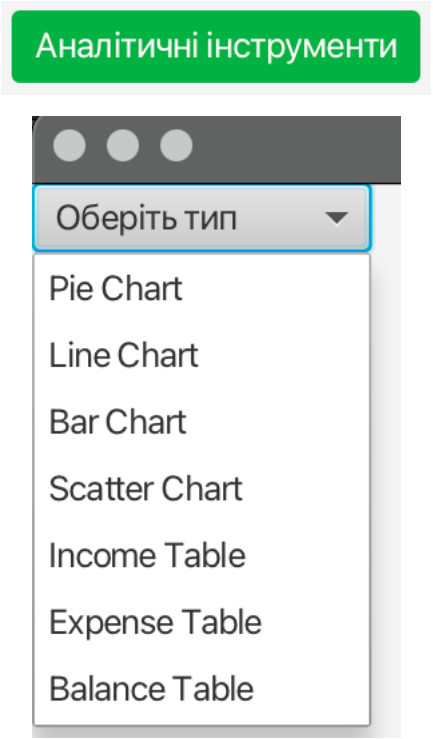

Рис. 3.11. Вибір аналітичних інструментів

Включає в себе такі інструменти візуалізація, як:

– Графік кругової діаграми (Pie Chart)

Вибір типу «Pie Chart» (рис. 3.12) надає інформацію про розподіл витрат за різними категоріями у вигляді кругової діаграми. Зручний для швидкого сприйняття головних напрямків витрат.

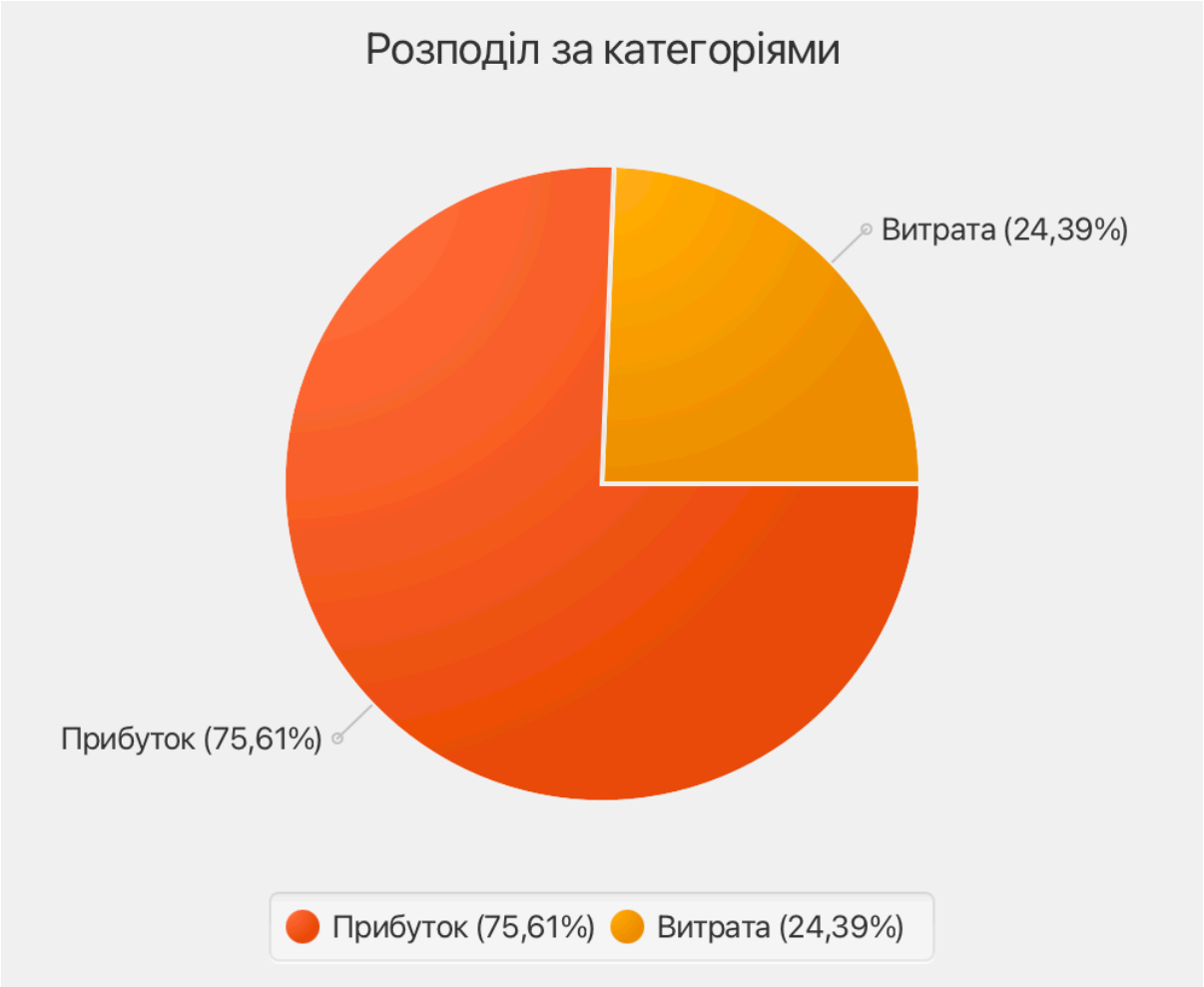

Рис. 3.12. Графік кругової діаграми

– Графік стовпчастої діаграми (Bar Chart)

Вибір типу «Bar Chart» (рис. 3.13) відображає кількість та обсяг фінансових транзакцій для кожної категорії у формі стовпчастої діаграми. Цей графік дозволяє користувачеві легко порівняти обсяги та виділити найбільш витратні чи прибуткові категорії.

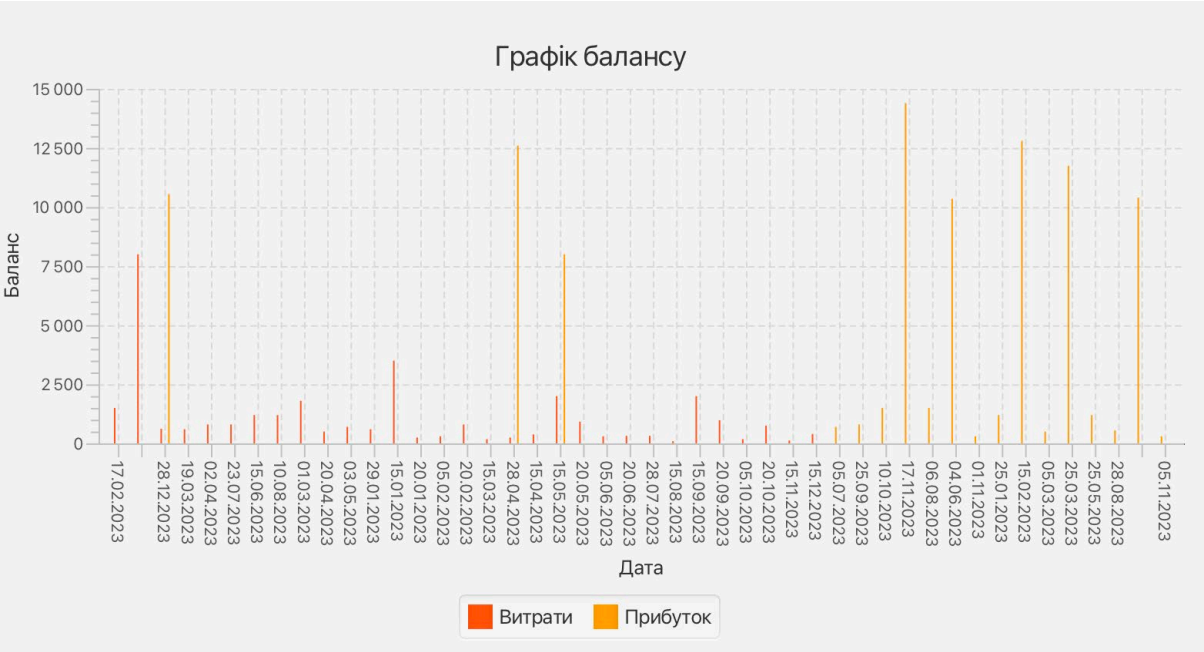

Рис. 3.13. Графік стовпчастої діаграми

– Лінійний графік (Line Chart)

Вибір типу «Line Chart» (рис. 3.14) демонструє динаміку зміни фінансового стану протягом певного періоду часу. Цей графік дозволяє виявляти тенденції та зміни в фінансових параметрах, що є важливим для планування бюджету.

