МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 19.00.00.000 ПЗ

Група ІПм-23-3

Романюк Володимир

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Романюк Володимир Романович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми імплементації програмного забезпечення для менеджменту персональних фінансів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Романюк В. Р.
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Григорчук Любомир Іванович, к. пед. н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри
доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль
доц. Вовк Р.Б.
(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Романюку Володимиру Романовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи **“**Моделі, методи та алгоритми імплементації програмного забезпечення для менеджменту персональних фінансів**”**
керівник проекту (роботи) Григорчук Любомир Іванович, к. пед. н., доцент
затверджені наказом закладу вищої освіти від **“** 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 12 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Архітектура, формальний опис і алгоритми функціонування систем

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

1. Теоретичні відомості про менеджмент персональних фінансів та актуальність розробки нових рішень
2. Дослідження сучасних рішень для менеджменту персональних фінансів та засобів їх розробки
3. Розробка програмного рішення
4. Експериментальна частина

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Діаграма взаємодії основних компонентів системи (рис. 3.1, ст. 49)
2. Use case діаграма системи (рис. 3.2, ст. 52)
3. Діаграма бази даних (рис. 3.5, ст. 55)
4. Структура проєкту в середовищі IntelliJ IDEA (рис. 3.7, ст. 59)
5. Індексна сторінка додатку (рис. 4.1, ст. 72)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури	15.09.2024	виконано
2	Аналіз сучасних рішень для менеджменту персональних фінансів	26.09.2024	виконано
3	Вибір основних технологій для реалізації програмного продукту	12.10.2024	виконано
4	Формулювання вимог, алгоритмів та основного функціоналу системи	30.10.2024	виконано
5	Розробка програмного рішення	26.11.2024	виконано
6	Затвердження пояснювальної записки роботи завідувачем кафедри	12.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 97 ст., 38 рисунків, 1 таблиця, 42 джерела.

Тема: Моделі, методи та алгоритми імплементації програмного забезпечення для менеджменту персональних фінансів.

Об’єкт дослідження: сфера управління персональними фінансовими ресурсами окремої особи, що охоплює питання доходів, витрат, планування бюджету та фінансового майбутнього.

Мета роботи: вдосконалення додатку, який буде допомагати користувачам слідкувати та керувати своїми фінансами, надавати зручні інструменти для організації і зручного представлення даних про власні фінансові рішення.

Предмет дослідження: моделі, методи і алгоритми, які можуть бути використані для виконання різноманітних задач управління персональними фінансами.

Результати дослідження:

Виконано аналіз існуючих систем для управління персональними фінансами і на їх основі було спроектовано власну архітектуру системи, визначено алгоритм її роботи та основний функціонал.

Висновок:

В результаті досліджень розроблено та протестовано веб-додаток для управління особистими фінансами, що вирішує проблему аналізу витрат персональних фінансових ресурсів, планування майбутніх витрат та визначення фінансових звичок.

ПЕРСОНАЛЬНІ ФІНАНСИ, ФІНАНСОВИЙ ОБЛІК, МОДУЛЬНА АРХІТЕКТУРА, ЦИФРОВІ ФІНАНСОВІ ІНСТРУМЕНТИ, КОНТРОЛЬ ВИТРАТ, ВЕБ-ДОДАТОК, SPRING FRAMEWORK, REST API.

ABSTRACT

Master's paper: 97 p., 38 figures, 1 table, 42 sources.

Title: Models, methods and algorithms for implementing software for personal finance management.

Object of research: the field of personal financial resource management of an individual, covering issues of income, expenses, budget planning and financial future.

Purpose: improving an application that will help users monitor and manage their finances, provide convenient tools for organizing and presenting data on their financial decisions.

Subject of research: models, methods and algorithms that can be used to perform various tasks of personal finance management.

Research results:

Performed analysis of existing systems for personal finance management and based on them new system architecture was designed, determined the algorithms of its operation and the main functionality.

Conclusion:

As a result of the research, a web application for personal finance management has been developed and tested, which solves the problem of analyzing the expenditure of personal financial resources, planning future expenses and detecting financial habits.

PERSONAL FINANCE, FINANCIAL ACCOUNTING, MODULAR ARCHITECTURE, DIGITAL FINANCIAL INSTRUMENTS, EXPENDITURE CONTROL, WEB APPLICATION, SPRING FRAMEWORK, REST API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

- HTTP (HyperText Transfer Protocol) – протокол передачі даних;
- JSON (JavaScript Object Notation) – текстовий формат даних. За допомогою нього описуються об’єкти або інші структури даних;
- REST (Representational State Transfer) – підхід до архітектури мережевих протоколів;
- API (Application Programming Interface) – прикладний програмний інтерфейс, засіб для двох або більше комп’ютерних програм комунікувати між собою;
- Фреймворк (Framework) - інфраструктура програмних рішень, що полегшує розробку складних систем;
- Ендпоінт (Endpoint) – в контексті веб-розробки, це специфічна URL-адреса або шлях на веб-сервері, який використовується для виконання певної операції або доступу до певних даних чи ресурсів.

ВСТУП

Актуальність роботи

Управління особистими фінансами є однією з найважливіших складових ефективного фінансового планування та гарантування стабільності для кожної людини. Сьогодні, коли доступ до численних фінансових можливостей супроводжується складними економічними умовами, управління власними фінансами стає серйозним викликом для багатьох людей.

Проте, сфера обліку особистих фінансів ще недостатньо розвинена, що створює прогалини у фінансовій грамотності та перешкоджає багатьом людям належним чином управляти своїми фінансами. Серед найпоширеніших проблем - відсутність доступу до необхідної інформації, складність процесів, великі витрати часу та відсутність автоматизації. Багато людей не мають достатньо знань про правильне управління фінансами. Невміння застосовувати базові принципи бюджетування, інвестування та заощадження, як правило, призводить до нераціонального використання ресурсів та небажаних фінансових ризиків. Традиційні підходи передбачають трудомісткі процедури, такі як створення бюджетів вручну або створення таблиць Excel, які забирають багато часу та зусиль.

Розробка додатків для управління особистими фінансами стає надзвичайно актуальною в сучасному світі. Такі програми забезпечують зручність, безпеку та розширюють можливості користувачів контролювати свої фінанси. Використання сучасних технологій дозволяє цим інструментам змінити підхід людей до управління грошима, допомагаючи їм приймати більш обґрунтовані фінансові рішення, розвивати хороші звички заощадження та покращувати загальний фінансовий добробут.

Мета і задачі дослідження

Метою даної роботи є вдосконалення додатку, який буде допомагати користувачам слідкувати та керувати персональними фінансами, а також надавати додаткові інструменти для організації і зручного керування фінансовими активами.

Мета роботи визначила необхідність виконання наступних задач:

- проаналізувати існуючі методи і засоби управління персональними фінансами та дослідити аналоги на ринку;
- проаналізувати сучасні методи реалізації подібних програмних рішень та вдосконалити необхідні для створення нового програмного продукту технології;
- провести системний аналіз, розробити структуру та алгоритми для вирішення задач управління фінансами;
- вдосконалити програмне рішення, яке могло б конкурувати з існуючими аналогами на ринку.

Об’єктом дослідження є сфера управління персональними фінансами, яка включає широкий спектр аспектів, пов'язаних з управлінням фінансовими ресурсами окремої особи, що дозволяє охопити питання планування бюджету: доходів та витрат, а також фінансового майбутнього.

Предметом дослідження є методи, засоби і алгоритми, які можуть бути використані для виконання різноманітних задач управління персональними фінансами. Вивчення інструментів для моніторингу, аналізу фінансових даних, а також розробку програмного забезпечення для керування грошовими ресурсами.

Наукова новизна одержаних результатів

Запропоновано нову архітектуру програмного забезпечення, яка поєднує принципи об'єктно-орієнтованого програмування, REST API та сучасні бібліотеки. Розроблено новий додаток Finance Manager, що удосконалює роботу автоматизації

обліку й планування фінансів та забезпечує ефективність і масштабованість системи.

Практична цінність

Розроблено платформу, яка відзначається гнучкістю та універсальністю, що дозволяє адаптувати її як для індивідуальних користувачів, так і для малих бізнесів. Завдяки модульній архітектурі та масштабованості система може бути налаштована відповідно до потреб користувача і забезпечує можливість ефективного управління. Такий підхід дозволяє в подальшому розширення функціональності платформи.

Структура магістерської роботи

Магістерська робота викладена на 100 сторінках друкованого тексту, який складається з вступу, чотирьох розділів, висновків, списку використаних джерел (42 найменування). Робота містить 38 рисунків, 1 таблицю.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО МЕНЕДЖМЕНТ ПЕРСОНАЛЬНИХ ФІНАНСІВ ТА АКТУАЛЬНІСТЬ РОЗРОБКИ НОВИХ РІШЕНЬ

1.1. Теоретичні відомості про менеджмент персональних фінансів

1.1.1. Вступ до менеджменту персональних фінансів

Фінансові проблеми можуть викликати у багатьох людей сильний стрес. Ця тривога може посилюватися під час економічних спадів або коли людям не вистачає заощаджень, щоб пережити такі бурі. Фінансовий стрес може затьмарювати судження, що призводить до прийняття рішень, які погіршують ситуацію. Але що, якби ви могли приймати розумні рішення, які допоможуть вам досягти фінансової безпеки? Як поставити цілі для фінансового благополуччя і побудувати фундамент для їх досягнення?

Ключ до цього лежить в оволодінні фінансовим менеджментом - навичкою, яка, на жаль, вислизає від більшості з нас. Ця робота заглиблюється у світ управління особистими фінансами, досліджуючи його основи, розкриваючи його переваги та надаючи вам найкращі поради щодо ефективного та результативного управління вашими фінансами [1-2].

1.1.2. Що таке управління особистими фінансами

Управління особистими фінансами передбачає ефективне управління грошима, активами та ресурсами. Це досягається за допомогою бюджетування, заощаджень, інвестування та управління боргами. Метою є оптимізація фінансових ресурсів для задоволення короткострокових потреб, досягнення довгострокових цілей і прийняття обґрунтованих фінансових рішень, які забезпечують стабільність і знижують ризики [3].

1.1.3. Історичний розвиток менеджменту персональних фінансів

Концепція управління особистими фінансами бере свій початок з того моменту, коли люди почали усвідомлювати важливість обліку ресурсів у спробі вижити і зробити своє існування стабільним. У Стародавній Месопотамії та Єгипті вели облік землі, врожаю та боргів. Ці перші форми фінансового обліку були пов'язані з необхідністю контролювати обмін ресурсами і велися від руки на глиняних табличках або папірусах.

Пізніше, у Середньовіччі, розвиток торгівлі та банківської системи призвів до появи більш досконалих підходів до управління фінансами. У цей період використовувалися системи подвійного запису, розроблені в Італії в 15 столітті, які лягли в основу сучасного бухгалтерського обліку.

Концепція управління особистими фінансами як самостійна сфера з'явилася в 19 столітті з поширенням індивідуального підприємництва. Люди почали розглядати свої доходи, витрати, інвестиції та заощадження як окремий об'єкт для аналізу та планування.

До епохи цифрових технологій фінансовий облік здійснювався вручну, за допомогою книг, електронних таблиць та журналів. З початку 20 століття, коли кількість друкованої продукції значно зросла, почали з'являтися шаблони для ведення сімейного бюджету. Люди записували доходи і витрати в спеціальні зошити або таблиці, розділені на категорії.

Перехід до цифрових систем почався в середині 20 століття з розвитком комп'ютерних технологій. Перші персональні комп'ютери дозволили використовувати для фінансового обліку такі електронні таблиці, як VisiCalc (1979) та Excel (1985). Це значно полегшило процес розрахунків та аналізу.

У 1990-х роках, з розвитком Інтернету, з'явилися перші програмні продукти для управління фінансами, такі як Quicken і Microsoft Money. Вони дозволили автоматизувати облік витрат, створювати бюджети і навіть планувати податки.

Пізніше почався перехід на онлайн-платформи та мобільні додатки. Веб-сервіси інтегрувалися з банківськими рахунками, як, наприклад, Mint.com у 2006 році, що дозволило користувачам автоматично відстежувати свої фінанси в режимі реального часу [4-5].

1.1.4. Вплив технологій на розвиток управління фінансів

Технологічний прогрес став одним з головних рушіїв змін в управлінні особистими фінансами. Він не тільки спростив всі види обліку та аналізу, але й відкрив широкі можливості для управління фінансами, яких раніше взагалі не існувало. Нижче висвітлені ключові аспекти впливу технологій [6].

1.1.4.1. Мобільні додатки

З появою смартфонів управління фінансами стало зовсім не недосяжним і несвоєчасним. Мобільні додатки, такі як Mint, YNAB або Money Lover, можуть робити те, що в минулому доводилося робити за допомогою досить просунутого програмного забезпечення, а іноді навіть фінансових експертів.

- Доступність та простота використання: Смартфони дозволяють людям управляти своїми фінансами будь-де і будь-коли. Замість ручного введення даних у таблиці користувачі отримали інтуїтивно зрозумілий інтерфейс для обліку доходів і витрат.
- Автоматизація обліку витрат: інтеграція з банківським рахунком дозволяє автоматично імпортувати транзакції, класифікувати їх за типами - їжа, транспорт, розваги - і відповідно генерувати звіти.
- Бюджетування: додатки допомагають встановлювати ліміти витрат, відстежувати їх у режимі реального часу та аналізувати, чи вдається вкластися у запланований бюджет.
- Сповіщення: нагадування про майбутні платежі або перевищення бюджету допомагають уникнути боргів і контролювати свої фінанси.

1.1.4.2. Штучний інтелект, персоналізовані рекомендації

Використання технологій штучного інтелекту спричинило революцію управління фінансами, зробивши його більш точним та персоналізованим.

- Аналіз фінансових звичок: ШІ може виявляти моделі витрат і прогнозувати майбутні фінансові потреби на основі історичних даних.
- Оптимізація заощаджень: Алгоритми автоматично підказують, як можна зменшити витрати або збільшити заощадження. Наприклад, такі сервіси, як Plum або Cleo, використовують чат-ботів, які даватимуть поради та взаємодіятимуть з користувачами у зручному для них форматі.
- Попередження про ризики: Додатки зі штучним інтелектом можуть аналізувати тенденції витрат і вказувати на появу можливих фінансових проблем, таких як ризики перевитрат або накопичення боргів.

1.1.4.3. Хмарні технології: доступність і синхронізація даних

Завдяки хмарним платформам користувачі можуть зберігати свої фінансові дані в безпечному середовищі та мати доступ до них з будь-якого пристрою.

- Синхронізація: Інформація автоматично надходить на всі ваші пристрої, і ви можете вести записи на смартфоні, планшеті або комп'ютері одночасно.
- Резервне копіювання: Хмарне рішення захистить дані у випадку втрати або пошкодження пристрою.
- Спільний доступ: Деякі сервіси дозволяють ділитися фінансовими даними з партнером або членами сім'ї для управління спільним бюджетом. Прикладом такого сервісу може бути GoodBudget.

1.1.4.4. Інтеграція з фінансовими інструментами та платформами

Технології дозволяють об'єднати всі аспекти вашого фінансового життя в одній системі.

- Банківські API: Більшість додатків інтегровані з банками, що дозволяє автоматично імпортувати транзакції, перевіряти залишки на рахунках і здійснювати платежі безпосередньо в додатку.
- Інвестиційні платформи: такі сервіси, як Robinhood або Acorns, спрощують інвестування для звичайних користувачів. Вони дозволяють інвестувати в фондові ринки без спеціальних знань.
- Податкові сервіси: Інтеграція з такими платформами, як TurboTax, автоматично створює податкові звіти на основі даних з бухгалтерських додатків.

1.1.4.5. Блокчейн і криптовалюти

Блокчейн відкрив нові горизонти в управлінні особистими фінансами, насамперед у сфері криптовалют.

- Криптогаманці: Ви можете зберігати, надсилати та отримувати криптовалюту за допомогою таких додатків, як Trust Wallet або Ledger Live. Вони дуже безпечні завдяки децентралізованому підходу.
- Смарт-контракти: блокчейн-системи дозволяють автоматизувати фінансові операції, такі як оренда або оплата послуг, без будь-яких посередників.
- Інвестування в криптовалюти: нові платформи спрощують доступ до ринку цифрових активів навіть для користувачів, які ніколи раніше не інвестували.

1.1.4.6. Гейміфікація: фінанси як гра

Гейміфікація в управлінні фінансами допомагає підвищити залученість користувачів і робить процес цікавим.

- Досягнення та винагороди: деякі додатки, такі як Monzo або Revolut, використовують систему досягнень, яка заохочує користувачів досягати фінансових цілей, наприклад, досягти 20% заощаджень.

- Ігрові сценарії: інтерактивні функції, включаючи симуляції інвестицій або планування великих покупок, дозволяють користувачам експериментувати, не ризикуючи реальними грошима.

1.1.4.7. Безпека та конфіденційність: виклики цифрової епохи

З розвитком технологій однією з найактуальніших тем є захист фінансових даних.

- Шифрування: Сучасні додатки використовують шифрування для захисту даних під час передачі та зберігання.
- Двофакторна автентифікація: Додатковий рівень безпеки забезпечується при вході у фінансові рахунки.
- Контроль доступу: Користувачі можуть обмежити доступ до своїх даних або переглянути, які сервіси мають до них доступ.

1.2. Ефективне управління фінансами

1.2.1. Сфери персональних фінансів

Доходи - це відправна точка персональних фінансів. Це вся сума грошових надходжень, яку ви отримуєте і можете розподілити на витрати, заощадження, інвестиції та захист. Дохід - це всі гроші, які ви приносите. Сюди входить заробітна плата, дивіденди та інші джерела грошових надходжень.

Витрати - це відтік готівки, і, як правило, це те, на що йде основна частина доходу. Витрати - це все, що людина купує на свій дохід. Сюди входить оренда, іпотека, продукти харчування, меблі для дому, ремонт житла, подорожі та розваги.

Здатність керувати витратами є критично важливим аспектом особистих фінансів. Люди повинні стежити за тим, щоб їхні витрати були меншими за доходи, інакше їм не вистачить грошей на покриття витрат або вони потраплять у

боргову яму. Борги можуть бути руйнівними у фінансовому плані, особливо через високі відсотки, які нараховують кредитні картки.

Заощадження - це дохід, який залишається після витрат. Кожен повинен прагнути мати заощадження для покриття великих витрат або надзвичайних ситуацій. Однак це означає не використовувати весь свій дохід, що може бути складно.

Незалежно від труднощів, кожен повинен прагнути мати принаймні частину заощаджень, щоб покривати будь-які коливання в доходах і витратах - десь між трьома і 12 місячними витратами.

Крім того, гроші, що просто лежать на ощадному рахунку, стають марнотратством, оскільки з часом вони втрачають купівельну спроможність через інфляцію. Натомість, гроші, не прив'язані до рахунків на випадок надзвичайних ситуацій або витрат, слід вкладати в те, що допоможе їм зберегти свою вартість або примножити її, наприклад, в інвестиції.

Інвестування передбачає придбання активів, зазвичай акцій та облігацій, з метою отримання прибутку на вкладені кошти. Інвестування має на меті збільшити багатство людини понад суму, яку вона вклала. Інвестування пов'язане з певними ризиками, оскільки не всі активи зростають у ціні, і ви можете зазнати збитків.

Інвестування може бути складним для тих, хто з ним не знайомий, тому варто присвятити деякий час для того, щоб отримати розуміння через читання та навчання. Якщо у вас немає часу, ви можете скористатися послугами професіонала, який допоможе вам інвестувати ваші гроші.

Захист - це методи, які люди використовують, щоб убезпечити себе від несподіваних подій, таких як хвороби або нещасні випадки, а також як засіб збереження багатства. Захист включає в себе страхування життя і здоров'я, а також майнове і пенсійне планування [7-8].