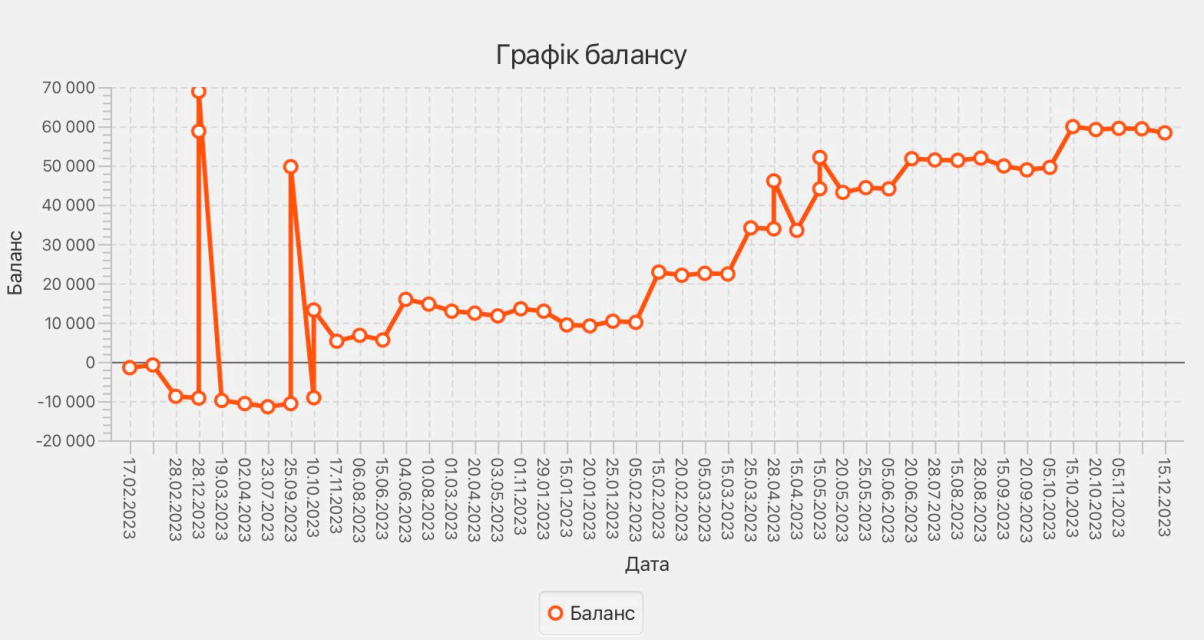

Рис. 3.14. Лінійний графік

- Діаграма розсіювання (Scatter Chart)

Обираючи «Scatter Chart» (рис. 3.15), користувач може вивчити взаємозв'язок між двома фінансовими параметрами. Це дозволяє виявити кореляції та визначити, як один параметр впливає на інший.

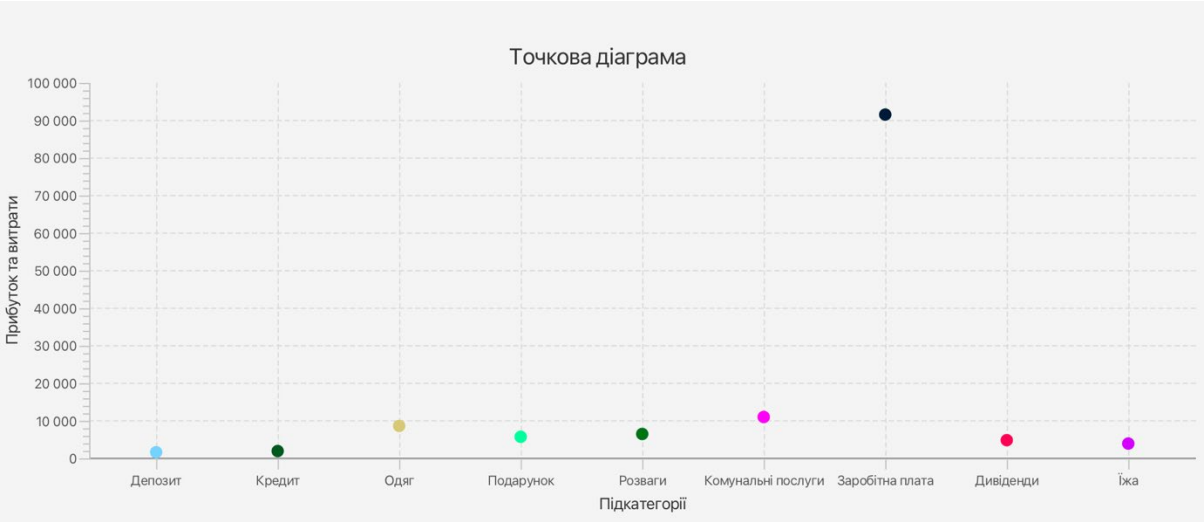

Рис 3.15. Діаграма розсіювання.

– Таблиця прибутку (Income Table)

Модуль «Таблиця Доходів» (рис. 3.16) надає детальний перегляд всіх доходів, включаючи дату, час, суму та опис. Ця таблиця дозволяє користувачеві систематизувати та аналізувати свої доходи, визначати їхні джерела та визначати, як різні доходи впливають на фінансовий стан.

Таблиця прибутку

Дата та час	Категорія	Підкатегорія	Сума
04.06.2023 01:45:30	Прибуток	Заробітна плата	10350,00
10.10.2023 05:20:20	Прибуток	Подарунок	1500,00
01.11.2023 22:22:33	Прибуток	Подарунок	300,00
25.01.2023 22:40:12	Прибуток	Дивіденди	1200,00
15.02.2023 14:20:45	Прибуток	Заробітна плата	12800,00
05.03.2023 19:30:45	Прибуток	Подарунок	500,00
25.03.2023 22:10:12	Прибуток	Заробітна плата	11750,00
28.04.2023 16:45:30	Прибуток	Заробітна плата	12600,00
25.05.2023 22:40:12	Прибуток	Дивіденди	1200,00
15.05.2023 14:20:45	Прибуток	Заробітна плата	8000,00
28.08.2023 16:45:30	Прибуток	Заробітна плата	550,00
25.09.2023 22:40:12	Прибуток	Дивіденди	800,00
15.10.2023 14:20:45	Прибуток	Заробітна плата	10400,00
05.11.2023 19:30:45	Прибуток	Подарунок	300,00
28.12.2023 16:45:30	Прибуток	Заробітна плата	10550,00

Загальна сума: 101700,00

Рисунок 3.16 – Таблиця прибутку.

– Таблиця витрат (Expense Table)

«Таблиця Витрат» (рис. 3.17) включає в себе усю інформацію про витрати, включаючи дату, час, суму та опис. Цей модуль надає можливість детального аналізу витрат за різними категоріями та підкатегоріями.

Рис. 3.17. Таблиця витрат

– Таблиця балансу (Balance Table)

«Таблиця Балансу» (рис. 3.18) відображає фінансовий баланс на певний момент часу. Вона включає в себе інформацію про наявність коштів, доходи та витрати, допомагаючи користувачеві отримати чіткий огляд свого поточного фінансового стану.

Річна таблиця загальних сум

Місяць	Дохід	Витрата	Баланс
Січень	1200.0	4350.0	-3150.0
Лютий	12800.0	10600.0	2200.0
Березень	12250.0	2580.0	9670.0
Квітень	12600.0	1930.0	10670.0
Травень	9200.0	3620.0	5580.0
Червень	10350.0	1820.0	8530.0
Липень	700.0	1120.0	-420.0
Серпень	2050.0	1290.0	760.0
Вересень	1600.0	2980.0	-1380.0
Жовтень	13400.0	930.0	12470.0
Листопад	15000.0	120.0	14880.0
Грудень	10550.0	1470.0	9080.0

Рис. 3.18. Таблиця щомісячного балансу

ВИСНОВКИ

В даній кваліфікаційній роботі була розроблена програма для обліку особистих фінансів. Метою проекту є полегшення ведення обліку фінансових транзакцій та надання можливості користувачеві аналізувати свій фінансовий стан.

Основні функції додатку включають додавання, редагування та видалення фінансових операцій. Важливою особливістю є можливість візуального представлення фінансових даних у вигляді графіків, діаграм та таблиць через інтерфейс користувача (GUI), що значно полегшує аналіз і сприяє кращому розумінню фінансового стану.