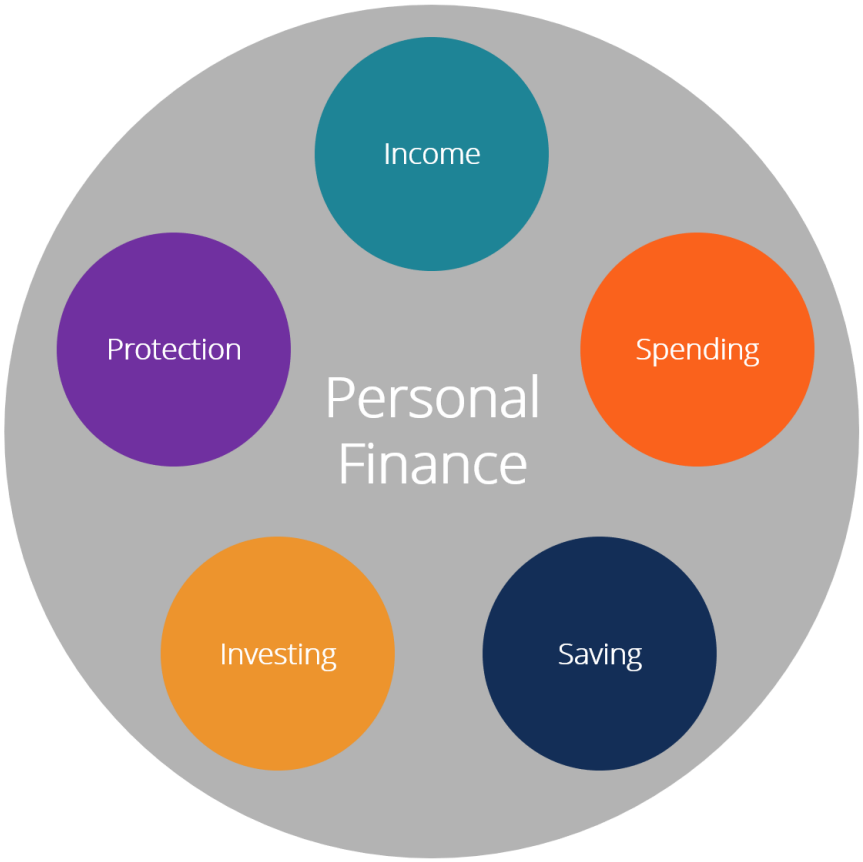


Рис. 1.1. Діаграма основних сфер персональних фінансів

1.2.2. Основні принципи управління персональними фінансами

УПМ (управління персональними фінансами) зводиться до основних принципів і практик, які допомагають людям ефективно управляти своїми грошима. До них можна віднести:

1.2.2.1. Відстежуйте свої доходи та витрати

Основою УПМ є розуміння ваших грошових потоків. Це означає ретельний облік усіх ваших джерел доходів (заробітна плата, підробіток тощо) та витрат (оренда, продукти, розваги). Аналізуючи свої доходи і витрати, ви можете визначити сфери для скорочення і вивільнити ресурси для досягнення своїх фінансових цілей. Точне знання того, куди йдуть ваші гроші, дозволяє ставити реалістичні фінансові цілі і гарантує, що ви будете жити за коштами.

1.2.2.2. Складіть бюджет

Бюджет - це детальний план, який відображає ваші доходи та витрати протягом певного періоду, як правило, помісячно. Він розподіляє ваш дохід на різні витрати (предмети першої необхідності, дискреційні витрати, заощадження) та погашення боргів. Існують різні методи складання бюджету, наприклад, правило 50/30/20 (50% - потреби, 30% - бажання, 20% - заощадження/борги), але головне - знайти той, який працює саме для вас. Реалістичний бюджет дає вам можливість приймати обґрунтовані рішення щодо витрат, розставляти пріоритети між потребами та бажаннями, а також уникати імпульсивних покупок.

1.2.2.3. Розумно керуйте боргом

Розумне управління боргом має вирішальне значення для підтримки фінансового здоров'я та досягнення довгострокових фінансових цілей. Це передбачає розуміння різниці між хорошим боргом, який може допомогти створити багатство (наприклад, іпотека або студентські позики, що сприяють інвестиціям у нерухомість чи освіту), і поганим боргом, який зазвичай є наслідком непотрібних витрат (наприклад, борг за кредитною карткою через імпульсивні покупки). Ефективне управління боргом починається з того, що ви берете позики в межах своїх можливостей і відповідально користуєтеся кредитом. Важливо здійснювати своєчасні платежі, щоб уникнути прострочення та високих відсоткових ставок, які можуть швидко збільшити борг, а також зберігати фінансову дисципліну, щоб уникнути необґрунтованих боргових зобов’язань.

1.2.2.4. Заощаджуйте на свої цілі

Заощадження на свої цілі гарантує, що ви будете готові до майбутніх фінансових потреб і прагнень. Почніть з визначення чітких, досяжних цілей і встановлення часових рамок для кожної з них. Наявність спеціальних ощадних рахунків для кожної цілі допоможе візуалізувати прогрес і залишатися

вмотивованим. Автоматизація переказів на ці рахунки з вашої зарплати забезпечить стабільні заощадження і зменшить спокусу витратити ці гроші. Визначивши пріоритети заощаджень і створивши достатній резервний фонд, ви зможете впоратися з несподіваними фінансовими труднощами і досягти своїх довгострокових цілей. Послідовність має вирішальне значення; навіть невеликі регулярні внески можуть з часом накопичити значну суму завдяки складним відсоткам.

1.2.2.5. Інвестуйте у своє майбутнє

Інвестування дозволяє вашим грошам зростати з часом, долаючи інфляцію та досягаючи довгострокових цілей, таких як вихід на пенсію. На відміну від заощаджень, які зазвичай включають в себе легкодоступні кошти з низьким рівнем ризику, інвестування передбачає вкладення грошей в такі активи, як акції, облігації, пайові інвестиційні фонди або нерухомість, які мають потенціал для отримання більш високого прибутку, але пов'язані з різним ступенем ризику. Ефективне інвестування починається з визначення ваших фінансових цілей та рівня прийнятного ризику, що допоможе уникнути імпульсивних рішень на ринку. Почніть з неризикових варіантів, таких як індексні фонди, і диверсифікуйте своє портфоліо відповідно до толерантності до ризику та віку. Зверніться за професійною фінансовою консультацією для розробки індивідуальної інвестиційної стратегії, яка врахує ваші потреби, часові горизонти та зміни економічних умов.

1.2.2.6. Відстежуйте свій прогрес і вносьте корективи

Фінансові ситуації змінюються. Регулярно переглядайте свої доходи, витрати та прогрес у досягненні цілей. Порівнюйте свій бюджет з фактичними витратами і вносьте необхідні корективи. Економічні фактори можуть вимагати коригування - збільшення доходу може дозволити зробити більші заощадження,

або непередбачувані витрати можуть вимагати тимчасових змін у бюджеті. Будьте гнучкими та адаптивними, щоб підтримувати фінансову стабільність.

1.2.2.7. Автоматизуйте свої фінанси

Технології можуть спростити управління особистими фінансами. Налаштуйте автоматичні перекази на заощадження, оплату рахунків та погашення боргів, щоб уникнути пропущених платежів та штрафів за прострочення. Використовуйте автоматичні перекази із зарплати на ощадні рахунки, щоб «спочатку заплатити собі». Автоматизовані процеси забезпечують послідовний прогрес і мінімізують ризик людської помилки. Пам'ятайте, що автоматизація вимагає регулярного моніторингу для коригування або скасування транзакцій у разі потреби [9-11].

1.3. Актуальність розробки нового додатку для менеджменту персональних фінансів

1.3.1. Зростання попиту на фінансову грамотність

Живучи в сучасному світі, люди так чи інакше роблять управління особистими фінансами невід'ємною частиною свого життя. Вони намагаються не позичати, планувати витрати, досягати фінансової стабільності. Але не всім вистачає фінансової грамотності, щоб правильно розпоряджатися ресурсами. Додатки для управління особистими фінансами можуть стати своєрідним наставником, який на основі реальних даних надасть рекомендації та допоможе розібратися у своїх фінансових звичках.

1.3.2. Недоліки існуючих рішень

Більшість успішних на сьогоднішній день додатків для управління фінансами призначені для задоволення загальних потреб і не враховують

особливостей конкретних ринків або цільових груп, таких як місцеві валюти, складні мультивалютні транзакції або інтеграція з місцевими банками. Крім того, більшість таких додатків не доступні рідною мовою для користувачів, що створює бар'єри для їх використання. Новий додаток може заповнити ці прогалини, пропонуючи функціональність і підтримку, локалізовану для певного регіону або громади.

1.3.3. Персоналізація

Користувачі очікують, що сучасні фінансові інструменти враховуватимуть їхні конкретні потреби. Це може бути функціонал, який підлаштовуватиметься під фінансову поведінку людини: автоматична категоризація витрат, персоналізовані поради щодо заощаджень, прогнозування майбутніх витрат чи доходів. Штучний інтелект та алгоритми машинного навчання дозволяють розробляти індивідуальні фінансові стратегії, зменшуючи потребу в ручному управлінні та спрощуючи взаємодію з додатком.

1.3.4. Інтеграція з іншими сервісами

Однією з головних проблем сучасних фінансових додатків є їхня ізольованість. Багато рішень не дозволяють інтегруватися з іншими сервісами, такими як банківські додатки, платіжні платформи, системи онлайн-інвестицій або податкові служби. Розробка програми, яка може автоматично імпортувати дані з банківських рахунків, синхронізувати платежі або надавати податкову звітність, значно підвищує її цінність для користувача.

1.3.5. Інноваційні функції

Ринок вимагає функцій, які роблять управління фінансами легким і навіть приємним: розумний бюджетник, який аналізує попередні витрати і пропонує найкращі способи заощадити гроші. Ще одна новинка - блокчейн-технології,

інтегровані для управління криптовалютою. Ви також можете запропонувати деякі елементи гейміфікації, які можуть заохочувати користувачів до досягнення фінансових цілей за допомогою винагород або змагань.

1.3.6. Зростання використання мобільних пристроїв

Мобільні пристрої вже стали основним засобом управління фінансами для більшості користувачів. Однак не всі додатки використовують можливості мобільних платформ повною мірою. Наприклад, підтримка офлайн-режиму, інтуїтивно зрозумілий дизайн, враховуючи обмежений розмір екрану, або використання біометричних даних для авторизації - це функції, які можна розвивати.

1.3.7. Популярність екосистеми фінансового планування

У сучасному світі люди все частіше розглядають фінансове планування як частину загального саморозвитку. Додаток може стати не тільки інструментом для управління грошима, але й частиною великої екосистеми, яка охоплює цілі, пов'язані із заощадженнями, інвестиціями, освітою чи благодійністю. Інтеграція з іншими додатками, такими як сервіси планування подорожей або фітнес-трекери, дозволяє створити єдину платформу для досягнення життєвих цілей [12-13].

1.4. Висновки до розділу 1

У першому розділі зроблено огляд теоретичних аспектів управління особистими фінансами, тобто фундаменту, на якому ґрунтуватимуться рішення у цій сфері. Описано поняття сутності управління особистими фінансами, його значення для забезпечення фінансової стабільності та роль у підвищенні якості життя громадян. Досліджено історичний розвиток концепції управління фінансами, яка зазнала значних змін від простих методів обліку на паперових носіях до сучасних комп'ютерних інструментів.

Особливу увагу приділено ключовим елементам належного управління особистими фінансами, які включають належне планування, бюджетування, уникнення боргів та частий аналіз фінансової ситуації. Визначили фундаментальні елементи особистих фінансів - доходи, витрати, заощадження, інвестиції та захист - формують основні аспекти добробуту людини.

Обґрунтовано актуальність розробки нового програмного забезпечення для управління особистими фінансами. Це визначає попит на інноваційні продукти, які забезпечують розширені можливості аналізу, адаптацію до індивідуальних запитів та більш інтуїтивно зрозумілий інтерфейс.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ СУЧАСНИХ РІШЕНЬ ДЛЯ МЕНЕДЖМЕНТУ ПЕРСОНАЛЬНИХ ФІНАНСІВ ТА ЗАСОБИ ЇХ РОЗРОБКИ

2.1. Аналіз аналогів на ринку

Проаналізувавши предметну область було виділено такі п’ять сервісів:

1) MoneyWiz 2;
2) Monefy;
3) CoinKeeper;
4) Mobills;
5) Spendee.

2.1.1. MoneyWiz 2

MoneyWiz 2 містить широкий набір функцій з управління фінансами: від простого підрахунку грошових витрат до повної автоматизації руху коштів. У застосунку можна знайти всілякі варіанти обліку фінансів, планування та управління бюджетом, повторюваними транзакціями, аналіз витрат, детальні звіти у вигляді графіків, діаграм або простої статистики з урахуванням різних готових і користувацьких фільтрів. При цьому кожен звіт можна зберегти в програмі або експортувати в PDF-файл. Для зареєстрованих користувачів MoneyWiz 2 пропонує можливості синхронізації своїх даних.

Що стосується інтерфейсу програми та зручності користування - тут не все так гладко (рис. 2.1.). Незважаючи на спроби зробити дизайн сучаснішим, додаток все одно виглядає трохи застарілим, а дрібні та незручні елементи інтерфейсу - результат поганої оптимізації для невеликих пристроїв. Крім того, щоб записати елементарну транзакцію, доводиться здійснювати цілу низку дій, деякі операції додаються через контекстне меню. Але варто похвалити розробників за додану

можливість налаштовувати макети транзакцій і виводити віджети з різною інформацією на головний екран програми.

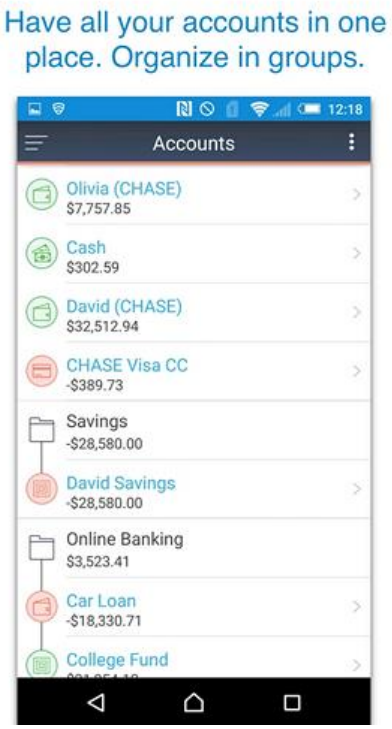


Рис. 2.1. Інтерфейс додатку MoneyWiz 2

MoneyWiz 2 – не найкращий застосунок для обліку фінансів, але, безперечно, один із найфункціональніших. Однак йому явно не вистачає клієнта для ПК або хоча б можливості керувати своїми фінансами в браузері. Це обмежує зручність використання для тих, хто надає перевагу роботі з фінансами на великому екрані і обмежує інтегрувацію фінансових операцій з іншими програмами на комп'ютері. Програма однозначно підійде людям, які звикли скрупульозно ставитися до фінансових витрат, але для простіших операцій краще звернути увагу на інші додатки. Спробувати MoneyWiz 2 в роботі можна, встановивши безкоштовну версію, що дає змогу зробити не більше 100 транзакцій. Перехід на платну версію обійдеться в 122 грн [14].

2.1.2. Monefy

Monefy – це додаток для управління особистими фінансами, який допомагає відстежувати ваші витрати та доходи. Він доступний на платформах Android та iOS.

Основні функції додатку включають можливість додавати транзакції як для доходів, так і для витрат, класифікувати їх за категоріями та відстежувати в часі. Користувачі можуть встановлювати бюджети та отримувати сповіщення, коли вони наближаються до бюджетних лімітів або перевищують їх. Monefy також надає детальні звіти та графіки, які допомагають користувачам зрозуміти свої витратні звички та фінансовий стан.

Користувацький інтерфейс Monefy розроблений так, щоб бути простим та інтуїтивно зрозумілим (рис. 2.2.).

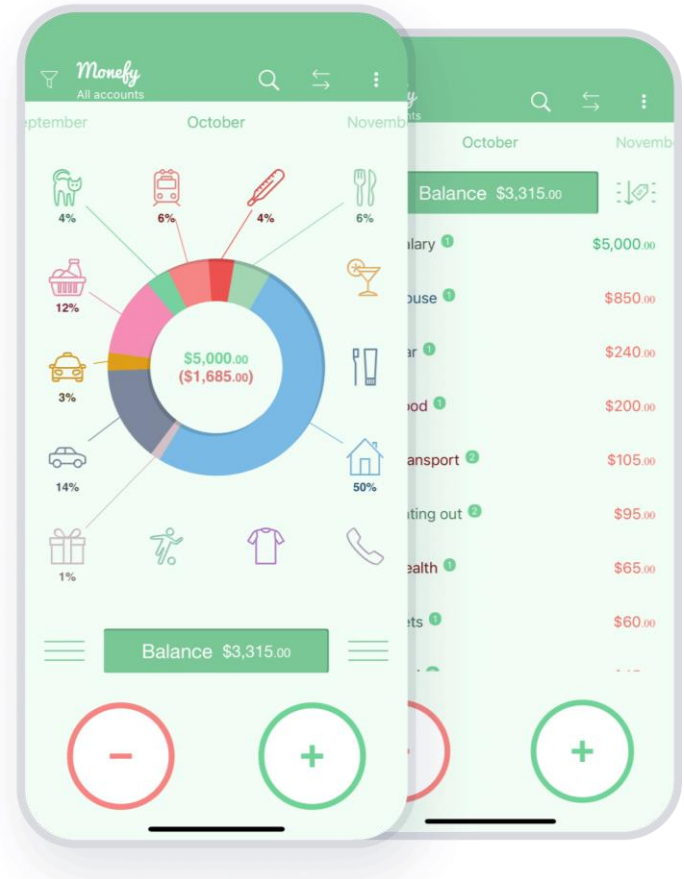


Рис. 2.2. Інтерфейс додатку Monefy

На головному екрані відображається підсумок балансу та список останніх транзакцій. Користувачі можуть додавати нові транзакції лише кількома дотиками та обирати з попередньо визначених категорій або створювати власні. Додаток також містить віджет для швидкого доступу до найважливіших функцій.

Monefy пропонує як безкоштовну, так і платну версію. Безкоштовна версія включає базові функції і підтримується рекламою. Платна версія, яка називається Monefy Pro, прибирає рекламу і додає деякі додаткові функції, такі як можливість синхронізації даних між кількома пристроями та експорт даних в Excel або Google Sheets. Ціна Monefy Pro коливається в залежності від платформи, але в середньому становить 3-5 доларів США [15].

2.1.3. CoinKeeper

Ветеран серед фінансових менеджерів, один з найбільш затребуваних застосунків на iOS і Android у категорії «Фінанси», що отримав перезапуск у 2014 році. Нова версія CoinKeeper змінила модель монетизації, стала виглядати більш сучасною і відповідати стилю iOS 7 і Android Lollipop. При цьому розробники продовжили підтримку старої версії додатка CoinKeeper Classic для користувачів з усталеними уподобаннями.

Найголовніша відмінність CoinKeeper від інших аналогічних програм – оригінальний і вельми незвичайний спосіб вести фінансовий облік за допомогою жестів (рис. 2.3.). Щоб додати витрати, необхідно перетягнути значок рахунку на необхідну категорію і ввести суму. Варто визнати, це найшвидший спосіб додавання транзакції – він позбавляє користувача зайвих дій і підходить для інших фінансових операцій, наприклад, під час переказу коштів між рахунками або їх поповнення. За допомогою жестів можна також упорядковувати категорії на екрані. При такому управлінні перевагою додатка є його інтерфейс, в якому всі основні елементи знаходяться на одному екрані.

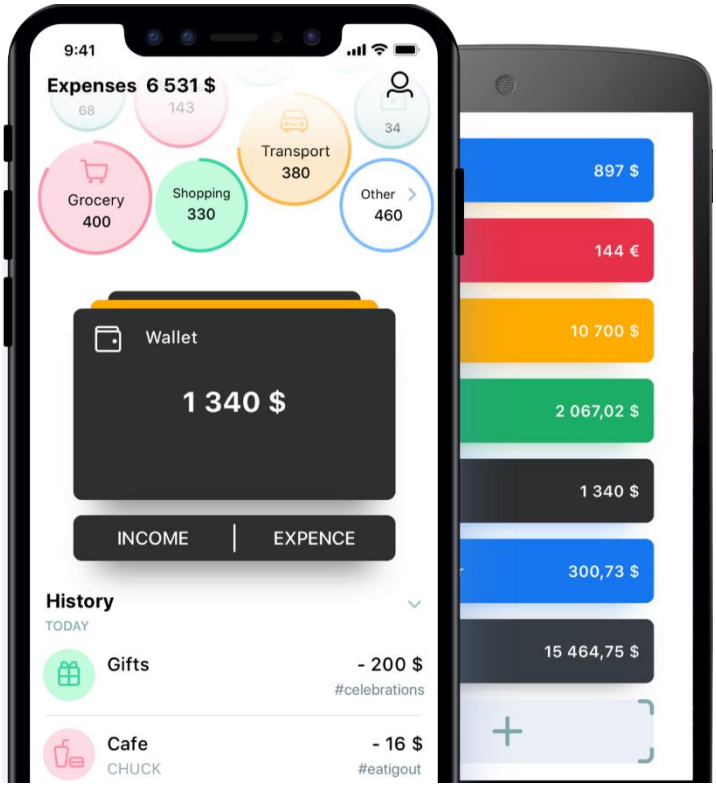


Рис. 2.3. Інтерфейс додатку CoinKeeper

Цікавою особливістю CoinKeeper є можливість вказувати місячний ліміт для кожної з категорій. Коли такий ліміт встановлений, значок категорії набуває колірної заливки, яка заповнюється в міру його вичерпання, сигналізуючи про можливу перевитрату коштів. Статистика менеджера не надто детальна: це загальна інформація щодо витрат і доходів у вигляді кругової діаграми, звіт по днях і базова динаміка фінансових операцій у вигляді графіка. Загалом виглядає все дуже солідно. До найістотніших недоліків CoinKeeper можна віднести відсутність мультивалютних рахунків і мультивалютних операцій, а також можливості імпортувати свої дані.

CoinKeeper поширюється за передплатою, яка відкриває функції, відсутні або обмежені в безкоштовній версії програми. До них належать: необмежена кількість рахунків, категорій витрат, статистика за будь-який період, додавання бюджету, експорт даних і синхронізація між пристроями [16].

2.1.4. Mobills

Mobills – це додаток для управління особистими фінансами, який допомагає користувачам відстежувати свої витрати, доходи та заощадження. Він доступний на платформах Android та iOS і може використовуватися у понад 150 країнах світу. Додаток має чистий і зручний інтерфейс, в якому легко орієнтуватися, що робить його чудовим варіантом для користувачів, які є новачками в керуванні своїми грошовими ресурсами (рис. 2.4.).