Під час розробки додатку були вирішені такі завдання:

- аналіз предметної галузі управління особистими фінансами;
- проектування та реалізація структури для зберігання фінансової інформації;
- створення інтерфейсу користувача (GUI) для зручного введення та відображення фінансових даних;
- розробка механізму візуалізації графіків та діаграм.
- розробка супровідної документації.

Додаток було створено з використанням мови програмування JavaFX для розробки графічного інтерфейсу та можливості збереження фінансової інформації. Під час розробки використовувалось інтегроване середовище розробки IntelliJ IDEA, що дозволило забезпечити продуктивний та зручний процес програмування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Белла, Я. К. (2017). "Оцінка фінансової стабільності підприємства на основі аналізу фінансового балансу." Науковий вісник НГУ, 1(2), 53-60.

2. Ольховик, О. В. (2018). "Аналіз доходів та витрат як основа фінансового планування." Економіка та управління підприємствами, 4(74), 63-68.

3. Дімітрова, Т. М. (2016). "Методи аналізу інвестиційних можливостей: оцінка ризиків та доходності." Вісник Львівського національного університету, 1(38), 136-143.

4. Павлов, В. І. (2019). "Аналіз кредитів та боргів: показники та методи." Фінанси, облік і аудит, 3, 108-115.

5. Красікова, І. С. (2017). "Аналіз ризиків у фінансовому управлінні: підходи та методи." Економіка та суспільство, 10, 248-254.

6. Макаров, П. А. (2018). "Методи аналізу портфеля інвестицій: теорія та практика." Фінансовий менеджмент, 4(60), 29-37.

7. Brigham, E. F., Houston, J. F. (200x). "Fundamentals of Financial Management Twelfth Edition." University of Florida.

8. Ramsey, D. (2003). "The Total Money Makeover: A Proven Plan for Financial Fitness."

9. Adusei, M. (2003). "Personal Financial Planning and Financial Satisfaction: Self-Control as a Mediator." Journal of Financial Counseling and Planning, DOI:10.1891/JFCP-2021-0074.

10. "Top languages used in 2022" [Електронний ресурс] – Режим доступу: https://octoverse.github.com/2022/top-programming-languages#:~:text=Top%20languages%20used%20in%202022,machine%20learning%20and%20data%20science (остання дата звернення 17.12.2023).

11. Eckel, B. "Thinking in Java" (4-е вид.). Нью-Джерсі: Prentice Hall, 2006.

12. Bloch, J. "Effective Java" (3-тє вид.). Pearson Addison-Wesley, 2019.

13. "Prototyping User Experience." UXmatters [Електронний ресурс] – Режим доступу: https://www.uxmatters.com/mt/archives/2019/01/prototyping–user–

experience.php (остання дата звернення 20.11.2023).

14. "Java Swing Vs JavaFX Comparisons — Which Is Better?" [Електронний ресурс] – Режим доступу: https://www.javaassignmenthelp.com/blog/swing-vs-javafx/ (остання дата звернення 12.11.2023).

15. "JavaFX UI Controls" [Електронний ресурс] – Режим доступу: https://docs.oracle.com/javafx/2/ui_controls/overview.html (остання дата звернення 15.11.2023).

16. "Maven Project Builder" [Електронний ресурс] – Режим доступу: https://maven.apache.org/what-is-maven.html (остання дата звернення 30.11.2023).

17. "What is a POM.file?" [Електронний ресурс] – Режим доступу: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html (остання дата звернення 28.11.2023).

18. "The 3 Best Java IDEs to Use in 2022" [Електронний ресурс] – Режим доступу: https://www.jrebel.com/blog/best-java-ide (остання дата звернення 01.12.2023).

19. "IntelliJ IDEA overview" [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html (остання дата звернення 10.12.2023).

20. Ушакова, І. О. "Проектування інформаційних систем: Практикум." Харків: Вид. ХНЕУ ім. С. Кузнеця, 2015. 250 с.

21. "Методологія IDEF0" [Електронний ресурс] – Режим доступу: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema6/tema6_2 (остання дата звернення 18.11.2023).

22. "IDEF0 як інструмент моделювання процесів – довідник з роботи з діаграмами IDEF0" [Електронний ресурс] – Режим доступу: http://www.logists.by/library/view/instrymenty-biznes-modelirovania (остання дата звернення 25.11.2023).

23. "Functional Decomposition" [Електронний ресурс] – Режим доступу: https://www.maxzosim.com/functional-decomposition/ (остання дата звернення 06.12.2023).

24. "Уніфікована мова моделювання (UML). Конструктивні блоки UML. Діаграми варіантів використання" [Електронний ресурс] – Режим доступу: http://iwanoff.inf.ua/oop_kn/LabTraining05.html (остання дата звернення: 12.12.2023).

25. Єрьоміна, Н. В. "Проектування баз даних: Навчальний посібник." – К.: КНЕУ, 1998. – 208 с.

26. "Entity Relationship Diagram (ERD) - What is an ER Diagram?"

[Електронний ресурс] – Режим доступу: https://www.smartdraw.com/entity-relationship-diagram (остання дата звернення: 08.12.2023).

27. "Принципи проектування, архітектури програмного забезпечення"

[Електроний ресурс] – Режим доступу: http://dspace.wunu.edu.ua/bitstream/316497/24194/1/опорний%20конспект%20лекцій.pdf (остання дата звернення: 02.12.2023).

28. "Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless?" [Електронний ресурс] – Режим доступу: https://rubygarage.org/blog/monolith-soa-microservices-serverless (остання дата звернення 10.12.2023).

29. Комп’ютерні науки та інформаційні технології: матеріали XVIII Міжнародної конференції з проблем використання інформаційних технологій в освіті, науці та промисловості, 8 грудн. 2023 р., Дніпро, Україна / Буток, В. О., Кабак, Л.В., Мороз, Б. І. "Сучасні технології управління особистими фінансами за допомогою аналітичних інструментів"

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

```
package com.dniprotech.financialaccounting;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.chart.*;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.input.KeyCombination;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.paint.Paint;
import javafx.scene.shape.Circle;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.util.StringConverter;

import java.io.*;
import java.text.SimpleDateFormat;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.util.*;
import java.util.stream.Collectors;

public class FinanceManagerApp extends Application {
    private Stage primaryStage;
    private BorderPane root;
    private String currentUser;
    private SimpleDoubleProperty balance;
    private ObservableList<Category> categories;
    private Map<Category, List<Subcategory>> categorySubcategoriesMap;
    private TableView<Transaction> transactionHistoryTableView;
    private TableView<BalanceCategory> balanceTableView;
    private PieChart pieChart;
    private TableView<Transaction> incomeCategoryTableView;
    private TableView<Transaction> expenseCategoryTableView;

    private LineChart<String, Number> lineChart;
    private BarChart<String, Number> barChart;
```

```
private ScatterChart<String, Number> scatterChart;
private TextField timeField;
private Label incomeTotalLabel = new Label();
private Label expenseTotalLabel = new Label();

public static void main(String[] args) {
    launch(args);
}

@Override
public void start(Stage primaryStage) {
    this.primaryStage = primaryStage;
    primaryStage.setTitle("Управління фінансами");
    String userName = getUserInfo();
    currentUser = userName;


    String userDataFileName = userName + "_transactions.txt";

    File userDataFile = new File(userDataFileName);
    if (userDataFile.exists()) {
        try {
            loadDataFromFile(userDataFile);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    root = new BorderPane();

    MenuBar menuBar = new MenuBar();
    menuBar.prefWidthProperty().bind(primaryStage.widthProperty());

    Menu fileMenu = new Menu("Файл");

    MenuItem saveItem = new MenuItem("Зберегти");
    saveItem.setAccelerator(KeyCombination.keyCombination("Ctrl+S"));
    saveItem.setOnAction(e -> {
        try {
            File userDir = new File("userdata");
            if (!userDir.exists()) {
                userDir.mkdir();
            }

            File userFile = new File(userDir, currentUser + "_transactions.txt");
            saveDataToFile(userFile);
        } catch (IOException ioException) {
            ioException.printStackTrace();
        }
    });

    MenuItem loadItem = new MenuItem("Завантажити");
```

```
        loadItem.setAccelerator(KeyCombination.keyCombination("Ctrl+L"));
        loadItem.setOnAction(e -> {
            try {
                FileChooser fileChooser = new FileChooser();
                fileChooser.setTitle("Оберіть файл для завантаження");
                File selectedFile = fileChooser.showOpenDialog(primaryStage);

                if (selectedFile != null) {
                    List<Transaction> loadedTransactions = readTransactionsFromFile(selectedFile);
                    addTransactionsToTable(loadedTransactions);
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        });

        MenuItem exitItem = new MenuItem("Вийти");
        exitItem.setAccelerator(KeyCombination.keyCombination("Ctrl+Q"));
        exitItem.setOnAction(e -> primaryStage.close());

        fileMenu.getItems().addAll(saveItem, new SeparatorMenuItem(), loadItem, new
SeparatorMenuItem(), exitItem);
        menuBar.getMenus().add(fileMenu);


        menuBar.getStyleClass().add("menu-bar");
        fileMenu.getStyleClass().add("menu");
        saveItem.getStyleClass().add("menu-item");
        exitItem.getStyleClass().add("menu-item");

        VBox userInfoBox = new VBox(10);
        userInfoBox.getStyleClass().add("vbox-user-info");
        Label userInfoLabel = new Label("Користувач: " + currentUser);
        userInfoLabel.getStyleClass().add("label-user-info");


        userInfoBox.getChildren().addAll(userInfoLabel);
        root.setTop(new VBox(menuBar, userInfoBox));

        VBox transactionInputBox = new VBox(10);
        transactionInputBox.setPadding(new Insets(20));
        Label transactionLabel = new Label("Додавання операції");
        transactionLabel.setStyle("-fx-font-size: 18px;");
        TextField amountField = new TextField();
        amountField.setPromptText("Сума");

        DatePicker datePicker = new DatePicker();
        datePicker.setId("datePicker");
        datePicker.setPromptText("Дата");
        datePicker.setValue(LocalDate.now());
```

```
timeField = new TextField();
timeField.setId("timeField");
timeField.setText(LocalTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")));
timeField.setPromptText("ЧЧ:ММ:СС");

categories = FXCollections.observableArrayList(
      new Category("Витрата", Arrays.asList(
            new Subcategory("Комунальні послуги"),
            new Subcategory("Їжа"),
            new Subcategory("Розваги"),
            new Subcategory("Одяг"),
            new Subcategory("Кредит"),
            new Subcategory("Депозит")
      )),
      new Category("Прибуток", Arrays.asList(
            new Subcategory("Заробітна плата"),
            new Subcategory("Дивіденди"),
            new Subcategory("Подарунок")
      ))
);

categorySubcategoriesMap = new HashMap<>();
for (Category category : categories) {
   categorySubcategoriesMap.put(category, category.getSubcategories());
}

ComboBox<Category> categoryComboBox = new ComboBox<>();
categoryComboBox.setPromptText("Виберіть категорію");
categoryComboBox.setConverter(new StringConverter<>() {
   @Override
   public String toString(Category category) {
      return category.getName();
   }

   @Override
   public Category fromString(String string) {
      return null;
   }
});
categoryComboBox.setItems(FXCollections.observableArrayList(categories));

ComboBox<String> subcategoryComboBox = new ComboBox<>();
subcategoryComboBox.setPromptText("Виберіть підкатегорію");

categoryComboBox.setOnAction(e -> {
   Category selectedCategory = categoryComboBox.getValue();
   if (selectedCategory != null) {
      List<Subcategory> subcategories = selectedCategory.getSubcategories();
      List<String> subcategoryNames = new ArrayList<>();
      for (Subcategory subcategory : subcategories) {
         subcategoryNames.add(subcategory.getName());
      }
```

```
subcategoryComboBox.setItems(FXCollections.observableArrayList(subcategoryNames));
            subcategoryComboBox.setDisable(false);
            subcategoryComboBox.getSelectionModel().clearSelection();
        } else {
            subcategoryComboBox.getItems().clear();
            subcategoryComboBox.setPromptText("Підкатегорія відсутня");
            subcategoryComboBox.setDisable(true);
        }
    });

    Button addRecordButton = new Button("Додати запис");
    addRecordButton.setOnAction(e -> {
        double amount = Double.parseDouble(amountField.getText());
        Category category = categoryComboBox.getValue();
        String subcategory = subcategoryComboBox.getValue();
        LocalDate selectedDate = datePicker.getValue();
        String selectedTime = timeField.getText();
        LocalDateTime dateTime = selectedDate.atTime(LocalTime.parse(selectedTime));
        addTransaction(amount, category, subcategory, dateTime);
    });

    addRecordButton.setStyle("-fx-background-color: green; -fx-text-fill: white;");
    addRecordButton.setMaxWidth(Double.MAX_VALUE);

    Button openChartsButton = new Button("Аналітичні інструменти");
    openChartsButton.setOnAction(e -> openChartsWindow());

    HBox buttonsBox = new HBox(10);
    buttonsBox.setAlignment(Pos.CENTER);
    buttonsBox.getChildren().addAll(addRecordButton, openChartsButton);

    transactionInputBox.getChildren().addAll(transactionLabel, amountField, datePicker,
timeField,
            categoryComboBox, subcategoryComboBox, buttonsBox);
    transactionInputBox.setAlignment(Pos.CENTER);
    root.setLeft(transactionInputBox);

    transactionHistoryTableView = createTransactionHistoryTableView();
    transactionHistoryTableView.setMaxHeight(Double.MAX_VALUE);
    transactionHistoryTableView.setPlaceholder(new Label("Історія транзакцій відсутня"));

    VBox transactionHistoryBox = new VBox(10);
    transactionHistoryBox.setPadding(new Insets(20));
    Label transactionHistoryLabel = new Label("Історія останніх транзакцій");
    transactionHistoryLabel.setStyle("-fx-font-size: 18px;");
    transactionHistoryBox.getChildren().addAll(transactionHistoryLabel,
transactionHistoryTableView);
    root.setRight(transactionHistoryBox);


    VBox rightBox = new VBox(menuBar, userInfoBox, transactionInputBox,
```

```
transactionHistoryBox);
        rightBox.getStyleClass().add("vbox-with-border");
        root.setRight(rightBox);

        primaryStage.setOnCloseRequest(e -> {
            try {
                File userFile = new File("userdata", currentUser + "_transactions.txt");
                saveDataToFile(userFile);
            } catch (IOException ioException) {
                ioException.printStackTrace();
            }
        });

        Scene scene = new Scene(root, 900, 700);
        scene.getStylesheets().add(getClass().getResource("/styles/styles.css").toExternalForm());
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void createAndConfigurePieChart() {
        pieChart = new PieChart();
        pieChart.setTitle("Розподіл за категоріями");
        updatePieChart();
    }


    private void createAndConfigureLineChart() {
        CategoryAxis xAxis = new CategoryAxis();
        NumberAxis yAxis = new NumberAxis();
        lineChart = new LineChart<>(xAxis, yAxis);
        lineChart.setTitle("Графік балансу");


        xAxis.setLabel("Дата");
        yAxis.setLabel("Баланс");


        updateLineChart();
    }

    private void createAndConfigureBarChart() {
        barChart = createBarChart();


        barChart.setTitle("Графік балансу");