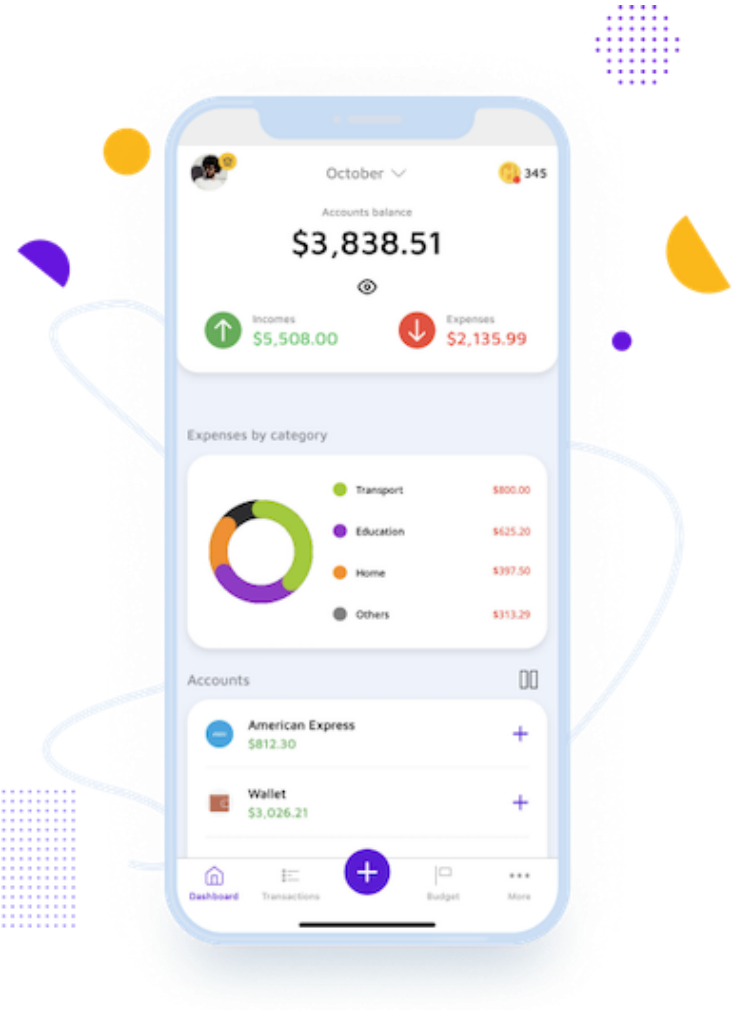


Рис. 2.4. Інтерфейс додатку Mobills

Однією з головних особливостей Mobills є його здатність автоматично розподіляти витрати за категоріями, що економить час і зусилля користувачів.

Додаток також дозволяє користувачам встановлювати бюджети та сповіщення, допомагаючи їм контролювати свої фінанси та уникати перевитрат. Mobills пропонує безліч корисних інструментів, таких як нагадування про рахунки, фінансові звіти та відстеження цілей, що робить його комплексним рішенням для управління особистими фінансами.

Mobills пропонує як безкоштовну версію, так і преміум-версію з додатковими функціями. Безкоштовна версія включає базові функції, такі як відстеження витрат і бюджетування, в той час як преміум-версія пропонує більш розширені можливості, такі як управління кредитними картками та відстеження інвестицій. Ціна на преміум-версію залежить від тривалості підписки, з можливістю щомісячної, щоквартальної та річної оплати.

Загалом, Mobills – це чудовий варіант для тих, хто хоче краще керувати своїми особистими фінансами. Інтуїтивно зрозумілий інтерфейс, автоматична категоризація та різноманітні функції роблять його комплексним рішенням для відстеження витрат та досягнення фінансових цілей [17].

2.1.5. Spendee

Цей сервіс для обліку фінансових витрат, представлений мобільними додатками для iOS і Android, позиціонується як найпростіший та найбільш інтуїтивно зрозумілий фінансовий менеджер. До цього варто додати визначення «лаконічний і стильний», оскільки дизайн Spendee чудово вписується в концепцію обох платформ (рис. 2.5.).

Spendee має базовий набір функцій, що включає облік фінансів за основним рахунком, роботу з бюджетом, експорт даних у CSV/XLS і хмарну синхронізацію, яка доступна під час реєстрації. Додаток підтримує мультивалютні операції з автоматичним конвертуванням в основну валюту. Головна ж перевага програми – простота і швидкість додавання фінансових операцій. У Spendee дійсно можна легко додавати транзакції в кілька кліків. Якщо ж ви маєте час, є можливість

додати до транзакції більше деталей: фотографію, місце розташування, примітку, нагадування, встановити повтор платежу або запланувати його на майбутнє. Ще однією перевагою застосунку є колірна диференціація транзакції відповідно до певної категорії витрат або доходів, інфографіка та проста, але наочна статистика.

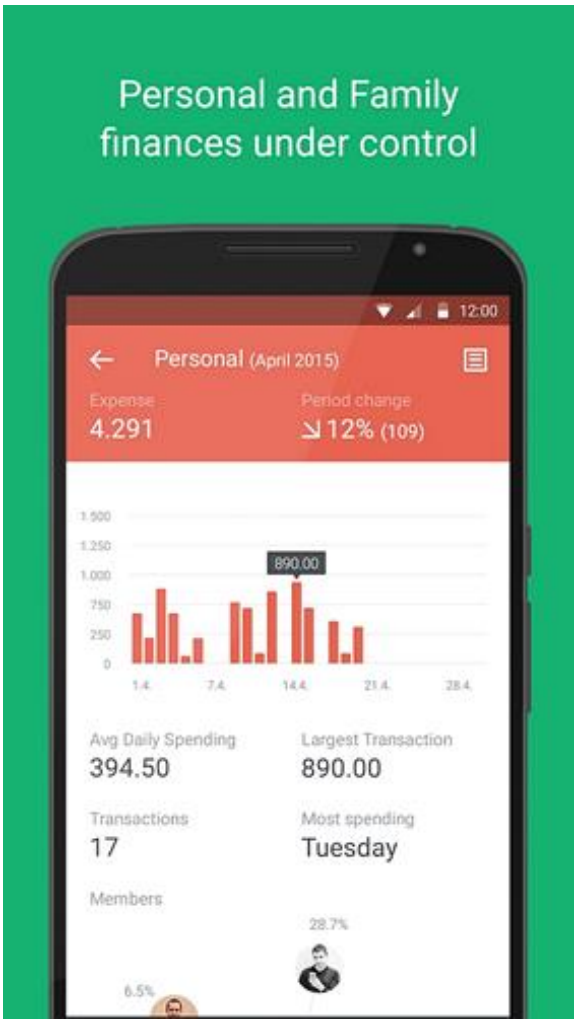


Рис. 2.5. Інтерфейс додатку Spendee

Загалом, Spendee – це справді простий фінансовий менеджер, який не претендує на звання найкращого або найбільш функціонального. Застосунок може бути доступний безкоштовно і за передплатою. Платна версія знімає обмеження на додавання гаманців, бюджетів, а також дає змогу додавати інших користувачів для загального управління фінансами [18].

2.2. Характеристика об’єкту розробки та постановка задачі

Метою магістерської роботи є створення веб-додатку для управління персональними фінансами. Цей додаток буде допомагати користувачам слідкувати за своїми фінансовими операціями та створювати зручні звіти для їх оцінки.

Програмне забезпечення застосунку розроблятиметься з урахуванням обробки даних користувачів, їх безпеки та можливості подальшого розширення. Дизайн застосунку буде простим, але функціональним, з інтуїтивно зрозумілим та адаптивним графічним інтерфейсом, який буде підлаштовуватись під різні пристрої.

В цій системі користувач може записувати транзакції, вказуючи деталі про дохід або витрату, дату, суму та короткий опис. Також користувач може групувати транзакції за категоріями для зручного управління своїми фінансами. Завдяки можливості розподілу транзакцій за категоріями, користувач може відстежувати, на що саме він витрачає свої гроші. Також додаток дозволяє користувачу створювати власні категорії, які найкраще будуть відповідати його потребам [19].

Окрім цього, фінансовий менеджер дозволяє користувачеві створювати звіти по транзакціям. За допомогою звітів, користувач може отримати уявлення про свої фінансові звички, зрозуміти, куди йдуть його гроші і як вони розподіляються між різними категоріями [20].

Взаємодія між клієнтом та сервером відбуватиметься за допомогою HTTP-запитів, зокрема GET, POST, PUT та DELETE запитів. Відповіді на запити надсилатимуться у форматі JSON, що спрощує обробку та передачу даних. Сервер матиме доступ до бази даних, де буде зберігатися інформація користувачів та їхні транзакції. Для забезпечення безпеки даних буде реалізовано захист від несанкціонованого доступу, включаючи шифрування паролів користувачів та використання токенів для автентифікації [21].

Таким чином, розробка додатку з урахуванням зазначених можливостей допоможе полегшити відслідковування персональних витрат, тримати хороший контроль над своїм фінансовим станом і допоможе планувати майбутні витрати. Це сприятиме підвищенню фінансової грамотності користувачів, надаючи їм зручні інструменти для аналізу та оптимізації фінансових рішень.

2.3. Вибір основних технологій та методів для проектування програмного продукту

2.3.1. Огляд проектування програмного забезпечення

Проектування програмного забезпечення - один з основних етапів розробки інформаційної системи. На цьому етапі приймається рішення щодо структури, функціональності та взаємодії компонентів системи. Застосування цих інструментів проектування представляє комплексний підхід у сфері розробки програмного забезпечення. Вони дозволяють чітко визначити вимоги до системи, спланувати архітектуру та передбачити логіку роботи всіх її частин. Це не тільки полегшує процес розробки, але й сприяє підвищенню якості кінцевого продукту.

Для проектування розроблюваного програмного продукту обрано такі технології та методи моделювання: діаграма прецедентів (use case діаграма), діаграми діяльності (activity діаграма), логічна модель бази даних та контекстна діаграма (IDEF0) [22].

2.3.2. Use-case діаграма

Одним з перших етапів проектування є створення Use Case діаграми, яка дозволяє визначити основні сценарії взаємодії користувача з системою. Ця діаграма є своєрідною візуалізацією функціональних вимог і дозволяє зрозуміти, які дії повинна виконувати система. У центрі діаграми розташовуються так звані «Варіанти використання» (Use Cases), які відповідають певній функції системи: додавання транзакцій, перегляд звітів або створення нових категорій витрат. Такі

варіанти використання матимуть так званих «акторів» - користувачів або зовнішні системи, що взаємодіють з програмою. Таким чином, розробник може побачити весь спектр можливостей системи і зрозуміти, як різні види ролей взаємодіють з функціоналом [23].

2.3.3. Activity діаграма

Ще одним потужним інструментом є діаграма діяльності, яка описує послідовність дій всередині системи. Цей тип діаграм є дуже корисним для моделювання динамічної поведінки програми, оскільки дозволяє побачити саму логіку певного процесу, наприклад, обробки транзакції або генерації фінансового звіту. Ця діаграма включає в себе окремі дії, які повинні послідовно виконуватися в системі, і рішення, на основі яких здійснюється логічне розгалуження - наприклад, у випадку перевірки коректності введених даних. Діаграми діяльності важливі для виявлення можливих вузьких місць у логіці процесу і допомагають забезпечити правильну взаємодію між модулями системи [24].

2.3.4. Логічна модель бази даних

Невід'ємною частиною проекту є логічна модель бази даних, призначена для визначення способу зберігання даних. Ця модель дозволяє структурувати інформацію у вигляді сутностей, їхніх атрибутів та зв'язків між ними. Для додатку персональних фінансів основними сутностями можуть бути, «Транзакції» та «Категорії». Правильна структура даних сприяє оптимальному управлінню інформацією та забезпечує ефективність запитів до бази даних, що особливо важливо для швидкої роботи додатку. Наприклад, сутність «Категорія» матиме ідентифікатор, ім'я та опис; а сутність «Транзакції» складатиметься з суми, дати, категорії та короткого опису. Зв'язки між сутностями забезпечують спосіб логічної інтеграції даних: кожна транзакція може бути віднесена лише до однієї з багатьох

категорій. Побудова моделі бази даних є важливою для ефективної і безпечної роботи з даними [25].

2.3.5. Контекстна діаграма

Контекстна діаграма (IDEF0) - ще один хороший інструмент проектування, який намагається описати взаємодію системи із зовнішнім середовищем. У загальному вигляді вона відображає основні функціональні блоки системи, її входи і виходи, механізми її реалізації та зв'язки із зовнішніми системами або користувачами. Якщо розглядати, наприклад, управління особистими фінансами, то основним функціональним блоком може бути «Додаток для менеджменту фінансів», що включає ряд користувацьких входів/транзакцій на вході, звіти та рекомендації на виході, а також внутрішній механізм, що являє собою базу даних або аналітичні алгоритми. Таким чином, при такому підході розробнику вдається зрозуміти, які ресурси необхідно розробити для роботи системи і як остання буде взаємодіяти із зовнішнім світом [26].

2.4. Вибір основних технологій для реалізації програмного продукту

2.4.1. Java – мова програмування для back-end розробки

Java була обрана з кількох причин: це одна з найбільш широко використовуваних мов програмування майже у всіх галузях, починаючи від веб-розробки та мобільних додатків і закінчуючи інтернет-рішеннями. Це забезпечує широку доступність ресурсів та документації для розробки проектів.

По-друге, мова Java не залежить від платформи, тобто код може працювати на кожному пристрої, де є віртуальна машина Java. Це надає коду більшої гнучкості та мобільності.

Крім того, існує велика кількість корисних бібліотек та фреймворків, які допомагають розробникам пришвидшити процес розробки та розширити функціональність своїх проектів.

Java є високозахищеною мовою програмування, що робить її більш стійкою до вразливостей, вірусів та інших загроз безпеки. Це особливо важливо в сучасному світі, де кібербезпека є великою проблемою [27].

2.4.2. Spring Framework

Обраний програмний фреймворк системи - Spring Framework. Це програмне забезпечення з відкритим вихідним кодом та широким інструментарієм, що дає змогу реалізовувати різноманітні складні та великі програми. Зокрема, існують модулі для розробки веб-додатків та мікросервісів, для забезпечення безпеки додатків тощо.

Spring Framework забезпечує простоту та гнучкість у розробці програм, що є важливим аспектом у сучасному програмуванні. Цей інструмент допомагає створювати зрозумілий і легко підтримуваний код. Spring відкритий для інтеграції з іншими інструментами та технологіями, що дає розробникам широкі можливості використовувати його для різноманітних потреб.

Однією з найбільших переваг використання цього фреймворку є велика спільнота розробників, яка постійно розвиває та оновлює цю технологію, що дозволяє створювати найсучасніші програмні продукти. Крім того, велика кількість документації та ресурсів, доступних в Інтернеті, робить Spring Framework доступним для розробників з будь-яким рівнем знань та досвіду.

Таким чином, Spring Framework є важливим інструментом для створення високоякісних додатків з високою гнучкістю та простотою. Потужність, гнучкість та підтримка роблять його ідеальним вибором для цього проекту. [28].

2.4.3. REST API

Для розробки веб-додатку було обрано REST (Representational State Transfer), оскільки це один з підходів до архітектури мережевих протоколів. Першим і найбільш очевидним є те, що він буде використовувати стандартні

протоколи, серед яких є HTTP, один з найпоширеніших в Інтернеті. Тоді у нього не буде проблем з інтеграцією з іншими сервісами, оскільки його специфікація гарантує, що проблеми сумісності не виникнуть.

REST дозволяє створити масштабовану архітектуру додатку. Можна ефективно розподіляти навантаження між серверами і масштабувати додаток в залежності від ваших потреб за допомогою стандартних HTTP-запитів. Це забезпечує високу продуктивність і гнучкість системи.

Крім того, використання REST дозволить розділити клієнтську та серверну частини додатку, щоб полегшити його розробку та подальше обслуговування. Таким чином, розробляти обидві частини можна незалежно, що забезпечить ще більшу гнучкість.

З REST буде безстанність, оскільки протокол REST не зберігає станів між запитами. Це означає, що кожен запит обробляється, не покладаючись на будь-який інший запит, що обробляється сервером. В процесі роботи це зменшує навантаження на сервер і ще більше підвищує продуктивність додатків.

Таким чином, використання REST при розробці веб-додатків забезпечує стандартизацію, масштабованість, розділення клієнта і сервера, безстанність і легку інтеграцію з іншими системами. Загалом, це дуже позитивно вплине на якість і продуктивність системи [29].

2.4.4. Система управління базами даних PostgreSQL

Додаток буде розроблено використовуючи базу даних PostgreSQL. Це потужна та надійна реляційна база даних, яка забезпечує стабільність в обробці даних та є ефективною у своїй роботі. Масштабованість і здатність обробляти великі обсяги інформації дозволяють ефективно працювати з даними в системі.

PostgreSQL - це продукт з відкритим вихідним кодом і великою спільнотою розробників. Це означає, що ви можете розраховувати на підтримку, оновлення та виправлення помилок від активної спільноти. Сьогодні це одна з найбільш

використовуваних баз даних у світі. Різноманітна документація та ресурси також допомагають розробникам ефективно працювати з PostgreSQL.

Обрана база даних є високофункціональною, підтримує всі види типів даних, і це дозволяє бути досить гнучким у роботі з різними типами даних, які будуть чудово працювати з іншими компонентами цього проекту.

Це важливо, оскільки дозволяє користувачам і додаткам безпечно зберігати, отримувати доступ і маніпулювати даними в системі. Її потужність, підтримка спільноти та розширені можливості роблять PostgreSQL ідеальним вибором для цього проекту [30].

2.4.5. HTML/CSS/JavaScript – технології програмування front-end

HTML, CSS та JavaScript - незамінні інструменти для розробки інтерфейсу веб-додатків. Кожен з них відіграє свою роль, але всі разом вони створюють зручний та ефективний користувацький інтерфейс.

HTML розшифровується як HyperText Markup Language (мова розмітки гіпертексту), і в основному використовується для структурування вмісту веб-сторінок. Ця мова визначає різні заголовки, абзаци, списки, таблиці та інші подібні елементи, які допомагають організувати інформацію. Це фактично основа кожного веб-додатку; без HTML неможливо розробити будь-який тип структурованого шаблону сторінки.

CSS відповідає за зовнішній вигляд сторінки. За допомогою CSS можна задавати кольори, шрифти, відступи, рамки та інші елементи дизайну. Ця мова дозволяє розробникам створювати привабливий і цілісний інтерфейс.

JavaScript - це мова програмування, яка забезпечує функціональність веб-додатків. З її допомогою можна реагувати на події, взаємодіяти з користувачем і динамічно змінювати вміст сторінки. Використовуючи JavaScript, можна створювати інтерактивні елементи, перевіряти форми, асинхронно спілкуватися з

сервером та багато іншого. Все це дозволяє зручно та ефективно взаємодіяти з бекендом системи.

На сьогоднішній день, використання HTML, CSS та JavaScript є фундаментом сучасної веб-розробки і дозволяють реалізовувати різноманітні проєкти, від простих веб-сторінок до складних додатків [31].

2.4.6. Середовище розробки IntelliJ IDEA

Для розробки цього проекту було обрано IntelliJ IDEA. IntelliJ IDEA - це потужне, багатофункціональне інтегроване середовище розробки (IDE) для продуктивної розробки. Завдяки таким функціям, як інтелектуальний редактор коду, автодоповнення, рефакторинг та іншим, підвищується продуктивність розробника та скорочується кількість помилок.

IntelliJ IDEA підтримує безліч технологій і мов програмування, серед яких Java, JavaScript, HTML, CSS - те, що потрібно проекту - і багато інших. Це дозволяє розробляти додаток з використанням різних технологій, підвищуючи гнучкість і можливості проекту.

Дуже зручно, що IntelliJ IDEA передбачає розширення та інтеграцію з іншими інструментами розробки, такими як системи контролю версій, конструктори проектів та засоби автоматизації. Це значно спрощує роботу з проектом і полегшує управління ним.

Підсумовуючи, IntelliJ IDEA пропонує зручне, потужне середовище розробки, яке допоможе ефективно працювати з додатком. Багатий набір функцій, підтримка різних технологій та інтеграція з іншими інструментами роблять його ідеальним для розробки нашого проекту [32].

2.4.7. Інструмент автоматизації Maven

Maven - це потужний інструмент в управлінні проектами з розробки ПЗ. Він автоматизує залежності, компіляцію, тестування та розгортання коду.

Основною перевагою використання цієї технології для розробки даного проекту є зручність управління його залежностями. Maven дозволить описати залежності у конфігураційному файлі pom.xml та автоматично завантажити бібліотеки з централізованого репозиторію. Це значно спростить розробку, оновлення та підтримку потрібних бібліотек.

Крім того, інструмент дозволяє автоматично компілювати, тестувати та збирати програмний код. Використання «виконавчих цілей» з Maven дозволить виконувати деякі операції, такі як компіляція, пакування, тестування, що дозволяє організувати повторне використання і автоматизувати процес збірки і розгортання програмного забезпечення.

Maven також керує життєвим циклом проекту. Різні типи конфігураційних етапів можуть бути використані для визначення покрокових процесів, що відбуваються під час розгортання проекту, таким чином допомагаючи розробникам створювати більш якісні результати стабільно швидше.