        barChart.getXAxis().setLabel("Дата");
        barChart.getYAxis().setLabel("Баланс");

        updateBarChart(barChart);
    }

    private void createAndConfigureScatterChart() {
```

```
    scatterChart = createScatterChart();

    scatterChart.setTitle("Точкова діаграма");
    scatterChart.getXAxis().setLabel("Підкатегорії");
    scatterChart.getYAxis().setLabel("Прибуток та витрати");
    scatterChart.setLegendVisible(false);
    updateScatterChart(scatterChart);
}

private void createAndConfigureIncomeCategoryTableView() {
    incomeCategoryTableView = createIncomeCategoryTableView();
    updateIncomeCategoryTable();
}

private void createAndConfigureExpenseCategoryTableView() {
    expenseCategoryTableView = createExpenseCategoryTableView();
    updateExpenseCategoryTable();
}

private void createAndConfigureBalanceTableView() {
    balanceTableView = createBalanceTableView();
    updateBalanceTable();
}

private void openChartsWindow() {

    Stage chartsStage = new Stage();
    chartsStage.setTitle("Аналітичні інструменти");


    BorderPane mainLayout = new BorderPane();


    VBox chartBox = new VBox(20);
    chartBox.setAlignment(Pos.CENTER);


    ComboBox<String> chartSelector = new ComboBox<>();
    chartSelector.setPromptText("Оберіть тип");
    ObservableList<String> chartOptions = FXCollections.observableArrayList(
            "Pie Chart", "Line Chart", "Bar Chart", "Scatter Chart",
            "Income Table", "Expense Table", "Balance Table");
    chartSelector.setItems(chartOptions);


    chartSelector.setOnAction(event -> {
        String selectedOption = chartSelector.getValue();
        chartBox.getChildren().clear();

        if ("Pie Chart".equals(selectedOption)) {
            createAndConfigurePieChart();
            chartBox.getChildren().add(pieChart);
```

```
        } else if ("Line Chart".equals(selectedOption)) {
            createAndConfigureLineChart();
            chartBox.getChildren().add(lineChart);
        } else if ("Bar Chart".equals(selectedOption)) {
            createAndConfigureBarChart();
            chartBox.getChildren().add(barChart);
        } else if ("Scatter Chart".equals(selectedOption)) {
            createAndConfigureScatterChart();
            chartBox.getChildren().add(scatterChart);
        } else if ("Income Table".equals(selectedOption)) {
            createAndConfigureIncomeCategoryTableView();
            Label incomeLabel = new Label("Таблиця прибутку");
            incomeLabel.setStyle("-fx-font-size: 18;");
            incomeTotalLabel.setStyle("-fx-font-size: 16;");
            chartBox.getChildren().addAll(incomeLabel, incomeCategoryTableView,
incomeTotalLabel);
        } else if ("Expense Table".equals(selectedOption)) {
            createAndConfigureExpenseCategoryTableView();
            Label expenseLabel = new Label("Таблиця витрат");
            expenseLabel.setStyle("-fx-font-size: 18;");
            expenseTotalLabel.setStyle("-fx-font-size: 16;");
            chartBox.getChildren().addAll(expenseLabel, expenseCategoryTableView,
expenseTotalLabel);
        } else if ("Balance Table".equals(selectedOption)) {
            createAndConfigureBalanceTableView();
            Label balanceLabel = new Label("Річна таблиця загальних сум");
            balanceLabel.setStyle("-fx-font-size: 18;");
            chartBox.getChildren().addAll(balanceLabel, balanceTableView);
        }

    });


    mainLayout.setTop(chartSelector);
    mainLayout.setCenter(chartBox);


    Scene chartsScene = new Scene(mainLayout, 800, 600);


    chartsStage.setScene(chartsScene);


    chartsStage.show();
  }


  private Transaction createTransactionFromRecord(TransactionRecord transactionRecord) {
    LocalDate date = transactionRecord.getDate();
    LocalTime time = transactionRecord.getTime();
    LocalDateTime dateTime = LocalDateTime.of(date, time);
    String category = transactionRecord.getCategory();
```

```
        String subcategory = transactionRecord.getSubcategory();
        double amount = transactionRecord.getAmount();
        String description = transactionRecord.getDescription();
        return new Transaction(dateTime, category, subcategory, amount, description);
    }

    private TableView<Transaction> createIncomeCategoryTableView() {
        TableView<Transaction> incomeCategoryTableView = new TableView<>();

incomeCategoryTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLIC
Y);

        TableColumn<Transaction, LocalDateTime> incomeDateColumn = new
TableColumn<>("Дата та час");
        incomeDateColumn.setCellValueFactory(new PropertyValueFactory<>("dateTime"));
        incomeDateColumn.setCellFactory(column -> new TableCell<>() {
            private final DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss");

            @Override
            protected void updateItem(LocalDateTime item, boolean empty) {
                super.updateItem(item, empty);
                if (empty || item == null) {
                    setText(null);
                } else {
                    setText(formatter.format(item));
                }
            }
        });

        TableColumn<Transaction, String> incomeCategoryColumn = new
TableColumn<>("Категорія");
        incomeCategoryColumn.setCellValueFactory(new PropertyValueFactory<>("category"));

        TableColumn<Transaction, String> incomeSubcategoryColumn = new
TableColumn<>("Підкатегорія");
        incomeSubcategoryColumn.setCellValueFactory(new
PropertyValueFactory<>("subcategory"));

        TableColumn<Transaction, Double> incomeAmountColumn = new TableColumn<>("Сума");
        incomeAmountColumn.setCellValueFactory(new PropertyValueFactory<>("amount"));
        incomeAmountColumn.setCellFactory(tc -> new TableCell<>() {
            @Override
            protected void updateItem(Double amount, boolean empty) {
                super.updateItem(amount, empty);
                if (empty || amount == null) {
                    setText(null);
                } else {
                    setText(String.format("%.2f", amount));
                }
            }
        });
```

```
        incomeCategoryTableView.getColumns().addAll(incomeDateColumn,
incomeCategoryColumn, incomeSubcategoryColumn, incomeAmountColumn);

        return incomeCategoryTableView;
    }

    private TableView<Transaction> createExpenseCategoryTableView() {
        TableView<Transaction> expenseCategoryTableView = new TableView<>();

expenseCategoryTableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLIC
Y);

        TableColumn<Transaction, LocalDateTime> expenseDateColumn = new
TableColumn<>("Дата та час");
        expenseDateColumn.setCellValueFactory(new PropertyValueFactory<>("dateTime"));
        expenseDateColumn.setCellFactory(column -> new TableCell<>() {
            private final DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss");

            @Override
            protected void updateItem(LocalDateTime item, boolean empty) {
                super.updateItem(item, empty);
                if (empty || item == null) {
                    setText(null);
                } else {
                    setText(formatter.format(item));
                }
            }
        });

        TableColumn<Transaction, String> expenseCategoryColumn = new
TableColumn<>("Категорія");
        expenseCategoryColumn.setCellValueFactory(new PropertyValueFactory<>("category"));

        TableColumn<Transaction, String> expenseSubcategoryColumn = new
TableColumn<>("Підкатегорія");
        expenseSubcategoryColumn.setCellValueFactory(new
PropertyValueFactory<>("subcategory"));

        TableColumn<Transaction, Double> expenseAmountColumn = new
TableColumn<>("Сума");
        expenseAmountColumn.setCellValueFactory(new PropertyValueFactory<>("amount"));
        expenseAmountColumn.setCellFactory(tc -> new TableCell<>() {
            @Override
            protected void updateItem(Double amount, boolean empty) {
                super.updateItem(amount, empty);
                if (empty || amount == null) {
                    setText(null);
                } else {
                    setText(String.format("%.2f", amount));
                }
```

```
            }
        });

        expenseCategoryTableView.getColumns().addAll(expenseDateColumn,
expenseCategoryColumn, expenseSubcategoryColumn, expenseAmountColumn);

        return expenseCategoryTableView;
    }

    private void updateIncomeCategoryTable() {
        List<Transaction> transactions = transactionHistoryTableView.getItems();
        List<Transaction> incomeTransactions = transactions.stream()
                .filter(transaction -> "Прибуток".equals(transaction.getCategory()))
                .collect(Collectors.toList());

        incomeCategoryTableView.getItems().clear();
        incomeCategoryTableView.getItems().addAll(incomeTransactions);

        double totalIncome =
incomeTransactions.stream().mapToDouble(Transaction::getAmount).sum();
        updateTotalLabel(incomeTotalLabel, totalIncome);
    }

    private void updateExpenseCategoryTable() {
        List<Transaction> transactions = transactionHistoryTableView.getItems();
        List<Transaction> expenseTransactions = transactions.stream()
                .filter(transaction -> "Витрата".equals(transaction.getCategory()))
                .collect(Collectors.toList());

        expenseCategoryTableView.getItems().clear();
        expenseCategoryTableView.getItems().addAll(expenseTransactions);

        double totalExpense =
expenseTransactions.stream().mapToDouble(Transaction::getAmount).sum();
        updateTotalLabel(expenseTotalLabel, totalExpense);
    }


    private void updateTotalLabel(Label totalLabel, double totalAmount) {
        totalLabel.setText(String.format("Загальна сума: %.2f", totalAmount));
    }

    public ScatterChart<String, Number> createScatterChart() {
        CategoryAxis xAxis = new CategoryAxis();
        NumberAxis yAxis = new NumberAxis();
        ScatterChart<String, Number> scatterChart = new ScatterChart<>(xAxis, yAxis);
        scatterChart.setTitle("Scatter Chart");

        updateScatterChart(scatterChart);