Загалом, Maven спрощує управління проектами та залежностями, автоматизує процеси збірки та розгортання, а також забезпечує чистоту та впорядкованість проекту, що значно полегшує життя розробникам програмного забезпечення. [33-34].

2.4.8. Система контролю версій Git

Git відіграє життєво важливу роль у сучасній розробці програмного забезпечення, пропонуючи численні переваги протягом усього процесу розробки. Однією з його головних переваг є можливості керування версіями. Розробники можуть ефективно відстежувати та керувати змінами у своєму коді з плином часу. Створюючи гілки, вони можуть працювати над конкретними функціями або виправленнями помилок незалежно, що дозволяє їм експериментувати і легко відкочувати зміни, коли це необхідно. Це допомагає створювати чистий і стабільний код.

Git також чудово підходить для розвитку співпраці та командної роботи серед розробників. Кілька членів команди можуть працювати над різними гілками одночасно, а Git надає інструменти для об'єднання та вирішення конфліктів, щоб безперешкодно інтегрувати їхні зміни. Це гарантує, що кожен має доступ до найновішого коду, сприяючи ефективній співпраці та координації.

Перегляд коду має вирішальне значення для підтримки якості коду, і цей інструмент спрощує цей процес. Розробники можуть створювати pull-запити та merge-запити для витягування або об'єднання, щоб поділитися своїми змінами коду з колегами. Це дозволяє отримати зворотній зв'язок, коментарі та пропозиції перед тим, як об'єднати зміни в основну кодову базу. Git-платформи, такі як GitHub або GitLab, часто пропонують вбудовані функції перегляду коду, що спрощує цикл зворотного зв'язку і покращує загальну якість коду.

Таким чином, Git є одним з найважливіших інструментів у розробці додатків, що забезпечує контроль версій, безперешкодну співпрацю, ефективний перегляд коду, відстеження та підтримку різних стратегій розгалуження. Його гнучкість, інтеграція з іншими інструментами розробки та здатність оптимізувати робочий процес роблять його наріжним каменем сучасної практики розробки програмного забезпечення.

2.4.9. Методологія розробки Agile

Потужність та гнучкість у розробці високоякісних програмних рішень - дві причини, чому методологія Agile здобула величезну популярність та визнання в індустрії розробки додатків. На відміну від типових розробок, ця методологія пропагує цінність співпраці, адаптивності та ітеративного розвитку для успішної реалізації проектів.

У сучасному динамічному бізнес-середовищі потреби клієнтів та ринкові тенденції можуть миттєво змінюватися. Гнучка методологія визнає цю реальність і сприяє зворотному зв'язку та адаптації на кожному кроці. Розбиваючи процес

розробки на невеликі, керовані кроки, так звані спринти, вона значно полегшує оцінку та коригування цілей проекту. Такий ітеративний підхід гарантує, що команда розробників завжди реагує на відгуки клієнтів і може легко вносити зміни протягом усього життєвого циклу проекту.

Ще однією значною перевагою цієї методології є її спрямованість на досягнення відчутного прогресу на кожній ітерації. На відміну від традиційних підходів, які часто вимагають тривалих циклів розробки, перш ніж будуть отримані будь-які результати, Agile сприяє ранньому і безперервному наданню робочого програмного забезпечення. Ця ітеративна модель розробки дозволяє швидко тестувати ідеї, визначати потенційні ризики та виявляти проблеми на ранніх стадіях.

Крім того, Agile заохочує культуру постійного вдосконалення. Вона передбачає цикли зворотного зв'язку, такі як ретроспективи, коли команда розмірковує над своєю практикою, визначає сфери, що потребують коригування, і впроваджує модифікацію. Наголос на вивченні досвіду та відповідному коригуванні змушує команду поступово вдосконалювати свої практики з кожною ітерацією.

Таким чином, методологія Agile виявилася дуже ефективною в розробці додатків завдяки своїй адаптивності, співпраці, поетапному впровадженню, розширенню можливостей команди та постійному вдосконаленню. Застосовуючи її принципи і практики, команди розробників краще справляються зі складністю проектів, швидше реагують на мінливі вимоги і розробляють програмні рішення, які відповідають потребам клієнтів і кінцевих користувачів, що постійно змінюються.

2.4.10. Фреймворк для тестування JUnit

JUnit - це, мабуть, один з найзручніших фреймворків для тестування при розробці програмного забезпечення. Цей інструмент надає розробникам надійні і

продуктивні методи для тестування коду невеликими частинами, гарантуючи правильну функціональність і відповідність бажаним специфікаціям. Незалежно від того, чи це малий або корпоративний рівень, цей інструмент у життєвому циклі розробки потенційно може принести багато переваг.

Одним з ключових позитивних моментів використання такого фреймворку є його простота: він забезпечує інтуїтивне розуміння тестів, тому сам інструмент є зрозумілим для більшості користувачів при розробці, підтримці, доопрацюванні тощо. За допомогою JUnit можна швидко писати тестові кейси, які перевіряють поведінку окремих частин коду, таких як методи або класи, забезпечуючи гарантію від регресії або збоїв протягом усього циклу тестування додатку, що розробляється.

Ще одним важливим аспектом JUnit є те, що він робить процес тестування автоматизованим. Коли додаток стає складним, ручна перевірка його функціональності займає багато часу і стає схильною до помилок. JUnit вирішує цю проблему, надаючи фреймворк для написання автоматизованих наборів тестів. Ці набори містять набір тестових кейсів, які можуть бути виконані однією командою. Автоматизуючи процес тестування, розробники заощаджують дорогоцінний час та забезпечують узгодженість і надійність результатів.

Крім того, JUnit добре поєднується з середовищами розробки та інструментами збірки. Він підтримує IntelliJ IDEA, що означає, що ви можете легко запускати і налагоджувати тестові кейси разом з вашим кодом. Він також досить добре працює з інструментами управління проектами, такими як Apache Maven, що дозволяє інтегрувати автоматизоване тестування в конвеєри безперервної інтеграції та розгортання. Це ще більше підвищує ефективність і продуктивність процесу розробки.

В цілому, JUnit - це дуже потужний фреймворк для тестування, який може значно підвищити ефективність процесу розробки. Його простота, можливості автоматизації, популяризація хороших практик кодування, безшовна інтеграція,

співпраця та стабільність зробили його надзвичайно корисним інструментом для розробників. Впроваджуючи цей фреймворк тестування в процес розробки додатків, стає доступним створення надійних програмних систем, які гарантують дотримання стандартів якості.

2.5. Висновки до розділу 2

В другому розділі обрані діаграми для опису різних аспектів системи:

- use-case діаграма для опису основного функціоналу системи;
- activity-діаграми для опису послідовності дій різного функціоналу;
- діаграма бази даних;
- контекстна діаграма системи для опису зовнішніх елементів, що взаємодіють із системою.

Проаналізовано можливі методи і засоби реалізації проєкту. Обрана мова програмування Java, основний фреймворк – Spring Framework, прийнято слідувати REST, обрана база даних PostgreSQL, графічний інтерфейс розроблено засобами HTML/CSS/Javascript, обрано середовище IntelliJ IDEA, Maven для допомоги у керуванні проєктом та JUnit як фреймворк для тестування. В якості системи для контролю версій обрано Git, а основною методологією розробки прийнято Agile.

РОЗДІЛ 3
РОЗРОБКА МОДЕЛЕЙ, МЕТОДІВ ТА АЛГОРИТМІВ ДЛЯ РЕАЛІЗАЦІЇ ПРОГРАМНОГО РІШЕННЯ

3.1. Модель структури системи

Системна структура додатку буде відповідати стандартній структурі додатку на Spring Framework. Таку структуру можна назвати модульною, в ній окремі модулі виконують конкретні передбачені ними функції і при необхідності викликають інші компоненти, які необхідні для обробки запиту користувача. Діаграма основних компонентів та їх взаємодія зображені на рис. 3.1.

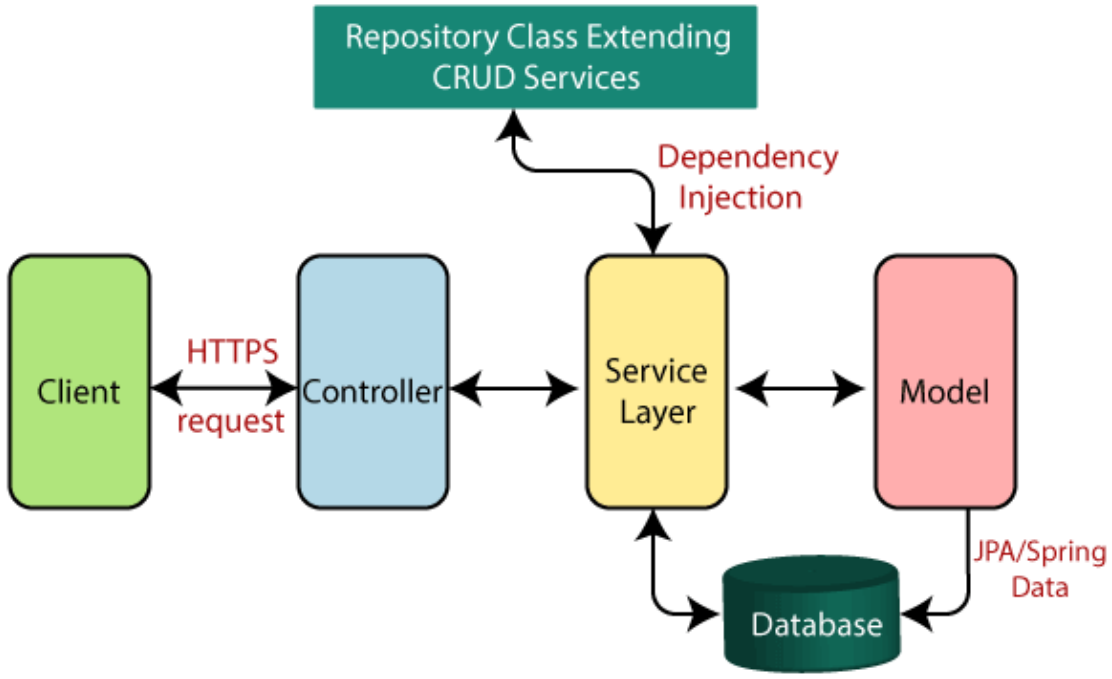


Рис. 3.1. Модель взаємодії основних компонентів системи

Клієнт: клієнтський компонент представляє зовнішні сутності, які взаємодіють з додатком, такі як веб-браузер, мобільний додаток або інший сервіс. Клієнт надсилає HTTP запити до серверної частини системи.

Контролер: компонент контролера отримує та обробляє вхідні запити від клієнта. Контролер відповідає за зіставлення вхідних запитів з конкретними методами та обробку параметрів запиту. Він також готує дані для відправки у відповідь.

Сервісний рівень: компонент сервісного рівня містить бізнес-логіку вашого додатку. Він інкапсулює складну обробку і виконує операції над даними. Сервісний рівень діє як міст між контролером і базою даних. Він отримує дані від контролера, виконує будь-які необхідні перетворення або перевірки, а потім взаємодіє з базою даних, щоб отримати дані або маніпулювати ними.

Модель: компонент моделі представляє структури даних або об'єкти, що використовуються у додатку. Він визначає структуру і поведінку даних. Класи моделі, як правило, є простими Java класами, які зберігають дані і надають геттери та сеттери для доступу до даних.

Репозиторій: компонент репозиторію відповідає за взаємодію з базою даних. Він забезпечує абстракцію над базою даних і пропонує методи для запитів, вставки, оновлення та видалення даних. У даній системі класи сховища створюються шляхом розширення операцій CRUD (Create, Read, Update, Delete), що надаються Spring Data JPA.

База даних: Компонент бази даних представляє систему зберігання, де зберігаються дані додатку. Для даного проєкту обрано реляційну базу даних PostgreSQL.

Взаємодія між цими компонентами буде відбуватись за такою схемою:

1) Клієнт надсилає HTTP запит на ендпоінт, визначений в контролері;
2) Контролер отримує запит і перевіряє всі вхідні дані;
3) Потім контролер викликає відповідний метод у сервісному рівні, передаючи всі необхідні дані;
4) Сервісний рівень виконує необхідну бізнес-логіку, таку як обробка даних, перетворення або перевірка;

5) Якщо сервісному рівню потрібно взаємодіяти з базою даних, він викликає відповідний метод у репозиторії;
6) Репозиторій взаємодіє з базою даних, виконуючи необхідні SQL-запити;
7) Після того, як репозиторій повертає запитувані дані або завершує запитувану операцію, сервісний рівень може виконати додаткову обробку, якщо це необхідно;
8) Нарешті, сервісний рівень повертає відповідь контролеру, який готує і надсилає відповідь назад клієнту.

Така взаємодія забезпечує структурований розподіл завдань: клієнт взаємодіє з контролером, який делегує бізнес-логіку сервісному рівню, а доступ до даних – репозиторію, забезпечуючи модульну архітектуру, що легко підтримується.

3.2. Моделі, які описують функціонування системи та її окремих блоків

3.2.1. Діаграма прецедентів (Use case diagram)

На рис. 3.2 зображена use case діаграма даної системи. В ній можна виділити двох акторів та 11 прецедентів. Актори: *Unauthorized User* та *Authorized User* та прецеденти: *Create new account, Sign in existing account, Manage categories, Manage transactions, Manage reports, Create new categories, Edit existing categories, Detele existing categories, Create new transactions, Edit existing transactions, Detele existing transactions*.

Зображені прецеденти описують основні можливості системи. Звідси можна зрозуміти, що неавторизований користувач не може використовувати всі можливості, для цього йому потрібно виконати одну з двох доступних дій: зареєструвати новий аккаунт або увійти в існуючий.

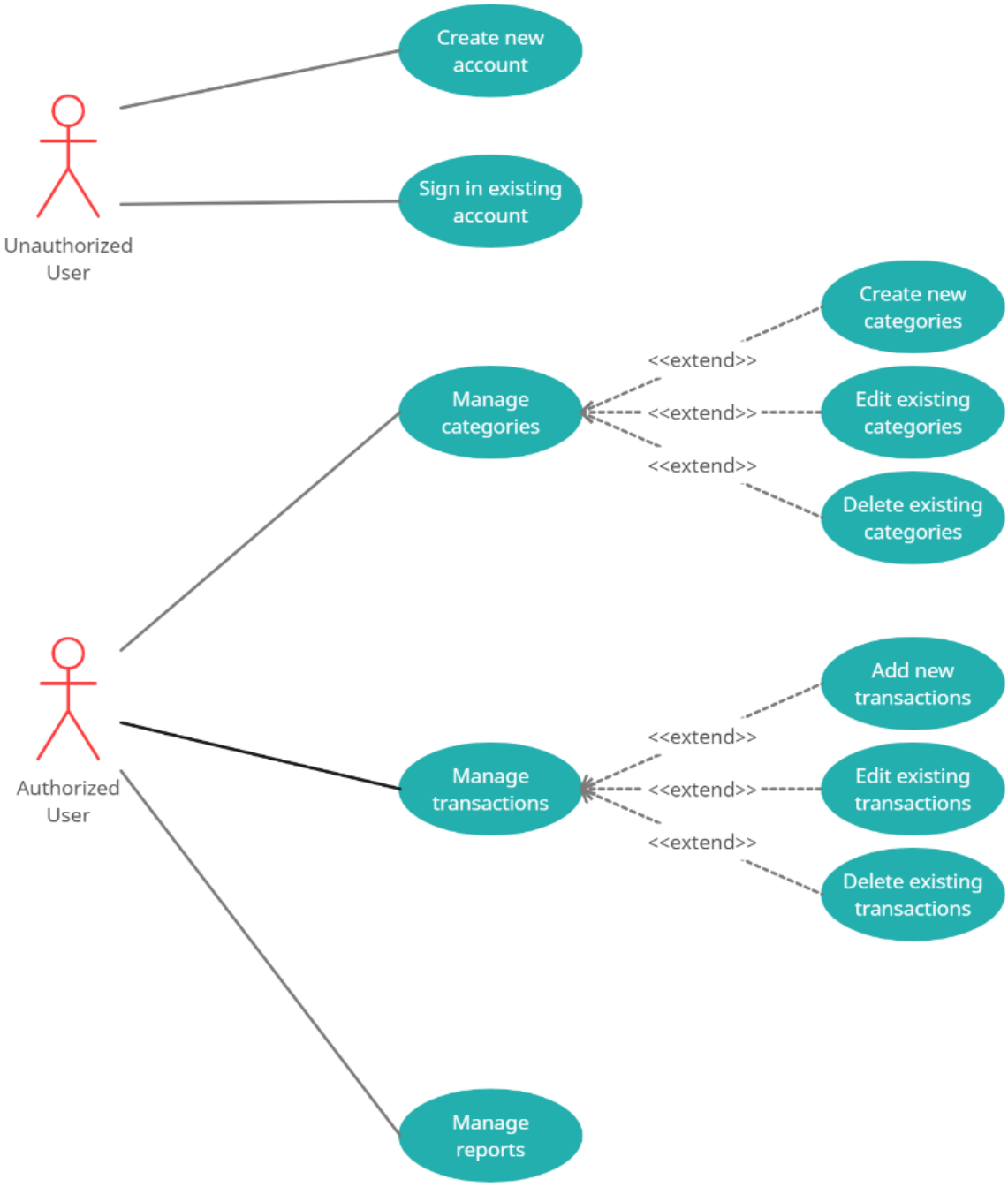


Рис. 3.2. Use case діаграма системи

3.2.2. Activity-діаграма

На рисунку 3.3 зображена activity-діаграма прецеденту Manage categories. Вона описує послідовність дій в системі та можливі розгалуження.

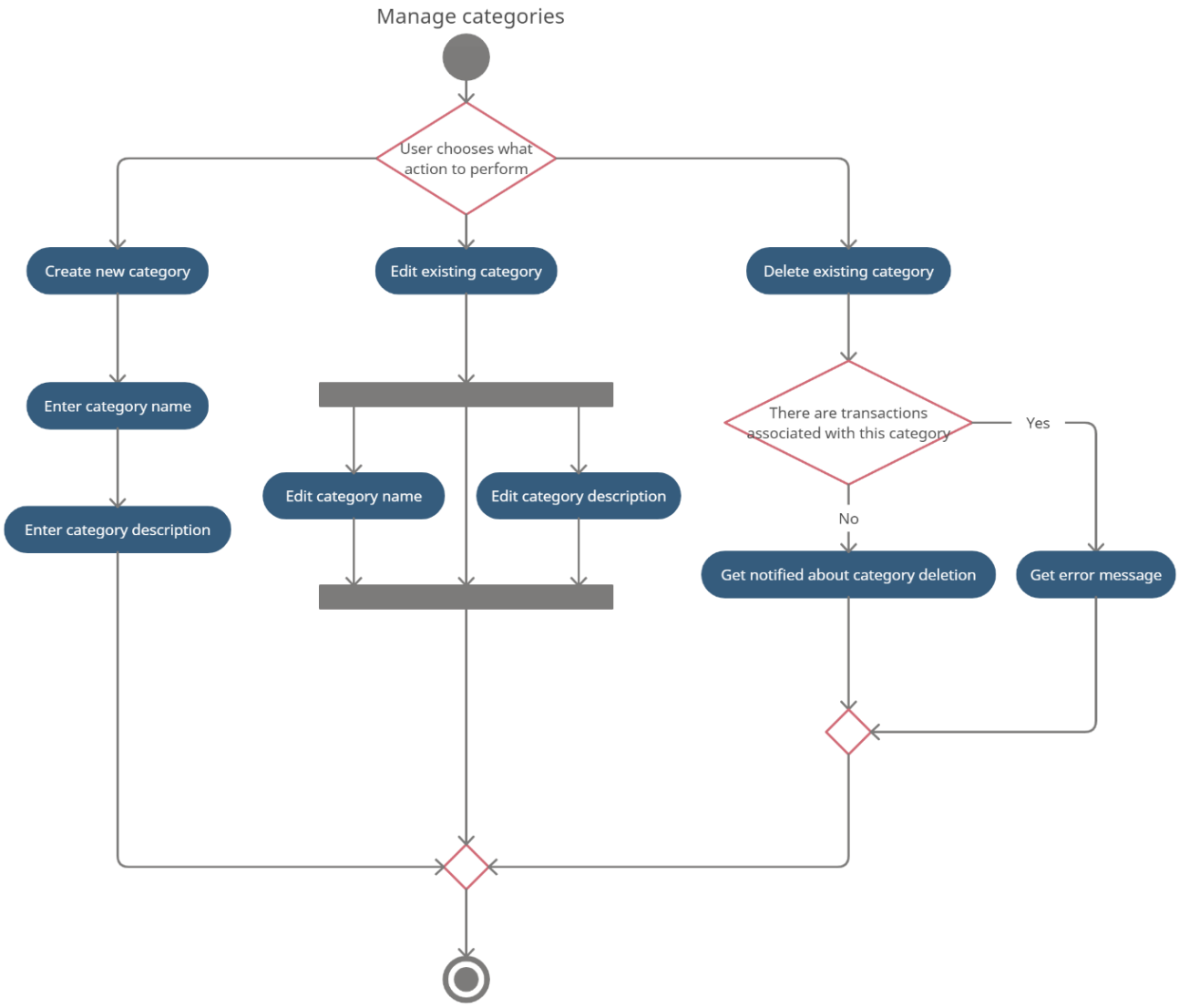


Рис. 3.3. Activity діаграма для прецеденту Manage categories

В логіка керування категоріями є досить прямолінійною. Користувач може виконати три базові дії: створити нову категорію для транзакцій, редагувати існуючу або видалити категорію. При обранні «створити нову категорію» користувачу потрібно ввести назву та опис категорії. При обранні «редагувати існуючу категорію» може змінити назву або опис категорії. При обранні «Видалити категорію», якщо до категорії не належить ніяких транзакцій, користувач отримає повідомлення про успішне видалення категорії.

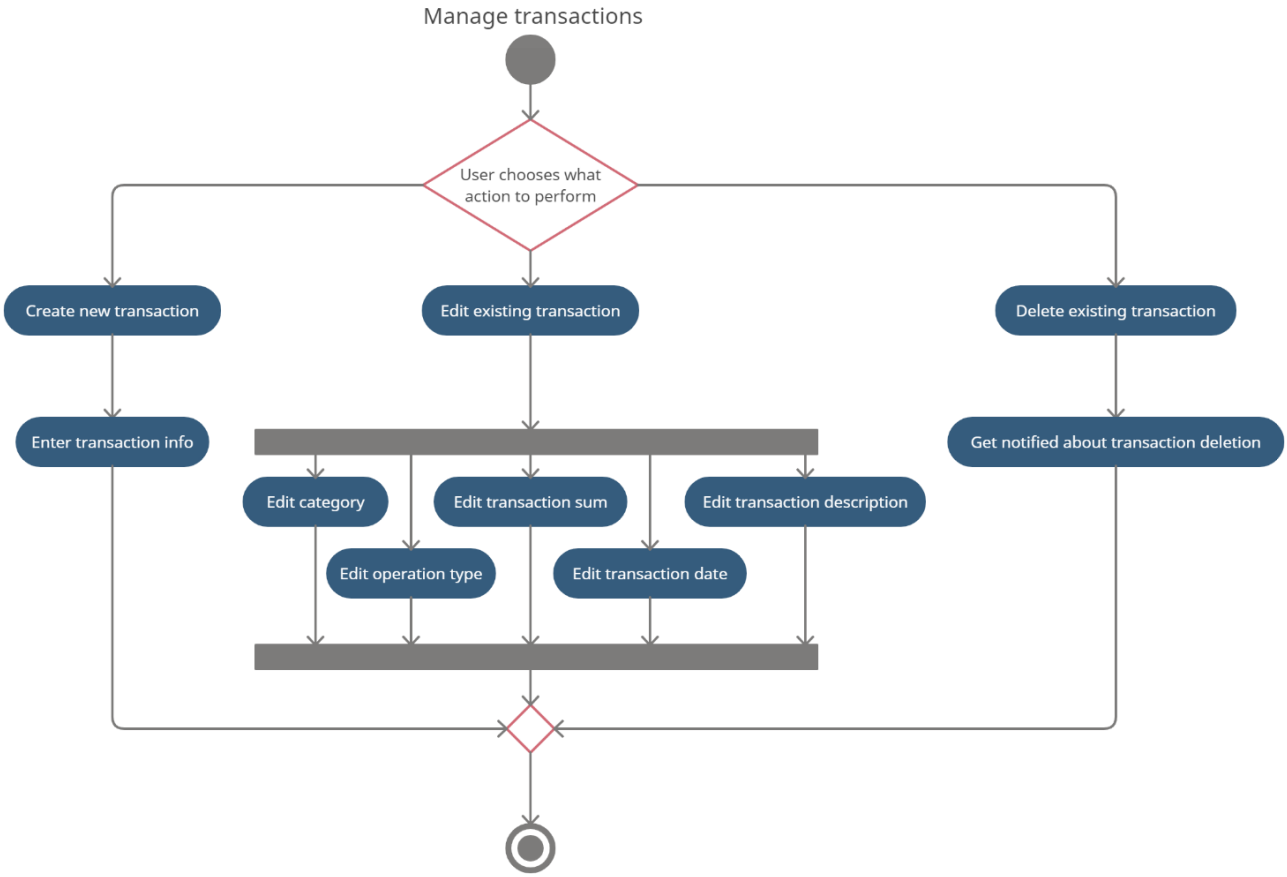


Рис. 3.4. Activity діаграма для прецеденту Manage transactions

Керування транзакціями в цілому слідує такій же логіці, як і керування категоріями, що було описане вище. Користувач може виконати три базові дії: створити нову транзакцію, редагувати існуючу транзакцію або видалити транзакцію. При обранні «створити нову транзакцію» користувачу потрібно вибрати категорію, тип операції, суму транзакції, дату та короткий опис транзакції. При обранні «редагувати існуючу транзакцію» може змінити ці ж властивості: категорія, тип, сума, дата і опис транзакції. При обранні «Видалити транзакцію», якщо в транзакції немає особливих обмежень або заборон – її буде видалено та користувач отримає повідомлення про успішне виконання операції та транзакція зникне зі списку.

3.2.3. Модель структури бази даних

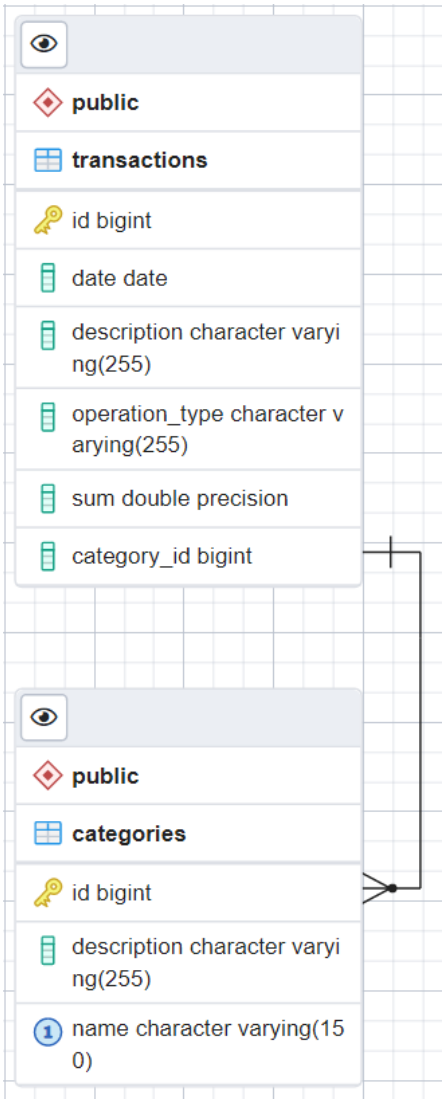


Рис. 3.5. Діаграма бази даних, створена за допомогою середовища pgAdmin 4 (СУБД для postgreSQL)

В цілому логіка бази даних дуже проста. В нас є категорії і до цих категорій належать транзакції. Оскільки додаток задумувався для персонального користування, то в нас немає необхідності робити прив’язку набору транзакцій до різних користувачів, що дуже спрощує структуру бази даних, що дуже зменшує навантаження на базу даних, а отже, система буде використовувати дуже мало ресурсів операційної системи в цілому.

3.2.4. Контекстна діаграма

На рис. 3.6 зображена контекстна діаграма. На ній зображений загальний опис системи і її взаємодія із зовнішнім середовищем.

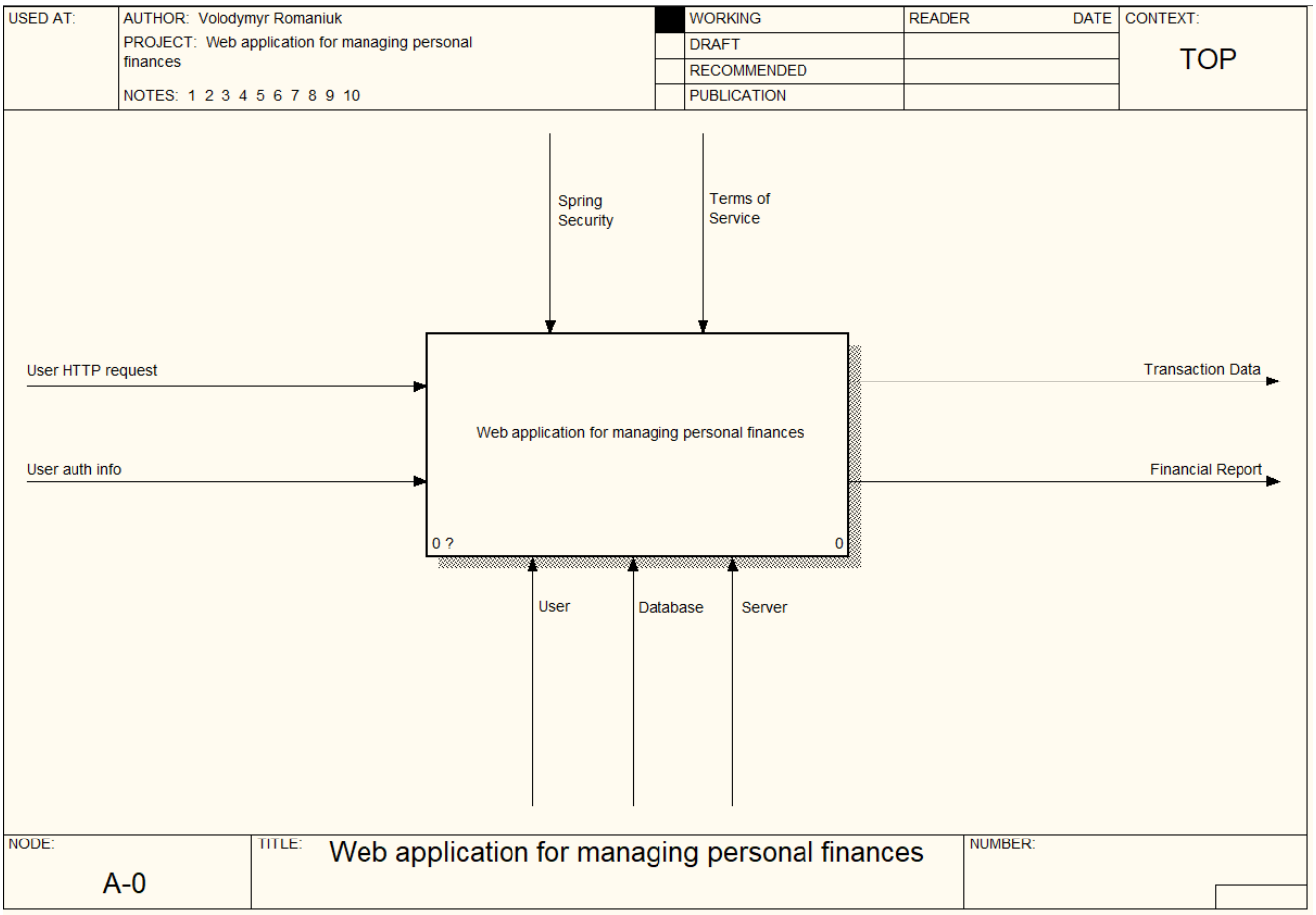


Рис. 3.6. Контекстна діаграма, створена за допомогою середовища AllFusion Process Modeler

З діаграми можемо зрозуміти, що в систему будуть входити HTTP-запити від користувача, за допомогою яких він буде спілкуватись із системою. На виході користувач буде мати впорядковані дані про свої транзакції та фінансові звіти. Взаємодіяти із системою буде користувач, база даних та сервер. Регулюють ці процеси *Умови використання* та *Spring Security* – модуль, який відповідає за безпеку.

3.3. REST API

Перш за все, необхідно конкретизувати поведінку REST API та методи взаємодії з ним. Потрібно реалізувати операції CRUD. Тому, логічним буде виділення двох сутностей у дві колекції: categories та transactions.

Згідно зі стандартами REST API, будуть прийматись та оброблятись такі запити, які зазначено у таблиці 3.1.

Таблиця 3.1

Перелік можливих HTTP запитів

Шлях	**Метод**	**Опис**
`/categories`	GET	Отримати список усіх категорій
	POST	Створити нову категорію
`/categories/<id>`	GET	Отримати інформацію про категорію
	DELETE	Видалити категорію
	PUT	Оновити інформацію про категорію
`/transactions`	GET	Отримати список усіх транзакцій
	POST	Створити нову транзакцію
`/transactions/<id>`	GET	Отримати інформацію про транзакцію
	DELETE	Видалити транзакцію
	PUT	Оновити інформацію про транзакцію

Приклади запитів:

GET запит на `/categories/1`:

```
{"id":1,"name":"categoryName1","description":"Description1"}
```

Отримаємо інформацію у вигляді JSON про категорію з id = 1;

DELETE запит на `/categories/1` видалить категорію з id = 1 з бази даних;

POST запит:

```
/categories?name=categoryName1&description=Description1
```

Створить нову категорію з ім’ям `categoryName1` та описом `Description1`.

PUT запит на `transactions/4` з наступним тілом:

```
{
"id":4,
"category":
     {
     "id":1,
     "name":"categoryName1",
     "description":"Description1"
     },

"operationType":"Spending",
"sum":"2",
"date":"2024-11-04",
"description":"Description 4"
}
```

Оновить дані про транзакцію з id 4.

REST вимагає, щоб контролери у відповідь для усіх запитів повертали статус коди 200 (OK), якщо операція успішна, 201 (CREATED) для POST запитів, та 404 (NOT_FOUND) якщо операція не виконана.

3.4. Структура проєкту

Структура проєкту в середовищі розробки згідно патерну MVC.

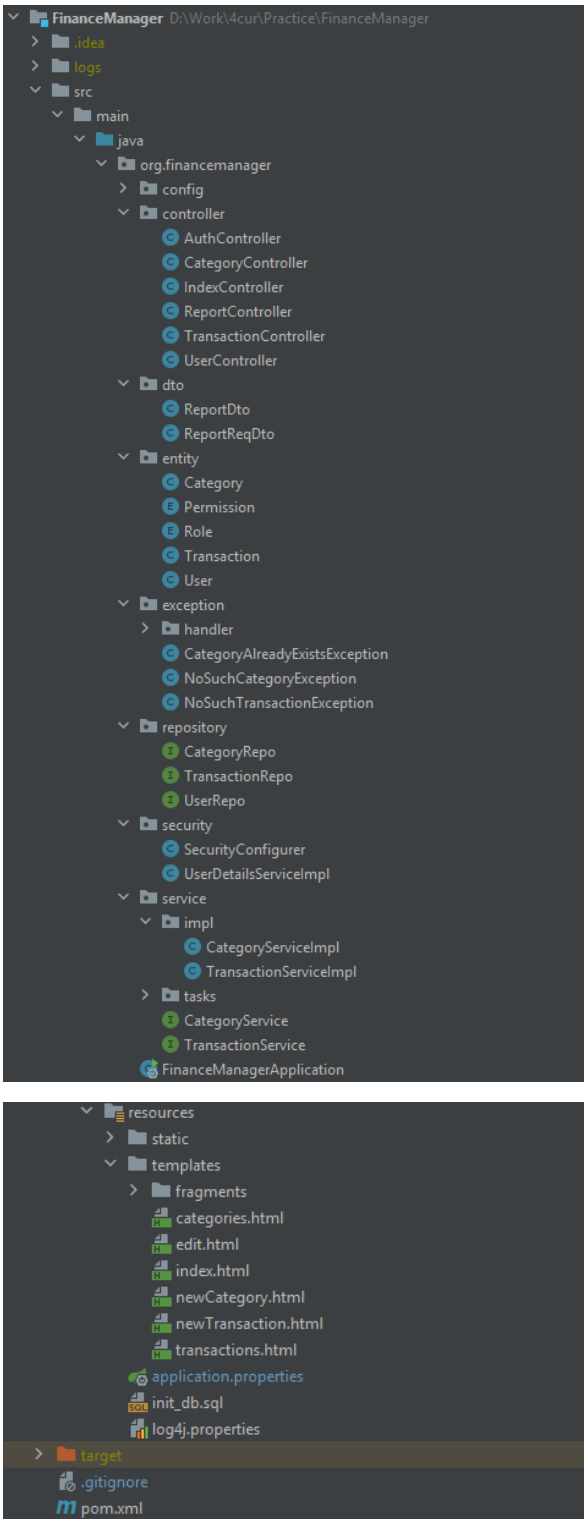


Рис. 3.7. Структура проєкту в середовищі IntelliJ IDEA

Опис структури:

java/org.financemanager:

- **config –** пакет містить в собі класи конфігурацій, в даному випадку для Spring MVC;
- **controller** – пакет містить контролери для обробки HTTP запитів;
- **dto** – пакет містить класи Data Transfer Object, які використовуються для передачі даних між різними компонентами програми;
- **entity** – пакет містить класи сутностей системи;
- **exception** – пакет містить класи виключень та їх обробників;
- **security –** пакет містить класи, які конфігурують модуль Spring Security;
- **repository** – пакет містить інтерфейси JPA-репозиторіїв;
- **service** – пакет містить інтерфейси сервісів для операцій з даними та імплементації цих інтерфейсів.

java/resources:

- **static** – містить статичні файли ресурсів, такі як файли CSS та JS-скрипти;
- **templates** – містить шаблони HTML-сторінок.

3.5. Діаграма класів

На рис. 3.8 представлена діаграма класів проєкту, сформована за допомогою середовища IntelliJ IDEA. Вона зображує зв’язки між різними класами системи. Можна побачити як `CategoryController, TransactionController, Report Controller,` які відповідають за обробку HTTP-запитів, звертаються до сервісного шару (`CategoryService, TransactionService`), який виконує бізнес-логіку, обробляючи запити від контролерів і здійснюючи відповідні операції над даними. Для доступу до бази даних сервісний шар взаємодіє з репозиторіями (`CategoryRepo, TransactionRepo`).

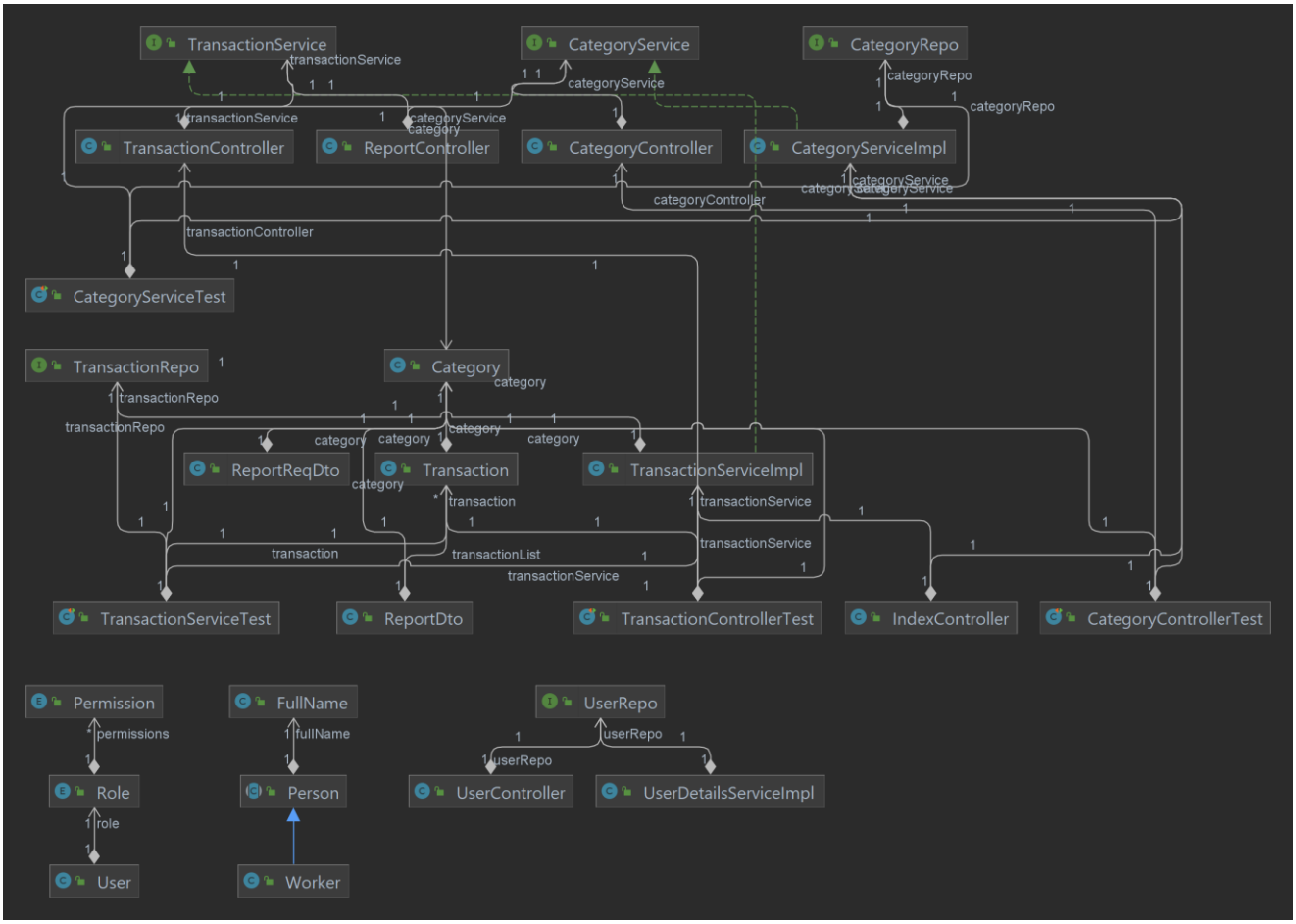


Рис. 3.8. Діаграма класів проєкту, сформована за допомогою середовища IntelliJ IDEA

3.6. Опис використаних сторонніх бібліотек та модулів

- **Lombok** – бібліотека для Java, додає набір анотацій, які при додаванні до Java-класів автоматично генерують відповідний код під час компіляції. Це усуває необхідність писати повторюваний код, такий як, наприклад, геттери та сеттери вручну, роблячи код лаконічнішим і простішим в обслуговуванні [35].
- **JUnit** – бібліотека для Java, надає набір анотацій, тверджень та засобів тестування, які спрощують процес написання та виконання тестів. Використовуючи цю бібліотеку, можна систематизувати тести і легко відстежувати їх результати [36].