        return scatterChart;
    }
```

```
    public void updateScatterChart(ScatterChart<String, Number> scatterChart) {
        scatterChart.getData().clear();

        List<Transaction> transactions = transactionHistoryTableView.getItems();


        Map<String, Paint> categoryColors = new HashMap<>();


        Map<String, Double> subcategorySums = new HashMap<>();

        for (Transaction transaction : transactions) {
            String category = transaction.getCategory();
            String subcategory = transaction.getSubcategory();
            double totalAmount = transaction.getIncome() - transaction.getExpense();


            if (!categoryColors.containsKey(subcategory)) {
                categoryColors.put(subcategory, getRandomColor());
            }


            if (subcategorySums.containsKey(subcategory)) {
                subcategorySums.put(subcategory, subcategorySums.get(subcategory) + totalAmount);
            } else {
                subcategorySums.put(subcategory, totalAmount);
            }
        }


        for (Map.Entry<String, Double> entry : subcategorySums.entrySet()) {
            String subcategory = entry.getKey();
            double totalAmount = entry.getValue();


            XYChart.Series<String, Number> series = new XYChart.Series<>();
            series.setName(subcategory);


            XYChart.Data<String, Number> dataPoint = new XYChart.Data<>(subcategory,
totalAmount);
            dataPoint.setNode(createDataPointNode(categoryColors.get(subcategory)));

            series.getData().add(dataPoint);

            scatterChart.getData().add(series);
        }
    }


    private Node createDataPointNode(Paint color) {
```

```
        Circle circle = new Circle(5);
        circle.setFill(color);
        return circle;
    }

    private Paint getRandomColor() {
        Random rand = new Random();
        return Color.rgb(rand.nextInt(256), rand.nextInt(256), rand.nextInt(256));
    }


    private PieChart createPieChart() {
        Map<String, Double> categoryBalances = new HashMap<>();
        double totalBalance = 0.0;

        for (Transaction transaction : transactionHistoryTableView.getItems()) {
            String category = transaction.getCategory();
            Double amount = transaction.getAmount();
            categoryBalances.merge(category, amount, Double::sum);
            totalBalance += amount;
        }

        PieChart chart = new PieChart();

        for (Map.Entry<String, Double> categoryEntry : categoryBalances.entrySet()) {
            String category = categoryEntry.getKey();
            double balance = categoryEntry.getValue();
            double percentage = (balance / totalBalance) * 100;
            String categoryLabel = category + " (" + String.format("%.2f%%", percentage) + ")";
            PieChart.Data categoryData = new PieChart.Data(categoryLabel, Math.abs(percentage));
            chart.getData().add(categoryData);
        }
        return chart;
    }

    private BarChart<String, Number> createBarChart() {
        CategoryAxis xAxis = new CategoryAxis();
        NumberAxis yAxis = new NumberAxis();
        BarChart<String, Number> chart = new BarChart<>(xAxis, yAxis);
        chart.setTitle("Суммарні витрати та прибутки");

        XYChart.Series<String, Number> expenseSeries = new XYChart.Series<>();
        XYChart.Series<String, Number> incomeSeries = new XYChart.Series<>();

        expenseSeries.setName("Витрати");
        incomeSeries.setName("Прибуток");


        for (Transaction transaction : transactionHistoryTableView.getItems()) {
            String date =
transaction.getDateTime().toLocalDate().format(DateTimeFormatter.ofPattern("dd.MM.yyyy"));
            double amount = transaction.getAmount();
```

```
            String category = transaction.getCategory();

            if ("Витрата".equals(category)) {
                expenseSeries.getData().add(new XYChart.Data<>(date, amount));
            } else if ("Прибуток".equals(category)) {
                incomeSeries.getData().add(new XYChart.Data<>(date, amount));
            }
        }

        chart.getData().addAll(expenseSeries, incomeSeries);

        return chart;
    }

    private void updateBarChart(BarChart<String, Number> barChart) {
        barChart.getData().clear();

        XYChart.Series<String, Number> expenseSeries = new XYChart.Series<>();
        XYChart.Series<String, Number> incomeSeries = new XYChart.Series<>();

        expenseSeries.setName("Витрати");
        incomeSeries.setName("Прибуток");


        for (Transaction transaction : transactionHistoryTableView.getItems()) {
            String date =
transaction.getDateTime().toLocalDate().format(DateTimeFormatter.ofPattern("dd.MM.yyyy"));
            double amount = transaction.getAmount();
            String category = transaction.getCategory();

            if ("Витрата".equals(category)) {
                expenseSeries.getData().add(new XYChart.Data<>(date, amount));
            } else if ("Прибуток".equals(category)) {
                incomeSeries.getData().add(new XYChart.Data<>(date, amount));
            }
        }

        barChart.getData().addAll(expenseSeries, incomeSeries);
    }


    private String getUserInfo() {
        TextInputDialog dialog = new TextInputDialog();
        dialog.setTitle("Авторизація");
        dialog.setHeaderText("Будь ласка, введіть ваше ім'я та прізвище:");
        dialog.setContentText("Ім'я та прізвище:");

        String result = "";
        while (result.isEmpty()) {
            Optional<String> userInput = dialog.showAndWait();
            if (userInput.isPresent()) {
                result = userInput.get();
```

```
            currentUser = result;

            File userFile = new File("userdata", currentUser + "_transactions.txt");
            if (userFile.exists()) {
                try {
                    loadDataFromFile(userFile);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            } else {
                System.out.println("Файл користувача не знайден: " + currentUser);
            }
        } else {
            System.exit(0);
        }
    }
    return result;
}

private void loadDataFromFile(File userFile) throws IOException {
    if (userFile.exists()) {
        try (BufferedReader reader = new BufferedReader(new FileReader(userFile))) {
            String line;
            while ((line = reader.readLine()) != null) {
                TransactionRecord transactionRecord = TransactionRecord.fromString(line);
                if (transactionRecord != null) {
                    Transaction transaction = createTransactionFromRecord(transactionRecord);
                    transactionHistoryTableView.getItems().add(transaction);
                }
            }
        }
    } else {
        System.out.println("Файл користувача не знайдено: " + currentUser);
    }
}

private LineChart<String, Number> createLineChart() {
    CategoryAxis xAxis = new CategoryAxis();
    NumberAxis yAxis = new NumberAxis();
    xAxis.setLabel("Дата");
    yAxis.setLabel("Баланс");

    LineChart<String, Number> chart = new LineChart<>(xAxis, yAxis);
    chart.setTitle("Зміна балансу з часом");

    XYChart.Series<String, Number> series = new XYChart.Series<>();
    series.setName("Баланс");


    for (Transaction transaction : transactionHistoryTableView.getItems()) {
        String date =
transaction.getDateTime().toLocalDate().format(DateTimeFormatter.ofPattern("dd.MM.yyyy"));
```

```
            double balanceChange = transaction.getAmount();
            series.getData().add(new XYChart.Data<>(date, balanceChange));
        }

        chart.getData().add(series);

        return chart;
    }

    private void updateLineChart() {
        XYChart.Series<String, Number> series = new XYChart.Series<>();
        series.setName("Баланс");

        double cumulativeBalance = 0.0;

        for (Transaction transaction : transactionHistoryTableView.getItems()) {
            String date =
transaction.getDateTime().toLocalDate().format(DateTimeFormatter.ofPattern("dd.MM.yyyy"));
            double balanceChange = transaction.getAmount();
            String category = transaction.getCategory();


            if ("Витрата".equals(category)) {
                cumulativeBalance -= balanceChange;
            } else if ("Прибуток".equals(category)) {
                cumulativeBalance += balanceChange;
            }

            series.getData().add(new XYChart.Data<>(date, cumulativeBalance));
        }

        lineChart.getData().setAll(series);
        lineChart.layout();
    }


    private void saveDataToFile(File userFile) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(userFile, true))) {
            for (Transaction transaction : transactionHistoryTableView.getItems()) {
                writer.write(transaction.toRecordString());
                writer.newLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void addTransaction(double amount, Category category, String subcategory,
LocalDateTime dateTime) {
        if (amount != 0 && category != null) {
            String operationType = category.getName();
            TextInputDialog descriptionDialog = new TextInputDialog();
```

```
            descriptionDialog.setTitle("Додавання запису");
            descriptionDialog.setHeaderText("Будь ласка, введіть опис операції (" + operationType +
"):");
            descriptionDialog.setContentText("Опис:");

            Optional<String> descriptionResult = descriptionDialog.showAndWait();
            if (descriptionResult.isPresent()) {
                String description = descriptionResult.get();
                String transactionInfo = dateTime.toString() + " - Категорія: " + category.getName() + "
- Підкатегорія: " + subcategory + ", Сума: " + amount + ", Опис: " + description;
                Transaction transaction = new Transaction(dateTime, category.getName(), subcategory,
amount, description);
                transactionHistoryTableView.getItems().add(transaction);

                if ("Витрата".equals(operationType) && balance != null) {