- **Hibernate** – бібліотека для Java, спрощує процес співставлення об'єктів Java до таблиць в реляційних базах даних і навпаки. Вона надає зручний спосіб взаємодії з базами даних за допомогою об'єктно-орієнтованого програмування, що робить операції з базами даних більш інтуїтивно зрозумілими та ефективними [37].
- **Bootstrap** – надає колекцію стилів CSS, компонентів JavaScript і заздалегідь розроблених шаблонів, які можна використовувати для створення привабливих і зручних користувацьких інтерфейсів. Він також пропонує адаптивну систему, яка дозволяє легко створювати макети, що адаптуються до різних розмірів екранів і пристроїв [38].
- **Thymeleaf** – розроблений для полегшення роботи з веб-фреймворками Java (в випадку даного проєкту це Spring Framework), що робить його хорошим вибором для полегшення взаємодії зі стороною сервера у веб-додатках на основі Java. Він забезпечує простий та інтуїтивно зрозумілий спосіб вбудовування динамічних даних у HTML-шаблони [39].

3.7. Розробка та опис методів і алгоритмів програмних модулів

Запуск програми буде відбуватись за допомогою класу `FinanceManagerApplication` (рис. 3.9). Цей клас можна назвати точкою входу в програму.

На перший погляд в ній майже нічого немає, але використовуючи Spring Framework дуже багато функціоналу можна реалізувати за допомогою анотацій. Особливо варто відмітити анотацію `@SpringBootApplication`, яка встановлює налаштування за замовчуванням. За допомогою однієї цієї анотації та одного виклику методу почне виконуватись вся програма. Також, для зміни налаштувань, в цю анотацію можна додавати параметри, які будуть застосовані під час виконання `FinanceManagerApplication`.

```java
package org.financemanager;

import ...

Volodymyr Romaniuk
@SpringBootApplication
public class FinanceManagerApplication {
    Volodymyr Romaniuk
    public static void main(String[] args) {
        SpringApplication.run(FinanceManagerApplication.class, args);
    }
}
```

Рис. 3.9. Код компоненту FinanceManagerApplication

3.7.1. Контролери

Контролери – це API, який будуть використовувати на стороні фронт-енду. З ними спілкування буде відбуватись за допомогою запитів.

```java
package org.financemanager.controller;

import ...

Volodymyr Romaniuk
@RestController
@RequestMapping("/categories")
public class CategoryController {

    public static final Logger logger = LogManager.getLogger(CategoryController.class);

    6 usages
    private final CategoryService categoryService;

    Volodymyr Romaniuk
    @Autowired
    public CategoryController(CategoryService categoryService) { this.categoryService = categoryService; }

    Volodymyr Romaniuk
    @GetMapping
    @PreAuthorize("hasAuthority('categories:read')")
    public ResponseEntity<List<Category>> findAll(Model model){
        logger.info("Getting categories list");
        List<Category> categories = categoryService.findAll();
        model.addAttribute( attributeName: "categoriesList", categories);
        return new ResponseEntity<>(categories, HttpStatus.OK);
    }
```

Рис. 3.10. Частина коду контролера CategoryController

Контролер на рисунку 3.10 представляє з себе клас з анотацією `@RestController`. Також анотація `@RequestMapping("/categories")` значить що цей контролер буде приймати запити з ендпоінтом categories. Всередині цього класу описані методи, до яких можна звертатись за допомогою HTTP-запитів. На цьому рисунку побачимо метод `findAll`, над яким знаходиться анотація `@GetMapping`, яка означає, що даний метод буде викликатись при отриманні GET-запиту. З коду можна зрозуміти, що метод `findAll` повертає список усіх категорій, які знаходяться в базі даних.

3.7.2. Сутності

Далі переходимо до сутностей, які є в системі. На рис. 3.11 представлений код класу сутності, що описує категорії.

```
1   package org.financemanager.entity;
2
3   import ...
12
    Volodymyr Romaniuk +1 *
13  @Builder
14  @AllArgsConstructor
15  @NoArgsConstructor
16  @Data
17  @With
18  @Entity
19  @Table(name = "categories")
20  public class Category implements Serializable {
21      @Id
22      @GeneratedValue(strategy = GenerationType.IDENTITY)
23      private Long id;
24      @NotBlank(message = "Name must not be empty")
25      @Size(min = 2, max = 150, message = "Name length must be 2 to 150 symbols")
26      @Column(name = "name", nullable = false, unique = true)
27      private String name;
28      @Column
29      private String description;
```

Рис. 3.11. Код класу сутності Category

Одразу можна помітити, що код сутності суттєво відрізняється від звичних класів у Java. В даному класі використовується багато анотацій для спрощення роботи й розуміння коду. За допомогою бібліотеки Lombok можна скоротити об'єм коду для класів сутностей, наприклад, необов'язково вручну писати всі методи для редагування та отримання даних з кожного поля класу. Завдяки анотаціям `@Data`, `@Builder`, `@AllArgsConstructor`, `@NoArgsConstructor` програма буде автоматично створювати всі необхідні для класу методи, конструктори, тощо. Це дозволяє легко і швидко додавати або змінювати існуючі поля і не турбуватись, що в пов'язаних з ними методах могли б виникати помилки. Анотації `@Entity` та `@Table` зазначають, що даний клас відображає сутність в базі даних та дозволяє зв'язати з ним конкретну таблицю. Також можна використовувати анотації на полях класу, щоб вказувати які з них відповідають колонкам таблиці в базі даних, задавати обмеження, наприклад, вказати можливий розмір назви категорії або заборонити залишати її порожньою.

3.7.3. Звертання до бази даних

Сучасний підхід до побудови запитів до бази даних в Spring реалізований через спеціальні інтерфейси – репозиторії. Код одного з таких репозиторіїв зображений на рис. 3.12.

```
1   package org.financemanager.repository;
2
3   import ...
9
        5 usages  Volodymyr Romaniuk +1 *
10  @Repository
11  public interface CategoryRepo extends JpaRepository<Category, Long> {
            Volodymyr Romaniuk +1
12      @Query("select c from Category c order by c.id asc")
13      List<Category> findAll();
14  }
```

Рис. 3.12. Код класу CategoryRepo

`@Repository` – анотація вказує на те, що це репозиторій;

`JpaRepository<`*`Сутність, тип_її_ID`*`>` - включає всі заготовлені запити до бази даних. Це дозволить легко звертатись до сутності навіть без необхідності самому прописувати всі потрібні SQL запити.

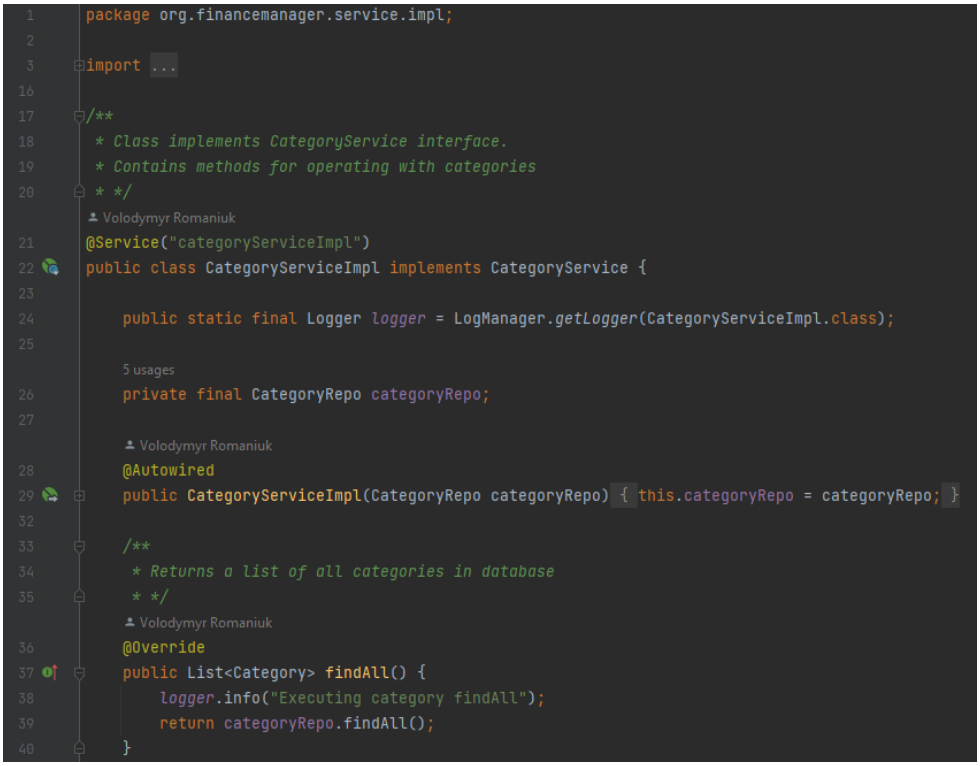
`@Query` – дозволяє створювати власні запити, якщо нас не задовільняють існуючі, або якщо потрібний запит відсутній в заготовлених. На рис. 3.12 її використано для вказання порядку сортування категорій при поверненні списку всіх категорій.

3.7.4. Сервіси

В контролерах було б занадто громіздким описувати всю логіку, тому для цього існують спеціальні класи, які позначаються анотацією `@Service` і уже з них будуть викликатись необхідні методи для запитів (рис. 3.7). Хорошою практикою вважається створення інтерфейсів з потрібними методами, а потім написання імплементації для цих інтерфейсів (рис. 3.13).

```
1   package org.financemanager.service;
2
3   import ...
7
    1 implementation  Volodymyr Romaniuk
8   public interface CategoryService {
        1 implementation  Volodymyr Romaniuk
9       List<Category> findAll();
10
        1 implementation  Volodymyr Romaniuk
11      Optional<Category> findById(Long id);
12
        1 implementation  Volodymyr Romaniuk
13      Category save(Category category);
14
        1 implementation  Volodymyr Romaniuk
15      Category update(Long id, Category category);
16
        1 implementation  Volodymyr Romaniuk
17      void delete(Long id);
```

Рис. 3.13. Код інтерфейсу сервісу CategoryService



```java
package org.financemanager.service.impl;

import ...

/**
 * Class implements CategoryService interface.
 * Contains methods for operating with categories
 * */
Volodymyr Romaniuk
@Service("categoryServiceImpl")
public class CategoryServiceImpl implements CategoryService {

    public static final Logger logger = LogManager.getLogger(CategoryServiceImpl.class);

    5 usages
    private final CategoryRepo categoryRepo;

    Volodymyr Romaniuk
    @Autowired
    public CategoryServiceImpl(CategoryRepo categoryRepo) { this.categoryRepo = categoryRepo; }

    /**
     * Returns a list of all categories in database
     * */
    Volodymyr Romaniuk
    @Override
    public List<Category> findAll() {
        logger.info("Executing category findAll");
        return categoryRepo.findAll();
    }
```

Рис. 3.14. Частина коду класу CategoryServiceImpl, який імплементує інтерфейс сервісу CategoryService

Дана імплементація (див. рис. 3.14) містить методи для роботи з категоріями, наприклад, отримання списку всіх категорій, отримання категорії по ID, збереження нової категорії тощо.

3.8. Розробка та опис інтерфейсу користувача

Розробка користувацького інтерфейсу відбувається за допомогою HTML – для структурування вмісту веб-сторінок, CSS – для відповідає за зовнішній вигляд сторінки, а JavaScript – для надання веб-додатку функціональності та динамічності.

Для спрощення реалізації адаптивного інтерфейсу буде використовуватись Bootstrap. За допомогою його інструментів можна дуже легко розробити інтерфейс, який буде автоматично підганяти свій розмір під розмір екрану пристрою користувача.

Щоб його використати, необхідно з офіційного сайту Bootstrap скачати його файли та додати до проєкту. Під'єднати відповідні .css та .js файли до конкретної веб-сторінки можна за допомогою тегів `<link>` всередині «голови» документу `<head>` та `<script>` в кінці «тіла» документу `<body>`. Відповідно:

```
<link rel="stylesheet" href="css/bootstrap.min.css">
<script src="js/bootstrap.js"></script>
```

Тепер можна використовувати вбудовані класи Bootstrap, щоб, наприклад, створювати кнопки за допомогою відповідних класів `btn`, `btn-success`, `btn-secondary`, `btn-danger`.

Таким чином можна легко отримувати різні кнопки для доступу до функцій додатку (рис. 3.15). Елементи, створені за допомогою вище описаних класів будуть одразу мати адаптивні властивості, тобто, вони будуть підлаштовуватись під розмір екрану пристрою.

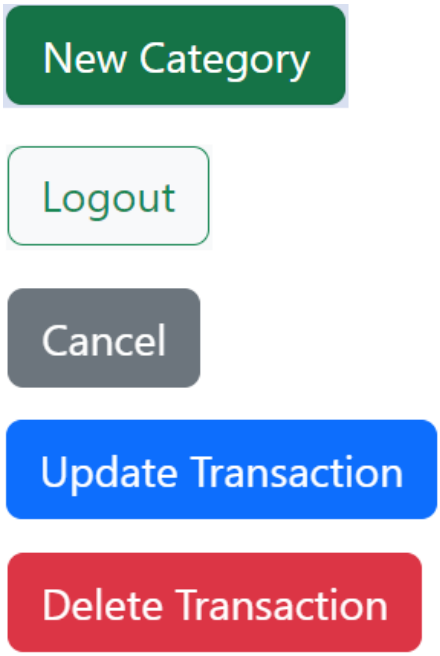


Рис. 3.15. Варіанти кнопок, створені за допомогою вбудованих CSS класів Bootstrap

Тепер можна приступати до розробки самого інтерфейсу користувача. Варто почати з макету сторінок. Сторінки будуть складатись з трьох основних елементів – header, в якому розташоване навігаційне меню, основна частина, в якій, власне, буде відбуватись основна взаємодія користувача із системою, та footer – блок, в якому зазвичай вказують власника, контактні дані та іншу додаткову інформацію.

Header та Footer будуть однакові або майже однакові на всіх сторінках, тому було б не дуже зручно в кожну нову HTML-сторінку додавати та налаштовувати ці блоки. Саме тому, хорошою вирішенням цього питання буде створити готові header та footer як окремі html-файли (рис. 3.16), а потім вже вставляти їх у потрібні сторінки за допомогою 1-2 рядків коду.

```
<div th:fragment="footer" xmlns:th="http://www.thymeleaf.org">
    <link rel="stylesheet" th:href="@{/../style/footer.css}">
    <footer class="footer">
        <div class="addressAndSite">
            <p>Volodymyr Romaniuk © 2024</p>
        </div>
        <div class="linkToEmail">
            E-mail:
            <a href="#">volodyaromanyuk@gmail.com</a>
        </div>
    </footer>
</div>
```

Рис. 3.16. Код блоку footer, записаний в окремому html-файлі

Для того, щоб вставляти ці фрагменти в інші веб-сторінки буде використаний Thymeleaf. За допомогою атрибуту `th:replace` можна всього за один рядок вставити необхідний фрагмент. В html-файлі це буде виглядати так:

```
<div th:replace="/fragments/navbar :: navbar"></div>
```

Тепер перейдемо до основної частини сторінок. Розглянемо створення сторінки з транзакціями. При переході на сторінку зі списком транзакцій

користувачу необхідно вивести усі його транзакції. Щоб отримати список транзакцій з бази даних, за допомогою JavaScript створимо та надішлемо HTTP-запит, який запросить з бази даних усі транзакцій для цього користувача (рис. 3.17).

```
29  const baseURL = 'http://localhost:8080/';
30  const requestURL = baseURL + 'transactions';
31  const xhr = new XMLHttpRequest();
32  xhr.open( method: 'GET', requestURL);
33  xhr.send();
```

Рис. 3.17. Фрагмент коду JavaScript файла, який створює та надсилає HTTP-запит

Як було визначено в REST API, для того, щоб отримати список всіх транзакцій потрібно надіслати GET-запит на ендпоінт `/transactions`. При успішній обробці запиту та отриманні відповіді, додаємо транзакції у таблицю, яка буде відображена на сторінці транзакцій. Подібним чином буде розроблені сторінки зі списком категорій та фінансовими звітами. Вигляд сторінки транзакцій зображений на рис. 3.18.

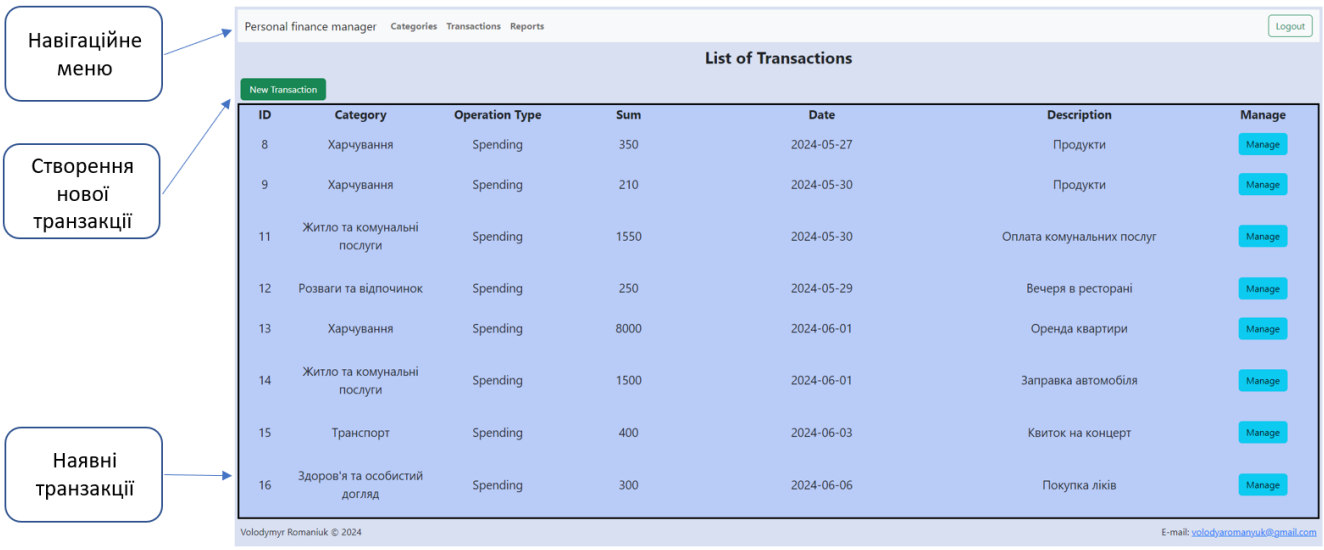


Рис. 3.18. Сторінка зі списком транзакцій

3.9. Висновки до розділу 3

У третьому розділі детально розв’язано питання проектування та реалізація програмного забезпечення для управління особистими фінансами. Розглянуто архітектурну структуру системи, засновану на Spring MVC, яка забезпечує чіткий розподіл функцій між моделлю, представленням та контролерами, а також дозволяє легко розробляти, тестувати та масштабувати систему.

В рамках проектування системи представлено ключові діаграми, покликані показати, як повинні функціонувати та взаємодіяти між собою її компоненти. Use case діаграма показала основні сценарії використання системи: створення транзакцій, управління категоріями витрат, можливість перегляду звітів. Діаграми діяльності показали логіку та порядок виконання ключових процесів. Модель бази даних дозволила чітко структурувати сховище даних, враховуючи всі сутності системи та їх взаємозв'язки, а контекстна діаграма дала уявлення про функціональний взаємозв'язок системи із зовнішнім середовищем. Створена діаграма класів для візуалізації об'єктно-орієнтованого підходу до реалізації системи.

Описано REST API: ключові ендпоінти, HTTP методи та формати передачі даних.

В роботі використані сторонні бібліотеки Lombok, JUnit, Hibernate, Thymeleaf, Bootstrap.

Розроблено та реалізовано роботу основних компонентів Spring Framework додатку, серед яких контролери, репозиторії, сервіси, сутності. Використано велику кількість анотацій, які заміняють звичне написання коду.

Описані основні технології та інструменти розробки користувацького інтерфейсу, такі як Bootstrap та Thymeleaf, структуру веб-сторінок та як користувач надсилає запити і взаємодіє з сервером.

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

4.1. Алгоритм дій для запуску програмного забезпечення

Для розгортання програмного забезпечення на власному пристрої необхідно виконати таку послідовність дій:

1. Завантажити проєкт з github-репозиторію (https://github.com/VolodymyrRomaniuk1/FinanceManager).
2. Завантажити та встановити PostgreSQL (https://www.postgresql.org/download).

3. Створити базу даних.