                    balance.set(balance.get() - amount);
                } else if ("Прибуток".equals(operationType) && balance != null) {
                    balance.set(balance.get() + amount);
                }

                updatePieChart();

                updateLineChart();
                updateBarChart(barChart);
                updateBalanceTable();
                updateScatterChart(scatterChart);
                saveTransactionToFile(currentUser, transactionInfo);

                if ("Прибуток".equals(operationType)) {
                    updateIncomeCategoryTable();
                } else if ("Витрата".equals(operationType)) {
                    updateExpenseCategoryTable();
                }
            }
        }
    }

    private void updatePieChart() {
        Map<String, Double> categoryBalances = new HashMap<>();
        double totalBalance = 0.0;

        for (Transaction transaction : transactionHistoryTableView.getItems()) {
            String category = transaction.getCategory();
            Double amount = transaction.getAmount();
            categoryBalances.merge(category, amount, Double::sum);
            totalBalance += amount;
        }

        pieChart.getData().clear();

        for (Map.Entry<String, Double> categoryEntry : categoryBalances.entrySet()) {
            String category = categoryEntry.getKey();
```

```
            double balance = categoryEntry.getValue();
            double percentage = (balance / totalBalance) * 100;
            String categoryLabel = category + " (" + String.format("%.2f%%", percentage) + ")";
            PieChart.Data categoryData = new PieChart.Data(categoryLabel, Math.abs(percentage));
            pieChart.getData().add(categoryData);
        }
    }

    private String getFormattedDateTime() {
        SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss");
        return sdf.format(new Date());
    }

    private void saveTransactionToFile(String user, String transactionInfo) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(user + "_transactions.txt",
true))) {
            writer.write(transactionInfo);
            writer.newLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private double calculateTotalIncome() {
        double totalIncome = 0.0;
        for (Transaction transaction : transactionHistoryTableView.getItems()) {
            if ("Прибуток".equals(transaction.getCategory())) {
                totalIncome += transaction.getAmount();
            }
        }
        return totalIncome;
    }

    private double calculateTotalExpense() {
        double totalExpense = 0.0;
        for (Transaction transaction : transactionHistoryTableView.getItems()) {
            if ("Витрата".equals(transaction.getCategory())) {
                totalExpense += transaction.getAmount();
            }
        }
        return totalExpense;
    }

    private TableView<Transaction> createTransactionHistoryTableView() {
        TableView<Transaction> tableView = new TableView<>();
        tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);

        TableColumn<Transaction, LocalDateTime> dateColumn = new TableColumn<>("Дата та
час");
        dateColumn.setCellValueFactory(new PropertyValueFactory<>("dateTime"));
        dateColumn.setCellFactory(column -> new TableCell<>() {
            private final DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy
```

```
HH:mm:ss");

        @Override
        protected void updateItem(LocalDateTime item, boolean empty) {
            super.updateItem(item, empty);
            if (empty || item == null) {
                setText(null);
            } else {
                setText(formatter.format(item));
            }
        }
    });

    TableColumn<Transaction, String> categoryColumn = new TableColumn<>("Категорія");
    categoryColumn.setCellValueFactory(new PropertyValueFactory<>("category"));

    TableColumn<Transaction, String> subcategoryColumn = new
TableColumn<>("Підкатегорія");
    subcategoryColumn.setCellValueFactory(new PropertyValueFactory<>("subcategory"));

    TableColumn<Transaction, Double> amountColumn = new TableColumn<>("Сума");
    amountColumn.setCellValueFactory(new PropertyValueFactory<>("amount"));
    amountColumn.setCellFactory(tc -> new TableCell<>() {
        @Override
        protected void updateItem(Double amount, boolean empty) {
            super.updateItem(amount, empty);
            if (empty || amount == null) {
                setText(null);
            } else {
                setText(String.format("%.2f", amount));
            }
        }
    });

    TableColumn<Transaction, String> descriptionColumn = new TableColumn<>("Опис");
    descriptionColumn.setCellValueFactory(new PropertyValueFactory<>("description"));
    dateColumn.setPrefWidth(125);
    categoryColumn.setPrefWidth(80);
    subcategoryColumn.setPrefWidth(125);
    amountColumn.setPrefWidth(70);
    descriptionColumn.setPrefWidth(180);


    tableView.getColumns().addAll(dateColumn, categoryColumn, subcategoryColumn,
amountColumn, descriptionColumn);
    return tableView;
  }

  private TableView<BalanceCategory> createBalanceTableView() {
    TableView<BalanceCategory> tableView = new TableView<>();
    tableView.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
```

```
        TableColumn<BalanceCategory, String> categoryColumn = new TableColumn<>("Місяць");
        categoryColumn.setCellValueFactory(new PropertyValueFactory<>("category"));

        TableColumn<BalanceCategory, Double> incomeColumn = new TableColumn<>("Дохід");
        incomeColumn.setCellValueFactory(new PropertyValueFactory<>("totalIncome"));

        TableColumn<BalanceCategory, Double> expenseColumn = new
TableColumn<>("Витрата");
        expenseColumn.setCellValueFactory(new PropertyValueFactory<>("totalExpense"));

        TableColumn<BalanceCategory, Double> balanceColumn = new TableColumn<>("Баланс");
        balanceColumn.setCellValueFactory(new PropertyValueFactory<>("totalBalance"));

        tableView.getColumns().addAll(categoryColumn, incomeColumn, expenseColumn,
balanceColumn);

        return tableView;
    }

    private void updateBalanceTable() {
        if (transactionHistoryTableView.getItems() == null) {
            return;
        }

        List<BalanceCategory> monthBalanceCategories = new ArrayList<>();

        List<String> monthLabels = Arrays.asList("Січень", "Лютий", "Березень", "Квітень",
"Травень", "Червень",
                "Липень", "Серпень", "Вересень", "Жовтень", "Листопад", "Грудень");

        for (int i = 0; i < 12; i++) {
            double totalIncomeForMonth = calculateTotalIncomeForMonth(i + 1);
            double totalExpenseForMonth = calculateTotalExpenseForMonth(i + 1);
            double totalBalanceForMonth = totalIncomeForMonth - totalExpenseForMonth;

            BalanceCategory monthBalanceCategory = new BalanceCategory(monthLabels.get(i));
            monthBalanceCategory.setTotalIncome(totalIncomeForMonth);
            monthBalanceCategory.setTotalExpense(totalExpenseForMonth);
            monthBalanceCategory.setTotalBalance(totalBalanceForMonth);

            monthBalanceCategories.add(monthBalanceCategory);
        }


        balanceTableView.getItems().setAll(monthBalanceCategories);
    }

    private double calculateTotalIncomeForMonth(int month) {
        List<Transaction> transactions = transactionHistoryTableView.getItems();
        return transactions.stream()
                .filter(transaction -> transaction.getMonth() == month &&
"Прибуток".equals(transaction.getCategory()))
```

```
            .mapToDouble(Transaction::getAmount)
            .sum();
    }

    private double calculateTotalExpenseForMonth(int month) {
        List<Transaction> transactions = transactionHistoryTableView.getItems();
        return transactions.stream()
                .filter(transaction -> transaction.getMonth() == month &&
"Витрата".equals(transaction.getCategory()))
                .mapToDouble(Transaction::getAmount)
                .sum();
    }


    public static class Transaction {
        private final LocalDateTime dateTime;
        private final String category;
        private final String subcategory;
        private final double amount;
        private final String description;

        public Transaction(LocalDateTime dateTime, String category, String subcategory, double
amount, String description) {
            this.dateTime = dateTime;
            this.category = category;
            this.subcategory = subcategory;
            this.amount = amount;
            this.description = description;
        }

        public LocalDateTime getDateTime() {
            return dateTime;
        }

        public String getCategory() {
            return category;
        }

        public String getSubcategory() {
            return subcategory;
        }

        public double getAmount() {
            return amount;
        }

        public String getDescription() {
            return description;
        }

        public double getIncome() {
            return amount > 0 ? amount : 0;
```

```
    }

    public double getExpense() {
        return amount < 0 ? -amount : 0;
    }


    public String toRecordString() {
        String formattedDateTime = dateTime.format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss"));
        return formattedDateTime + "\tКатегорія: " + category + "\tПідкатегорія: " + subcategory
+ "\tСума: "
                + amount + "\tОпис: " + description;
    }


    public int getMonth() {
        return dateTime.getMonthValue();
    }
}


public static class Category {
    private final String name;
    private final List<Subcategory> subcategories;

    public Category(String name, List<Subcategory> subcategories) {
        this.name = name;
        this.subcategories = subcategories;
    }

    public String getName() {
        return name;
    }

    public List<Subcategory> getSubcategories() {
        return subcategories;
    }
}

public static class Subcategory {
    private final String name;

    public Subcategory(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