4. Щоб підключити базу даних, у файлі налаштувань application.properties (FinanceManager/src/main/resources/application.properties) встановити посилання на вашу базу даних та логін з паролем до бази даних.

5. Для запуску програми потрібно запустити FinanceManagerApplication.java (FinanceManager/src/main/java/org/financemanager/FinanceManagerApplication.java)

6. (Опціонально) Заповнити базу даних рядками для швидкої перевірки роботи програми запустивши SQL-скрипт data.sql (FinanceManager/src/main/ resources/data.sql).

4.2. Вимоги до апаратно-програмного забезпечення

Для запуску програми мають бути задоволені такі мінімальні системні вимоги [40-42]:

1. RAM 128МБ;
2. 128МБ місця на диску;
3. Процесор Intel Pentium 2 266МГц;
4. Java 8 або вище;

5. Java Runtime Environment;
6. Spring Framework 5.0 або вище.

4.3. Тестування програми вручну

Для демонстрації роботи програми пройдемось по його основному функціоналу.

При переході по посиланню на сайт, що у даному випадку localhost:8080, бо проєкт запущений локально, потрапляємо на індексну сторінку (рис. 4.1).

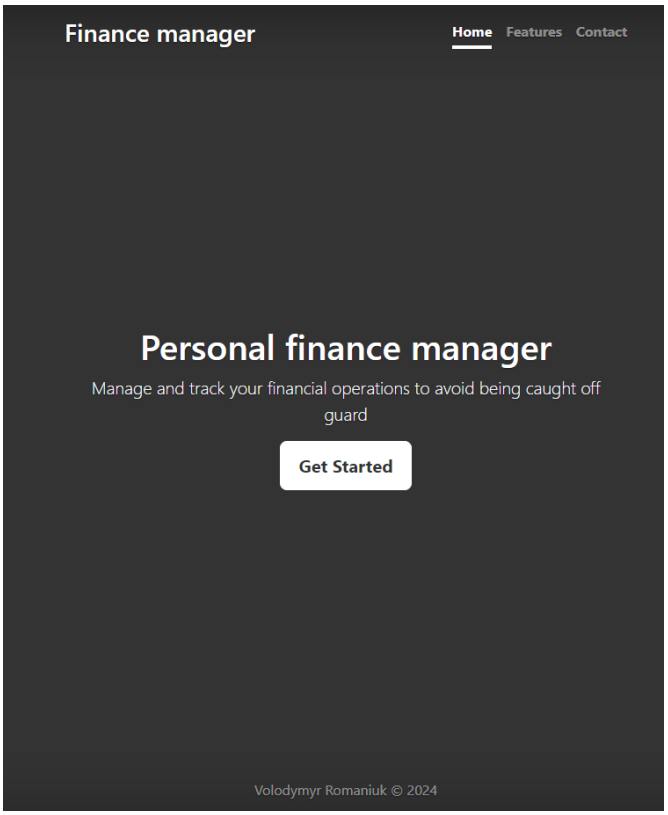


Рис. 4.1. Індексна сторінка додатку

При натисканні кнопки Get Started відбувається перехід до управління категоріями та транзакціями. До бази даних було додано декілька рядків у таблиці categories та transactions для демонстрації. При переході на сторінку категорій

бачимо список усіх категорій в базі даних (рис. 4.2). Тут можна переглянути інформацію про існуючі категорії, змінити їх або створити нові.

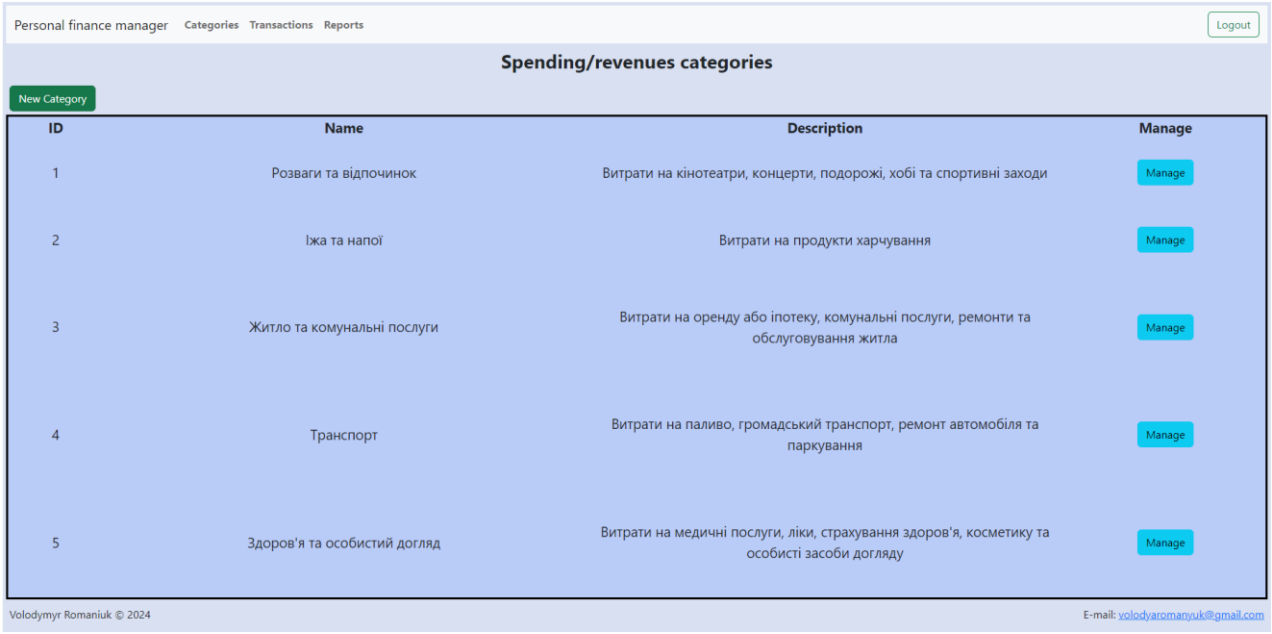


Рис. 4.2. Сторінка зі списком усіх категорій

Наприклад, спробуємо змінити категорію 2. Натискаючи на кнопку Manage біля неї, бачимо модальне вікно, в якому є два поля: Name і Description (рис. 4.3). Змінюючи наповнення цих полів можна змінити інформацію про категорію. Три кнопки знизу відповідають за скасування змін, підтвердження змін або видаляють категорію.

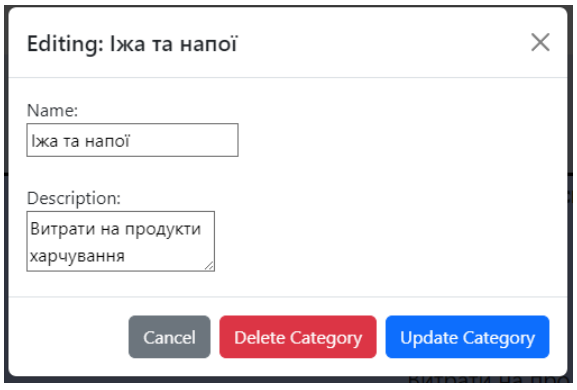


Рис. 4.3. Модальне вікно управління категорією

Спробуємо змінити ім’я та опис категорії: змінимо ім’я на «Харчування» та опис на «Витрати на продукти харчування» та натиснемо Update Category. Сторінка оновиться і, якщо не виникло помилок, дані оновляться (рис. 4.4).

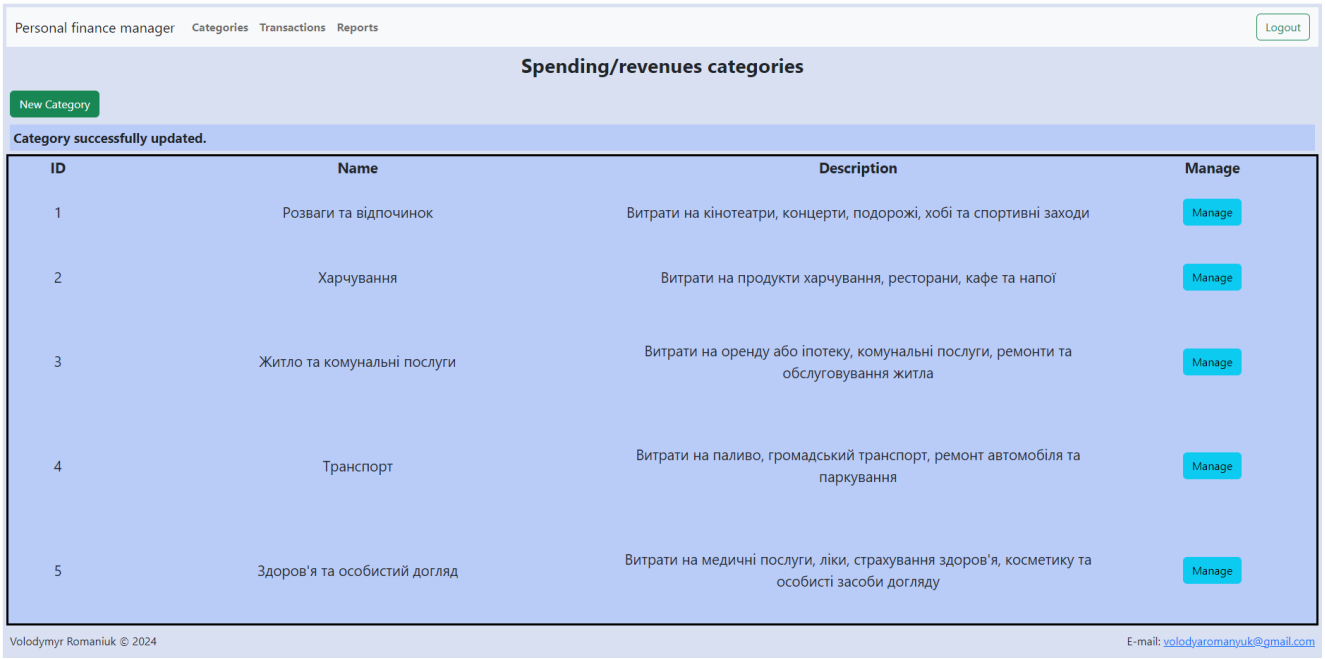


Рис. 4.4. Сторінка з оновленою категорією

Зміни відобразились у таблиці та з’явилось повідомлення про успішне оновлення. У базі даних зміни також відбулись (рис. 4.5).

id [PK] bigint	description character varying (255)	name character varying (150)
1	Витрати на кінотеатри, конц...	Розваги та відпочинок
2	Витрати на продукти харчув...	Харчування
3	Витрати на оренду або іпоте...	Житло та комунальні ...
4	Витрати на паливо, громадс...	Транспорт
5	Витрати на медичні послуги,...	Здоров'я та особисти...

Рис. 4.5. Рядки таблиці categories

Тепер перейдемо на сторінку з транзакціями, натиснувши відповідну кнопку в навігаційному меню зверху (рис. 4.6). Тут також є кілька транзакцій для демонстрації.

Рис. 4.6. Сторінка зі списком усіх транзакцій

Спробуємо видалити транзакцію з id 2, яка відноситься до категорії «Харчування». Натиснемо кнопку Manage поряд з необхідною транзакцією. Відкриється модальне вікно, подібне до вікна управління категоріями (див. рис. 4.3). У вікні бачимо поля Category, яке позначає до якої категорії відноситься транзакція, поле Operation Type, яке позначає тип транзакції (витрата/нарахування), поле Sum означає суму грошей в транзакції, поле Date позначає дату і поле Description – короткий опис транзакції (рис. 4.7). Для прикладу обрано категорію «Харчування», тип операції Spending (витрата), сума - 350 грн., дата проведення операції – 27.05.2024 та введено короткий опис – «Продукти».

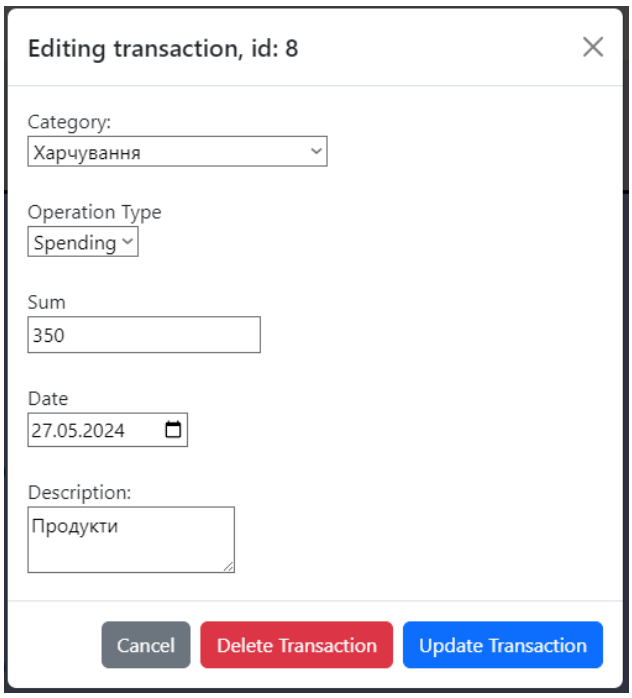


Рис. 4.7. Модальне вікно управління транзакцією

Натискаємо кнопку Delete Category знизу модального вікна. Якщо не виникло помилок, сторінка оновиться і можемо побачити, що категорія була успішно видалена і було отримане відповідне повідомлення (рис. 4.8).

Рис. 4.8. Сторінка після видалення транзакції

Тепер спробуємо створити нову транзакцію. Натискаємо кнопку “New Transaction” на сторінці перегляду транзакцій. Переходимо на нову сторінку, де користувачу необхідно ввести дані про транзакцію (рис. 4.9). Потрібно ввести коректні дані та натиснути кнопку «Create new transaction».

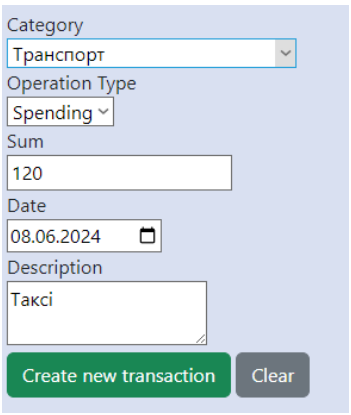


Рис. 4.9. Сторінка створення нової транзакції

Після введення всіх необхідних даних та підтвердження створення кнопкою «Create new transaction» знову повертаємось до списку всіх транзакцій (рис. 4.10). Для цього використовуємо кнопку в навігаційному меню зверху.

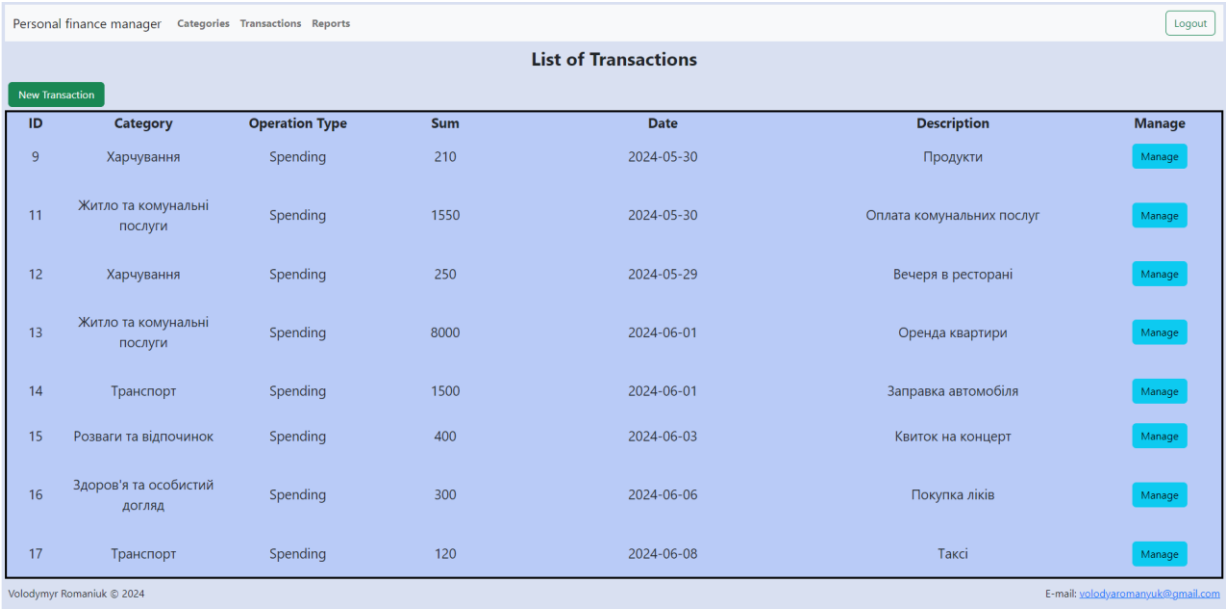


Рис. 4.10. Список із новою транзакцією

Утворилась нова транзакція з id 8, розташована в кінці списку. Можемо впевнитись в цьому подивившись в самій базі даних (рис. 4.11).

id [PK] bigint	date date	description character varying (255)	operation_type character varying (255)	sum double precision	category_id bigint
9	2024-05-30	Продукти	Spending	210	2
11	2024-05-30	Оплата комунальних ...	Spending	1550	3
12	2024-05-29	Вечеря в ресторані	Spending	250	2
13	2024-06-01	Оренда квартири	Spending	8000	3
14	2024-06-01	Заправка автомобіля	Spending	1500	4
15	2024-06-03	Квиток на концерт	Spending	400	1
16	2024-06-06	Покупка ліків	Spending	300	5
17	2024-06-08	Таксі	Spending	120	4

Рис. 4.11. Рядки таблиці transactions

Тепер протестуємо можливості створення фінансових звітів. Спробуємо створити звіт по витратам. Натиснувши відповідну кнопку в навігаційному меню зверху відбувається перехід на сторінку генерації звітів. Необхідні дані та можливі варіанти графіків зображені на рис. 4.12.

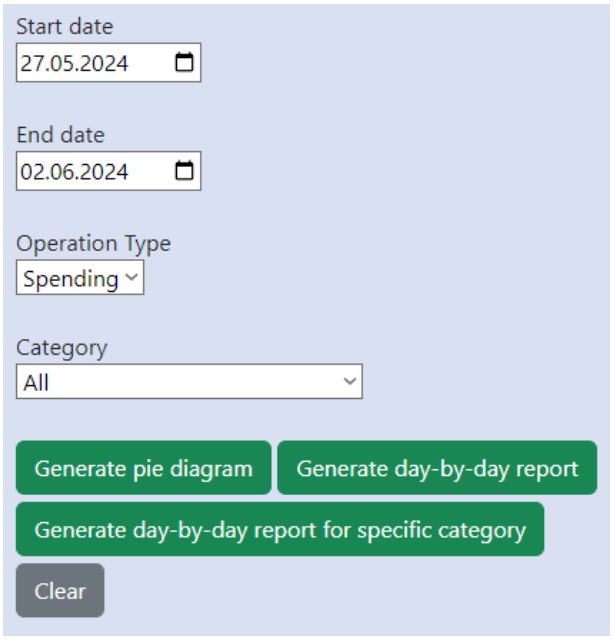


Рис. 4.12. Меню створення звітів

Для створення звіту потрібно встановити проміжок дат, тип операцій та категорію (опціонально). Для демонстрації створимо по-денний графік витрат для перших двох тижнів травня (рис. 4.13).

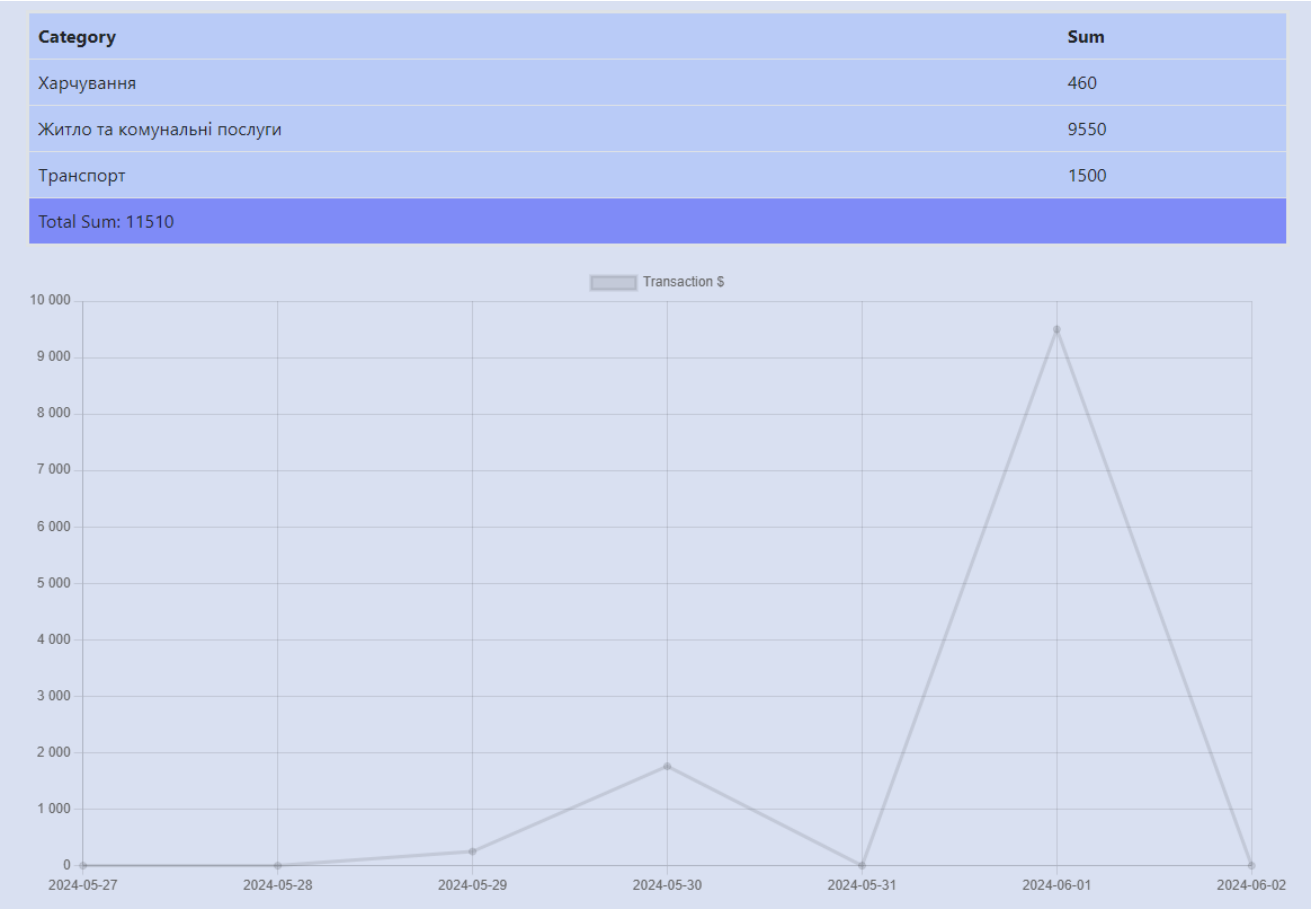


Рис. 4.13. Графік створений при натисканні кнопки Day-by-day report

4.4. Тестування програми за допомогою автоматизованих засобів

Проводячи тестування вручну шляхом показу роботи програми на скріншотах та описування послідовності кроків, які були виконані, була перевірена функціональність веб-додатку, а також отримане візуальне уявлення про те, як користувачі будуть взаємодіяти з графічним інтерфейсом.

Однак, щоб остаточно впевнитись у правильності роботи всіх функцій системи цього недостатньо. У всіх сучасних проектах автоматизовані засоби

тестування є невід'ємною частиною процесу розробки, вони дозволяють дуже швидко, надійно та ефективно перевірити весь функціонал в повному обсязі, виключаючи людський фактор та гарантуючи достовірність результатів тестування. Ще одна їх зручність в тому, що їх можна автоматично запускати при кожному новому запуску додатку.

За допомогою фреймворку JUnit, для тестування розроблюваного веб-додатку було написано 32 тести, які охоплюють весь функціонал найважливіших частин додатку – контролерів та сервісів, де відбуваються обробка HTTP-запитів та робота з даними.

На рис. 4.14 представлено список цих тестів та результати їх виконання. Можемо побачити, що вони всі успішно виконуються, що підтверджує коректну роботу веб-додатку.

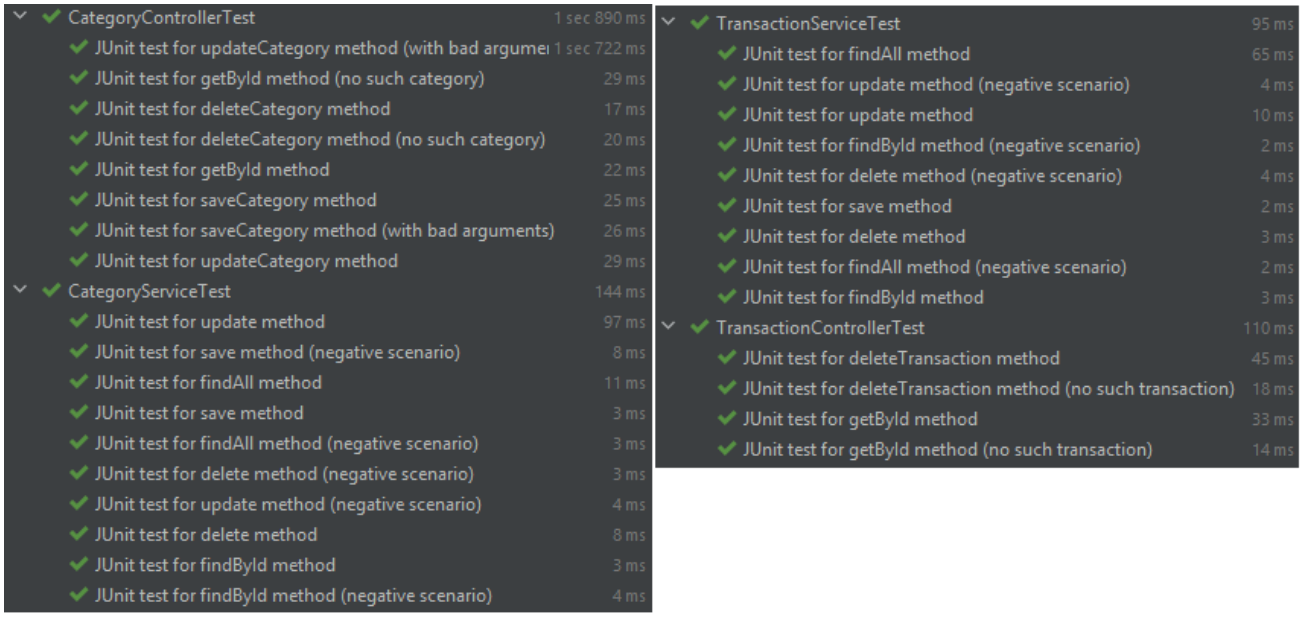


Рис. 4.14. Список тестів та результати їх проходження

4.5. Висновки до розділу 4

В четвертому розділі проведено тестування роботи веб-додатку. Розроблене програмне забезпечення характеризується низькими системними вимогами, що

забезпечує стабільну роботу додатку навіть на застарілих операційних системах і апаратних платформах.

Створено тестовий фінансовий звіт, який показує витрати з кожної категорії та графік витрат по днях.

Тестування функціоналу проводилося із використанням бібліотеки JUnit, що дозволило перевірити коректність роботи REST API, основних бізнес-логік, операцій із базою даних та інших компонентів. Розроблені автоматизовані тести дали змогу оперативно виявити та усунути потенційні помилки в коді, що значно підвищило загальну стабільність та якість додатку.

ВИСНОВКИ

У зв’язку з досить високим попитом на використання систем для менеджменту фінансів прийнято рішення розробити новий додаток для управління персональними фінансами. Розроблено модель структури системи і алгоритм взаємодії її компонентів, змодельовано основний функціонал та визначено методи його імплементації.

Для опису структури, бізнес-логіки та процесів, що відбуваються у системі використані UML діаграми: use case діаграма для моделювання основного функціоналу; activity діаграми для опису алгоритмів та послідовності дій окремих процесів; модель бази даних для опису структури та таблиць бази даних. Також прийнято використовувати REST API та конкретизовано його поведінку.

Розробку серверної частини виконано на мові Java з використанням Spring Framework як програмного каркасу, а розробку користувацького інтерфейсу засобами HTML/CSS/JavaScript. Завдяки Spring Framework системна структура додатку була модульною, що дозволило легко організовувати процес розробки та полегшило майбутнє обслуговування.

Розроблений веб-додаток для управління персональними фінансами. Спілкування користувача із системою відбувається через графічний інтерфейс надсилаючи HTTP-запити. Серед основних функцій додатку є запис транзакцій з інформацією про них, розподіл транзакцій по категоріях, які користувач також може створити сам, та можливість створювати фінансові звіти за допомогою збережених даних, звідки можна дізнатись розподіл витрат по категоріях, кількість витрат кожного дня, загальну динаміку витрат у часовому проміжку. Такий функціонал допоможе користувачу аналізувати свої витрати, планувати майбутні витрати та помітити свої фінансові звички.

Розроблена платформа відзначається гнучкістю та універсальністю, що дозволяє адаптувати її для індивідуальних користувачів і малих бізнесів.

Запропоновано нову архітектуру програмного забезпечення, яка поєднує принципи об'єктно-орієнтованого програмування, REST API та сучасні бібліотеки.

Розроблений веб-додаток рекомендований для використання індивідуальним користувачам, що прагнуть ефективно управляти своїми фінансами. Це дасть можливість ефективно використати розроблену систему для ефективної організації та управління фінансового обліку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Atkinson, A. F., & Messy, F. “Measuring Financial Literacy: Results of the OECD / International Network on Financial Education (INFE) Pilot Study,” // OECD Working Papers on Finance, Insurance and Private Pensions – 2012 - №15 – С. 10–33.
2. Behrman, Jere R., Mitchell, Olivia S., Soo, Cindy K. and Bravo, D. “How Financial Literacy Affects Household Wealth Accumulation,” // 2012 AER Papers and Proceedings
3. Bhattacharjee, B. J. “Financial Literacy And Its’ Influencing Factors: An Empirical Study of Investors,” // International Journal Of Research In Commerce, IT & Management [Електронний ресурс] – Режим доступу: http://ijrcm.org.in/
4. Куцяк В. О. Особисті фінанси в умовах розвитку ринкових відносин в Україні [08.00.08 «Гроші, фінанси і кредит»] / В. О. Куцяк. – ТНЕУ, 2016. – 220 с.
5. Calderon. M. “The Role of Financial Literacy and of Financial Education intervention in developing countries,” // DIW Roundup #34 – 2014. - DIW Berlin – Deutsches Institut für Wirtschaftsforschung
6. Манцуров І. Г. Методологія статистичного оцінювання економічного зростання та конкурентоспроможності країни [Текст] / Манцуров І. Г. // КНЕУ – 2006. - 306 с.
7. Слепов В. А., Екшембіев Р. С. Персональні фінанси [Текст] / Слепов В. А., Екшембіев Р. С. // Фінанси і кредит - № 40 (280) - 2007. - С. 2-7.
8. What Is Personal Finance, and Why Is It Important? [Електронний ресурс]. – Режим доступу: https://www.investopedia.com/terms/p/personalfinance.asp
9. Master Your Money: The Ultimate Guide to Personal Finance Management [Електронний ресурс]. – Режим доступу: https://qsalary.com/blog/personal-finance-management/

10. Robb, C. A. & Wodyard, A. S. “Financial Knowledge and Best Practice Behaviors,” // Journal of Financial Counseling and Planning – 2011 - Vol. 22 (1), C. 60-70
11. McCormick, Martha. H. The Effectiveness of Youth Financial Education: A Review of the Literature // Journal of Financial Counseling and Planning – 2009 - Volume 20, Issue 1 - C. 70-83
12. Nalini, G.S. “Financial Literacy of Micro, Small and Medium Entrepreneurs,” // International Journal of Management Research and Review – 2011 - Vol.1 (5) / №9 - C. 189-197
13. D. Bialaszewski et al. Finance requirements and computer utilization at AACSB accredited schools // Financial Practice and Education – 1993.
14. MoneyWiz 2023 - personal finance [Електронний ресурс]. – Режим доступу: https://www.wiz.money.
15. Monefy Handy personal finance management tool for iOS and Android [Електронний ресурс]. – Режим доступу: https://monefy.me.
16. CoinKeeper - best budgeting app, money manager, personal finance, budget & expenses [Електронний ресурс]. – Режим доступу: https://about.coinkeeper.me/eng.
17. Get complete control over your money - Mobills [Електронний ресурс]. – Режим доступу: https://www.mobillsapp.com.
18. Money Manager & Budget Planner | Spendee [Електронний ресурс]. – Режим доступу: https://www.spendee.com.
19. Кізима, Т. О. Особисті (персональні) фінанси: необхідність виокремлення та перспективи розвитку в умовах ринку [Текст] / Т. О. Кізима // Актуальні проблеми економіки. – 2008. – № 11. – С. 194-203

20. Кушнір, Ю. Б. Необхідність виділення персональних фінансів в окремий об'єкт дослідження [Текст] / Ю. Б. Кушнір, Р. Р. Буяк // Наукові записки Львівського університету бізнесу та права. – 2012. – Вип. 8. – С. 234-237.

21. Бурлаков А. А. Організація гнучкого доступу до даних в додатках на java-платформі / А. А. Бурлаков // Вісник ХНУ. – 2016. – № 2. – С. 30–36.

22. Cruzes, D.S., Dybå, T. Research synthesis in software engineering: a tertiary study. // Inf. Softw. Technol. – 2011 - №53(5) - С. 440–455

23. Why use a UML diagram? [Електронний ресурс]. - Режим доступу : https://www.lucidchart.com/pages/uml-use-case-diagram

24. What is Activity Diagram? [Електронний ресурс]. - Режим доступу : https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/

25. What is a logical data model? [Електронний ресурс]. - Режим доступу : https://www.tibco.com/glossary/what-is-a-logical-data-model

26. Presley, A.; Liles, D.H. The use of IDEF0 for the design and specification of methodologies. // In Proceedings of the 4th Industrial Engineering Research - Conference, Nashville, TN, USA - 24-25 May 1995 - С. 442–448

27. .Який Java Web фреймворк обрати? [Електронний ресурс]. // Спільнота програмістів. DOU : портал. – Режим доступу : https://dou.ua/forums/topic/11227/.

28. Офіційна документація. Spring [Електронний ресурс]. – Режим доступу : http://spring.io/docs.

29. Rosenstock, L. "OpenAPI and Design-First Principles", [Електронний ресурс]. – Режим доступу до матеріалу: https://stoplight.io/blog/openapi-and-design-first-principles-96e7c4b2aec1/

30. Вдовичин Т. Я., Лазурчак Л. В. Проєктування інформаційно-пошукових систем як засіб використання сучасних технологій // Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. Том 33 (72). № 4. 2022. 66-71.

31. Шакуров Є. О. Сучасні тенденції побудови веб-сайтів / Є. О. Шакуров, В. К. Пономарьова // Наумовські читання : матеріали XIX наук.-метод. конф. здобувачів вищої освіти та молодих учених, присвяч. року мат. освіти в Україні, Харків, 23-24 листоп. 2021 р. / Харків. нац. пед. ун-т ім. Г. С. Сковороди ; [редкол.: Н. О. Пономарьова та ін.]. – Харків : [Б. в.], 2022. – С. 242–246.

32. Базюк Р. С. Порівняльний аналіз середовищ програмування мовою Java / Р. С. Базюк, С. В. Завгородній, А. В. Ковтун // Збірник наукових праць «Актуальні питання природничо-математичної освіти». – 2017. – № 9. – С. 155–159.

33. Муляр І.В. Аналіз підходів до структурної збірки web-додатків / І. В. Муляр, В. М Лоза, С. Б. Войнарович // Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка. - К.: ВІКНУ, 2017. - Вип. № 56.- С. 132- 138.

34. Офіційна документація. Apache Maven [Електронний ресурс]. – Режим доступу: https://maven.apache.org/guides/index.html.

35. Офіційна документація. Project Lombok [Електронний ресурс]. – Режим доступу: https://projectlombok.org/features/.

36. Офіційна документація. JUnit [Електронний ресурс]. – Режим доступу: https://junit.org/junit5/docs/current/api/.

37. Офіційна документація. Hibernate [Електронний ресурс]. – Режим доступу: https://hibernate.org/orm/documentation/6.2/.

38. Офіційна документація. Bootstrap [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/docs/4.1/getting-started/introduction/

39. Офіційна документація. Thymeleaf [Електронний ресурс]. – Режим доступу: https://www.thymeleaf.org/documentation.html.

40. What are the system requirements for Java? [Електронний ресурс]. – Режим доступу: https://www.java.com/en/download/help/sysreq.html.

41. Spring Boot 2.0.0. System Requirements [Електронний ресурс]. – Режим доступу: https://docs.spring.io/spring-boot/docs/2.0.0.RELEASE/reference/html/getting-started-system-requirements.html.

42. System requirements IntelliJ IDEA [Електронний ресурс] // IntelliJ IDEA Help. – Режим доступу: https://www.jetbrains.com/help/idea/prerequisites.html.

ДОДАТКИ

ДОДАТОК А

Програмний код елементу FinanceManagerApplication.java

org\financemanager\FinanceManagerApplication.java

```
package org.financemanager;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class FinanceManagerApplication {
    public static void main(String[] args) {
        SpringApplication.run(FinanceManagerApplication.class,
args);
    }
}
```

ДОДАТОК Б

Програмний код елементу TransactionController.java

org\financemanager\controller\TransactionController.java

```
package org.financemanager.controller;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.financemanager.entity.Transaction;
import org.financemanager.service.TransactionService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;

@RestController
@RequestMapping("/transactions")
public class TransactionController {

    public static final Logger logger =
LogManager.getLogger(TransactionController.class);

    private final TransactionService transactionService;

    @Autowired
    public TransactionController(TransactionService
transactionService) {
        this.transactionService = transactionService;
    }

    @GetMapping
    @PreAuthorize("hasAuthority('transactions:read')")
    public ResponseEntity<List<Transaction>> findAll(Model model){
```

```
        logger.info("Getting transactions list");
        List<Transaction> transactions =
transactionService.findAll();
        model.addAttribute("transactionsList", transactions);
        return new ResponseEntity<>(transactions, HttpStatus.OK);
    }

    @GetMapping("{id:[\\d]+}")
    @PreAuthorize("hasAuthority('transactions:read')")
    public ResponseEntity<Optional<Transaction>>
getById(@PathVariable("id") Long id){
        logger.info("Getting transaction by id " + id);
        return new ResponseEntity<>(transactionService.findById(id),
HttpStatus.OK);
    }

    @PostMapping
    @PreAuthorize("hasAuthority('transactions:write')")
    public ResponseEntity<Transaction>
saveTransaction(@ModelAttribute("transaction") @Valid Transaction
transaction, BindingResult bindingResult) {
        logger.info("Creating new transaction");
        if(bindingResult.hasErrors()){
            logger.error("Provided transaction has errors");
            logger.info(bindingResult.getAllErrors());
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        return new
ResponseEntity<>(transactionService.save(transaction),
HttpStatus.CREATED);
    }

    @PutMapping("{id:[\\d]+}")
```

```
    @PreAuthorize("hasAuthority('transactions:write')")
    public ResponseEntity<String> updateTransaction(@PathVariable
Long id, @RequestBody @Valid Transaction transaction, BindingResult
bindingResult){
        logger.info("Updating transaction id " + id);
        if(bindingResult.hasErrors()){
            logger.error("Provided transaction has errors");
            logger.info(bindingResult.getAllErrors());
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        transactionService.update(id, transaction);
        logger.info("Transaction id " + id + " successfully
updated");
        return new ResponseEntity<>("Transaction successfully
updated.", HttpStatus.OK);
    }

    @DeleteMapping("{id:[\\d]+}")
    @PreAuthorize("hasAuthority('transactions:write')")
    public ResponseEntity<String>
deleteTransaction(@PathVariable("id") Long id){
        logger.info("Deleting transaction id " + id);
        transactionService.delete(id);
        logger.info("Transaction id " + id + " deleted");
        return new ResponseEntity<>("Transaction deleted.",
HttpStatus.OK);
    }
}
```

ДОДАТОК В

Програмний код елементу Category.java

org\financemanager\entity\Category.java

```
package org.financemanager.entity;

import lombok.*;
import org.hibernate.annotations.GenericGenerator;

import javax.validation.constraints.NotBlank;

@Builder
@AllArgsConstructor
@NoArgsConstructor
@Data
@With
@Entity
@Table(name = "categories")
public class Category implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotBlank(message = "Name must not be empty")
    @Size(min = 2, max = 150, message = "Name length must be 2 to
150 symbols")
    @Column(name = "name", nullable = false, unique = true)
    private String name;
    @Column
    private String description;
}
```

ДОДАТОК Г

Програмний код елементу TransactionRepo.java

org\financemanager\repository\TransactionRepo.java

```
package org.financemanager.repository;

import org.financemanager.entity.Category;
import org.financemanager.entity.Transaction;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.sql.Date;
import java.util.List;

public interface TransactionRepo extends JpaRepository<Transaction,
Long> {
    @Query("select t from Transaction t order by t.id asc")
    List<Transaction> findAll();

    List<Transaction> findAllByDateBetween(Date dateStart, Date
dateEnd);

    List<Transaction> findAllByDateBetweenAndOperationType(Date
dateStart, Date dateEnd, String operationType);

    List<Transaction>
findAllByDateBetweenAndOperationTypeAndCategory(Date dateStart, Date
dateEnd, String operationType, Category category);
}
```

ДОДАТОК Д

Програмний код елементу TransactionService.java

org\financemanager\service\TransactionService.java

```
package org.financemanager.service;
import org.financemanager.entity.Category;
import org.financemanager.entity.Transaction;
import java.sql.Date;
import java.util.List;
import java.util.Optional;

public interface TransactionService {
    List<Transaction> findAll();

    List<Transaction> findAllByDateBetween(Date dateStart, Date
dateEnd);

    List<Transaction> findAllByDateBetweenAndOperationType(Date
dateStart, Date dateEnd, String operationType);

    List<Transaction>
findAllByDateBetweenAndOperationTypeAndCategory(Date dateStart, Date
dateEnd, String operationType, Category category);

    Optional<Transaction> findById(Long id);
    Transaction save(Transaction transaction);
    Transaction update(Long id, Transaction transaction);

    void delete(Long id);
}
```