```
public static class BalanceCategory {
    private List<Transaction> transactions;
    private String category;
    private double totalIncome;
    private double totalExpense;
    private double totalBalance;

    public BalanceCategory(String category) {
        this.category = category;
    }

    public void setTransactions(List<Transaction> transactions) {
        this.transactions = transactions;
    }

    public Double getTotalAmountForMonth(int month, List<Transaction> transactions) {
        return transactions.stream()
                .filter(transaction -> transaction.getMonth() == month)
                .mapToDouble(Transaction::getAmount)
                .sum();
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public double getTotalIncome() {
        return totalIncome;
    }

    public void setTotalIncome(double totalIncome) {
        this.totalIncome = totalIncome;
    }

    public double getTotalExpense() {
        return totalExpense;
    }

    public void setTotalExpense(double totalExpense) {
        this.totalExpense = totalExpense;
    }

    public double getTotalBalance() {
        return totalBalance;
    }

    public void setTotalBalance(double totalBalance) {
        this.totalBalance = totalBalance;
```

```
    }

    public Double getTotalAmountForMonth(int month) {
        return transactions.stream()
                .filter(transaction -> transaction.getMonth() == month)
                .mapToDouble(Transaction::getAmount)
                .sum();
    }
}

public class CategoryData {
    private String category;
    private List<SubcategoryData> subcategories;

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public List<SubcategoryData> getSubcategories() {
        return subcategories;
    }

    public void setSubcategories(List<SubcategoryData> subcategories) {
        this.subcategories = subcategories;
    }
}

public class SubcategoryData {
    private String subcategory;
    private List<Double> incomes;
    private List<Double> expenses;

    public String getSubcategory() {
        return subcategory;
    }

    public void setSubcategory(String subcategory) {
        this.subcategory = subcategory;
    }

    public List<Double> getIncomes() {
        return incomes;
    }

    public void setIncomes(List<Double> incomes) {
        this.incomes = incomes;
    }
```

```
    public List<Double> getExpenses() {
        return expenses;
    }

    public void setExpenses(List<Double> expenses) {
        this.expenses = expenses;
    }
}

private void addTransactionsToTable(List<Transaction> transactions) {
    Platform.runLater(() -> {
        transactionHistoryTableView.getItems().clear();

        for (Transaction transaction : transactions) {
            if (transaction != null) {
                addTransaction(transaction);
            }
        }

        transactionHistoryTableView.refresh();
    });
}

private void addTransaction(Transaction transaction) {
    Platform.runLater(() -> {
        transactionHistoryTableView.getItems().add(transaction);
    });
}

private List<Transaction> readTransactionsFromFile(File file) {
    List<Transaction> transactions = new ArrayList<>();
    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = reader.readLine()) != null) {
            System.out.println("Read line: " + line);
            Transaction transaction = parseTransactionFromString(line);
            transactions.add(transaction);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    return transactions;
}

public static Transaction parseTransactionFromString(String line) {
    String[] parts = line.split("\t");
    LocalDateTime dateTime = LocalDateTime.parse(parts[0],
DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss"));
    String category = parts[1].substring(parts[1].indexOf(':') + 2);
    String subcategory = parts[2].substring(parts[2].indexOf(':') + 2);
    double amount = Double.parseDouble(parts[3].substring(parts[3].indexOf(':') + 2));
```

```
        String description = parts[4].substring(parts[4].indexOf(':') + 2);
        return new Transaction(dateTime, category, subcategory, amount, description);
    }
}
```

```
package com.dniprotech.financialaccounting;

import java.time.LocalDate;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;

public class TransactionRecord {
    private LocalDate date;
    private LocalTime time;
    private String category;
    private String subcategory;
    private double amount;
    private String description;

    public TransactionRecord(LocalDate date, LocalTime time, String category, String subcategory,
double amount, String description) {
        this.date = date;
        this.time = time;
        this.category = category;
        this.subcategory = subcategory;
        this.amount = amount;
        this.description = description;
    }

    public static TransactionRecord fromString(String line) {
        String[] parts = line.split(" - ");
        if (parts.length == 5) {
            String dateTimeStr = parts[0];
            String category = parts[1].split(": ")[1];
            String subcategory = parts[2].split(": ")[1];
            String amountStr = parts[3].split(": ")[1];
            String description = parts[4].split(": ")[1];

            String[] dateTimeParts = dateTimeStr.split(" ");
            if (dateTimeParts.length == 2) {
                LocalDate date = LocalDate.parse(dateTimeParts[0],
DateTimeFormatter.ofPattern("dd.MM.yyyy"));
                LocalTime time = LocalTime.parse(dateTimeParts[1],
DateTimeFormatter.ofPattern("HH:mm:ss"));
                double amount = Double.parseDouble(amountStr);
                return new TransactionRecord(date, time, category, subcategory, amount, description);
            }
        }
        return null;
    }

    public LocalDate getDate() {
```

```
        return date;
    }

    public LocalTime getTime() {
        return time;
    }

    public String getCategory() {
        return category;
    }

    public String getSubcategory() {
        return subcategory;
    }

    public double getAmount() {
        return amount;
    }

    public String getDescription() {
        return description;
    }
}
// CSS:
.root {
    -fx-background-color: #f4f4f4;
}

.table-view {
    -fx-background-color: white;
}

.table-row-cell {
    -fx-background-color: #e0e0e0;
}

.menu-bar {
    -fx-background-color: #666;

.menu-bar .menu .menu-item {
    -fx-text-fill: white;
}

.button {
    -fx-background-color: #4CAF50;
    -fx-text-fill: white;
}

.table-view .column-header {
    -fx-background-color: #666;
    -fx-text-fill: white;
}
```

```
.text-field,
.date-picker {
   -fx-alignment: CENTER;
}

#timeField {
   -fx-alignment: CENTER;
}

.vbox-user-info {
   -fx-background-color: #f4f4f4;
   -fx-padding: 10px;
}

.label-user-info {
   -fx-font-size: 18px;
   -fx-text-fill: #333;
}


// pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.dniprotech</groupId>
   <artifactId>FinancialAccounting</artifactId>
   <version>1.0-SNAPSHOT</version>
   <name>FinancialAccounting</name>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <junit.version>5.9.2</junit.version>
   </properties>

   <dependencies>
      <dependency>
         <groupId>org.openjfx</groupId>
         <artifactId>javafx-controls</artifactId>
         <version>19.0.2.1</version>
      </dependency>

      <dependency>
         <groupId>com.jfoenix</groupId>
         <artifactId>jfoenix</artifactId>
         <version>9.0.10</version>
      </dependency>
```

```
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.14.2</version>
    </dependency>

    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.10.1</version>
    </dependency>

    <dependency>
        <groupId>org.controlsfx</groupId>
        <artifactId>controlsfx</artifactId>
        <version>11.1.2</version>
    </dependency>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-fxml</artifactId>
        <version>19.0.2.1</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.15.1</version>
    </dependency>

    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>com.jfoenix</groupId>
        <artifactId>jfoenix</artifactId>
        <version>9.0.10</version>
    </dependency>

</dependencies>
```

```
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.10.1</version>
        <configuration>
          <source>17</source>
          <target>17</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-maven-plugin</artifactId>
        <version>0.0.8</version>
        <executions>
          <execution>
            <!-- Default configuration for running with: mvn clean javafx:run -->
            <id>default-cli</id>
            <configuration>
              <mainClass>

com.dniprotech.financialaccounting/com.dniprotech.financialaccounting.HelloApplication
              </mainClass>
              <launcher>app</launcher>
              <jlinkZipName>app</jlinkZipName>
              <jlinkImageName>app</jlinkImageName>
              <noManPages>true</noManPages>
              <stripDebug>true</stripDebug>
              <noHeaderFiles>true</noHeaderFiles>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

ДОДАТОК Б

Перелік файлів на диску

Ім’я файлу	**Опис**
Пояснювальні документи	
Кваліфікаційна робота Буток В.О 122-22-3.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Буток В.О 122-22-3.docx	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Теза-доповідь Буток В.О 122-2-3.docx	Тези конференції «Сучасні технології управління особистими фінансами за допомогою аналітичних інструментів»
Програма	
Буток В.О 122-22-3.rar	Архів. Містить код програми та скомпільований додаток.
Презентація	
Буток В.О 122-22-3.pptx	Презентація кваліфікаційної роботи